Karlsruher Institut für Technologie

Performance Isolation in Multi-Tenant Applications

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Rouven Krebs
aus Speyer, Deutschland

Tag der mündlichen Prüfung:	26.6.2015
Erstgutachter:	Prof. Dr. Samuel Kounev
Zweitgutachter:	Prof. Dr. Ralf Reussner

DOI: 10.5445/IR/1000049479

Hiermit versichere ich wahrheitsgemäß, die Dissertation selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten Anderer und eigener Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, den 28.04.2015

Rouven Krebs

Abstract

Multi-tenancy is an approach to share one application instance among multiple customers by providing each of them a dedicated view. This approach reduces the costs for service provisioning. Tenants expect to be isolated in terms of the application performance they observe, and the providers' inability to offer performance guarantees is a major obstacle for potential cloud customers. A direct relation between observed performance and customer satisfaction is well documented for interactive web applications. Performance can even become critical to economic success.

Providing performance isolation when sharing at the application layer is a challenge. The layers responsible for controlling resource access (e.g., operating system) are normally unaware of entities defined at the application level. Thus, these layers cannot distinguish between different tenants. Furthermore, it is hard to predict how tenant requests propagate through the multiple layers of the execution environment, down to the physical resource layer. The intended decoupling of the application from the resource controlling layers does not allow moving the resource control to the application.

Nowadays, no methods exist to efficiently enforce performance isolation between tenants sharing one multi-tenant application. Furthermore, the providers know neither which characteristics of the methods are important nor which method provides the best isolation. Thus, quantifying the degree of performance isolation a system achieves is essential. Additionally, there is no feasible documented approach of how potential performance isolation methods can be implemented in an application.

This thesis aims at providing performance isolation mechanisms, a methodology to quantify performance isolation, as well as insights and methods that support multi-tenant application providers in realizing performance isolated multi-tenant applications.

Performance isolation is enforced by means of request based admission control. The degree of performance isolation is quantified by using a benchmarking approach. Its metric captures the influence of an increasing workload from one tenant upon the observed performance of other tenants. To support providers in applying the correct isolation method, the solutions are evaluated using the proposed benchmarking approach. Beyond this, additional relevant aspects of the methods, e.g., their informational requirements, are identified. To implement the isolation method, a framework is considered that allows to easily add the application-specific aspects of a concrete isolation method.

Finally, the results of the thesis are validated in environments motivated by a real world PaaS runtime, using representative benchmarks. In an enterprise service, it was possible to reduce one tenant's impact upon the response time of another from 141% to 11% by applying methods presented in the thesis. The benchmarking approach was additionally validated in a case study that quantified the Xen hypervisor's isolation capabilities.

Kurzfassung

Cloud Computing gewinnt im Bereich der Geschäftsanwendungen immer mehr an Bedeutung. In infrastrukturbezogenen Angeboten wird die gleiche Hardware zwischen mehreren Kunden geteilt. Hierfür werden Virtualisierungstechniken verwendet, welche jedem Kunden eine virtuelle Maschine zur Verfügung stellen. Dadurch lassen sich Ressourcen teilen und einzelnen Kunden direkt zuweisen.

Mehrmandantenfähigkeit teilt eine Applikationsinstanz zwischen mehreren Kunden, wobei jeder seine eigene und isolierte Sicht auf die Anwendung erhält. Dies erhöht den Anteil der geteilten Ressourcen, da nicht nur die Hardware, sondern auch darüberliegende Schichten geteilt werden. Somit können Kosten weiter gesenkt werden. Trotz dieses erhöhten Anteils geteilter Ressourcen erwarten die Mandanten, bezüglich der Performance von anderen Mandanten isoliert zu sein.

Die Unfähigkeit der Anwendungsanbieter Performancegarantien sicherzustellen, ist ein Hemmnis für potentielle Kunden. Speziell im Falle interaktiver Web-Anwendungen ist der Zusammenhang von Performance und Benutzerzufriedenheit bekannt, und schlechte Performance kann den ökonomischen Erfolg einer Anwendung verhindern.

Im Fall einer geteilten Infrastruktur sind sowohl die Ressourcen, als auch die Entitäten, die voneinander getrennt werden sollen, bekannt. Performanceisolation bei einer geteilten Anwendungsschicht zu garantieren, ist im Gegensatz dazu eine Herausforderung mit speziellen Aspekten. Die ressourcenkontrollierenden Ausführungsumgebungen, wie z.B. das Betriebssystem, haben in der Regel keine Kenntnisse über Entitäten, welche auf Ebene der Applikationsinstanz definiert sind. In der Regel ist das Mandantenkonzept jedoch nur der Anwendung oder bestenfalls der Middleware bekannt. Somit können die ressourcenkontrollierenden Schichten nicht zwischen Mandanten unterscheiden. Desweiteren ist es schwer vorherzusagen, wie eine Anfrage durch die verschiedenen Schichten des Systems bis hin zu den Ressourcen propagiert wird. Anwendungen sind absichtlich von der Ressourcenverwaltung getrennt, um eine effiziente Entwicklung zu ermöglichen. Deswegen erscheint es nicht schlüssig, die Ressourcenkontrolle in die Anwendung zu integrieren. Diese Schichtendiskrepanz verhinderte bislang die Performanceisolation von Mandanten, welche sich eine Applikationsinstanz teilen.

Einem Anbieter von Mehrmandantenanwendungen fehlen die Methoden, um Performanceisolation sicherzustellen. Desweiteren wissen die Anbieter nicht, welche Charakteristiken der Methoden wichtig wären, noch welche Methoden die beste Isolation bieten würden. Deshalb ist es essentiell, im ersten Schritt Performanceisolation quantifizieren zu können. Erschwerend kommt hinzu, dass kein umfassender Ansatz dokumentiert ist, der beschreibt, wie die Methoden zur Sicherstellung der Performanceisolation implementiert werden können.

Diese Dissertation hat zum Ziel, Methoden zur Performanceisolation zu entwickeln, Ansätze zur Quantifizierung von Performanceisolation zu erstellen, und Wissen zu erarbeiten, das die Anbieter von interaktiven Webanwendungen befähigt, isolierte Anwendungen zu erstellen.

Ein Einfluss zwischen zwei Mandanten tritt auf, wenn diese gleichzeitig im Wettbewerb um eine Ressource stehen. In einer interaktiven Anwendung beeinflussen sich Mandanten somit immer dann, wenn ihre Anfragen gleichzeitig vom System bearbeitet werden. Durch Einflussnahme wann und ob eine eingehende Anfrage bearbeitet wird, lässt sich somit kontrollieren, welcher Mandant auf eine Ressource zugreifen kann. In der vorliegenden Arbeit wird Performanceisolation durch eine anfragebasierte Zugangskontrolle gewährleistet. Diese basiert auf zwei Bestandteilen. Zum einen stellt ein arbeitserhaltender Scheduler sicher, dass die Anzahl der zugelassenen Anfragen pro Mandant einem konfigurierten Verhältnis entsprechen. Zum anderen stellt eine asynchron arbeitende Optimierung sicher, dass die Verhältnisse so angepasst werden, dass möglichst alle Mandanten eine faire Zuteilung der Ressourcen erhalten. Dies beinhaltet unter anderem auch eine Erhöhung des Anteils eines Mandanten, wenn andere Mandanten ihre Performancegarantien dadurch nicht verletzen, um nicht unnötig Ressourcen brachliegen zu lassen.

Die Zweiteilung erlaubt es, in kurzfristigen Zeitabständen Isolation sicherzustellen, sowie in langfristigen Zyklen komplexere Optimierungen vorzunehmen, welche für jede Anfrage einen zu hohen Mehraufwand bedeuten würden.

Zwei konkrete, neuartige Verfahren werden in der Arbeit vorgestellt. Das erste basiert darauf, den Ressourcenbedarf einer Anfrage abzuschätzen. Dieses Wissen wird genutzt, um im Scheduler sicherzustellen, dass ein Mandant nicht mehr als den ihm zugewiesenen Anteil einer Ressource nutzen darf. Dies erlaubt es, Mandanten effektiv zu isolieren, falls der Ressourcenbedarf pro Anfrage sich zwischen den Mandanten stark unterscheidet. Ein zweites Verfahren beobachtet die Performancedaten der Anwendung, und leitet darauf basierend ein Blackbox-Modell der Anwendung ab. Dieses wird genutzt, um mit Hilfe einer analytischen Optimierung, eine Gewichtung für ein faires Queuing zu ermitteln. Das letztere Verfahren erlaubt es, Garantien bezogen auf extern sichtbare Performanceeigenschaften zu erhalten.

Zur Ermittlung der Isolationsqualität wird ein benchmarkorientierter Ansatz vorgestellt. Dieser basiert auf der Separierung der Mandanten in zwei Gruppen. Die störende Gruppe, welche die maximal erlaubte Last überschreitet, und der rechtschaffenden Gruppe von Mandanten, welche die erlaubte Last nicht überschreiten.

Mit dieser Grundlage lassen sich zwei Typen von Metriken erstellen. Zum einen wird in der vorliegen Arbeit der Einfluss von steigender Arbeitslast eines Mandanten auf die beobachtete Performance eines anderen ins Verhältnis gesetzt. Zum anderen wird ein Messansatz vorgestellt, bei dem Lastkombinationen ermittelt werden, für welche die beobachtete Performance der rechtschaffenden Mandanten unverändert bleibt. Aus dem Verhältnis der Lasten beider Gruppen lässt sich die Performance zwischen vollständig isoliert und absolut nicht isoliert einordnen. Weitere Metriken umfassen die Anpassungsgeschwindigkeit der Verfahren im Falle einer Laständerung.

Zur Unterstützung des Entwicklers einer mehrmandantenfähigen Anwendung wurden die Isolationseigenschaften verschiedener Isolationsverfahren analysiert. Die Ergebnisse werden in der Arbeit vorgestellt. Darüber hinaus werden weitere relevante Aspekte, wie z.B. benötigte Informationen der Isolationsmethoden, besprochen. Zusätzlich sind Designempfehlungen für die Anwendung und Implementierung der Isolationsmethoden in der Arbeit enthalten. Mithilfe eines in der Arbeit vorgestellten Prozesses, basierend auf dem Analytical Hierarchy Process, kann auch ein unerfahrener Entwickler das passende Isolationsverfahren für sein Szenario auswählen. Für die konkrete Implementierung der Methoden wird ein Framework vorgeschlagen, das es erlaubt, die anwendungsspezifischen Aspekte und konkreten Methoden schnell einzubauen.

Alle Experimente der Arbeit wurden auf Systemlandschaften durchgeführt, welche durch eine reale Mehrmandantenfähigkeit unterstützende Platform-as-a-Service motiviert ist. Dies umfasst eine Virtualisierung der Hardware, um einzelne Anwendungen voneinander zu trennen, sowie den Einsatz realistischer Technologien, wie z.B. des SAP Hana Cloud Platform Laufzeitcontainers.

Eine repräsentative mehrmandantenfähige Benchmarkanwendung wurde hierfür entwickelt, und wird ebenfalls in der Arbeit vorgestellt. Darüber hinaus zeigt eine Fallstudie basierend auf einer realen Anwendung die Anwendbarkeit der vorgestellten Lösungen.

Die Anwendung der Ergebnisse aus dieser Arbeit erlaubt es, den Einfluss eines Mandanten auf die Performance eines anderen einzuschränken. Dies erhöht die Qualität der angebotenen Dienste. Somit besteht die Möglichkeit, höherwertigere SLAs zu garantieren und einen Wettbewerbsvorteil zu erhalten. Weiterhin ermöglichen die vorgestellten Ansätze, die Ressourcenpuffer für Lastspitzen zu reduzieren, und somit die Effizienz der Systeme zu erhöhen. Weiterhin bietet sich die Möglichkeit, die Performance einzelner Mandanten zu kontrollieren und die Grundlage für Elastizität auf Mandantenbasis einzuführen.

In einer konkreten Fallstudie, konnte der der Einfluss eines Mandanten auf einen anderen Mandanten von 141%, im nicht isolierten Fall, auf 11% gesenkt werden. Der benchmarkorientierte Ansatz wurde zudem in einer Fallstudie verwendet, welche die Isolationsfähigkeit der Xen Virtualisierung untersucht.

Zusammengefasst bietet die vorliegende Arbeit Methoden, um einzelne Mandanten, welche eine Anwendungsinstanz teilen, bezüglich ihrer Performance zu isolieren. Hierfür wird eine anfragebasierte Zugangskontrolle verwendet. Darüber hinaus werden Methoden zur Messung der Isolation vorgestellt, und die Verfahren entsprechend analysiert. Weitere Beiträge der Arbeit ermöglichen es auch unerfahrenen Entwicklern, ein passendes Isolationsverfahren für ein gegebenes Szenario auszuwählen und zu realisieren.

Danksagung

Viele Personen haben zum Erfolg dieser Arbeit beigetragen. Sicherlich hätte diese Arbeit mehr Zeit in Anspruch genommen, ohne einen Doktorvater, der sich so viel Zeit nimmt, und dem die Ausbildung seiner Doktorranden so wichtig ist, wie Prof. Dr.-Ing. Samuel Kounev. Von seinen Rückmeldungen und Diskussionen konnte ich vieles lernen. Ich danke ihm vielmals für sein Engagement, trotz stressigen Zeiten und seine Unterstützung.

Eine Dissertation ist nicht möglich ohne einen zweiten Gutachter. In diesem Fall bot mir dieser auch die Möglichkeit, mich als volles Mitglied seiner Forschungsgruppe zu fühlen. Das regelmäßige Feedback und die Diskussionen mit Prof. Dr. rer. nat. Ralf Reussner und den Mitgliedern des SDQ am Karlsruher Institut für Technologie verbesserte meine Forschung in vielerlei Hinsicht.

Der Rückhalt und die Diskussionen in einer Forschungsgruppe sind wichtig, um eine Arbeit zu stärken und kreative Gedanken einzubringen. Die ehemalige Descartes Forschungsgruppe, inzwischen Softwareengineering II der Julius-Maximilians-Universität Würzburg, hat diese Rolle größtenteils übernommen. Besonders hervorzuheben ist Nikolas Herbst, der trotz knapper Zeit viel Feedback zu meiner Dissertation gab. Danke auch den anderen Mitgliedern: Fabian Broßig, Nikolaus Huber, Joakim v. Kistowski, Aleksandar Milenkoski, Piotr Rygielski, Simon Spinner und Jürgen Walter.

In der Anfangszeit eines Promotionsvorhabens ist das Fokussieren auf ein Thema schwer und Unsicherheiten sind gegeben. Christof Momm hat mir dabei geholfen, mein Forschungsfeld zu definieren, und dieses in Einklang mit Ansprüchen aus verschiedenen Projekten zu bringen.

Danke gilt auch den Studenten Bernhard Neu, Undral Ganbat, Manuel Lösch, Nadia Ahmed, Arpit Mehta, Mehran Saliminia, Philipp Schneider, Florian Gauger, Corinna Malthaner und Elena Pyatkova für die förderlichen Diskussionen, und für ihre mühevolle Umsetzung vieler Ideen.

Die SAP SE und mein Vorgesetzter Wolfgang Theilmann gaben mir jahrelang Unterstützung und den benötigten Freiräume für meine Forschung. Auch meinen Kollegen bei der SAP SE danke ich für ihr zahlreiches Feedback und die angenehme Zeit.

Neben den Personen, die unmittelbar mit den Inhalten meiner Forschung in Kontakt kamen, haben einige schon vor vielen Jahren, und während der Zeit als Doktorand, die unverzichtbaren Grundlagen für das Gelingen meiner Promotion gelegt. Ohne diese Personen, wäre diese Arbeit nicht zu Stande gekommen. Eine ganz besondere Person ist meine Frau Tanja, welche mich über die Jahre meines Studiums und der Promotion unterstütz hat, und vielen persönlichen Verzicht in Kauf genommen hat, um mir diese Möglichkeit zu schenken. Dafür danke ich dir âĂŞ ich werde es nie vergessen. Vielen Dank gilt auch meinen Eltern Helga und Richard, die mich immer ermutigt haben, Ziele zu verfolgen und mir den Glauben schenkten, dass ich diese auch erreichen kann. Ohne ihre Unterstützung wäre diese Arbeit nicht entstanden.

Contents

1. Introduction

1.1. Motivation

Cloud computing offers ubiquitous, convenient, and on-demand access to computing resources [Mell and Grance, 2011]. Cloud computing became adopted as mainstream in recent years. Cloud providers leverage economies of scale to decrease costs for operation, development, and procurement. Furthermore, sharing resources yields cost benefits in cloud environments. Referring to [Brey and Lamers, 2009, Glanz, 2012], servers typically run between 5-12% utilization. They are oversized to have resources left for unexpected load peaks. Sharing resources on a large scale allows computing systems to operate more efficiently. One aspect is the possibility to reduce oversizing by utilizing workload fluctuations. By sharing these resources, hardware as well as software, cloud users and providers have financial benefit [Momm and Krebs, 2011, Jackson, 2011].

Sharing of computing resources is possible at different abstraction levels [Lenk et al., 2009, Mell and Grance, 2011]. Infrastructure as a Service (IaaS) provides access to resources that can be used for any purpose by consumers. The provider adds value by hosting the technologies below. Hardware virtualization technologies, such as hypervisors [Xen, 2012, VMWare, 2012], provide the technical foundation for such a service provisioning.

Platform as a Service (PaaS) is a cloud service model that provides a middleware for applications created by the consumers. In such an environment, the platform provider adds value to the pure infrastructure provisioning by administering the whole runtime environment including required services. Thus, the application provider can fully focus on orchestrating these services and value-adding business logic, without the overhead of operating the complex runtime environment including persistence or other required services. The technical foundations for this are application runtime containers, deployed in distributed environments, and additionally provided services like persistence to add value to the offering.

Software as a Service (SaaS) is a cloud service offering, comprising a ready-to-run hosted application, like an entire business suite or web mailing application. SaaS is a fast-growing market. It is expected that 50% of all CRM systems will be deployed as SaaS by the end of 2015 [Davies et al., 2014]. A SaaS can be operated on a PaaS to take advantage of the aforementioned benefits [Lenk et al., 2009]. To reduce the total costs of ownership, SaaS providers often allocate users of different customers to the same application instance. Thus, they share all layers, including the application instance. Consequently, one-time overheads on the all layers divided among the number of customers. The technical foundation for such applications is a multi-tenant architecture [Wilder, 2012, Koziolek, 2011, Koziolek, 2010].

A *tenant* is a group of users sharing the same view onto an application. This view includes the data they access, the configuration, the user management, particular functionality and related non-functional properties. Usually the tenants are members of different legal entities. This comes with restrictions (e.g., data security and privacy). In this thesis, *multi-tenancy* is an approach to share an application instance between multiple tenants by providing every tenant a dedicated share of the instance, isolated from other shares with regard to performance, appearance, configuration, user management and data privacy [Krebs et al., 2012a].

Applications designed to serve multiple tenants with a single runtime instance are known as multi-tenant applications. In general, it is possible to provide SaaS without creating Multi-Tenant Applications (MTAs). In these cases, a separate application instance is running for each tenant, in a PaaS or IaaS based environment. However, MTAs are the most efficient approach with regard to operating costs [Momm and Krebs, 2011], as in this case the general overhead is minimal and it is possible to perform a very fine-grained workload and resource management (e.g., based on requests or threads). Gartner recognized the economic relevance of various sharing approaches, especially multi-tenancy [Smith, 2011, Natis, 2012]. Several sources generally emphasize the economic benefit of multi-tenancy (e.g., [Aiken, 2011, Schuller, 2009]). Furthermore, in [Krebs et al., 2013] a case study showed that significantly more tenants could be served by one computing node when using multi-tenancy instead of a Xen based hardware virtualization.

Despite the shared resources, cloud users expect to have the feeling to control their own and separate environment, with their own Service Level Agreements (SLAs) and regulations as known from private data centers. In addition, they expect to be isolated from other customers with regard to functional and non-functional aspects, including the performance they observe.

Especially for e-commerce applications, where users interactively use the system, performance is essential for high user satisfaction. Several sources confirm a direct relation between response times and economic success. Amazon recognized that every 100 ms delay reduces sales by one percent, Google and Microsoft observed a reduction in revenue of 4.3% for each two-second delay [Schurman and Brutlag, 2009]. Other companies observed up to three percent conversions increase for every second the response times have been reduced [Hung and Danson, 2013].

Due to the sharing of resources, performance related issues are often caused by a minority of customers sending a high number of requests, since the load generated by one customer competes for the resources also used by others. Especially in the cloud context, where resources are shared intensively among customers, it is not easy to maintain reliable performance. This is a serious obstacle for cloud customers. The challenge was already recognized when the cloud computing paradigm started to gain attention [Armbrust et al., 2009, bitcurrent, 2011]. In 2010, 65% of the respondents in a survey answered that server performance process management is critical [IBM, 2010]. That shows the importance of the topic. Not too much changed in the recent years; performance is still a major point of interest. In [Internap, 2014], 30% of the IT infrastructure experts answered that one of their challenges with cloud services they currently use is related to performance. It was the highest ranked issue in the survey. These surveys focused on cloud use cases not limited to SaaS.

However, since multi-tenancy is a technological foundation for SaaS, they also show the importance for MTA. The mutual performance related influences of different tenants sharing one MTA are a well-known problem (e.g., [Wilder, 2012]). In existing multi-tenancy-related research papers, the primary future research topic identified is related to SLA compliant services, especially with regard to the performance of individual tenants [Kabbedijk et al., 2015].

In summary, using multi-tenancy provides the technical foundation for an economically efficient provisioning of SaaS, while performance is one of the major obstacles for cloud customers. Moreover, reliable performance is a key differentiator between providers, as 44% of cloud customers see this as their number one criterion when selecting a cloud provider [Internap, 2014]. As outlined, a major problem is the mutual influence of different customers sharing one system. Thus, providing the means to maintain the performance of one tenant isolated from the performance observed by other tenants within an MTA lays the foundation for SaaS providers to significantly differentiate their services.

1.2. An Example

This section outlines a motivating example, describing the problem from various viewpoints inspired by the PaaS SAP Hana Cloud Platform (HCP). Figure 1.1 depicts a simplified view, focusing on the relevant aspects of this thesis. Each customer accessing the HCP requires an account. One account may have several users and it is the billing entity. An account can be used for deploying and operating an application or for subscription to an existing application hosted in another account. In the HCP, an MTA provider creates an application, which can be subscribed by different tenants. Each of these tenants subscribes his account to an MTA. The MTA provider may provide several applications. In principle, it is also possible to host a non-shared *application instance* for each customer within his own account.

Although it is not depicted in the figure, each application instance could be deployed on one or more *Virtual Machines (VMs)* to cope with the load. Each of these deployments is referred to as *application node*. The applications run within the HCP runtime container, the *Lean Java Server (LJS)*. The virtual machines may share the same host. They may also use it together with other applications, even from other accounts. The LJS runtime additionally provides libraries to access platform services used by the application. Examples are the document service, to store unstructured data, or a relational database system, as well as services to identify tenants and accessing tenants' metadata. The platform services transparently provide data isolation for the various tenants and thus they are MTAs, too.

Tenants in this scenario ask for SLA guarantees and expect their performance to be independent of other tenants. The PaaS provider has already added value to the platform by supporting multi-tenancy on a functional basis. Now he will also support the SaaS provider by creating performance isolated applications, to again add value to the platform. Additionally, the PaaS provider tries to isolate the platform services among SaaS provider accounts or even tenants to foster a proper

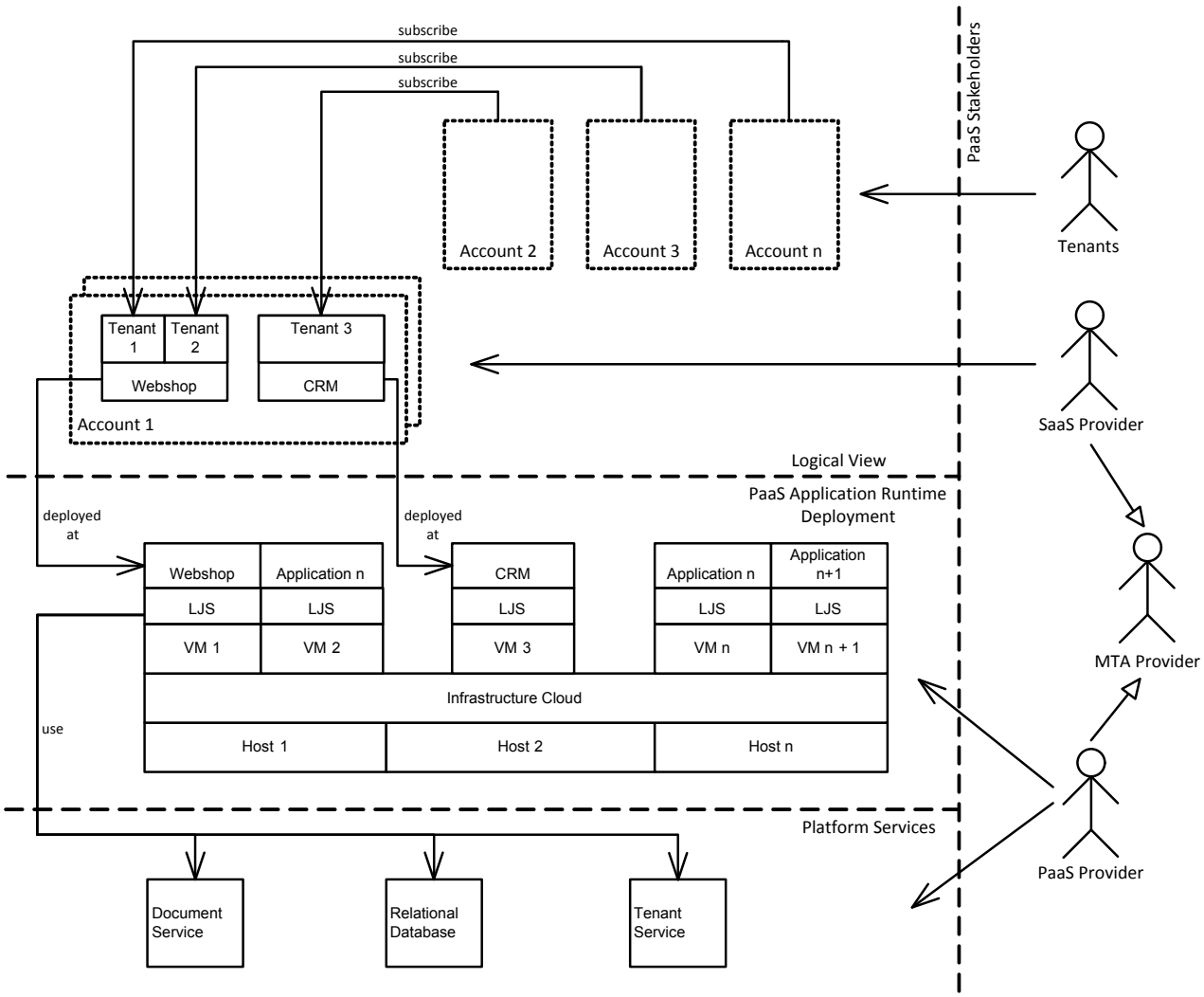


Figure 1.1.: Motivating example, inspired by the SAP Hana Cloud Platform.

isolation on the MTA using it. An MTA has to do the following tasks to performance isolate the MTA.

1. Search for existing approaches to make an MTA performance isolated.
2. Select an appropriate isolation method for the given scenario.
3. Design how this method can be integrated with the MTA.
4. Implement and deploy the solution.
5. Validate the solution.

An MTA provider's problem is that only a few methods tackling the performance of individual tenants are described in the literature so far. These neither cover all scenarios nor provide sufficient performance isolation in relevant situations. Furthermore, the providers know neither which characteristics of the methods are relevant nor how the methods behave according to these characteristics. For a proper selection, it is essential to be able to quantify the degree of performance

isolation a potential solution achieves. However, no metrics exist to quantify this. This hinders MTA providers to find an appropriate method for the scenario they consider. Additionally, there is no feasibly documented approach of how potential performance isolation methods can be implemented in an application, especially not if it runs on a PaaS. For the validation, the methods to evaluate the performance isolation are required again.

Although it is clear which tasks have to be executed to make an MTA performance isolated, the providers struggle with many difficulties and they are likely to miss their goal.

1.3. Goals, Challenges and Research Questions

This section defines the overall goals of the thesis, the resulting challenges and research questions related to these, and sets these aspects in relation to each other.

1.3.1. Assumptions and Context

The focus of this thesis is on MTAs. They provide the basis technology for SaaS, but they are considered without a concrete SaaS context in the following. Since performance is very critical in interactive web applications, the contributions of the thesis focus on this class of applications. It is expected that no batch jobs or asynchronous tasks that influence the application performance will be triggered. The primary focus within this thesis is on response time, throughput and arrival rate. It is worth mentioning that, despite these limitations, the contributions made are still applicable for other scenarios. The performance isolation guarantees are observed from the provider side and the client side performance logic is not considered. Thus, the response time of individual requests sent to the server are considered and not the response of the user interface. This is a valid approach, since the different tenants and users share only the server part and not the client side parts of the application. Moreover, the application instances are expected to be performance isolated to each other and required services are performance isolated among applications. Malicious attacks of a certain tenant aiming at reducing the service quality are related to the field of security and thus not part of this work. Malicious attacks are, e.g., distributed denial of service attacks, which generate load far beyond any usual workload fluctuation, or manipulated requests.

1.3.2. Goals

An overall goal for the thesis is formulated first, which is continually refined within this section and the sections providing the actual contributions. Additionally, relevant definitions are introduced, which specify the goal in more detail.

This thesis aims at providing performance isolation mechanisms, a methodology to quantify performance isolation and insights based on these methods that support MTA providers realizing performance isolated MTAs.

To avoid distrust in the solutions an MTA provider will apply, it is necessary to ensure fair behavior of the system with respect to its different tenants. In this thesis, the following definition of fairness is used.

Definition 1

A system is considered to be fair, if all of the following conditions are met:

1. *Tenants working within their assigned quota should not suffer performance degradation from tenants increasing their workload.*
2. *Tenants exceeding their quotas more should suffer higher performance degradation than tenants exceeding their quota less.*
3. *Tenants exceeding their quota should not suffer performance degradation, if tenants that comply with their quota are unaffected.*

Within this thesis *quota* refers to the amount of workload a tenant is allowed to execute. The term *guarantee* refers to the promised performance from the provider. The main focus of the thesis is on the first fairness criterion, which is achieved by performance isolation. Performance isolation is defined as follows.

Definition 2

A system is performance isolated, if for each tenant working within his quota the performance is not negatively affected when other tenants increase their workload.

In comparison, a non-isolated system can be defined as follows.

Definition 3

A system is non-isolated, if the performance for a tenant working within his quota is negatively affected by other tenants increasing their workload, as if all users of the MTA would belong to the same tenant.

Thus, every *abiding* tenant, with a workload lower or equal to the quota, may suffer from the high workload caused by one or more single *disruptive* tenants exceeding their quota.

A system is usually expected to be somewhere between these two boundaries. A system where the influence of one tenant on another is less provides a better isolation compared to a system where the influence is more.

SLAs are of major importance for shared services. Therefore, it can be useful to reflect this in the previous definitions. This means, the performance of customers working within their quota is allowed to be reduced as long as the guaranteed performance is maintained. This is essential to allow over-commitment. Note that, in a non-isolated system, abiding tenants will inevitably exceed their guarantee if the disruptive tenants increase the load enough. An isolated system will maintain the guarantee, independent of the disruptive load.

Subsequently, the goal of the thesis is outlined in more detail. More details in the form of derivable goals are provided in the respective chapters.

Goal 1 — *Methods for Performance Isolation*

Define a general concept how performance isolation can be achieved in an interactive multi-tenant web application and provide concrete methods to enable performance isolation. These methods have to reflect the three fairness criteria.

Success Criteria: Comparable approaches for MTAs do not exist. However, approaches from related fields can be applied with some modifications. The proposed isolation methods, discussed in detail later, should outperform them in either the isolation, adaptation speed or efficient utilization of resources.

Goal 2 — *Quantification of Performance Isolation*

Define metrics that quantify the performance isolation capabilities of an isolation method. To provide a level playing field for comparison; suitable workloads have to be recommended, for which the metric is determined. Ideally, the metrics are not limited to multi-tenant applications.

Success Criteria: It should be possible to quantify the quality of the methods resulting from **Goal 1**. Furthermore, they should provide results that reflect the expectations in well-understood systems of different technical domains.

The first two goals pose the most important aspect of the thesis. The following two goals aim at providing additional insights based on the methods from **Goal 1** to enable the MTA provider in developing performance isolated MTAs. This means that the results of the next goals are consequences from previous decisions. Nevertheless, the interpretation of these consequences is posed as own goals in the following, since not covering it leaves unanswered questions concerning the realization of the isolation methods.

Goal 3 — *Decision Support for Method Selection*

Provide an analysis of isolation methods, to help MTA providers to select the most appropriate isolation method for a given scenario. Therefore, criteria for a comparison of the methods must be determined. The behavior of the various isolation methods with respect to performance isolation aspects has to be studied accordingly. Finally, a suitable approach to create a figure of merit identifying the best solution for a scenario is required.

Success Criteria: It should be possible to find an isolation method that increases the isolation for a given scenario.

Goal 4 — *Architectural Recommendations*

Prospect multi-tenancy specific architectural decisions that have an influence upon performance isolation and define an architecture that allows a multi-tenant application provider to implement application-specific performance isolation methods.

Success Criteria: The sketched architecture should be applicable in various scenarios for all isolation methods resulting from **Goal 1**.

1.3.3. Challenges and Requirements

Challenges and requirements are not considered separately in the following listing. The fine-grained discussion provides insights into the differences of the challenges and allows a clear discussion of the solutions in Section 1.5.1.

Challenge 1 — *Layer Discrepancy Hinders a Direct Resource Control*
The concept of tenants is only known within the application or middleware layer. The resources are controlled in the lower layers like the operating system or hypervisor. Resource control allows influencing performance. However, lower layers are not aware of tenants, while the application is intentionally abstracted from the resource control. Additionally, tenants may have different data volumes they access, or different configurations of the application, leading to different resource demands for the same request type. This challenge is primarily relevant for **Goal 1**.

Challenge 2 — *Over-Commitment Requires an Efficient Utilization of Existing Resources*
Multi-tenant systems are usually operated in an over-committed state to increase utilization. This leads to lower margins to counteract against increasing demands of tenants. Furthermore, if all tenants try to utilize their SLA-defined quota, the system cannot maintain the guarantee. Additionally, unused resources from one tenant must be reallocated to tenants who need them to fulfill their guarantees. Since the operating costs of a system are widely independent of its load, tenants should generally profit from other tenants' unused resources to maintain an efficient usage. This challenge is primarily relevant for **Goal 1**.

Challenge 3 — *Unpredictability of Tenants' Workloads Reduces Reaction Times*
Tenants usually belong to different legal entities, independent of the provider. This means, information about workload changes or upcoming configuration modifications are not known in advance. Thus, little time is given to react. This can lead to overcompensating a sudden workload change and results in situations where the system begins to oscillate between two suboptimal states. This challenge is primarily relevant for **Goal 1** and for a metric that refers to **Goal 2**.

Challenge 4 — *Multi-Tenant Application Providers Become More Cost Aware*
With the increasing number of services hosted centrally by a single provider, the relevance of the operational costs has increased. Therefore, isolation methods should have low performance related overheads to reduce operational costs. Development costs are of similar importance, thus they should also be low. This challenge is primarily relevant for **Goal 1** and **Goal 4**.

Challenge 5 — *Interactive Workload Model Increases Complexity*
Due to the closed character of the system, the observed performance has an influence uponthe arrivals behavior. This adds additional complexity to the system. This challenge is primarily relevant for **Goal 1**

Challenge 6 — *Versatile Bottlenecks Lead to Various Reasons for Mutual Performance Influences*
Due to the tight coupling between tenants, hardware and software, resources are shared. Therefore, bottlenecks can occur at any of the shared layers. Whenever a resource becomes a bottleneck, it is likely that tenants compete for it and thus performance isolation is violated. Thus, performance isolation methods must be able to deal with versatile reasons for performance isolation problems. This challenge is primarily relevant for **Goal 1** but also influences **Goal 4**.

Challenge 7 — *Different Stakeholder Expectations Lead to Different Expectations for the Metric*
While a developer of an MTA is interested in the potential space for improvement of the isolation capabilities, an administrator rather needs to estimate the impact of one tenant upon others for adequate capacity planning or SLA definition. Both interests have to be reflected. This challenge is primarily relevant for **Goal 2**.

Challenge 8 — *Black Box View at the Application Limits the Available Information*
Performance isolation was defined from the end users'/tenants' perspective. Moreover, when a provider compares various isolation methods, possibly provided by third parties, he may not have internal knowledge of the methods. Therefore, only the information measurable from outside the application should be accessible. This challenge is primarily relevant for **Goal 2**.

Challenge 9 — *Separation of Quality Aspects for the Measurement of Performance Isolation Mechanisms*
While comparing systems' performance isolation capabilities it must be avoided that different aspects interfere with each other; e.g., a fast adaptation speed does not imply that the isolation is good. These are two separate aspects that should

not be mixed. A metric has to reflect this. This challenge is primarily relevant for **Goal 2** and has some influence on **Goal 3**

Challenge 10 — *Unknown Future Performance Isolation Methods and a Variety of Requirements*
It is expected that more performance isolation methods will arise. Thus, an analysis of the requirements and isolation qualities might not be valid in the future. For the selection of a performance isolation method, a vast number of functional and non-functional requirements are of potential interest. Furthermore, their importance may change for different scenarios, thus identifying the most suitable isolation methods for a certain scenario. This challenge is primarily relevant for **Goal 3** and **Goal 4**.

Challenge 11 — *Diversity of Isolation Methods Leads to Different Requirements*
The isolation methods proposed in the thesis, or derived by existing approaches from related fields, are diverse with regard to the information they need and the ways they enforce performance isolation. Thus, finding common design recommendations for applications and the implementation of the isolation methods becomes more complex. This challenge is primarily relevant for **Goal 4**.

Challenge 12 — *Lack of Control About Lower Layers when Hosted on PaaS*
If an MTA is hosted on a PaaS, the layer discrepancy becomes more evident. On one hand, the MTA provider has fewer opportunities to influence the underlying hardware, middleware behavior or request flow. On the other hand, the PaaS provider lacks in knowledge of the application scenario to automatically provide a performance isolation method that fits the needs. This challenge is primarily relevant for **Goal 4**.

1.3.4. Research Questions

The research questions to be answered to achieve the thesis goals are summarized in the following. A summary of the answers is given in Chapter 7.

RQ 1 — *What is an appropriate approach providing performance isolation in multi-tenant applications?*
The question concerns methods limiting the mutual influence of tenants on several layers, while providing a good utilization of the system and best performance with regard to given guarantees in case of over-commitment. This question covers concrete isolation methods as well as how they are realized in existing or new MTAs. To answer this question, solutions for **Challenge 1**, **Challenge 2**, **Challenge 3**, **Challenge 4**, **Challenge 5**, **Challenge 6** and **Challenge 12** have to be provided.

RQ 2 — *What are appropriate metrics to quantify the level of performance isolation a system provides?*
In other words, which metrics are able to describe the relevant aspects of performance isolation methods for different stakeholders, and how can aspects like the timeliness of the isolation quality be quantified? To answer this question, solutions for **Challenge 7**, **Challenge 8** and **Challenge 9** have to be provided.

RQ 3 — *What are the relevant characteristics of MTAs and the isolation methods to select and realize a feasible isolation method for a given scenario?*
Potential subquestions that have to be answered are related to an abstraction of the methods and the requirements to cover potential future approaches. Additionally, functional and non-functional requirements not related to the isolation metrics might be of interest and have to be identified. To answer this question, solutions for **Challenge 10** and **Challenge 11** must be provided.

RQ 4 — *What is the best isolation method for a particular scenario?*
First, it has to be evaluated how the various methods behave with regard to the identified characteristics. Then it has to be discussed if a best method does exist. If this is not the case, the question is how in a given scenario the best isolation method can be found. This research question is directly associated with **Goal 3** and can be answered if the previous questions are answered.

1.4. Contribution and Methodology

A brief excerpt from existing approaches is given at first. Then the contributions of the thesis are listed and set into relation to the four goals from Section 1.3.2. After that, it is discussed how the different contributions interplay to enable an MTA to create performance isolated MTAs. The research methods used and validation environments are outlined in separate subsections.

1.4.1. Limitations of Existing Approaches

State-of-the-art solutions provide each tenant with a separate application instance, deployed in different virtual machines with a fixed set of resources, if the performance influence between tenants should be low. This comes along with high overheads and the inability to easily shift unused resources from one tenant to another. Only few authors discuss the performance aspects of individual tenants in MTAs. Most of the existing works focus on the SLA-aware placement of tenants onto existing application nodes. These solutions assume to have knowledge of the tenants' demands before the allocation or have to do time-consuming reallocations at runtime. Thus, they cannot provide isolation on a shared node level and show a serious delay before a new configuration is settled. The

few existing solutions that consider tenant individual performance within one application node neither reflect the impact one tenant has upon another nor relate to concrete quotas or guarantees. They usually do not consider a concrete bottleneck resource or analysis of the application's behavior. Instead, the focus is on the overall performance of the application and service level differentiation between tenants.

A few authors discuss how performance isolation between tenants could be achieved on a conceptual level and provide architectural concepts. However, they do not provide a clear relation to performance isolation methods and their requirements. Thus, they are of limited use for relevant scenarios. Furthermore, they do not mention MTAs' realizations on PaaS environments.

No approach that quantifies the influence between two tenants exists so far. Existing work from the field of virtualization comprises case studies that use a white box view and do not define reusable metrics. They also do not provide insight on appropriate and representative workload profiles for MTAs. Related works that focus on performance aspects of individual tenants do not set the performance of these tenants in relation to each other and do not analyze the observed performance by creating further metrics providing insight for different stakeholders.

For more details concerning related work on individual publications see Section 2.1.3.

1.4.2. Contributions and Approach

The following list describes the contributions and how they approach the corresponding goals for the thesis. A more detailed overview of each approach is given at the beginning of the respective chapters.

Contribution 1 — *Request Admission Control Based Performance Isolation Methods for Multi-Tenant Applications*

Methods that control the admission of incoming requests for an MTA aiming at reducing the influence of one tenant on another were developed. A fast-reacting work conserving scheduling mechanism working on a request basis is combined with a mechanism dynamically adjusting the scheduler's configuration. Thus the impact of potential SLA violations can be limited on a short-term basis by the scheduler. In the long-term, a more complex optimization can be done, which would be computationally too expensive for each request.

One concrete implementation ensures that tenants cannot utilize more than a defined share of a time-sharing bottleneck resource. This method applies resource demand estimation techniques to gain insight into the MTA's internal behavior. Another concrete implementation analytically models the MTA as a black box. An optimization working on this model continuously updates the scheduler's configuration. Further approaches from related fields were implemented in addition, as basis for comparisons.

Scientifically, the major insight is that layer discrepancy can be dissolved by actively controlling the admission of requests to the application. Besides that, a model describing the performance behavior and a corresponding fitness function applicable for the sake of performance isolation in MTAs is presented. Furthermore, insights into the applicability of selected resource demand estimation methods in the given context are given.

This, contribution is primarily related to **Goal 1**. The respective parts of this contribution have already been published in [Krebs et al., 2014d, Krebs et al., 2014c, Krebs and Mehta, 2013].

Contribution 2 — *Metrics and Methods to Quantify Performance Isolation in Multi-Tenant Applications*

This contribution defines a set of metrics to quantify performance isolation, by measuring the impact of one tenant upon the observed performance of the others. A second class of metrics quantifies performance isolation by setting the abiding workload in relation to the disruptive, such that the expected QoS is maintained for the abiding tenants. The ratio can be used as a metric. Two further metrics, which are related to the field of control theory, quantify the adaptation speed of the performance isolation methods and their variability in the observed performance. Additionally, suitable load profiles for the measurement of performance isolation are discussed and a representative work for MTAs is defined.

Technical results of this contribution are a concrete benchmark application and a framework enhancement of the SoPeCo [Westermann et al., 2010], which allows the measurement of performance isolation to be automated.

The scientific contributions are the definition of novel metrics, the application of known metrics in a new domain and the insights on how to collect the values of these metrics, such that they are representative for real MTAs.

This contribution is primarily related to **Goal 2**. Parts of this contribution have already been published in [Krebs et al., 2014b, Krebs et al., 2012b, Krebs et al., 2013].

Contribution 3 — *Decision, and Design Support for the Selection of a Performance Isolation Method and its Implementation*

This contribution provides the missing links between the formulated performance isolation methods and their application in MTA. Therefore, the requirements of the isolation methods are outlined. These requirements primarily focus on the information required by the methods and outline under which conditions a method will not provide sufficient isolation. These results are applied in a problem-specific decision process, based on Analytical Hierarchical Pro-

cess (AHP), to find the most appropriate isolation method for a particular scenario. For the realization of the performance isolation method selected, a reference architecture is briefly sketched, which is primarily designed as an enhancement for a PaaS supporting performance isolated MTAs. Design recommendations that help MTA providers to build the MTA such that performance isolation can be easily realized for them are also provided.

Scientific insights are a classification schema of the methods and the classes the respective informational requirements. This also covers a discussion concerning the implementation in case of MTAs hosted on a load-balanced cluster of several application nodes. The identification of components that are not scenario specific, and thus part of the framework, is another relevant insight.

A technical result of this contribution is a concrete framework that can be applied in various scenarios to enable performance isolation for MTAs in different contexts.

This contribution is primarily related to **Goal 3** and **Goal 4**. Parts of this contribution have already been published in [Krebs and Loesch, 2014, Krebs et al., 2014a, Krebs et al., 2012a, Loesch and Krebs, 2014, Loesch and Krebs, 2013].

Contribution 4 — *Evaluation of the Performance Isolation Methods for Multi-Tenant Applications*
This contribution comprises the insights gathered from the evaluation of the performance isolation methods. Moreover, the overhead and scalability of the scheduler and the corresponding optimization was evaluated. This was done in an environment motivated by the example from Section 1.2, for the quantification of the isolation capabilities. The analysis of the overhead and scalability was done by simulating a vast number of tenants, application nodes and request types.

This contribution is primarily related to **Goal 3**, since it provides the foundations for an adequate selection of a performance isolation method. From the scientific point of view, this provides important insights about the pros and cons of the various performance isolation methods from **Contribution 1**.

In Figure 1.2 the rectangles depict the contributions made in this thesis and how they are related to each other. The *Evaluation* uses the *Metrics and Measurement* methods to evaluate the performance isolation quality of the *Request Admission Control* based performance isolation mechanisms. Overall, two novel methods based on this concept were developed (*Model Based* and *Resource Isolation*) and existing methods were adapted for the MTA use case. All of them were evaluated. The contributions related to *Decision and Design Support* rely on the knowledge about the isolation method-specific pros and cons, gathered from the evaluation, to select an appropriate performance isolation method. The design support-related part provides the measures to implement request ad-

mission control based performance isolation methods in various scenarios of interactive multi-tenant web applications.

1.4.3. Research Methodology

Glass et al. [Glass et al., 2002] analyze articles from six leading software engineering journals. They identify the established research methods applied in software engineering. Moreover, the authors define three more general research approaches to classify the overall approach of scientific contributions. The *descriptive* approach is based on primarily describing a system or an opinion, but also includes development-oriented approaches generating knowledge to solve problems. The *formulative* approaches formulate methods, algorithms or guidelines, taxonomies and others. The *evaluative* approaches are based on comparisons, e.g., based on data gathered from surveys. Of the same relevance are the research methodologies, where the authors identified 22 in total. The relevant methodologies in this thesis are the *conceptual analysis (CA), conceptual analysis mathematical (CAM), concept implementation/proof of concept (CI), case study (CS), laboratory experiment (LE), mathematical proof (MP), data analysis (DA)* and *simulation (SI)*. Often several aspects can be found within one publication and even the same contribution. This is the same in the present thesis. Formulative research approaches and the research methods' conceptual analysis and concept implementations are identified as the most common by Glaas et al. Both also have a significant portion in the work presented in the thesis.

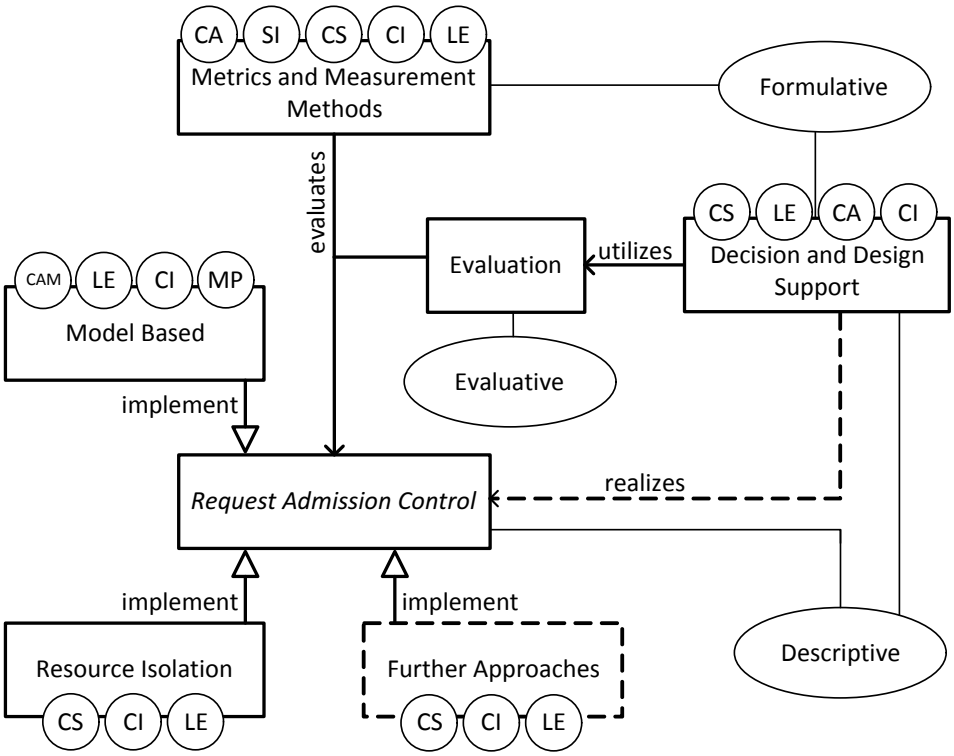


Figure 1.2.: Overview of the contributions and research methodologies/approaches.

Figure 1.2 also depicts the relation of the contributions in this thesis to the identified research methods and approaches from Glass et al. The ovals depict the major research approach used for a

particular contribution and the circles identify the research methodologies applied in this context. It is worth mentioning that not all research methods were applied to every part of a contribution. It is rather the case that the methodologies were selected such that they fit best to particular parts of the contributions.

1.5. Problem Solution and Validation

In the following, it is summarized how the challenges of Section 1.3.3 are reflected by the contributions and decisions taken in the thesis. It is also emphasized how the MTA provider's problems are solved by the results of the thesis.

1.5.1. Mastering the Challenges

The performance isolation method conducted in this thesis uses a request based admission control. This avoids modifications on established concepts separating the concerns of resource management and application logic (i.e., operating systems) and avoids a tight coupling of the MTA to a dedicated underlying technology. Admission control mechanisms are already provided by runtime containers. Thus, the implementation overhead of the solution is reduced (**Challenge 4**). The proposed request admission control allows regulating the influence of tenants on each layer, including required services and hardware resources, since each request has to pass the admission control. Consequently, even the tight coupling of the tenants is overcome and mutual performance influence at different places is limited (**Challenge 6**). The resource isolating method estimates the resource demands of requests. In principle, this can be done for any bottleneck resource. This information closes the discrepancy between the layers (**Challenge 1**). The model based isolation method treats the system as a black box, using data that is measurable from outside the application, and thus abstracts from where performance isolation is violated (**Challenge 6**). Both approaches allow shifting unused resources from one tenant to another. In case of resource isolation, the work-conserving request admission allows the over-commitment of resources. The model based approach allows over-commitment based on application level SLAs. It uses a work-conserving scheduler for fast and sudden changes in the load, and a fitness function reflecting the guarantees and quotas of all tenants, to allow an efficient and overcommitted operation (**Challenge 2**). Since the proposed admission control schedulers immediately react to load fluctuations on a short-term basis, and the configuration of them is continuously updated, exact predictions and forecasts of the tenants workloads are not necessary (**Challenge 3**). Additionally, the admission control has a low overhead (**Challenge 4**), as shown in Section 6.1.4. The model-based approach includes the users into the model. The resource isolation method does not have to reflect users, and works for interactive (**Challenge 5**) and open workload models.

As part of this work, metrics are described that set the abiding workload in relation to the disruptive. By reducing the disruptive tenants' workloads such that they maintain a constant QoS, the potential for improvement can be quantified. Other metrics measure the impact of one tenant upon the

performance observed by the others. This allows for insights into potential guarantee violations. By providing two separate metric types, the requirements of various stakeholders are reflected (**Challenge 7**). Furthermore, adequate workload profiles are defined for the measurement of performance isolation by analyzing the specific characteristics of MTAs and the situations where performance isolation becomes most important. Therefore, no reference to existing metrics is required. The metrics are based on the system's behavior, visible from an external point of view (**Challenge 8**). Besides the isolation metrics, which cover the system in steady state, metrics to quantify the timeliness of the system are derived from existing metrics. Since the different aspects are gathered by different metrics, the characteristics can be determined independently (**Challenge 9**).

To discuss the various isolation methods' requirements, they are first described by a more abstract classification schema. Furthermore, the informational requirements are also abstracted by classes of information. Thus, the results and outcomes are more generic, and it is possible to compare various isolation methods more easily, since a similar behavior within a class can be expected (**Challenge 10** and **Challenge 11**). The informational requirements and actuators of the methods are identified. In combination with the analysis of architectural concerns of MTAs, this allows a PaaS provider to develop a system that allows providing a certain degree of isolation. The same mechanisms can also be applied if the runtime environment is controlled by the MTA developer. Consequently, the relevant control mechanisms and monitoring information to enable performance isolation are made available (**Challenge 12**) with the proposed architecture.

1.5.2. Achievement of the Goals

The goals of the thesis were successfully achieved by the contributions. In the following, the results are summarized, and it is discussed how these results support the MTA providers.

1.5.2.1. Validation Results

A variety of environments are conducted in the laboratory experiments and case studies.

The most relevant, used for the comparison of the isolation methods, are motivated by the architecture of a state-of-the-art PaaS that supports MTAs (cf. Section 1.2). Artificial workloads required by the isolation metrics are applied, and real load profiles are replayed to gain results under realistic conditions. The novel isolation methods outperform the approaches transferred from other fields in these environments. They provide a better isolation quality, either in case of variable resource demands per requests originating from different tenants, or by adapting twice as fast as a comparable mechanism to a changing context. In other scenarios they provide a more efficient distribution of the resources, enabling over-commitment.

Case studies showing the applicability of the measurement-related contributions either rely on the already-mentioned environments or evaluated the performance isolation capabilities in a Xen virtualized environment. Moreover, experiments using simulated performance isolation methods were conducted.

The decision and design support-related contributions and the results of the evaluation are used to provide a solution for an apparent problem scenario in a case study on the SAP HCP Document Service. It was possible to significantly increase the performance isolation in the case study when reflecting a realistic problem scenario. In the non-isolated system, the response times increased by 141%. Applying a performance isolation method, which achieved good results in most of the evaluation cases, resulted in an increase of 97%. Applying the isolation method recommended by **Contribution 3** to the specific scenario, the increase was below 11%. For an overhead and scalability analysis of the isolation methods, large-scale application environments were emulated. The processing time consumed by the isolation methods for each request was measured in the dimension of μs. Therefore, no significant reduction of the overall system's performance was observable in the conducted case studies. The isolation measurement-related contributions were also successfully used to determine the isolation of the hypervisor Xen. The proposed architecture was used to realize five different performance isolation methods in an environment motivated by a realistic PaaS environment.

Comparing the proposed performance isolation mechanisms with a Xen-virtualized environment providing similar features as the MTAs investigated, the proposed isolation methods were able to provide similar isolation capabilities.

1.5.2.2. Supporting the Providers of Multi-Tenant Applications

The motivating example in Section 1.2 identifies five steps a provider of an MTA has to follow to make the application performance isolated.

Performance isolation methods, a methodology to quantify performance isolation, and contributions to apply these in practice are provided by the thesis. With help of these contributions, MTA providers can create performance isolated MTAs, since each of the tasks listed in Section 1.2 can be successfully mastered. With this thesis, the MTA provider has several relevant approaches to ensure performance isolation. The provided decision support and evaluation results, based on the defined metrics, help to find the proper isolation method for a given scenario. The proposed architecture and other design recommendations help to implement the methods in various environments. Finally, the measurement-related contributions help to validate the solution, if required.

Overall, the contributions help to sustain a stable performance in face of fluctuating workloads originating from different tenants. This enables providers of MTAs to offer high quality SLAs that give them an economic advantage.

1.6. Thesis Organization

To address the overall goals of the thesis, outlined in the previous section, the document is separated into seven chapters. The structural organization of the thesis is outlined in the first subsection and some hints for the readers are given in an extra subsection.

1.6.1. Structure

The introduction in Section 1 already motivated the work and defined the goals of the thesis. Moreover, it identified the respective challenges and research questions to be answered within the thesis.

Chapter 2 starts with the presentation of the relevant foundations on a high level and defines required terms. Details of the foundations are introduced when required throughout the document. After the foundations, publications and concepts from related fields are discussed. The differences in the thesis contexts are outlined. An extra section provides an analysis of approaches controlling performance on a web server's or web application's level. The fourth section in the foundations tackles existing work that considers performance measurements in shared environments. Finally, a summary outlines the limitations of existing approaches and concepts.

The Chapters 3, 4 and 5 provide contributions to achieve the goals of the thesis. At the beginning of each of these chapters, a fine-grained listing of the particular goals for the chapter's objective is listed. These goals can be derived from the thesis' overall goals. Each of these chapters includes a critical discussion about the presented solutions and a summary of the chapters. Additionally, Chapters 3 and 4 present a first proof of concept of the contributions discussed.

Chapter 3 begins with the presentation of metrics developed to quantify performance isolation and useful load profiles for their measurement. It further introduces an approach, based on an existing solution, to automate the isolation measurement process, followed by the definition of a benchmark application. The latter defines the work for a multi-tenant scenario. Finally, the chapter describes two case studies, in which the contributions were applied, to show their usefulness and applicability. A summary section critically discusses the approach and potential drawbacks. Chapter 4 introduces the overall approach to enforce performance isolation, based on a request admission control. Then, dedicated sections describe the model and resource isolating methods. Each of these sections includes a critical discussion. An additional section introduces three minor adapted approaches from related fields. Finally, the chapter presents a proof of concept for the two novel approaches and a conclusion. Chapter 5 identifies common architectural concerns in MTAs and how they influence the performance isolation capabilities. After that, the approach to find performance isolation methods for a particular application scenario is introduced. Besides that, the implementation hints in the form of an architecture are given. The conclusion summarizes the chapter and contains a critical discussion.

Chapter 6 evaluates the developed performance isolation methods. The chapter starts with the definition of the goals for the evaluation. The largest part considers the experiments and discusses the results concerning the isolation methods. A separate section summarizes the results, concerning the measurement of isolation. The summary comprises the insights from Chapter 3 and the experiences made applying the contributions within the first part of the evaluation. The case study based on the SAP HCP Document Service is the last section including measurement results. Finally, a critical discussion and summary concludes the evaluation.

The last chapter concludes the thesis. It starts with a summary of the thesis. The chapter also includes a critical discussion on a higher abstraction level, compared to the discussions in the previ-

ous sections, considering the overall approach. Furthermore, it gives some ideas for future research directions and examples of potential other applications of the contributions.

1.6.2. Information for Reading the Thesis

Particular importance was placed on the independence of the Chapters 3, 4 and 5. That means, the basic concepts of these chapters can be understood without reading other chapters. Therefore, they include a short repetition of the most relevant aspects from previous chapters, a brief definition of the chapter's fine-grained goals, the actual contribution, a proof of concept, and a summary including a critical discussion.

It is possible to follow the evaluation chapter without reading Chapters 3, 4 and 5, since the isolation methods' specific details were validated in the respective chapter of the isolation methods. Nevertheless, the evaluation utilizes the measurement methods (Chapter 3) and it is easier to understand the differences in the method's behavior if their functionality is understood (Chapter 4). Therefore, it is recommended to read these chapters in advance. The insights from Chapter 5 are useful to understand the evaluation's subsection 6.3. The other chapters do not necessarily have to be read in advance.

Parts of the thesis have already been published by the author. In the remainder of the thesis, for each section where the content or text relates to one of the author's publications, the respective references are listed at the beginning of exactly that section. Additionally, the relevant publications were already listed together with the contributions in Section 1.4.2

At the end of the thesis, a list of abbreviations and a glossary are presented for terms that were not explicitly introduced in Chapter 2 or defined later as part of the contributions. In general, new terms introduced or terms directly referring to a figure are written in italics.

2. Foundations and State-of-the-Art

In the first half of this chapter, the foundations of the thesis are discussed. It will introduce relevant concepts and definitions. In the second part, an analysis of existing work in the same or related research fields is presented to reflect the state-of-the-art. Some topics in the state-of-the-art section may also provide some background information. Thereby, related work from academia and industry is conducted. Multi-tenancy still increases in attention. Thus, most of the related work focuses on functional aspects and only few publications conduct performance isolation in MTA. This is also emphasized by the research from [Kabbedijk et al., 2015]. The authors analyzed scientific publications considering multi-tenancy. The existing publications emphasize the importance of quality of service enforcement, especially with regard to performance, as future work.

However, related problems to performance isolation in MTA also appear in other fields. Therefore, related approaches and concepts are discussed as well. Additionally, an excerpt on existing and relevant measurement approaches is given.

2.1. Foundations

This section presents the foundations on which the further contributions are built on. At first, an overview of the technical aspects of multi-tenant systems is provided, before the field of performance engineering is conducted in more detail, followed by a brief definition of closed control loops. The section aims at providing a general understanding of the relevant aspects, while concrete details are introduced, when required within the thesis.

2.1.1. Common Properties of a Multi-Tenant Application

This section starts with an overview of common multi-tenant architectures, followed by a definition of common operational paradigms.

2.1.1.1. Technical Details

Figure 2.1 depicts the artifacts that are most common for existing MTA. These artifacts were identified by [Koziolek, 2011, Koziolek, 2010], who analyzed existing multi-tenant applications. The architecture relies on the common three-tier web application model, which is enhanced to support multi-tenancy.

Requests from the *Client Tier* are usually distributed by a *Load Balancer* onto several application nodes. Each application node runs several *Application Threads* in parallel, to service the requests.

Depending on the load balancing paradigm, the responses are directly sent to the client, or via the load balancer. The latter sustains a single entry point to the application.

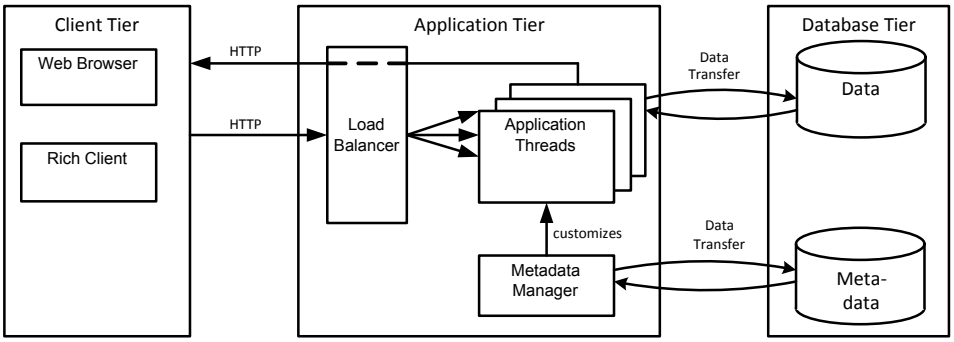


Figure 2.1.: Multi-tenant architecture based on [Koziolek, 2011].

The *Metadata* contains tenant specific information. The kind of information stored in the metadata is widely spread. Examples are customizing, database id, tenant name, credentials and tenant specific SLAs. Another essential component is the *Metadata Manager*, which enables access to this data and adjusts the application according to the information stored in the metadata.

When a request arrives at an MTA, not only the user and the client have to be identified, but also the tenant. Various approaches exist to identify the tenant. One solution is to connect the tenant specific information to the user. However, this approach requires an authentication of the user and duplicate usernames in different tenants are not possible. Thus, this violates isolation. Another widely used approach (e.g., SAP HCP [SAP SE, 2014]) is the identification by the host name. In this scenario, various host aliases point to the same application instance. On the application, the tenant specific host name is used to identify the tenant. A common approach to transfer the tenant's identifier along the program execution path uses the thread context to which the information is bound.

Three major approaches to separate a tenant's data from the data persisted by other tenants [Wang et al., 2008, Chong et al., 2006] exist. The *dedicated database* approach provides an own, completely separated database system for each tenant. In a *dedicated table/schema* approach, every tenant uses the same database management system, but separate tables or schemas. The highest degree of sharing, and thus efficiency, is established by the *shared table/schema* approach [Wang et al., 2008, Jacobs and Aulbach, 2007]. To provide a flexible customizing, some researchers propose to mix the approaches to a certain degree [Aulbach et al., 2008], while a few companies provide an additional, tenant-individual view onto a rather simple and generic data structure storing the data [Weissman and Bobrowski, 2009].

In principle, scalable web applications should be *stateless* [Fehling et al., 2014]. That means, that incoming requests do not refer to a state which is held by one application node only. If an application is stateless, a request can be served by any node in the cluster. This reduces the risk for hot spot nodes. However, several business applications rely on a state for each user's activity to handle complexity. Usually, this dependence on a state is expressed in a user *session*, which means

that all the requests from one user have to be served by one node. The same principle holds for MTAs as well.

Besides the isolation of non-functional properties, the isolation of data, configuration and customizing is an essential requirement for MTAs.

MTA providers often develop their application for a specific middleware or PaaS that actively supports the development of MTA. This decreases the development efforts. The middleware provides integrated measures for the identification of the tenant, to access metadata and their administration. Even the isolation of data is often supported (e.g., Google App Engine [Google, 2014], SAP HCP [SAP SE, 2014]).

2.1.1.2. Operations

Service Level Agreements A SLA is a negotiated document between consumer and provider that defines the quantitative and qualitative terms of a service. Usually it is legally binding. The quality aspect may relate to performance, reliability, incident management or other non-functional properties. The SLA also regulates the responsibility for the parties, e.g., how the service has to be used. A common simplified model in literature orders SLAs into gold, silver and bronze with descending qualities. Several approaches utilize such knowledge to decide which customer should be provided better performance, in case of an overloaded situation. Mechanisms enforcing the preference of customers using a high-order SLA enable *Service Differentiation*. Note that service differentiation does not enforce performance isolation. However, most performance isolation mechanisms can easily differentiate the tenants importance, since they already provide measures to control the performance. The following paragraphs discuss the performance relevant aspects of SLAs considered in this thesis.

The two most essential parts of SLAs are the description of the *qualifying condition* and the *guarantee*. A part of the qualifying condition describes the workload a service consumer is allowed to generate, in the present thesis this refers to as quota. The guarantee refers to the promised performance. Typical quota related metrics are, e.g., the request arrival rate or the amount of used storage. The *arrival rate* describes the number of requests arriving within a period of time. Guarantee metrics may refer to maximum response times, throughput or availability.

In general, SLAs can be defined on various sharing levels. At the application level, they usually refer to application entities such as number of transactions or response times. On the resource sharing level, they rather refer to a certain amount of resources allowed to be used. The mapping of low level, resource oriented, SLAs to application level SLAs is a research question in itself.

Quota and Guarantee as Optimization Goal In this thesis, the essential goal is the isolation of performance. In the context of Definition 2 and Definition 3, a possible relation to a defined quota and guarantee is explicitly mentioned. It is worthwhile to differentiate between two contexts where the concepts quota and guarantee are used. First in the context of the SLAs, where they are part of a legally binding contract. Second, as a goal definition for a self aware system's optimiza-

tion. Such systems dynamically change their configuration, or even deployment to achieve a certain optimization goal [Kounev et al., 2010]. An SLA based guarantee and quota may have different values, compared to a guarantee and quota used for the system configuration, e.g., to maintain s safety margin. Furthermore, they can be of a completely different nature. While e.g., the application's adaptability mechanism guarantees are related to a certain amount of hardware resources, the legally binding guarantee may refer to response times. In this thesis, unless otherwise specified, the terms *guarantee* and *quote* refer to the values used to define the optimization goal.

Resource Provisioning To economically provide a service in shared resource paradigms, providers consider various ratios of provided and promised resources. *Over-commitment* or *under-provisioning* refers to a setup where the legal binding guarantees for customers cannot be achieved with the provisioned resources, if all tenants use their entire quota. Providers may accept this risk under the assumption, that never all customers request their full quota at once. Over-commitment can either refer to hardware resources, or higher/application level SLAs. Thus, the shared resource paradigm allows a generally higher resource utilization and thus reduces costs. The providers have to find a trade-off between low operational costs and the risk of potential SLA guarantee violations. The opposite is *under-commitment* or *over-provisioning*. In such a setup, the provider has enough resources, that guarantees can be overfulfilled. Consequently, the system can provide the guarantees, although all tenants use their entire quota. Thus resources are likely to be low utilized, most of the time.

2.1.2. Performance Engineering

Performance Engineering or Software Performance Engineering (SPE) is a supporting process in software engineering, which aims at capturing and ensuring non-functional performance requirements within the life cycle of IT systems [Smith and Williams, 1993]. In this thesis, the term performance engineering is seen as synonymous for SPE. Through the assessment of systems designs, SPE can intervene at an early stage of the development process to assess various design decisions, reducing the costs for fixing issues later. For existing systems, the SPE can collect performance characteristics and point to bottlenecks or provide solutions for existing performance problems. The term *bottleneck* refers to a resource that has the lowest throughput of all parts in the system. Thus, it is the limiting factor for the performance. A bottleneck can be a hardware resource like CPU or software related resources, such as a connection pool.

To support these goals, the methods of SPE can be classified into measurement based, simulation and analytical methods, as well as educated guessing of experts. In the next subsections relevant aspects for the thesis are introduced.

2.1.2.1. Measurement Based Performance Engineering

The measurement based performance engineering methods are based on observations of a system's behavior. They aim at gathering systematic insights, by comparing the observed performance as

output of the system, while the workload, the configuration or the context in which the system runs is changing. In principle, this information can be gained by the observation of the parameters in a productive system while it is used by real users, or by executing a benchmark, which generates artificial load onto a system under test. While the first, is perfectly representative for the scenarios observed, it lacks in flexibility when conclusions for non-observed scenarios should be made. In the following, several terms related to benchmarking are further considered.

Benchmarking a System "A benchmark is a test, or set of tests, designed to compare the performance of one computer system against the performance of others." [SPEC, 2015]. Besides that, it can be used to gain insights of the system's behavior for a well defined and representative usage scenario. In this context, the *System Under Test (SUT)* refers to the deployed, and runnable entity, that is investigated by a benchmark. Usually this is a single hardware resource, an application or the whole system combining both. To stress the system, a certain *workload* is imposed. The definition of the workload provided by [Jex, 1998], in the context of human performance management, divides the word in two parts and is applicable to SPE as well. The term *load* refers to the quantitative value and thus the amount of work to be done. The term *work* refers to qualitative values, and thus the difficulty of the work. [Dirlewanger, 1994] differentiates three load creating models. A real load is generated by the real usage scenario, e.g., human users accessing the SUT to stress it. Pseudo real load replays logged call profiles from the real application scenario. The first lacks in economic efficiency and the latter is difficult to modify, to study the system under different conditions. The *synthetic abstracted load* is recommended by [ISO/IEC, 2011a], since it provides a behavior covering the major characteristics of the real scenario and thanks to the abstracted description allows to adjust it for various scenarios. When creating the underlying models to generate synthetic load, a good trade-off between accuracy and complexity must be found. At least two large classes for the specification of the work can be defined. *Application benchmarks* and *micro benchmarks*. The first class is based on either a real application, or an application which represents the relevant features of a particular class of applications (e.g., an interactive web application). Different functions of this application are called, to execute work on the SUT. Micro benchmarks instead, test only a very limited set of functions on a low level of abstraction. An example is the execution of certain Central Processing Unit (CPU) instruction.

Describing Workload Jain [Jain, 1991] provides an overview of techniques to model a synthetic abstracted load. *Averaging* reproduces the average workload, which can be enhanced by adding *dispersion* with the help of e.g., the standard deviation. More detailed methods use multi-parameter histograms to model the workload or being based on principal component analysis. However, only *Markov Models* can also reflect the order in which different work is executed by the system, e.g., in a bookshop application users are likely to follow a certain navigation path through the application. This can have a significant impact upon the performance observed, e.g., due to caching effects or pre-load mechanisms.

A Markov chain is a discrete stochastic process, for which the prediction of future states, based on a limited knowledge about the past, is as good as if the whole history of the process is known. It undergoes transitions from one state to another. The probability for a particular transition depends on *n* states visited before. For modeling workload, the behavior is usually assumed to be *memoryless*, which means that the probability for a certain transition depends only on the present state. This is also referred to as *Markov Property*.

Memoryless Markov models are widely used to model user behavior in benchmarks. In TPC-W [TPC, 2002] the current state represents the page an emulated user currently sees and the transitions describe the probability of following a certain link on that page. When the transition is triggered, it ends up at a defined new state, corresponding to the web page received from the benchmark application.

When a human user behavior is modeled, it is essential to consider the user's physiological and psychological demand. When a user receives a response, he has to think about the response of the system before he can trigger the next activity. This phenomenon is usually emulated by an artificial delay between two transitions and referred to as *think time*. The delay is often described by a random variable to reflect reality.

A further approach to simplify the emulation of real workload is to cluster different activities, by their impact upon the system. An example is a bookstore application with two different requests to search for either an author or a publisher. If both have the same demand onto the system resources they can potentially be considered as the same *request type*.

Running a Benchmark Caching or garbage collection are examples where the performance of an SUT depends on previously executed actions. Such mechanisms lead to changing performance over time. Therefore, a benchmark defines a *warm up phase* [SPEC, 2015], which is a period of time prior to the actual measurement, to get the SUT in a steady and consistent state. After this, the measurements are taken, followed by a *cool down phase* in which the load is decreased to get the SUT again in a steady and consistent state. Such a procedure is often iteratively repeated, either to make sure the results are reliable or to evaluate the system with different workloads or configurations. Similar to Westermann [Westermann, 2013] an *Experiment Series* refers to a set of experiments stressing one SUT to achieve a certain goal. This means, that within one experiment series, different workloads and configurations of the SUT may be used. Each concrete combination of the *parameters*, which describe this configuration, are an *Experiment*. Each may be repetitively executed in *Experiment Runs*, to ensure a statistically reliable result. For each experiment run, a warm up and cool down phase may occur.

Technical Components Usually three to four components are used to run an application benchmark (e.g., [SPEC, 2012]). The *load driver* emulates users, which trigger the application functions. The *benchmarking application* is the application deployed on the SUT to create the work, by serving the load driver triggered requests. Sometimes, emulated third party services are added to the test landscape, to emulate the applications references to other systems. All these com-

ponents are controlled by the *benchmarking harness*, which starts and stops the load generation, collects relevant samples of the performance metric under investigation and computes a figure of merit.

Typical Performance Behavior of Systems While the workload on the SUT increases, one can usually observe one of the two patterns depicted in Figure 2.2. In the trashing case, the system enters an overloaded state, at which the load cannot longer be adequately processed. Moreover, due to context changes and synchronization among parallel execution paths, it comes to a visible reduction of the system's performance. In case of a non-trashing system this is not the case. The introduction of admission control mechanisms can limit the amount of work processed by the SUT in parallel, and thus avoids a trashing state. In case the workload is defined by requests, these are either rejected without being served, or they are queued to be served at a later point in time. Before the point, where the throughput starts to decrease, the response times usually increase slowly. Consequently, if a system is significantly below its maximum saturation, an increase in load has only a low influence upon the response times of the system. Therefore, performance isolation aspects are more important for systems already running at high utilization.

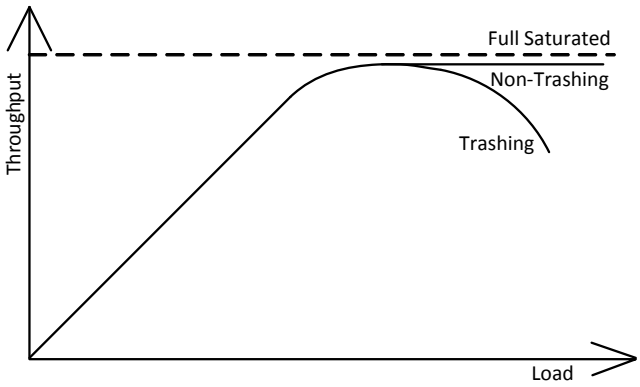


Figure 2.2.: Typical performance pattern for increasing load based on [Kounev, 2011]

2.1.2.2. Further Performance Engineering Methods

Model based approaches provide other possibilities to gain insights into a system's performance behavior. Model based approaches are often assumed to use simulation techniques, to obtain insights. However, an analytical approach abstracts the system's behavior for a special purpose and omits unnecessary information, thus it is a model as well. Therefore, a differentiation between simulateable models and analytically solvable models is necessary. These models vary in the modelling assumptions, the effort to create them or to solve them, and the level of abstraction [Noorshams, 2014]. Examples of analytic models are regressions, interpolations or extrapolations of existing measurements. Such models can be used to get estimates for unmeasured parameter configuration [Courtois and Woodside, 2000]. The corresponding functions refer to the term *performance curves* [Westermann and Momm, 2010]. Since these models do not use any system internal knowledge, they belong to the group of *black box* models, while others allow to describe internal relationships

of the investigated system. Correspondingly, these approaches refer to *white box* models. A modeling paradigm which can be used for white box modeling relies on *queuing theory*. It is a discipline of operations research and dates back as far as the early 20th century. One of the first applications was the analysis and optimization of phone exchange nodes. The field of applications is very wide. Whenever waiting lines occur, it is a potential approach to optimize processes. The same it is in computer science (e.g., [Menasce et al., 2004, Bolch et al., 1998]). Queuing networks allow to use simulations or analytical methods for investigating the performance. However, analytic approaches usually rely on less detail. If the models are created manually, the complexity of the analytic approach limits its use. Besides that, analytic solvers support less functionality as simulations. Examples are the missing support to annotate individual requests with a payload, e.g., parameter settings, and missing reflection of parameter dependencies in the modeled system behavior [Brosig et al., 2015]. Furthermore, their results are often limited to mean values. The advantage is the rather fast solving.

Queueing Networks Numerous variations of queuing theory exist, and all rely on the concept of a queue and a server. In the following, the terminology for typical computer science related scenarios are used. This is depicted in Figure 2.3. A *queue* holds the arriving requests and forwards them with a defined scheduling strategy. The most common one is the First In First Out (FIFO), where requests are served in the same order they arrive. Others prefer requests based on a certain priority related to a particular request type. At the server, a request is delayed a defined time frame. Usually it is described by a random variable and refers to the term *service time* distribution. The *inter arrival time* defines the time between the arrival of two requests. It is usually described by a random variable, too. However, deterministic delays for service time and inter arrival time are also supported. Advanced methods also provide service time models considering the internal behavior of a time sharing resource (e.g., CPU). In these cases the time required to service a request increases with the number of requests served in parallel. Moreover, a limitation of the queue length is possible.

In most cases a *Poisson process* is assumed for the arrival rates and service times. Poisson processes assume the occurrence of random, discrete and independent events within a time frame where the average amount of occurrences is known. This implies a memoryless process. Thus, the *Poisson distribution* describes the probability of a certain number of events for a defined interval length. The *exponential* or *negative exponential* distribution assumes a Poisson process and describes the probability density of the time between the occurrence of two events. It is common use to assume user think times and service rates fulfilling the Poisson conditions.

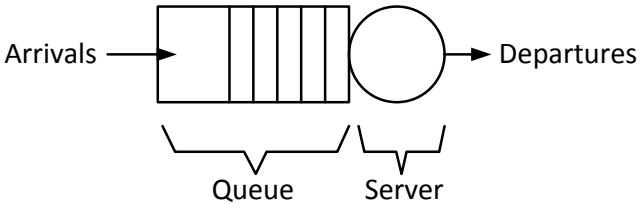


Figure 2.3.: Queue and a server.

Such a queue and server can also be used to black box model complex systems. However, sometimes it is necessary to analyze a certain detail of a more complex system and thus a detailed model is required. Queuing Networks (QNs) are bipartite graphs connecting queues and servers as depicted in Figure 2.4. If requests enter the system and leave it after being served (cf. Figure 2.3) the system is *open*. In Figure 2.4 requests that have been processed by the modeled application do not leave the system. Instead, they are returned to a part of the network, which describes the users' behavior before the request again arrives the application server. In such a *closed system* the *population* of requests is constant. This approach is often used when interactive applications are modeled, since users usually send only one request at a time. To model users' behavior, very often an *infinite server* queue is chosen. Hence, the user related queue is always empty, since the requests can be forwarded to an infinite number of servers. The servers delay the request using an exponential function. Another name for such closed systems refers to the characterization of the workload, that is a *closed workload* system. *Open workload* is defined respectively.

A closed system as depicted in Figure 2.4 will always converge to a steady state over time, since the population of requests is constant. *Steady state* means, that the systems observable parameters do not change over time. This does not exclude that the system follows random processes and thus produces some random noise, when being observed. Note that, steady state is actually the state at which a system's performance is observed in a benchmark. A related term is *flow equilibrium*, it defines that the arrival rate of a system, or part of the system is the same as its throughput. Two of the most important measures in this context s are the arrival rate and the *departure rate*. The latter expresses the number of requests that depart a server in a defined time interval. In case the system is not in flow equilibrium, usually due to the fact the arrival rate is larger than the departure rate, it ends up in increasing queue lengths and thus it cannot be in steady state. However, even in a system with steady state *backlogged periods* can occur.

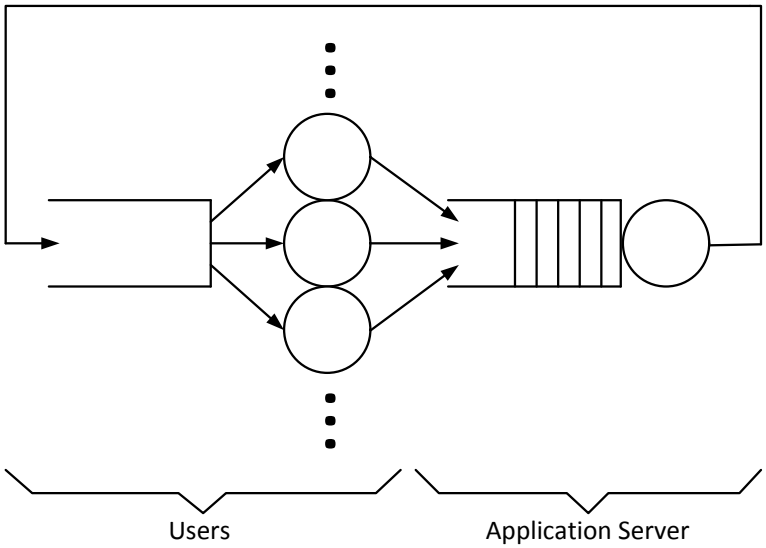


Figure 2.4.: Closed queuing network.

Assume that each server in the user model omits requests Poisson distributed. The merge property states, that the superposition of these processes is again a Poisson process whereby the mean arrival

rate is the sum of the mean arrival rates of all sub processes. The same holds for the split property. If a Poisson process is randomly split in sub processes, the resulting processes are again Poisson processes [Ross, 2006].

The Poisson Arrivals See Time Averages (PASTA) property states, that requests arriving a system, see it in a state as if they came into the system in a random instant of time. This is equivalent to the state a random external observer would see the system [Wolff, 1982]. This is an important property. It allows to conclude from the average state (e.g., queue length), the waiting time of an arriving request in the queue.

A convolution of two Poisson processes is still a Poisson process. However, in the example from Figure 2.4 it cannot directly be concluded that the arrivals at the application server follow a Poisson process, even if the think time and the service time are. This is, because the system is not memoryless. Assuming a system's population of only 1 request, it becomes immediately clear. An external observer would in average see the application server model utilized at least a small portion in time, while the arriving request would always observe a free system. However, for a sufficient large number of requests in steady state, the system arrival rate converges to a Poisson arrival process.

Resource Demand Estimation When a QN is created, one of the important steps is to determine the demand, a request has onto the resources simulated by the servers. In a black box approach, this might simply refer to the observed service time. However, if resources are explicitly modeled, it often lacks adequate monitoring information. If the system's workload can be controlled, resource demands can be easily determined by sequentially sending requests to the system and monitoring the consumed resources. This task becomes more complicated with an increasing number of simultaneously served request types. Resource Demand Estimation (RDE) methods use statistical tools, often combined with particular domain knowledge to estimate the hidden resource demand a request has. Examples are based on linear regressions [Rolia and Vetland, 1995, Pacifici et al., 2008, Casale et al., 2008] or Kalman filters respectively [Zheng et al., 2008, Kumar et al., 2009].

2.1.3. Closed Control Loop

Adaptive computer systems dynamically and automatically change their configuration or deployment. They aim to adapt to a changing context to maintain a defined observable behavior. The context can comprise the workload, hardware and everything that has an influence upon the system's performance. The fundamental mechanisms enabling such features are known for centuries now.

In the year 1784 James Watt invented a steam engine that is able to maintain a constant drive by applying a centrifugal governor [Karl-Dieter and Oliver, 2012, pp. 1-3]. This centrifugal governor is driven by the steam engine. If the drive increases, it closes a valve to reduce the drive. Thus a constant drive is maintained, independently from disturbing factors like changing heat. Such a measure refers to closed control loops. Such systems "continuously compare the actual output to its

desired reference value; then apply a change to the system inputs that counteracts any deviations of the actual output from the reference." [Janert, 2013, p. 15]. Consequently, these systems have the objective that an output follows a desired reference value, although the system underlies disturbing influences.

Figure 2.5 depicts a schematic view onto a *closed control loop*. A *sensor* measures the system's output. The output should maintain a value defined by the *reference* value. The measured output is used to calculate an error between the desired value and the real value of the system. This error is used by the controller to adapt the system's behavior using the *actuating* variables. These variables configure an *actuator* to control the system's input.

If a perfect system knowledge would exist and no disturbances would occur, the feedback and the computation of the error would not be required. Such systems are referred to as *open control loop* or *non-feedback controllers* or *feedforward controllers*. However, the system underlies continuous changes, like the number of users per-tenant, or a reduced processing speed caused by high temperatures. Especially in physical environments, a disturbance onto the sensor can occur as well, at least the sensor adds some delay before the error can be calculated.

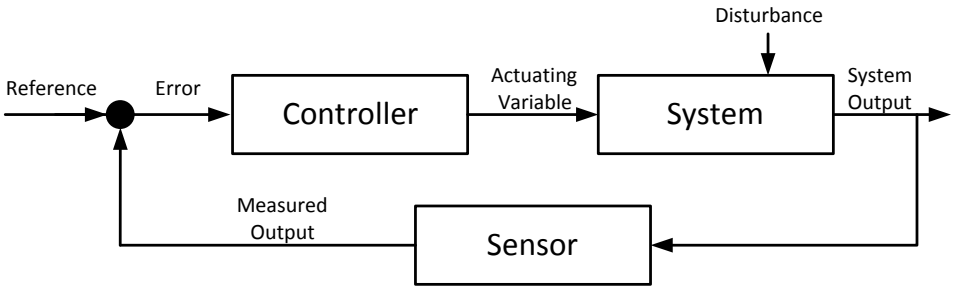


Figure 2.5.: Schematic view onto a closed control loop.

Closed loop controllers can be classified in two large groups. Multiple Input Multiple Output (MIMO) controllers observe several outputs and control several actuating variables. They come along with a high complexity and thus with several limitations in practice [Janert, 2013, pp. 65-66]. MIMO approaches often use solutions which do not require a detailed system model. One example is fuzzy logic based controllers. Single Input Single Output (SISO) controllers observe one output and control a singe actuating variable.

A widely used controller is the Proportional-Integral-Derivative (PID) controller. It controls the input by a proportional factor based on the current error, by using an integral over the past errors and the derivative of the change in the error signal. The integral part is able to reduce the steady state error and the derivative control can increase the adaptation speed [Hellerstein et al., 2004, 293ff.]. Some controllers use only a subset of the PID provided functionality. Accordingly, these refer to, e.g., PI controllers.

2.2. Related Fields

This section discusses existing approaches that are related to the goals of the thesis, but not directly focusing on multi-tenancy. At first, virtualization is discussed. It is the most used alternative approach to multi-tenancy. Then follows a discussion how processes, threads and I/O are controlled. Additionally, mechanisms from the network domain are conducted. A brief discussion, why elasticity does not achieve performance isolation, presents the last topic discussed. It is not possible to discuss every approach and method in the particular field. Instead, some representative examples are selected to explain the principles in that domain. These shared principles are used to differentiate the goals of the thesis from those of the related work.

2.2.1. Hardware Virtualization

Hardware virtualization technologies can be used to run different operating systems and applications isolated from each other, on a shared hardware or host operating system. A hypervisor is the control instance that supervises the execution of the guest systems and their access to resources. In general, it is possible to differ between two fundamental approaches, to host several guest operating systems. Virtualization and paravirtualization methods are distinguished. The first leaves the guest system completely unaware of its virtualized execution. Emulation can be used to achieve this. This usually comes along with rather high overheads for the system's performance. Paravirtualization instead is an approach where the guest operating systems are aware of their execution context. Therefore, they can be implemented with special functions to communicate with the hypervisor. This increases performance, since the guests may directly access the hardware. Calls from guests to the hypervisor refer to hyper calls. Since this concept allows to deploy the same application multiple times onto the same hardware, by deploying them into separate guest machines, this is an alternative approach to multi-tenancy.

Exemplary the CPU isolation among guests is discussed in the following. The physical cores are abstracted by so called VCPUs. Each guest uses one or more VCPUs. The hypervisor controls the execution and thus the resources consumed by the guest systems. One option is to pin a guest systems VCPUs directly to one of potentially many real CPUs. Thus, it is possible to dedicate the CPU for the usage by one guest only. Therefore, resources can be isolated. However, this does not allow over-commitment and reduces efficiency. Usually CPUs are shared among several guest operating systems.

To ensure isolation in these situations, hypervisor manufactures and the academia developed various schedulers to manage the access onto the CPUs. The Xen Credit scheduler is an example that allows to enter a share of the resources per guest system and a cap value [Xen Project, 2015]. Even if no other guest competes for resources, the cap cannot be exceeded. The credit scheduler is a *proportional weighted fair share* CPU scheduler, aiming at providing each guest resources proportional to their share. In Xen, one queue for every physical core is created, in which the VCPUs of the guests are queued accordingly. Each VCPU has an amount of credit it is allowed to

consume within an accounting period. An *accounting period* is the time for which a scheduler tries to balance the usage of a resource according to the weights. The VCPUs are primarily selected by their state which indicates, if it currently process I/O, has credits left or already used all of its credit. Within these defined groups a round robin like approach selects the next VCPU. A currently running VCPU is being pre-empted, once it exceeds the quantum or release the resource. This allows other VCPUs to process their instructions [Xen Project, 2015]. *Quantum* refers to the time a process is allowed to use a resource, without interruption.

Additionally to this approach, the academia developed schedulers for a variety for situations to cover real time applications (e.g., [Xi et al., 2011]), or to cover I/O bound aspects (e.g., [Zeng et al., 2013, Gulati et al., 2010]). Isolating I/O is a difficult task. Especially in case of Xen, since the I/O related drivers are centrally provided by a privileged guest system which can become a bottleneck. Additionally, the I/O activities require CPU in the shared privileged guest system. [Gupta et al., 2006] tackle this problem. In their particular case, they determine the CPU overhead for I/O of a Xen guest on the shared privileged domain, by advanced monitoring features. They use this information in an enhanced credit scheduler to achieve lower interference of guests.

Xen was discussed as the sole example within this section. Nevertheless, the methods and approaches mentioned here are of general relevance for hypervisors.

Relevance for Performance Isolation If a guest system hosts an application instance serving one customer with several users, it is conceptually comparable to multi-tenancy. Although virtualization enables measures to provide performance isolation between different guest systems, several differences compared to an MTA exist. The hypervisors unify the hardware control and the knowledge about the entities they isolate (guest systems), by connecting the VCPUs with real CPUs in the schedulers. VCPUs and CPUs are conceptually the same resource type. In case of MTAs the entities are conceptually different. At least a mapping between a tenant and a resource representative entity would have to exist. As one example, the CPU isolation is considered. The hypervisors scheduled VCPUs are *preemptive*, which means that the processing can be interrupted at any point in time. Thus, immediately leaving the resource for others. Moreover, the control is very fine-grained on a cycle's basis. Only one entity allocates the CPU at a time, whereas several tenants have to allocate the application at the same time, to ensure a proper overall performance.

Generally, using virtualization to provide performance isolation adds additional resource overhead, and the methods used by virtualization technologies cannot directly be applied for performance isolation in MTA. However, the method from Gupta et al. [Gupta et al., 2006] is potentially relevant, since they provide a mapping between the CPU resource required to process guest individual I/O activities, which are two different kinds of resources, where no direct mapping was available before their work.

2.2.2. Scheduling of Processes, Threads and I/O

A process is executed and controlled by an operating system. Different states of a process are specified. The most relevant are *running*, in which it actively uses resources. Another is *waiting*, in which usually other processes allocate the resource. A similar concept are threads. Several threads belong to one process, whereby a thread cannot exist outside a process. Threads are the smallest set of instructions that are scheduled by common operating systems. Processes are isolated with regard to the address space and thus data from other processes. Threads share the same process context, meaning they share the same data. An operating system scheduler manages, when a process or thread is allowed to run. Simplistic approaches provide a round robin, in which each process is allowed to process for a quantum of time. However, usually a preference of interactive processes should be given. Several queues with waiting processes exist in Linux environments. The queues have different priorities, according to the process priority. Each queue serves its processes round robin, whereby queues with higher priorities are preferred.

Another interesting field relates to I/O scheduling where requests, once they are forwarded to the device driver, cannot longer be preempted. Moreover, the devices often process several requests in parallel. A common approach is deadline based scheduling. In such a system each I/O request has an expiration time and the scheduler selects requests based on the expiration time. Thus, it is aimed, that requests will not exceed a certain response time. More advanced schedulers provide a round robin like scheduling for each process, additionally to the deadline queue. Hereby, each process is allowed to send as many I/O requests as possible within its quantum. It is worth mentioning, that these schedulers can differentiate between request types and their potential impact to optimize the resource usage, by not mixing request types. More details about scheduling can be found in [Stallings, 2008].

Relevance for Performance Isolation Similar to virtualization, the schedulers control the resources/threads and know the entities consuming them (cf. Section 2.2.1) and the processing can be preempted. However, I/O schedulers aim at ensuring a proper response time for all processes without a direct control of the resource, but by limiting the admission to the resource. A difference is, that an I/O request usually only refers to one resource, whereas application requests can generate load on several resources. A widely used and representative approach uses deadline based scheduling. In this mechanism, the information about the request originating is not longer of relevance. Advanced methods cover this, but still lack in considering a certain quota for a process and differ only priorities of processes. Since a quota is not reflected, the schedulers will fail to maintain performance isolation among processes, if the device is overloaded. This becomes even more evident, since requests close to the deadline will be preferred in processing, independent of the source. Simple and static schedulers, like a round robin, lack in efficiency and over-commitment support. Using these approaches, one queue can be long and providing guarantee violations, while another is short and far below its guarantee. A dynamic adaptation of scheduling weights would be able to fix this issue. Furthermore, although I/O schedulers are aware of request types, they do not really know the

resource internal impact and the potential influence upon the performance observed by the requests served next.

However, it seems that a request based scheduling or active control of the admission of requests is able to provide measures to control performance. This could be a way to provide performance isolation with reflecting quotas.

In summary, in most cases the requests can be preempted and a direct control of the hardware layer exists. There is still a significant difference in the granularity between user level requests and single I/O requests. The latter is much finer and allocates only one resource.

2.2.3. Networks

In networks the achieved QoS is of interest, since the establishment of the first phone lines. It is differentiated between connection and packet oriented networks. In connection oriented networks, a dedicated path between two communication endpoints is established. Thus the resources are shared by a strict allocation of them. These approaches are not considered to be relevant, because of their drawbacks in over-committed scenarios.

In packet oriented networks, the quality observed for a certain packet depends on the quality provided on each of the transportation layers of the OSI model [Zimmermann, 1980]. If an application relies on a certain quality of the network, it also has to communicate this through the stack to the point, where finally a medium is accessed. In IP packets, the differentiated services code (DSC) or type of service field (TOS) [Nichols et al., 1998, Moy, 1994] is used to prioritize different IP packets. Network elements like routers may use this information to prioritize selected packets. Virtual LANs (VLAN) enable a separation into virtually decoupled LANs based on a frame id field, at the second layer. Additionally, they allow to prioritize Ethernet frames by a three bit field [IEEE, 2006] which can be used by the network elements to adjust their scheduling. Thus, both concepts are quite similar. If an application requires a certain quality from the network, this information is downwarded the stack into each of the underlying layers. These approaches enable the differentiation of service qualities, but cannot ensure isolation. If isolation should be achieved, approaches where resources become explicitly reserved on all network elements, on the communication path, are required (e.g., RFC 2205 [Braden et al., 1997]).

A certain QoS and isolation can be guaranteed between data streams, multiplexed onto a medium between two directly connected points, [Zimmermann, 1980] by using scheduling mechanisms, which organize the frames or packets according to their priority. Common *work-conserving* Weighted Fair Queuing (WFQ) schedulers are used for this task. Work-conserving means, that empty queues are skipped, to serve queues with pending requests, although this might result in an imbalanced service rate for different queues. The advantage is a higher utilization of the shared medium. Besides the work-conserving WFQ schedulers, quota enforcing approaches like a leaky bucket [Tanenbaum and Wetherall, 2011, pp. 407ff.] scheduling may also be applied. To relieve the disadvantages of a static behavior of the scheduler approaches, academia provides approaches that dynamically adapt the weights based on various optimization techniques (e.g., [Grzech et al., 2009]

A recent development are Software Defined Networkss (SDNs). SDN is a concept, that manages the routing of packages or Ethernet frames by flexible software logic, accessing several OSI layers, instead of hard wired circuits on the forwarding plane, accessing only one layer. The concept additionally allows to enable a fast forwarding of packages, by utilizing low level information. Whereby potentially complex software defined rules on the control plane fill the switching tables. An example is the OpenFlow standard [ONF, 2013]. OpenFlow assumes an architecture, where the switching of packages is still implemented by a fast forwarding plane with specialized technologies. However, the concrete information how a frame or a package is routed, is dynamically defined at runtime by some higher level control logic. If for example a new package arrives and no rule exists on the switching/data plane, the upper level software defined logic is triggered for a decision. This investigates more details, like package details, and defines a rule for the lower level switching tables, in case a package with the same characteristics arrives again. The benefit of the differentiation into these two abstraction levels allows high throughput, combined with complex processing and decision making.

Relevance for Performance Isolation There are several significant differences between the work done in the field of networks and the applicability on MTA. First, in networks the information about the applications is not available in lower layers ensuring a certain QoS. Indeed, only different classes of priority can be encoded. Even more interesting, this information is pushed from the application downwards to the lowest levels, really accessing a concrete medium. Additionally, this usually does not provide isolation, but a differentiation of the service qualities only. Furthermore, once the packet is on its way in the network, the quality depends on a volunteer support of the standards by each network node. For an MTA this would mean that information about a tenant is pushed down the stack to the operating system or hypervisor. This is imaginable, e.g., by adding some meta information to the thread context, which is used by the operating system for an additional accounting in the scheduler. However, it becomes infeasible once other services in other processes on other nodes are used to process a request. Even within one process and operating system one would have to enhance all resource schedulers, including I/O, CPU and memory allocation mechanisms to support this concept. That seems not to be feasible without enormous efforts.

When the medium is accessed, one has to rely on the voluntary support of other nodes accessing the medium. Assuming a shared medium, such as in a WIFI network, one node can easily flood the medium. If a node solely sends multiplexed packages on one physical medium connected to one receiving node, they again apply common scheduling techniques. A MTA which strongly benefits from the parallel processing of requests, would not be operated efficiently if only one request would be allowed to be processed at once. Furthermore, the users of tenants do not follow a defined protocol, and one cannot assume that they will decrease their load following certain algorithmic patterns like adjusted receive window sizes in TCP.

However, scheduling mechanisms were proved to provide a certain handle on controlling performance. Furthermore, the separation of complex processing for package individual decisions and the fast rule based processing of requests in context of SDNs brings several benefits.

2.2.4. Elasticity

Essential features for the cloud users are the illusion of infinite computing resources, the elimination of an up-front commitment by cloud users, and the ability to pay for the resources used in a short-term basis [Armbrust et al., 2009]. The combination of these features is referred to as *elasticity* and enables users to rapidly adapt the resources provided by the cloud. They use these features to adapt to variable workload in an automated way, maintaining stable service qualities. The lowest granularity of resources allocated, is often measured in terms of instances, nodes or the number of CPUs. In the most literal sense of the word elasticity, it would mean, that absolutely no over-provisioning occurs, to ensure the pay for what you need expectation of a cloud consumer. Advanced methods use controllers, including a prediction of the future workloads and queuing models to estimate the impact upon the system for proactive adaptation of the used resources [Ali-Eldin et al., 2012]. However, state-of-the-art industry solutions, neither provide an automatic adaptation of the application at all, or use event triggered predefined rules to adjust the application [Galante and Bona, 2012]. Such an event could be an SLA guarantee violation. Researchers already started to add these rule based adaptations into models (e.g., [Suleiman and Venugopal, 2013]). Others focus on the technological enablement. Das et el. and Sousa [Das et al., 2011, Sousa and Machado, 2012] provide measures to support the replication and migration of a shared/multi-tenant databases at runtime, to adapt to increasing or decreasing load.

To clearly separate elasticity from performance isolation, the following example helps. Assuming a system in an overloaded situation, because of one disruptive tenant, it could be a solution to provision and allocate additional resources by leveraging underlying layers, to maintain SLA guarantees. This is elasticity and it is acceptable, if the disruptive tenant pays for the increased capacity. If the disruptive tenant does not pay for the extra resources, the system is no longer fair. Moreover, new resources are not instantaneously available and at least within this duration the performance has to be isolated.

Relevance for Performance Isolation Although related work mostly focuses on elastic provisioning of hardware related resources, it is also used for applications running on a PaaS. In this case, usually application nodes are added. However, elasticity for a particular tenant was not yet investigated. Existing elasticity mechanisms focus on an entity and granularity they can control. Usually this is much more coarse-grained as required when tenants should be controlled. The possibility to control a tenant's performance individually, would lay the foundation to provide elasticity on a tenant basis.

However, there are also some similarities. Elasticity and performance isolation face a dynamic environment in a similar context, in which they have to adapt an existing system. The results from the field of elasticity show that controllers can be applied to this task. Since in the isolation methods presented later, the actuators can be operated very fast, the benefit of predictive methods becomes less. Nevertheless, using a queuing model, calibrated by real time information to find a suitable system setup, may be used in the context of performance isolation. Compared to [Ali-Eldin et al.,

2012], the model would not be used to predict the impact of workload changes, but for the system optimization only.

2.3. Web Server Based Performance Control

This section considers related work, where performance is actively controlled at the application level, which is at a level playing field with MTAs. First, related aspects from a wider context are discussed before multi-tenancy is investigated in detail.

2.3.1. Performance Driven Deployment

The proposed solutions in this group aim at achieving certain performance objectives by a goal oriented deployment of the relevant entities. These entities can be tenants, components or others. However, these approaches fail in case of unpredictable workload behavior and guaranteeing isolation within a shared node. In the following an excerpt of such approaches is given.

Fehling et al. [Fehling et al., 2010] analyzed challenges arising from multi-tenant scenarios. Furthermore, they provided a method to place tenants onto application nodes with different QoS, in a way resources are used optimally, without SLA violations. Thereby, tenants with lower requirements are sometimes located onto application nodes providing better qualities, if the provisioning of additional resources can be avoided. The method selects the locations by a simulation of the tenants impact and a hill climbing algorithm. Also, this approach targets at maintaining tenants SLAs, it cannot ensure isolation between tenants sharing the same resource. Additionally, it requires a perfect forecast of the tenants behavior and demands, which seems hardly possible considering fluctuating request rates of an application. Furthermore, a combination of simulation and hill climbing is slow and fails if demands change fast. Additionally, a dynamic relocation at runtime might have to many overheads to react to sudden changes. The work of [Zhang et al., 2010] aims at maximizing the overall resource utilization of a set of computing nodes, with respect to guaranteed SLAs for different tenants. The aim is to place a set of sequentially on-boarding tenants to a set of available resources, without violating the SLAs. The consumption of a particular resource is assumed to be linear on the number of the tenant's active users, while a certain response time is guaranteed. However, tenants allocating the same node still influence each other, and variable demands are not covered at all. Lang et al. [Lang et al., 2012] applies comparable approaches to databases. The approaches from [Schroeter et al., 2012, Schroeter, 2013, Ruehl, 2013] assume a multi-tenant application allowing a distributed deployment of its components. They provide models to describe the application components, the tenants' requirements and the offerings of various deployment locations. In principle, these approaches could be used to deploy performance critical components of an application isolated for separate tenants. However, in this case resources are no longer shared and the drawbacks of multiple static overheads appears again. Furthermore, approaches which belong to locating tenants onto computing nodes are usually NP-hard bin-packing problems, using known heuristics.

2.3.2. Closed Control Loops

The papers presented in the following use closed control loops to regulate the performance observed by the users. The main difference to the thesis objective is, that they focus on the overall performance only, but not on the influence of one tenant onto another. In the following insights into representative approaches are given.

One paradigm is to control and enforce a certain performance of the overall application, by using a request based admission control. [Iyer et al., 2001] recognize an overloaded situation and simply block the establishment of additional connections. To dynamically adapt the admission control, in case of variable workload, several approaches apply closed control loops. [Qin and Wang, 2005] use a closed control loop to adjust the ratio of requests, that have to be rejected to ensure a proper performance of those accepted for processing by the application. To do so, they apply a continuously adapting controller, to tackle non-linear behavior of the response time. Other approaches are based on fuzzy controllers. In [Diao et al., 2002b] the maximum number of connections an Apache web server process keeps open is controlled by the human expert knowledge transferred into a fuzzy controller. Requests that arrive, after the maximum number of connections exceeded, are rejected. This ensures a proper performance. [Chandra et al., 2006] apply a PI controller for a similar goal. Several derivatives of such approaches exist. They all do not consider different sources of incoming requests and consequently these approaches cannot provide tenant-individual performance measures.

MIMO based controllers seem more appropriate for the sake of performance isolation, since several inputs and outputs have to be controlled. [Diao et al., 2002a] present a controller where the MaxClients and KeepAlive settings of an Apache web server are adjusted to maintain a certain utilization of the hardware resource. Both settings relate to the admission of requests. The authors overcome the layer discrepancy and enforce a certain utilization of resources, although no direct resource control is given. However, in this and similar approaches from other domains the actuating variables are independent. Transferring this to the multi-tenancy domain would mean that the admission of requests from one tenant could be adjusted independently from the admission of the others. This is not the case in reality. An increase in admitted requests requires a reduction of the others, since the overall load onto the system should not become too high. Moreover, the system's output is not a single metric, but the impact one tenant has onto another. Furthermore, a binary quota like concept is not supported by these approaches. Another example is [Liu et al., 2007]. The authors use a MIMO based controller to adjust resource shares for each tier in a 3-tier application. This controller ensures that each customer, using one separate instance of the application, observes different performance, when it comes to overloaded situations.

2.3.3. Source Depending Performance Control

In contrast to the previously presented approaches, the measures depicted in this subsection are aware of the source of incoming requests. They control these requests in order to achieve a certain

performance goal. However, usually the focus is on the overall system performance and/or service differentiation. Exemplary, three publications are discussed in the following.

[Welsh and Culler, 2003] actively promote the use of request based admission control to ensure a proper performance of web applications in case of extensive load peaks. They apply on each computing node, which is part of the service, different controllers to ensure a proper performance. Since they do not apply this at one single entry point, only requests that use a bottleneck service are rejected or delayed. In general, they have one closed control loop, dynamically adjusting the arrival rate of requests admitted for processing, with the help of a token bucket algorithm. Additionally, they adjust the number of threads processing the incoming traffic to avoid overloaded situations. Furthermore, the authors discuss an approach, providing service differentiation, where the requests are ordered into classes. The quality difference is enforced by dedicated bucket sizes. However, the class specific bucket sizes are determined by fixed factors and thus lack in efficiency with regard to over-commitment. Even worse, as long as tokens exist in both buckets, there is a significant influence between the request classes. This is why the authors aggressively reduce the tokens of low priority classes in case response time problems occur. Similar to the previous approaches, a defined quota is not considered.

Lu et al. [Lu et al., 2006] present an example of work where different requests/connections are given different priorities, depending on their origination. In particular their work focus on TCP connections and differentiating QoS, but does not provide isolation as previously defined.

So far, these kind of approaches focus on service differentiation and avoidance of overloaded situations. However, they do not consider multi-tenancy at the application level or performance isolation.

[Karlsson et al., 2005] applies a closed control loop on storage devices to adjust the maximum number of requests an entity is allowed to be sent. They define various working points of the system. Each has different ratios of requests allowed to send by the different entities. Once a working point is defined, the overall throughput, and the derived throughput for each source is reduced such that the delay of the response maintains a certain value. Nevertheless, it is not discussed what should happen with requests exceeding the adjusted throughput. Queueing would immediately violate isolation, since the queues are not reflected in the mechanism. Further, the approach does not consider a quota and individual tenants exceeding it.

2.3.4. Multi-tenancy

This section discusses existing work considering the isolation in MTAs. At first, an overview of work to achieve isolation of non-performance related aspects is discussed. Secondly, an overview of existing performance isolation mechanisms for MTAs is given.

2.3.4.1. General Isolation Aspects

Several publications focus on how to consolidate different tenants onto the same resources. Some of these publications focus on the database level and how the data is isolated between tenants.

[Weissman and Bobrowski, 2009] discusses an approach using simplistic and rather generic tables, adding a metadata based defined view onto it. This allows to provide tenants individual views onto the data and even individual schemas. Aulbach et at. [Aulbach et al., 2008] present an approach where parts of the data from different tenants is shared in the same table and other parts are stored in separate tables, to enable customization by providing isolated data and isolated customizing. [Jacobs and Aulbach, 2007] discusses various approaches, how the database runtime can support various tenants and [Yaish and Goyal, 2013] presents an access control to ensure that tenants can only access their data.

At the application level, various authors discuss how isolation of non-performance aspects can be achieved. Bezemer and others [Bezemer and Zaidman, 2010, Guo et al., 2007, Koziolek, 2010, Koziolek, 2011, Strauch et al., 2013] present more or less coarse-grained architectures and design ideas. Examples are file path separation and tenant-individual credential storages to access data. They aim at ensuring a proper isolation of the application data and customizing, which comes along with increased code complexity. Other approaches allow to have some components being shared among tenants, while others are separately deployed to keep tenants isolated. These separated component instances may even rely on tenant specific implementations [Mietzner et al., 2008, Mietzner et al., 2009, Schroeter et al., 2012, Schroeter, 2013, Ruehl, 2013].

Relevance Bezemer [Bezemer and Zaidman, 2010] recognized that several aspects, such as security, are cross cutting concerns and thus linked with different layers of the application provisioning. This is similar to performance. However, especially in case of data isolation or customization, the subject to be isolated for different tenants is intentionally controlled by the application layer. This allows to rather easily ensure their isolation. Other non-functional properties like security or availability related issues are also relevant for the whole stack. Approaches as those from [Schroeter et al., 2012, Ruehl, 2013] might take this into account and deploy crucial components on locations with higher qualities. Thus, the authors fill the gap of the layer discrepancy by an automated deployment of the application components. Whereby the solver, finding a suitable deployment, has information about requirements and provided qualities of different locations. However, the research in this thesis focus on shared application instances and their nodes. Thus, it is complementary to the approaches from [Schroeter et al., 2012, Ruehl, 2013]. Isolation methods presented in this thesis might be used to achieve performance isolation within one of these shared components. Overall, the related work tries to separate multi-tenancy specific concerns, like tenant authentication, from the application logic. Ideally, this would also be possible for performance isolation.

2.3.4.2. Performance Isolation

In the following, the existing publications which aim at MTAs and performance isolation are discussed on an individual basis.

[Guo et al., 2007] outline their position on how performance isolation between tenants could be achieved. The authors recommend to locate tenants on application nodes, by considering their

resource requirements and load profiles. This should reduce the influence upon each other. Since this cannot ensure performance isolation sufficiently, they additionally recommend static or dynamic admission control mechanisms. However, no details about concrete methods are given.

Shue et al. [Shue et al., 2012] aim at reducing interference between tenants and unpredictable performance. Thereby, they aim at key-value databases shared among several tenants. The approach is composed of four parts. On a long timescale, the partitions of the database are reallocated, thus it avoids unfair distribution of load. On a medium timescale, global weights for the tenants, representing their relative share, are adjusted. These weights are used to determine local weights per node. The mechanism increases the weight of one tenant on one node, if his demand on this node is higher compared to other nodes. This information is shared with other nodes for compensation. To enforce the weights, a WFQ is applied. Moreover, a decision mechanism decides which replica, if existing, should be used to answer a tenant's request. Although, scheduling weights are adjusted dynamically, they do not reflect the fairness criteria defined for this thesis. The authors do not consider application level guarantees and quotas like response time, but simply try to enforce a defined share of the resources, which reduces the capabilities for over-commitment on application level SLAs. The approach fails if tenants have different demands per request.

In [Lin et al., 2009], the authors propose a closed control loop based performance regulation for MTAs. The focus is to guarantee different QoS to different tenants. Therefore, they propose two closed control loops. The first loop ensures that the overall performance of the system is not exceeding a certain limit. This is enforced by applying an admission control mechanism. The second loop, dynamically adjusts the thread priorities of tenants to diversify quality. However, in situations with high load, the method will fail to guarantee isolation, because the mechanism does not provide isolated queues. Even with high differences in the thread priorities, an isolation is not possible, if the disruptive tenants allocate a large number of threads. Therefore, requests are rejected instead being backlogged, although the response times would still be sufficient. If the bottleneck is in another process, not controlled with the thread priority, the approach will not be sufficient.

In [Wang et al., 2012], a Kalman filter is used to estimate the CPU consumption of tenants sending different request types. The isolation strategy checks, if the current CPU utilization exceeds a predefined threshold. If this is the case, it identifies the tenant and request type that causes the largest CPU utilization. As a consequence, the allowed request rate for the request type of the disruptive tenant is decreased by rejecting requests. Their approach assumes, that the same request type sent by different tenants has the same demand. Furthermore, the demand estimation is only used to identify malicious workload, but not to define individual shares. Incoming malicious request types are rejected. A better approach would enable real resource shares, and handle requests with some delay, instead of rejecting them. This would rather reflect the behavior of stand alone, completely isolated systems. If a resource demand estimation has to be applied on a tenant-individual basis, further problems arise. Previous evaluations of resource demand estimation methods are limited to a low number of request types. There are case studies with linear regression [Zhang et al., 2007, Rolia et al., 2010] using 14 different request types. Kraft et al. [Kraft et al., 2009] evaluate the influence

of the number of request types (between one and five) on different linear regression and maximum-likelihood resource demand estimation methods. In [Rolia and Vetland, 1995] experiments using linear regression and varying the number of request types between three and ten are described. In [Spinner, 2011], several methods for resource demand estimation are evaluated for up to 16 workload types.

The SPIN framework [Li et al., 2008] realizes three major features. First, it recognizes performance anomalies, which will lead to significant performance problems. Second, it provides fine-grained monitoring information to account resource consumptions to tenants, which is used to identify the responsible tenant for the performance anomalies. The used metric is the ratio of the resources used. At last point an adoption decision is taken to reduce the impact of the tenant. However, the authors miss to discuss how such an adoption might look like. Neither the mechanism how the adoption process might be controlled, nor potential actuators are defined. This makes the definition of a framework less helpful, since it does not consider how potential algorithms to enable performance isolation may look like, e.g., for a closed loop controller the accounting of resources may not be required. Furthermore, no measures to reflect SLAs are mentioned.

Similar to the paper just discussed, Walraven [Walraven et al., 2012] presents a framework to provide performance isolation in MTAs. Compared to Li [Li et al., 2008], the authors provide more insights on how the actuator works. They first assume a tenant profiler, which gathers various monitoring information. Based on a comparison with tenant specific SLAs each tenant is put into a category. Tenant's requests are put into different queues, according to their category. A scheduler selects requests from different queues, following an undefined strategy, to ensure proper isolation between categories. Although the approach provides more information about how performance isolation should be engaged, it misses a detailed discussion how the tenant profiler is structured internally. Furthermore, the limitation of the scheduler to a limited set of categories hinders a fine-grained, tenant-individual optimization. A centrally controlled modification of the scheduler's behavior at runtime would be beneficial, but was not discussed. The reasons leading to the various design decisions are not discussed and a detailed analysis of potential performance isolation mechanisms and their requirements was not done.

Both frameworks neither describe, how they collect and combine information distributed among several processes and application nodes, nor how they support performance isolation methods.

2.4. Performance Measurement in Shared Environments

Most existing publications, in the wider context of this thesis, focus on the benchmarking of single aspects of cloud services, like databases (e.g., [Cooper et al., 2010]). Several publications discuss metrics for cloud features, like elasticity (e.g., [Kupperberg et al., 2011, Herbst et al., 2013, Islam et al., 2012, Dory et al., 2011]).

The most relevant related work comes from the field of virtualization. Most of the benchmarks focus on general performance aspects (e.g., [SPEC, 2013]). Another example is VMmark [Herndon et al., 2006], a benchmark developed by VMWare. It defines a tile as a set of VMs serving different

applications (e.g., mail server and SPECweb2005). Several tiles are deployed on a virtualized hardware. The benchmark score is based on a normalized, overall throughput of the applications. The total throughput increases with the number of tiles deployed, as long as the system is not saturated. VMWare publishes the number of tiles in addition to the throughput. However, VMmark focuses on the overall performance of a hosting platform, and misses to quantify the mutual influence, of the different workloads or tiles. Others investigate in detail, which parameters influence performance in virtual environments (e.g., storage [Noorshams et al., 2013]) but do not consider isolations aspects.

Georges et al. [Georges and Eeckhout, 2010] developed two metrics to express the efficiency of a virtualized environment. One is similar to VMmark. The other, Average Normalized Reduced Throughput (ANRT), reflects the loss of throughput on a per VM basis, when additional VMs are deployed. Nevertheless, they do not set the amount of changed workload in relation to ANRT, and use static amount of workload for the VMs. Thus, these metrics are not feasible to be used for quantifying performance isolation between tenants already deployed.

Koh et al. [Koh et al., 2007] collected data within an experimental environment to characterize the performance inference of workloads in different VMs. In addition, the mechanism predicts the inference of these workloads. Huber et al. [Huber et al., 2011] created a feature tree that captures the mutual influences of different VMs, with different resource requirements. This is done in an automated way. Nevertheless, Huber and Koh do not extract a single figure of merit describing the system's isolation behavior. Furthermore, their approaches focus on hardware related resources, only available in white box scenarios. Consequently, the approaches are hard to be used in SaaS or PaaS scenarios and do not cover the black box view used for the definition of performance isolation.

One of the most interesting work in this field is from Matthews et al. [Matthews et al., 2007]. The authors propose a performance isolation benchmark. They deploy a web application benchmark in a VM, hosted together with a VM creating disruptive load. At first a reference measurement without disruptive load is done. Then, various micro benchmarks (e.g., fork bombs or arithmetical computation) are started in the malicious VM, and the reduction of performance at the application was observed. However, the authors do not control the amount of malicious load and do not set the impact upon the observe performance in relation to it. Furthermore, the execution of micro benchmarks on a tenant basis, sharing one application instance is not possible.

Iosup et al. [Iosup et al., 2011b] observe the performance of systems running on amazon web services and google app engine on the long-term. They observe a certain variability and were able to identify recurring patterns. However, Iosup does not define a metric which expresses the influence of foreign workload onto the own.

Guo et al. [Guo et al., 2007] defines performance isolation based on their understanding of a fair system behavior. From their point of view, a system should prevent high performance for one tenant at the cost of another. However, in this thesis, the definition of fairness was explicitly divided into three different aspects. Finally, Guo does not define a metric nor a benchmark.

[Wang et al., 2008] describes and evaluates the performance of known database patterns for MTAs introduced in Section 2.1.1.1. Others also consider how the overall performance in case of multi-

tenant databases can be increased [Gao et al., 2011]. Nevertheless, the influence of one tenant onto another tenant is not investigated.

Work that evaluates existing performance isolation methods uses standard benchmarks (e.g., [Lin et al., 2009, Wang et al., 2012]. The authors directly compare observed performance metrics in various situations. Neither metrics nor specific benchmarks for the isolation quality were published so far.

2.5. Summary of Related Work

Measurement Existing work considers only general performance objectives of the shared systems. Only few authors discuss the variability of the performance in cloud environments and nobody quantifies the impact of tenants onto each other, from an applications perspective.

Deployment and Elasticity Approaches which locate entities, SLA aware, onto a set of computing nodes, cannot ensure a proper isolation within the shared node. However, it might support additionally applied methods. As it was outlined, elasticity might add resources at runtime, to compensate increasing demands. However, this is conceptually different and solves not the problems envisioned in the thesis.

Scheduling Scheduling approaches, mostly based on WFQ, are used in many disciplines to ensure a fair usage of resources. Usually it is applied where direct access to the resource is given. Therefore, the information about the entity, which belongs to the particular fine-grained resource request has to be known. In the field of networks, this information is pushed downwards through the stack. In case of process scheduling, the isolated entity is already known by the operating system. Both is not possible in case of MTAs. Furthermore, network related technologies usually provide service differentiation and rely on the cooperation of other nodes in the network. Most static scheduling solutions fail, if the demand of the entity they schedule varies a lot, and the processes are non-preemptive. A preemption is not possible, if such a mechanism is used to schedule tenant requests. To bind such schedulers to certain application level performance objectives, the scheduling has to be adapted at runtime.

Dynamic Adaptation Some solutions apply an admission control and adjust its configuration dynamically, to adapt to changes in the system's usage. Usually closed control loops are used for that. However, these approaches do not reflect the different demands per scheduled entity, or lead to unstable behavior as, e.g., shown in the work from Hellerstein [Hellerstein et al., 2002]. They are hard to construct, since various controllers are required for different working points of the system. Related work does not consider a dependency between the actuating variables and does not reflect over-commitment and quota definitions.

Besides the already mentioned disadvantages, no work in the field of multi-tenancy conducts a systematic and coherent solution, including the isolation methods, their evaluation with problem

specific metrics, the identification of the method's/application's requirements and characteristics, and the methods realization in real world scenarios.

3. Measurement of Isolation

This section discusses methods to quantify the degree of performance isolation a system achieves. To address this goal, metrics expressing the degree of performance isolation are required. In an ideal case, these metrics can optionally be derived externally, running benchmarks from the outside and treating the system as a black box. This enables them for a broad application, because internal knowledge of the system is not necessarily required.

To provide a level playing field for comparisons, it is important to explicitly consider the workload profiles used, when applying the metrics. This is the same for other metrics, e.g., a given response time for a system is meaningless without considering the system's workload, at which the response time was measured. Consequently, for a comparison of response times, the request arrival rate/throughput must be considered and the service called on the SUT have to be standardized. The goals of this chapter can be summarized as follows.

Measurement Goal 1 — *Metrics to Quantify the Degree of Performance Isolation*
Develop metrics that are able to quantify the isolation capabilities of a system. These metrics should reflect Definition 2, including their relation to SLAs from Section 1.3.2, as well as the timeliness of methods which enforce performance isolation. Since the performance isolation definition was made from an user's perspective, the metrics are expected to trade the SUT as black box. As a foundation, the questions the metrics should answer have to be identified.

Measurement Goal 2 — *Load Profile Recommendations*
Recommend suitable load profiles to provide a level playing field for comparison. A concrete load profile strongly depends on the questions and evaluation goal of a particular scenario, where the metrics are applied. Consequently, only some argumentative rules of thumbs can be expected.

Measurement Goal 3 — *Work Definition and Measurement Harness*
Definition of a representative application and call behavior for interactive, data centric multi-tenant applications. In addition, to make the metrics easily usable beyond the scope of multi-tenant systems, they should be embedded into an existing measurement framework.

In this chapter, two different methodologies and several alternative metrics, along with appropriate measurement techniques, for quantifying the isolation capabilities of IT systems are proposed. The

metrics presented are applicable to performance benchmarks and allow measurements without internal knowledge. They are preferable in situations where different request sources use the functions of a shared system, with a similar call probability and demand per request, but different load intensity. These characteristics are typical for MTAs. Other load generating entities could be suitable for the metrics too, as long as the limitations are considered. Examples of such entities are a single human user or another technical system that sends requests.

This chapter also shares approaches to determine a good load profile and introduces a multi-tenancy specific benchmark application, as well as a generic framework, based on the existing Software Performance Cockpit [Westermann et al., 2010], to measure the isolation in various domains. The chapter concludes with a first assessment of the metrics, providing additional insights into some basic performance isolation methods for MTAs and Xen.

3.1. Metrics and Load Profiles

According to the IEEE Standard 1061 a software quality metric is: "A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality." [IEEE, 1998].

An analogous definition can be derived for performance metrics: A function whose input is a computing system and whose output is a numeric value that can be interpreted as the degree to which useful work can be accomplished by the system compared to the time and resources used. Hereby, the computing system includes the software, hardware and deployment information.

The proposed metrics are not necessarily coupled to performance, and they do not express the systems capability to accomplish useful work. They rather express the influence of one tenant onto the capabilities of another one to accomplish useful work. Consequently, the metrics are not covered by these definitions and provide an independent quality aspect.

Existing benchmarks and metrics in the field of shared resources and cloud computing focus on single aspects like databases (e.g., [Cooper et al., 2010]). Others discuss metrics for cloud features like elasticity (e.g., [Kupperberg et al., 2011, Herbst et al., 2013, Islam et al., 2012]) or the performance variability [Schad et al., 2010, Iosup et al., 2011a]. The latter observes the changes in performance over time while the workload is constant. However, the authors do not set this in relation to the workload induced by others and thus a new approach is required.

The subsequent sections identify the goals and requirements for these novel metrics, followed by their definition. To achieve this, the Goal Question Metric (GQM) [van Solingen et al., 2002, van Solingen and Berghout, 2001] approach is used to derive suitable and generic goals, and question for the metric. Related work is considered to find general requirements for metrics in the field of software and performance engineering. After the identification of the requirements, the general idea of the isolation metrics is introduced, followed by the definition of the metrics. Section 3.1.5 explains some metrics that are already known from other fields, to address the time behavior of isolation methods. Furthermore, relevant load profiles are discussed for the measurement.

This section has already been published to large extent [Krebs et al., 2014b, Krebs et al., 2012b].

3.1.1. Goal of the Metrics

To improve an existing performance isolation mechanism, MTA developers need an isolation metric to compare different variants of the solution, including an estimate on how much the method can be improved. For stakeholders involved in operations, the impact of an increasing load onto the other tenants can be of interest, to define SLAs, or to manage the capacity.

A system is performance isolated, if for each tenant working within his quota the performance is not negatively affected, when other tenants increase their workload. A decreasing performance for the tenants exceeding their quotas is fair with regard to the second fairness property (cf. Section 1.3.2). Moreover, it was outlined that it is possible to link the definition to the performance guarantees with SLAs. This allows, that a decreased performance for the tenants working within their quota is acceptable, as long as it is within their SLA defined guarantee. These aspects have to be reflected by the metric. The Definition 2 and Definition 3 provide upper and lower bounds for the metrics.

To support stakeholders in building performance isolated applications, or estimating the impact tenants have onto each other, the subsequent questions have to be answered.

Q1 How much potential to improve a system's isolation exists?

Q2 What is the impact of one tenant onto the others?

Q3 Which isolation method is better?

Beside these metric specific requirements, several generic criteria for metrics can be defined. Most publications (e.g., [von Kistowski et al., 2015, Marco et al., 2012]) focus on benchmarks and list aspects like independence from a manufacturer. Although these aspects are of high interest when a concrete benchmark is created, they are not necessarily of relevance in the context of a single metric.

Obviously a metric should allow conclusions about the characteristic under investigation and reflect the real facts. This refers to the **validity** of the metric. In the literature (e.g., [Hoecker et al., 1984]) additional relevant requirements for metrics are listed:

Objectivity No subjective influence possible.

Reliability Provide the same result when measured multiply times.

Comparative Should be able to be set in relation to other metrics.

Economy Low cost of measurement.

Useful Fulfill practical needs.

The authors identified these requirements for software metrics, but they are of a general nature and most of them are applicable for the purpose of performance isolation. However, a metric describing an aspect that was not described by other metrics before, can hardly be set in relation to other metrics.

3.1.2. General Idea of the Isolation Metrics

In the following Section, a set of metrics in order to to quantify the isolation capabilities of a system is introduced.

When conducting performance measurements and benchmarking, most approaches vary an input parameter of the system, while observing the impact upon a performance objective under investigation. Hereby, the independent input parameter is usually related to the entity for which the performance objective is relevant. An example is the observation of the response time for a group of users, while the number of users of this group increases.

For the measurement of performance isolation, one has to distinguish between groups of *disruptive* and *abiding* tenants (cf., Section 1.3.2). The metrics can be computed by a function whose input is a computing system, including load generating entities and whose output is a numeric value that can be interpreted as the degree to which these entities influence each other's observed quality.

Consequently, the metrics are based upon the influence of the disruptive tenants on the abiding tenants. Thus the two groups exist, and the influence of one group as a function of the workload of the other group is observed. This, is a major difference to traditional performance measurements. For the definition of the metrics, a set of symbols is defined in Table 3.1.

Symbol	Meaning
t	A tenant in the system. $t_i \in D \oplus t_i \in A$
D	Set of disruptive tenant exceeding their quotas (e.g., contains tenants inducing more than the allowed requests per second). $\|D\| > 0$
A	Set of abiding tenants not exceeding their quotas (e.g., contains tenants inducing less than the allowed requests per second).$\|A\| > 0$
q_{t_i}	Quota for tenant t_i. A numeric value describing the QoS required by the SLA.
w_t	Reflects the workload caused by tenant t represented as numeric value $\in \mathbb{R}_0^+$. The value is considered to increase with higher loads on the system (e.g., request rate and job size). $w_t \in W$
W	The total system workload as a set of the workloads induced by all individual tenants. Thus, the load of the disruptive and abiding ones.
$z_t(W)$	A numeric value describing the QoS provided to tenant t. The individual QoS a tenant observes depends on the composed workload of all tenant W. QoS metrics with lower values of $z_t(W)$ correspond to better qualities (e.g., response time). $z_t : W \to \mathbb{R}_0^+$
I	The degree of isolation provided by the system. An index is added to distinguish different types of isolation metrics. The various indices are introduced later. Furthermore, a numeric suffix to the index is used at some places to express the load of the isolation measurement.

Table 3.1.: Overview of variables and symbols for performance isolation metrics.

3.1.3. QoS Impact Based Metrics

QoS-oriented metrics depend on at least two measurements. First, the observed QoS results for every abiding tenant $t \in A$ at an application wide reference workload W_{ref}. Second, the results for every abiding tenant $t \in A$ at a global workload W_{disr}, when a subset of the tenants have increased their load to challenge the system's isolation mechanisms. As previously defined, W_{ref} and W_{disr} are composed of the workload of the same set of tenants, which is the union of A and D. At W_{disr} the workload of the disruptive tenants is increased.

The relative difference of the QoS (Δz_A) for abiding tenants at the reference workload compared to the disruptive workload is considered.

$$\Delta z_A = \frac{\sum\limits_{t \in A} [z_t(W_{disr}) - z_t(W_{ref})]}{\sum\limits_{t \in A} z_t(W_{ref})} \tag{3.1}$$

Additionally, the relative difference of the load induced by the two workloads is relevant.

$$\Delta w = \frac{\sum\limits_{w_t \in W_{disr}} w_t - \sum\limits_{w_t \in W_{ref}} w_t}{\sum\limits_{w_t \in W_{ref}} w_t} \tag{3.2}$$

Based on these two differences, the influence of the increased workload on the QoS of the abiding tenants is expressed as follows.

$$I_{QoS} := \frac{\Delta z_A}{\Delta w} \tag{3.3}$$

A low value of this metric represents a good isolation, as the difference of the QoS in relation to the increased workload is low. If the value is 0, the isolation is perfect. Accordingly, a high value of the metric expresses a bad isolation of the system. In principle, the upper bound of the metric is unlimited. A value of 1 means that the performance decreases exactly by the factor the workload increases. A negative value may occur if a mechanism reduces the performance of the disruptive tenant more than required. Thus providing the abiding tenants even a better performance.

The metric provides a result for two specified workloads (W_{ref} and W_{disr}) and thus the selection of the workloads plays an important role. On the one hand, this provides a good evidence for exactly this setup and thus provides detailed information. On the other hand, only one measurement for a given workload tuple (W_{ref}, W_{disr}) is not sufficient, if the exact workloads of interest are unknown or variable. Thus, they can be enhanced by considering the arithmetic mean of I_{QoS} for m disruptive workloads. Hereby, the disruptive tenants increase their workload equidistantly within a lower and upper bound.

$$I_{avg} := \frac{\sum\limits_{i=1}^{m} I_{QoS_m}}{m} \tag{3.4}$$

This metric provides an average isolation value for the entire space of measurements and provides one representative numeric value. The curve's shape is not reflected and thus the value might lead

to misleading results within some ranges of disruptive load. The metric's interpretation is the same as for I_{QoS}

It is conceivable that a provider is interested in the relative difference of disruptive workload Δw, at which abiding tenants receive a predefined proportion of the promised QoS Δz_A. This is conceptually similar to the already described metrics and could be used as one additional approach.

3.1.4. Workload Ratios Based Metrics

Assume a non-isolated system and the situation in Figure 3.1. The disruptive tenant increases its load over time. Since the system is not isolated, the response time for the abiding increases, the same as if its users would be part of the disruptive tenant. This influence was used for the previous metrics.

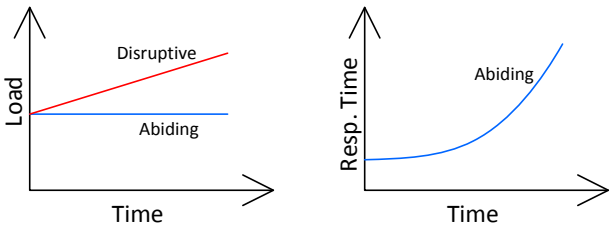


Figure 3.1.: Influence of the disruptive tenant onto the response time.

The following metrics are not directly associated with the QoS impact resulting from an increased workload of disruptive tenants. The idea is to compensate the increased workload of disruptive tenants, by decreasing the workload of the abiding ones such that they keep the QoS for the abiding ones constant. Figure 3.2 depicts an example. Since in a non-isolated system, the performance and load is equally shared among the tenants, the response time would maintain a constant value if the abiding tenant decreases its workload by the same amount as the disruptive tenant increases it. This is depicted in Figure 3.2. In case of better isolation, the abiding tenants have to reduce their load less.

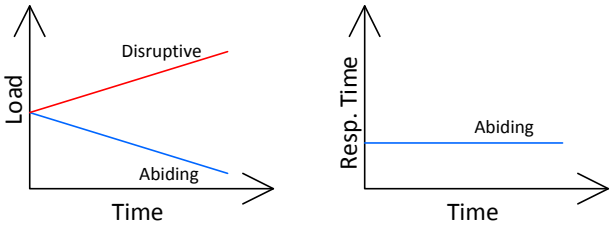


Figure 3.2.: Influence of the disruptive tenant onto the response time with a load adapting abiding tenant.

Naturally, this is only possible with the support of the abiding tenants and such a behavior does not reflect productive systems. Thus, these metrics are planned to be applied in benchmarks with artificial workloads, where a load driver simulates the tenants and can be enhanced to follow the described behavior.

In the following this idea is introduced in more detail. One starts measuring the isolation behavior of a non-isolated system, by continuously increasing the disruptive workload W_d. In such a situation,

$z_t(W)$ remains unaffected, if the workload of the abiding tenants W_a is adapted accordingly, to compensate for the increase in the disruptive workload. Plotting W_a as a result of W_d, describes a pareto optimum of the system's total workload, with regard to constant QoS for the abiding tenants.

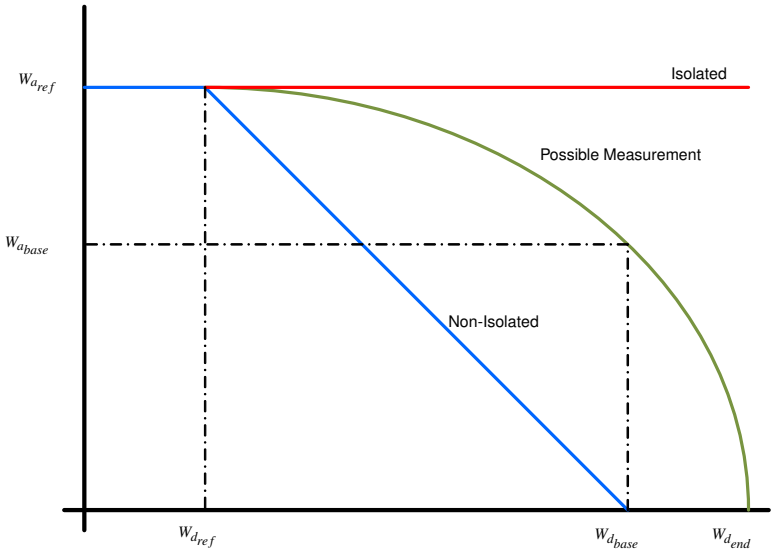


Figure 3.3.: Fictitious isolation curve, including upper and lower bounds.

The x-axis in Figure 3.3 shows the amount of workload W_d caused by the disruptive tenants, whereas the y-axis shows the amount of the workload W_a caused by the abiding tenants. The non-isolated line depicts how W_a has to decrease to maintain the same QoS as in the beginning. In a *Non-Isolated* system this function decreases linearly. For every additional amount added to the disruptive load, one has to remove the same amount at the abiding load, since in a non-isolated system the various workload groups would behave as they were one. In a perfectly isolated system, the increased W_d has no influence on $z_t(W)$ for all $t \in A$. Thus, W_a would be constant in this case, as shown by the *Isolated* in the figure. The isolated and non-isolated lines provide exact upper and lower bounds, corresponding to a perfectly isolated and a non-isolated system, respectively. Figure 3.3 shows some important points referenced later and defined in Table 3.2.

Based on this approach, several metrics are defined in the following. As discussed before, the workload scenarios play an important role, and thus it may be necessary to consider multiple different workload scenarios and average over them as previously.

3.1.4.1. Significant Points

The significant points marked in Figure 3.3 provide several ways to define an isolation metric by themselves. I_{end} is a metric derived from the point at which the workloads of abiding tenants have to be decreased to 0, to compensate the disruptive workload. The metric sets $W_{d_{end}}$ and $W_{a_{ref}}$ in relation. Due to the discussed relationship of the workloads in a non-isolated system and the definition of the various points, the condition $W_{a_{ref}} = W_{d_{base}} - W_{d_{ref}}$ holds. This relation helps to simplify the formulas. With Figure 3.3 in mind, I_{end} is defined as follows:

Symbol	Meaning
W_d	The total workload induced by the disruptive tenants: $W_d := \sum_{t \in D} w_t$
$W_{d_{base}}$	The level of the disruptive workload at which the abiding workload in a non-isolated system is decreased to 0 due to SLA violations.
$W_{d_{end}}$	The level of the disruptive workload at which the abiding workload is decreased to 0 in the system under test.
$W_{d_{ref}}$	The value of the disruptive workload at the reference point in the system under test. This is the point to which the degree of isolation is quantified. It is defined as the disruptive workload, at which in a non-isolated system the abiding workload begins to decrease.
W_a	The total workload induced by the abiding tenants: $W_a := \sum_{t \in A} w_t$
$W_{a_{ref}}$	The value of the abiding workload at the reference point $W_{d_{ref}}$ in the system under test. $W_{a_{ref}} := W_{d_{base}} - W_{d_{ref}}$
$W_{a_{base}}$	The value of the abiding workload corresponding to $W_{d_{base}}$ in the system under test.

Table 3.2.: Overiew and definition of relevant points for performance isolation.

$$I_{end} := \frac{W_{d_{end}} - W_{d_{base}}}{W_{a_{ref}}} \tag{3.5}$$

A value of 0 for I_{end} reflects a non-isolated system. Higher values reflect better isolated systems. A value of 1 is interpreted as being twice times better as a non-isolated system. In case of a perfect isolated system the value tends to ∞. This makes the metric rather difficult to be used in case of good isolation. A negative value may occur, if for some reason the performance of the abiding tenants is reduced more than the disruptive load increased.

Another approach uses $W_{a_{base}}$ as a reference. Setting this value and $W_{a_{ref}}$ in relation, results in an isolation metric having a value between $[0,1]$. The formula for metric I_{base} is below:

$$I_{base} := \frac{W_{a_{base}}}{W_{a_{ref}}} \tag{3.6}$$

A value of 0 for I_{base} reflects a non-isolated system, while a value of 1 expresses a perfect isolation. Both metrics have some drawbacks resulting from the fact, that they do not take the curve's progression into account. This means, that in a system which behaves linear until a short distance from $W_{d_{base}}$ and then suddenly drops to $W_a = 0$, both metrics would have the same value as in the case of a completely non-isolated system, which is obviously unfair in this case. Moreover, a well isolated system, requires a very high disruptive workload before W_a drops to 0, making it hard to measure the metric in an experimental environment. I_{base} has some further disadvantages, given that it is only representative for the behavior of the system, within the range of $W_{d_{ref}}$ and $W_{d_{base}}$. Given that the metric does not reflect what happens after $W_{d_{base}}$, it may lead to misleading results for well isolated systems, of which respective $W_{d_{end}}$ points differ significantly.

For systems that exhibit a linear degradation of abiding workload, and isolation metrics based on the angle between the observed abiding workloads line segment and the line segment which represents a non-isolated system can be used.

3.1.4.2. Integral Metrics

In the following two metrics addressing the discussed disadvantages of the above metrics are defined. They are based on the area under the curve derived for the measured system $A_{measured}$, and set in relation to the area under the curve corresponding to a non-isolated system $A_{nonIsolated}$. The area covered by the curve for a non-isolated system is calculated as $W_{a_{ref}}^2/2$.

Integral Limited to $\mathbf{W_{d_{base}}}$ The first metric $I_{intBase}$ represents the isolation as the ratio of $A_{measured}$ and $A_{nonIsolated}$ within the interval $[W_{d_{ref}}, W_{d_{base}}]$. $f_m : W_d \rightarrow W_a$ is defined as a function which returns the residual workload for the abiding tenants based on the workload of the disruptive tenants. Thus, it is possible to define the metric $I_{intBase}$ as follows:

$$I_{intBase} := \frac{\left(\int_{W_{d_{ref}}}^{W_{d_{base}}} f_m(W_d) dW_d \right) - W_{a_{ref}}^2/2}{W_{a_{ref}}^2/2} \tag{3.7}$$

$I_{intBase}$ has a value of 0 in cases the system is not isolated and a value of 1 if the system is perfectly isolated within the interval $[W_{d_{ref}}, W_{d_{base}}]$. The metrics major advantage is that they allow to set the system directly in relation to an isolated and non-isolated system. This metric again has the drawback, that it only captures the system behavior within $[W_{d_{ref}}, W_{d_{base}}]$. Again, a negative value may occur, if for some reason the performance of the abiding tenants is reduced more than the disruptive load increased.

Integral Without Predefined Intervalls In a well isolated system, it is not be feasible to measure the system behavior only up to W_{dbase}. Thus, the following metric $I_{intFree}$ allows to use any predefined artificial upper bound $p_{end} > W_{d_{base}}$, which represents the highest value of W_d that was measured in the system under test. The metric is defined as follows:

$$I_{intFree} := \frac{\left(\int_{W_{d_{ref}}}^{p_{end}} f_m(W_d) dW_d \right) - W_{a_{ref}}^2/2}{W_{a_{ref}} \cdot (p_{end} - W_{d_{ref}}) - W_{a_{ref}}^2/2} \tag{3.8}$$

This metric quantifies the degree of isolation provided by the system for a specified maximum level of injected disruptive workload p_{end}. A value of 1 represents a perfect isolation and a value of 0 a non-isolated system. Negative values for $I_{intFree}$ have the same interpretation as negative values for $I_{intBase}$.

3.1.5. Further Quality Aspects

Although the metrics described in Section 3.1.3 and Section 3.1.3 allow to quantify isolation, they do not adequately describe the time behavior of a system. Several methods ensuring performance isolation are based on an adaptive approach, which dynamically adapts the system configuration to ensure a proper isolation, often based on closed control loop. Consequently, one can assume the existence of situations where the system requires a certain amount of time to adapt to changes in the workload. Therefore, two additional metrics that are relevant to quantify the dynamic aspects of the performance isolation mechanisms are discussed.

Commonly discussed issues of controlled system in the literature are stability/oscillation, settling time/performance and accuracy/steady state error [Janert, 2013, pp. 19-21]. In context of the present work, the steady state error/accuracy is already covered by the metrics in Section 3.1.3 and Section 3.1.3. The other two aspects are covered in the following.

3.1.5.1. Settling Time

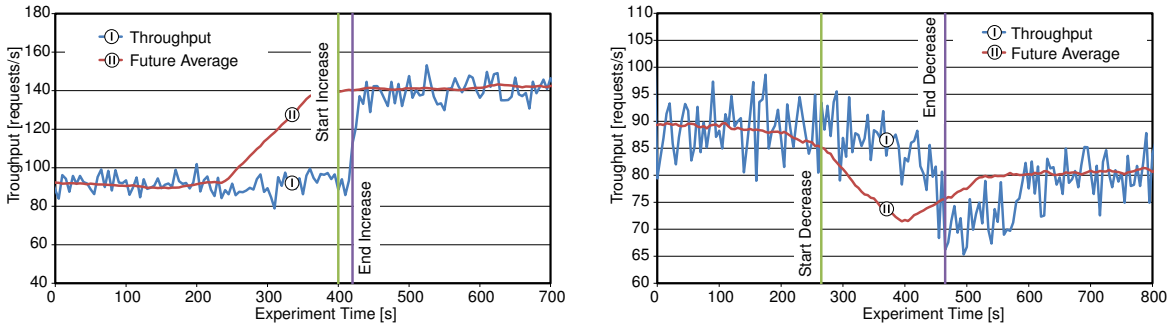
The settling time is a relevant aspect in the given context. The settling time, describes the time a system needs, after an instantaneous step at the input, to again achieve an output value within a defined error range. The faster this happens, the better is the system.

Ideally, a Dirac impulse would be used for the input. In the given context, the input value is the workload of the tenants, while the output is the observed performance of the quality metric under investigation (i.e., response time). Naturally, it is not possible to generate a Dirac impulse for such a system. Neither in physical systems, nor in a discrete system comparable to MTAs. Therefore, a step function can be used for this kind of analysis. However, even a significant increase of workload, to a constant value in very short time is not feasible. Therefore, the start event to measure the settling time is defined as the point in time at which the workload becomes stable again. The event to stop the measurement of the settling time is, when the observed quality metric is back at a stable value. Thereby, a certain error is acceptable. It is possible to relate this start and stop event to the guarantees relevant for a tenant. In this case, the start event is triggered, if the observed QoS is worse than the guarantee. The end event is triggered when it reaches the guarantee again.

However, if the value should not be related to the guarantee, another definition of stable is required. Therefore, the proposed metric considers the average response time of the next m to n samples in the future and compares it with the current one. If the load increases, one can expect a higher response time, which will decrease as the method tries to compensate this problem. At some point in time, this value will be close to the computed average or even cross this line, which marks the end event.

Figure 3.4a/b shows an example of one of the abiding tenants based on the evaluation results from Section 6.1.2.8. The two horizontal lines mark the beginning and the end of the time span, where the workload was modified.

Settling time similar metrics for adaptive IT-systems in the context of QoS metrics were already used before. One example is the CloudScale consortium [Brataas, 2014], which uses a metric re-



(a) Load increased.

(b) Load decreased.

Figure 3.4.: Example of measuring settling times.

ferred to as Mean Time To Quality Repair (MTTQR), to describe the time an elastic system needs to be SLA compliant, after an SLA violation occurred. Although MTTQR focuses on different scenarios, it is comparable to the here presented interpretation of settling time. However, one difference is the precise definition of a metric, even for situations where no guarantee is provided, or the system will never come back to a guarantee compliant state.

3.1.5.2. Oscillation

Oscillation can happen, if feedback from the system is used to adapt it to changing scenarios. Figure 3.5 shows an example of the abiding tenants, from the evaluation results in Section 6.1.2.6.

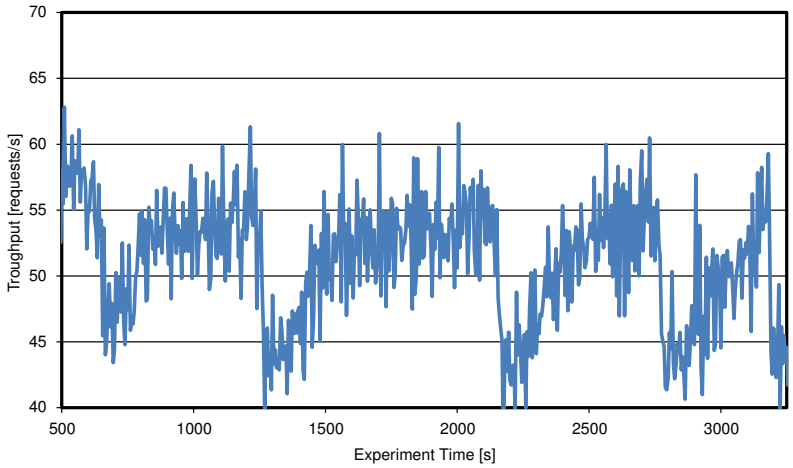


Figure 3.5.: Example of oscillation.

Oscillation describes an oscillating output of the system, while the input is constant. It is a known phenomenon [Janert, 2013, pp. 19-21]. The controllers are usually designed to damp the oscillation and the amplitude converges to zero. If this is the case, the settling time is a useful metric. If this is not the case, the controller maintains an unstable state.

Discrete systems with random inputs like an interactive web application, can be in a steady state concerning the average characteristics, while the input is still subject to random processes. Furthermore, if a closed system is investigated, the output influences the input. This increases the risk, that

the isolation method never converges to a steady state. The amplitude and the frequency of resulting oscillations are indicators to compare various solutions. For the purpose of performance isolation mechanisms, the amplitude would be based on the average, relative change of the quality metric under investigation. An average value for all tenants is valuable. Although this seems intuitively correct, the metric lacks in objectivity. In real systems, the oscillation is mixed with noise in the measurement and a clear oscillation might not be visible at all. Furthermore, if the system reacts very fast to minor and possible random changes, no repeatable pattern may occur. Thus, it is likely that a precise identification of the highest and lowest point of the oscillation is not possible. Consequently, it is difficult to clearly identify patterns coming from the active control mechanisms and usual random processes. Thus, a human has to define which signals are relevant and which are not. This is a violation of the objectivity and reliability. Moreover, it results in high efforts limiting its usefulness. Furthermore, the distribution of the measured data is unknown and potentially different for various isolation mechanisms.

Therefore, the size of the 25%-75% percentile in relation to the observed arithmetic mean or median value is conducted. In case of a high oscillation or high variability of the performance, the size is higher in comparison to one with low oscillation. This approach does not rely on the assessment of a human. The drawback is, that very strong noise is considered as oscillation. This metric is closely related to an existing approach referred to as performance variability (e.g., [Iosup et al., 2011b]).

3.1.6. Load Profiles

To provide a level playing field for comparisons, a description of a relevant workload profile is required together with the metrics. Although, the concrete definition of the relevant workload is a case to case decision and usually defined to answer a dedicated question, this subsection will share some general thoughts concerning this issue.

3.1.6.1. Reference Load

MTAs are operated at rather high utilization for economic reasons. Another important reason to run the test system under high utilization is the goal to evaluate performance isolation aspects. In a system with low utilization of the resources, the increased workload of one tenant has a low impact upon the performance of the others, as no bottleneck exists. Another aspect is related to existing guarantees. If the provider wants to maintain a certain QoS, it is possible to configure the overall reference workload in a way the average systems QoS is close to this value. In this case, a small increase of workload at the disruptive tenant, immediately results in violations for the abiding ones, in case of a weak isolation. In case no SLA based guarantee exists, and the bottleneck resource is unknown, a measurement to identify the system's maximum throughput, by increasing all tenants workloads in parallel is feasible. To increase the speed finding this point, a binary like search can be used (cf. [Shivam et al., 2008]). Usually this workload is accompanied with the highest possible utilization of the bottleneck resource. This means, that increasing the workload in a non-isolated

system, will immediately result in less performance for all other tenants. Consequently, it would immediately cause guarantee violations. Another argument is, that an isolation mechanisms should intervene latest at this point. The key findings for the reference workload are summarized as follows:

1. High load/utilisation preferable.
2. QoS observed should be close to the guarantee.
3. The systems maximum throughput is an additional indicator.

3.1.6.2. Divergent Work

The metrics definitions are based on a unit of workload. Assuming a situation where an application benchmark controls the number of users in a closed workload, the load is defined by the number of users. This works well for the defined metrics, as long as the call probability for the application functions and the related resource demand is the same for each tenant. If an MTA does not provide the means for intense customizing, the work induced by the tenants is rather homogeneous (except the load). This means, the requests from different tenants have a comparable impact upon the application's behavior and resources, which is usually the case, since they are using the same application. Therefore, the comparability between tenants is given and therefore the amount of load correlates with the amount of workload.

If this is not the case, the metrics are still valid, as they compare the amount of workload. In practice, additional measurements in a non-isolated system have to be taken, to compute the impact per amount of load of different tenants onto the system. This can be expressed as the tenants' impact ratio. To determine this ratio, additional measurements comparing various proportions of load, with steady QoS, are required. Additionally, if internal knowledge about the system's behavior and dependencies are given, it is possible to compute the metrics without adjusting the load at all.

3.1.6.3. Resulting Measurement Process

A coarse-grained description of four major steps is summarized in the following as starting point. However, a detailed description of a measurement process is not considered to be part of this thesis since it follows common patterns (e.g., [Herbst et al., 2015]). At first, the system has to be analyzed to gather a better understanding of the performance characteristics. At second, the reference workload has to be identified. Either, by referring to existing SLA definitions or by the insights from the first step. In both cases, the results from the first step are utilized to calibrate the reference workload for each tenant. At third, the load of the disruptive tenants is stepwise increased during an experiment series execution . Depending on the metric, the abiding load is adjusted accordingly. As outlined in Section 2.1.2.1, a warm up period after each load change has to be maintained. It must be at least as long as the settling time. In case the system has a high oscillation, the measurement duration has to be increased accordingly, to ensure a representative mean value is gathered. Based on the results from the previous step the performance isolation is derived.

3.1.7. Further Applications and Limitations

The proposed isolation metrics are not limited to MTAs. They may also be applicable in other scenarios where a system is shared, e.g., a web service triggered by other components, virtual machines hosted on the same hypervisor instance, or network devices serving packages from various sources. However, practical limitations might appear, because of missing a uniform behavior of the workload originating from different sources. As long as the requests have the same impact upon the system (e.g., network packages onto a switch) the application of the metrics should be possible.

Furthermore, the metrics are not specifically defined for one distinct QoS metric. The primary focus in this thesis is on response times. Nevertheless, it can be applied to other quality metrics as well. The proposed method to find an adequate reference workload might be adapted, if the QoS metric under investigation is not related to the workload.

3.2. Domain Independent Framework for Performance Isolation Measurement

The proposed metrics are derived from fundamental QoS metrics, and a dynamically adjusted measurement process. Consequently, they come along with a higher complexity compared to basic metrics such as response time. The metrics are independent of the domain they are applied to. To reduce the barrier to entry using these metrics, an environment for measuring the isolation in different domains was developed. It encapsulates the logic and algorithms for the metrics and measurement process in a technical/domain independent way. The result is a framework environment which provides predefined measurement approaches, but has to be enhanced with a concrete connector for each environment. As a basis, the Software Performance Cockpit (SoPeCo) was enhanced, which will be shortly introduced at the beginning of this subsection. After that, the isolation relevant aspects are presented.

This section has already been published in [Andrikopoulos et al., 2013a, Andrikopoulos et al., 2013b, Andrikopoulos et al., 2013c].

3.2.1. Overview of the Existing Software Performance Cockpit

The Software Performance Cockpit (SoPeCo): "... encapsulates knowledge about performance engineering, the system under test, and analyzes in a single application by providing a flexible, plug-in based architecture." [Westermann et al., 2010]. The goal of the domain independent framework for performance isolation measurement is to encapsulated the complexity of the measurement process and the analysis. Therefore, it is assumed that the SoPeCo can be used, because of the overlap in the goals.

The SoPeCo defines a metamodel for the definition of performance related experiments and interprets these models to run measurements on systems. The SoPeCo metamodel is defined generically, for arbitrary scenarios. However, for the usage in dedicated scenarios, a limitation of the functionality, by hiding irrelevant aspects, increases the usability. Another drawback of the SoPeCo is its usage to derive Performance Curves [Westermann and Momm, 2010, Wert et al., 2012] of

the observed performance parameter describing the analyzed system. In this situation the observed parameter is not the value of interest, it is just the foundation for the computed isolation metric. Therefore, the SoPeCo was enhanced.

Figure 3.6 describes the most important components of the SoPeCo and the enhancements for the purpose of performance isolation. The explanation of the original SoPeCo components in the following is based on [Westermann et al., 2010].

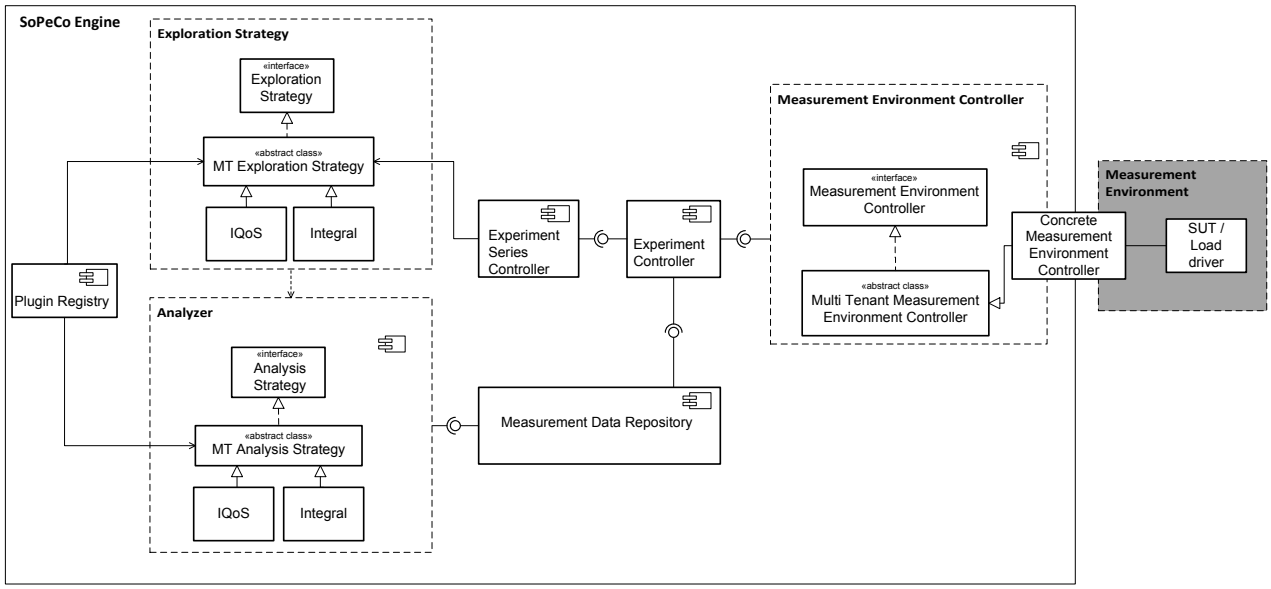


Figure 3.6.: Enhancements and Components of the Software Performance Cockpit.

The *SoPeCo Engine*s core artifacts read required configuration files, initially instantiate the required components and load the experiment series configuration, according to its definition described by an external file.

The *Experiment Series Controller* controls the execution of a concrete experiment. It loads the required plugins and artifacts and ensures that the experiment series is started.

A *Exploration Strategy* is an algorithm or heuristic defining which experiments should run within an experiment series. Usually it defines how the space spanned by the possible configuration parameters should be traversed. Examples are randomly selected parameter configurations for the experiment, or an equidistant selection of configurations. It is worth mentioning, that the selection of the next configuration can be dynamically adapted, based on the results of previous experiments.

The *Experiment Controller* runs one single experiment, which is defined by one dedicated selection of parameters. It triggers the measurement environment and stores the observed data in the repository.

The *Measurement Data Repository* stores all the relevant data for an experiment. This comprises the concrete observed performance metrics, as well as the configuration and metadata.

The *Analyzer* analyzes the measured data and supports various statistical analysis methods. This can be easily extended by providing additional methods as a plugin. The methods analyze the samples to create a performance curve of the system behavior. It can be used to predict parameter configurations not measured by using interpolation techniques.

The *Plugin Registry* is a management component to load and instantiate the plugins required at different locations.

The *Measurement Environment Controller* enables the communication with the SUT. It provides functionality to initialize the environment, generate load and to gather the relevant data. For each SUT a concrete implementation of the required interfaces is necessary.

The interplay of all these components allows to automate the measurement process to a large extent.

3.2.2. Multi-Tenancy Enhancements

This Section describes the elements that enhance the SoPeCo, to enable it for an easier measurement of performance isolation specific aspects. The SoPeCo was enhanced in two ways. At first, a wrapper was created which hides irrelevant aspects, groups relevant information and adds a performance isolation specific naming to make the usage more feasible. As second, functionality dedicated to the measurement of performance isolation was added to be reused in various contexts.

In detail, the following components were modified to enable the measurement of performance isolation: Measurement Environment Controller, Exploration Strategy, and Analysis Strategy, (cf. Figure 3.6). To keep it decoupled, the original components were only enhanced by functionality. Thus, adapters were implemented for the three relevant components. The *MT Exploration Strategy*, *MT Analysis Strategy*, and *MT Measurement Environment Controller*. The SoPeCo uses a generic mechanism to make it independent of any type of application or measurement goal. The developed adapters convert the generic input parameters (e.g., workload) into tenant-specific/isolation-specific parameter values covered in explicit objects, which are easier to understand, for a developer enhancing the framework.

The *Multi-Tenant Measurement Environment Controller* is the abstract super class for the system specific adapter class (*Concrete Measurement Environment Controller*) which communicates with the SUT.

3.2.3. Exploration Strategy Extensions

For each isolation metric, one Exploration Strategy and one Analysis Strategy is required. The first experiments conducted, define the reference measurement. The number of experiments is defined by an incrementally increasing workload for the disruptive tenant. Each time the load is increased, the MT Experiment Controller is conducted to run an experiment. The stop criteria for the experiments are either a maximum experiment time, or a maximum disruptive load, or in case of the Integral Exploration Strategy, the point at which the sum of abiding workload is zero.

The *IQoSMeasurementExplorationStrategy* continuously increases the load of the disruptive tenant for each experiment.

The *IntegralMeasurementExplorationStrategy* tries to find, for increasing disruptive loads, corresponding abiding workloads. Corresponding means: the observed quality metric of the abiding

tenants is the closest to reference load measured by an experiment. Therefore, the strategy continuously reduces the workload of abiding tenants by a configured step width, until it is below or equal the observed quality at the reference measurement.

3.2.4. Analysis Strategy Extensions

The *IQoS Strategy* and *Integral Analysis Strategy* classes realize the computation of the actual metric's value, once all experiments finished. In a first step they compute the isolation value for the parameter configurations measured. In a second step they return a function interpolating the isolation result between the measured configurations. Thus, it is possible to derive the isolation of the system over the whole range of disruptive load. This is required by the SoPeCo for further processing of the results. In case of the integral analysis strategy the necessary steps are:

1. Find the reference points in the data.
2. Extract measurements defining suitable workload configuration. They are identified by those measurements, where the abiding tenants QoS is closest to those at the reference workload, or the defined guarantee. This step is required, since all measurement results are stored.
3. Calculate the isolation for different disruptive workloads.
4. Linear interpolation splines are applied to derive a function of the whole range.

3.3. Benchmark Application

Besides a metric and a defined load profile, the work must be defined to compare a system's performance and consequently its performance isolation capabilities. It is important to define a representative work for the question which should be answered by a benchmark. Thus, the artificially generated load should represent the latter usage of the SUT. The focus of the thesis is in interactive multi-tenant web applications. Thus, an ideal work is a performance benchmark reflecting the performance behavior of an MTA. It is worth mentioning that such a benchmark application also provides the means for additional experiments. One potential use case is the comparison of different application runtime containers, or even lower level infrastructure elements, with regard to their performance for multi-tenant applications. Another option is the evaluation of a runtime container, which already provides means for MTAs, with regard to performance.

No benchmark was described before that concerns multi-tenancy support. Consequently, an extended version of an existing benchmark to support multi-tenancy was specified and implemented. This application can be used to evaluate the performance of an on premise middleware or PaaS, supporting MTAs, or for other experiments requiring a representative multi-tenant load.

In this section, an extended version of an existing benchmark to support multi-tenancy is presented. It is applied in a case study to show its feasibility.

This section has already been published in [Krebs et al., 2013].

3.3.1. Characteristics of a Multi-Tenant Benchmark Application

It is essential to represent the main characteristics of MTAs in a benchmark. In the foundations (cf. Section 2.1.1.1) some aspects were already discussed. Especially with regard to isolation aspects, MTA providers have to make several architectural decisions that have a major influence upon the performance. Furthermore, in case the application should be deployed on a PaaS scenario, it comes up with some additional, technical requirements concerning the actual implementation. In the following, a short overview of the most interesting architectural aspects for building a representative benchmark are outlined and discussed in the relevant depths.

3.3.1.1. Identification

One solution to identify a tenant is to connect the used information to the user. However, this approach requires an authentication of the user and duplicate users in different tenants are not possible. Therefore, applications usually identify the tenant by the host name. Whereby various host names point to the same IP. On the application, the tenant specific host name is used to identify the tenant. Usually the tenant's identifier information is transferred along the program execution path by binding it to the thread context.

3.3.1.2. Database

Three major approaches to separate a tenant's data from the data persisted by other tenants (Wang et al. [Wang et al., 2008] and Chong et al. [Chong et al., 2006]) exist. The highest degree of sharing, and thus efficiency, is established by the shared table/schema approach, which shares the same tables and schemas. This comes along with the largest consequences upon the application or platform, since the application or the platform provided persistence APIs have to take care of the tenant ID in each database statement. If the platform provides an abstraction of the database, it often handles the additional tenant ID in a transparent way (e.g., EclipseLink [Eclipse, 2013]). Koziolek [Koziolek, 2011][Koziolek, 2010] identified the shared table as the most common approach.

3.3.1.3. Tenant Metadata

In Section 2.1.1.1, it was outlined that an MTA usually reflects a 3-tier Web Application Architecture with an additional metadata storage for the tenant specific metadata (e.g., customizing, database id, tenant name, specific SLAs). Another essential element is the metadata manager, which enables access to this data and adjusts the application, according to the information stored in the metadata. The variability of information stored in the metadata is widely spread. However, at least an ID for the tenant, a display name of the tenant and a database identifier should be available. Dependent on the type of database multi-tenancy, the database identifier refers to an ID or a database connection. Platforms with multi-tenancy support usually provide these tenant metadata via defined interfaces (e.g., Google App Engine [Google, 2014], SAP HCP [SAP SE, 2014]).

3.3.1.4. Metrics for a Multi-Tenant Benchmark

In traditional application benchmarks, usually one or several QoS metrics are observed in relation to the number of simulated users or a request rate. A cost factor is sometimes included in the overall benchmark metric (e.g., [TPC, 2002, Islam et al., 2012]). Based on this information, a figure of merit describing the system is derived. In general, all these metrics can also be applied for a multi-tenant benchmark.

In a multi-tenant system, one can additionally consider the number of tenants. One potential metric is the throughput and response time, based on the number of tenants. Whereby, one tenant has a fixed workload. This metric might be of interest, when the per-tenant overhead and the total number of tenants a system can serve is important. Furthermore, it may answer the question about the optimal number of tenants for one application. Another metric defines a fixed number of tenants, by observing the QoS, based on the number of users for each. The proposed metrics for performance isolation are the most relevant for this thesis.

3.3.2. A Multi-Tenancy Benchmark

Subsequently, the TPC Web benchmark is introduced. It defines the basis for the multi-tenancy benchmark. The specification of the multi-tenant benchmark follows and some hints on its implementation are given.

3.3.2.1. TPC Web Benchmark

The Transaction Processing Performance Council (TPC) developed a transactional web e-Commerce benchmark (TPC-W) [TPC, 2002]. Its focus is on business oriented transactional web servers. The workload follows a controlled Internet commerce environment and simulates a bookshop. The benchmark simulates multiple on-line browser sessions by calling dynamically generated pages. The benchmark simulates three profiles, that differ by the browse to buy request ratio, resulting in various proportions of database reads or inserts/updates: primarily shopping, browsing and web-based ordering. The load can be varied by the number of Emulated Browsers (EBs). One EB simulates one user calling various application transactions, in a closed workload. Every EB realizes a user think time with an exponential distribution and a mean of 7s. In the following, the term emulated user, user and EB are used synonymously in the context of applying a benchmark. It is worth mentioning, that the TPC-W only specifies the benchmarking application, including implementation details like indices etc. to ensure the portability.

TPC-W is widely used in the industry and academia for similar goals as in this thesis. In Hellersteins book [Hellerstein et al., 2004] about control theory, and other control theory related publication (e.g., [Zhu et al., 2009]), TPC-W is used for the evaluation of the approaches. In Section 5.2.1 various classes for performance isolation will be introduced, which partially rely on feedback information from the system. Thus, TPC-W is an adequate reference. Padala et al. [Padala et al., 2009] used it to evaluate the service differentiation capabilities of their approach in shared environments.

As mentioned in Section 2.1.1.2, performance isolation methods can be used as a foundation to achieve service differentiation. Islam et al. apply TPC-W to quantify elasticity in Clouds [Islam et al., 2012] and Suleiman [Suleiman and Venugopal, 2013] validate their rule based elasticity models with its help on a widely used public cloud offering. The industry and academia driven project CloudScale [Brataas, 2014] uses it as basis for building scalable cloud applications as a part of their validation. Finally, Wang et al. [Wang et al., 2012] use TPC-W to validate their approach to achieve performance isolation in MTAs. In [Musabbir et al., 2013] multi-core and multi-threaded web servers' performance is optimized at runtime for web applications. As outlined in Section 2.1.1.1, this is the same basis application type as in MTAs. Summarized, several publications already used TPC-W in a similar context.

However, Binnig [Binnig et al., 2009] identified requirements for a cloud benchmark and analyzed TPC-W accordingly. Although, the authors identified some achievements, they see some problems if TPC-W should run against a cloud environment. The TPC-W requires a database with ACID [Elmasri and Navathe, 1999] properties, which they see as not present in cloud platforms. Further, they criticize the TPC-W metric, since in a cloud scaling environment the throughput will always increase with the load, and a relation to costs does not make sense for the given context. Other arguments tackle the missing support of modern technologies and the resulting call frequency patterns onto the server. Finally, the benchmark lacks in metrics for relevant cloud features (e.g., fault tolerance).

It is worth mentioning again, that the focus of the thesis is on MTAs and not on cloud environments in general. Moreover, ACID databases are widely provided as cloud services (e.g., SAP HCP and Amazon [Amazon, 2014]. The metrics of interest are redefined in the context of this thesis, or other appropriate metrics. Furthermore, infinite scalability and elasticity, which were described as a problem, play a negligible part in the thesis context (cf. Section 2.2.4). The benchmark interaction with the clients is based on HTTP, and similar to asynchronous technologies the workload strongly relies on the selection of the requested business data and transferring it to the client. In the present context, the benchmark is not used for general purpose cloud evaluations. Therefore, the missing metrics for relevant cloud features are not a problem.

TPC-W is a benchmark widely used for situations comparable to the thesis work and later proposed approaches. Thus, it is a representative application. Although Binnig et al. criticizes its application of cloud environments, it was clearly shown that the mentioned drawbacks are negligible compared to the advantages in the thesis context. Consequently, TPC-W is an appropriate basis for the validation of performance isolation aspects.

3.3.2.2. Specification of the MTTPC Web

The existing specification of the TPC-W is extended in several points to cover the relevant conceptual aspects described in Section 3.3.1.

The *Tenant Metadata Manager* (cf. Section 3.3.1.3) provided by the platform, is used to render the tenants display name as part of various web pages (Home Page, Customer Registration Page,

Buy Confirm Page). On one page (Buy Confirm) the tenant identifier is also rendered. This ensures the usage of a data element, which should be available in the metadata for any platform.

For environments with a native connection to one schema, in one Relational Database Management System (RDBMS) a tenant ID (integer) column is added to all tables. The primary key is a combination of the tenant ID and the entity specific ID field. In addition to the standard, an index is added to the tenant ID. Before each SQL request from the application, the metadata manager is called to get the tenants database ID. This ID is added to the conditional clause of every database request, to ensure data isolation.

Some database management systems do not support the auto generation of combined primary keys, whereby one element of the key is predefined. To ensure portability, the usage of a key-value table, with segment support, to reduce overhead is indicated. This solution uses a database table, which contains a key counter for each table and each tenant. To avoid overhead, this counter is not increased for every single insert. The key-value table is only accessed via an application local cache. This cache increases the counter by a count of 1000. Thus, it can return 1000 IDs before the next update on the key table. It has to be ensured, that increasing the database key counter by several application instances does not result in unsolved conflicts. This key counter mechanism is used to generate the primary keys. It is worth mentioning, that the tenant ID part of the key must not be generated, as this is a value given by the origin of the request that invoked the database update.

For environments with a native connection to various SQL servers or schemas for each tenant, the auto generation of the keys can be reused and the additional column for the tenant ID becomes obsolete. In these situations the database connection/schema is either provided in a transparent way by the platform or is stored in an application specific configuration, where it is mapped to the tenant. In the latter case, for every SQL request the appropriate connection must be selected on the basis of the tenants ID returned by the tenant metadata manager.

For environments with an API based access to the persistence layer, where the data isolation aspect is transparent to the application, the aforementioned methods lose their relevance. If the data isolation aspect is not transparent, the aforementioned solutions have to be considered.

The load driver has to support the platform specific identification mechanism of a tenant (cf. Section3.3.1.1). Possible variations of workload definitions follow the standard TPC-W benchmark, whereby every tenant uses the same workload definition. Since every tenant uses the same application, similar workload profiles can be expected.

Potential metrics were already discussed in Section 3.3.1.4. The relevant metric and the exact setup concerning the number of users for each tenant depends on the goal of the measurement.

For situations where the runtime environment does not provide a tenant identification, nor a metadata storage for the tenants, additional actions have to be taken. The tenant should be identified by the hostname as previously described. Due to the tight coupling to runtime server specific technologies no specification is provided here. However, the identification of the hostname and mapping to the tenant relevant data should follow a common approach. If available, it is recommended to use a thread context variable to identify the tenants while processing a request. The variable value

should be set by an application server specific technology, e.g., valve [Craig McClanahan, 2015] or request filter [Shing Wai Chan, 2013], since this reflects common implementations. Furthermore, an additional table has to be defined. This table consists of tenant ID (String), tenant database ID (Integer), tenant name (String) and an identifier (String) that is the host name. A metadata manager has to be implemented in addition. This manager accesses the table and the thread contexts variable to provide the information.

3.3.2.3. Realization of the MTTPC Web Benchmark

This section provides a detailed overview of the implementation for the sake of reproducibility and reimplementation. The basis of the version was implemented by Cain et al. [Cain et al., 2001]. It provides a java servlet based application that relies on a JDBC. These servlets access the database with the help of one central class. Figure 3.7 shows a simplified overview of the elements used in the enhanced version of the TPC-W benchmark. In the following, the function of the various elements, and how they are related to each other is explained.

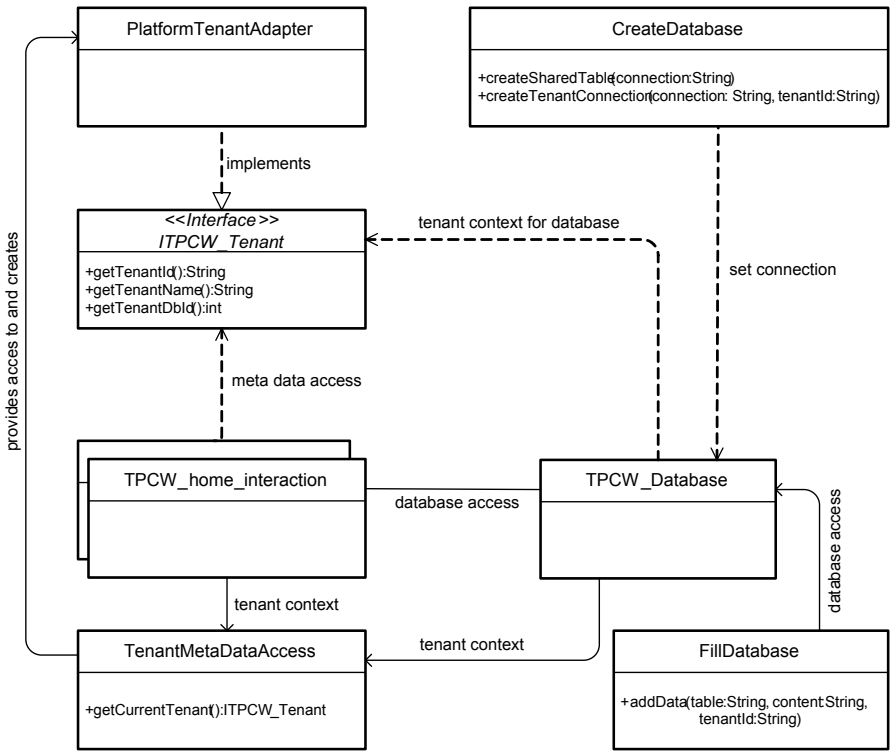


Figure 3.7.: Overview of the multi-tenant TPC-W benchmark.

TPCW_home_interaction is one example of 14 servlets available in the implementation. The servlets are responsible for rendering the html pages and the control flow. Every servlet has a reference to the *TenantMetaDataAccess* and uses an implementation of the interface *ITPCW_Tenant* implementations to access the concrete meta information for the tenant who owns the current thread.

TPCW_Database implements the communication with the database using JDBC. Its implementation follows Section 3.3.1.2. It also encapsulates the key generator aforementioned.

The *TenantMetaDataAccess* class enables the access to the platforms tenant metadata. It hides the platform specific implementation, to access the tenant specific information. Thus, it is possible to port the implementation onto another platform, by changing the implementation of this class. The *TenantMetaDataAccess* creates platform bound implementations of the interface *ITPCW_Tenant*.

The *PlatformTenantAdapter* uses the runtime container's mechanism to gather the tenant specific data. It has to be implemented specifically for each environment. In case one runs this implementation in a runtime environment which does not support multi-tenancy, this would be the place to access the tables providing the necessary information.

ITPCW_Tenant defines the interface which represents a concrete tenant. The class encapsulates the communication with the metadata manager to provide tenant specific information.

CreateDatabase also extends *HttpServlet* and is a proxy to create the required schema in the platform environment, if no direct access is available. The method *createSharedTable* creates a schema where the tables are shared among the tenants. Method *createTenantConnection* stores the tenant specific connection, and creates a schema without tenant ID at each connection. The corresponding connection and type of database multi-tenancy is then set at the *TPCW_Database*. Thus, using *createTenantConnection* enables separate schema and separate databases to be used. If the platform provides the tenant specific connections in a transparent way, one has to modify *TPCW_Database*.

In some PaaS environments, the access to the storage is only possible from the application runtime container. This becomes of importance, when a benchmark for PaaS environments is implemented, as usually the database must be set into a defined state. If a benchmark should be able to run on a PaaS, it has to provide technical measures to handle this. *FillDatabase* also extends HttpServlet and is a proxy to initialize the database data for the actual benchmark run. It uses the TPCW_Database to access the database by executing insert commands.

The *Load Driver* is based on the implementation of Cain [Cain et al., 2001]. The target platform under investigation differentiates tenants by the host name. Therefore, one instance of the load driver is created for each tenant, with a tenant specific hostname as the target.

3.4. First Assessment of the Metrics

The goal of this section is to assess the isolation metrics. The primary question of this evaluation is: *Do the metrics answer the questions they were defined for?*

To answer this, two case studies applying the metrics are provided, followed by a critical discussion and an analysis of the design goals on a conceptual level. In the following, the focus is in the QoS and Integral based metrics. As outlined in Section 3.1.5, the metrics covering the dynamic aspects of the systems do not necessarily imply the measurement of performance isolation and similar metric had already been applied in other contexts. In both case studies, setups are selected for which the order of their isolation capabilities is known, to validate the metrics. Beside the assessment provided here, all metrics, including the time dependent, are applied on the assessment of the developed performance isolation methods in Section 6.

This section has already been published in [Krebs et al., 2014b, Krebs et al., 2012b, Andrikopoulos et al., 2013a].

3.4.1. Simulation Based Case Study

This section presents the results of a simulation-based evaluation. This allows evaluating different concepts for isolation, and the metrics efficiently, without disturbing influences. It is an example, how a developer or architect may use the metrics to decide for one implementation.

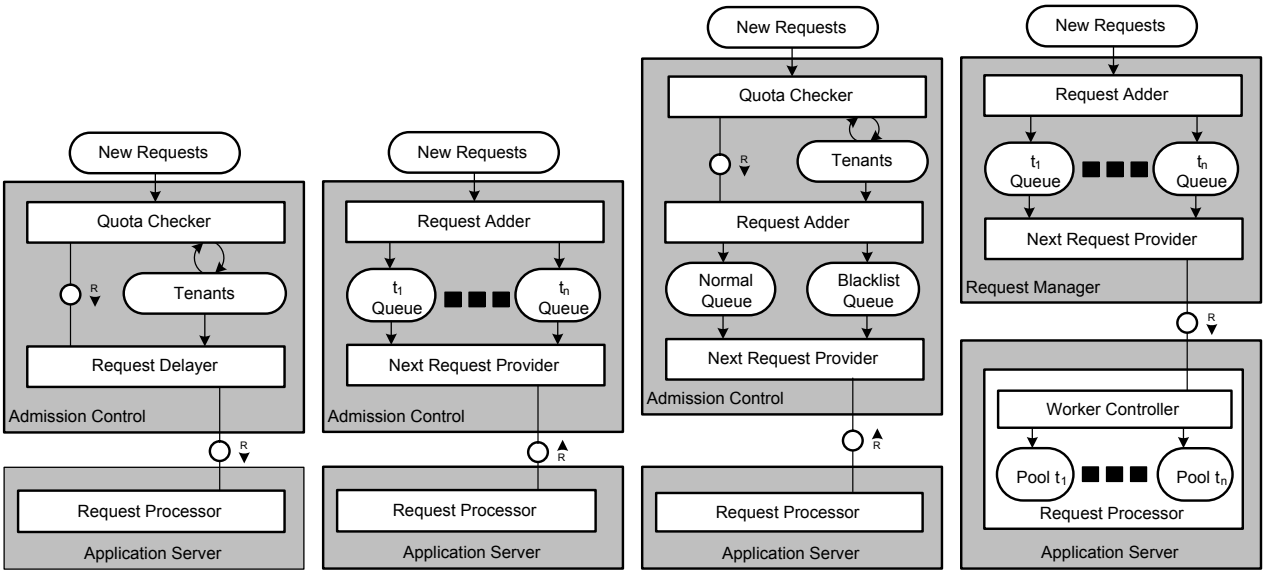
3.4.1.1. Simulation

The ssj discrete event simulation framework, including the provided stochastic features [L'Ecuyer and Buist, 2005] is employed. The major artifacts developed to simulate the shared system are the *RequestManager*, *RequestProcessor*, *Tenant* and *Scheduler*. The *RequestManager* is responsible to realize the different approaches for performance isolation and checks if the *RequestProcessor* has free resources to forward the next request. If no resources are available, the request is queued until the *RequestProcessor* signals that resources became free again. The *RequestProcessor* is responsible to simulate the request processing behavior, according to the predefined scheduling strategy. The *Scheduler* used within in the evaluation, simulates a resource, which is partially shared and assumes that the capacity of requests it can handle is limited. Further, the processing time for a request is not increasing linearly with the number of requests, to simulate trashing. Thus, the calculation of the residual service time for each request is based on the number of requests in the *RequestPocessor* and a user defined factor for the proportion of shared resources as well as a reference service time for the request in an unused system. The value becomes actualized every time a request arrives or departs the *RequestProcessor*. Once the request is processed, it is sent back to the corresponding *Tenant*. The *Tenant* instance, then simulates the think time and initializes the request for the next iteration.

To run the experiment, and analysis of the data an adapted version of the Performance Isolation Framework (PIF) is used (cf. Section 5.3.2).

3.4.1.2. Isolation Methods and Expected Results

In this section, four isolation methods are presented and their expected qualities are discussed. Later, this is used to compare the metrics' values with the expectations. All implementations try to enforce the isolation of response time. The concepts are based on request admission control and thread pool management mechanisms. In the following, figures explain the structure of the different approaches. Additionally, a short description will be found. Every approach implements two top level components: An *Admission Control* handling the incoming request and an *Application Server* providing the *Request Processor*. In the default case, the application server's request processor has one thread pool, with restricted size processing the requests.



(a) Artificial Delay (b) Round Robin (c) Black list (d) Thread pool

Figure 3.8.: Simulated methods to achieve performance isolation.

Artificial Delay This approach (Figure 3.8a) artificially delays incoming requests depending on the request rate of the corresponding tenant. In closed workload scenarios this result in artificially increased response times for tenants exceeding their quotas and generates some backpressure. Thus, the overall workload induced by a tenant is controlled. A new request arrives the admission controller's *Quota Checker* which evaluates the tenants currently used quotas, and stores the results together with the allowed quotas in the tenants metadata. After that, the quota checker triggers the request delayer, which possibly delays the processing of a request before it is forwarded to the Request Processor for processing. The duration of the artificial delay is constant.

Optionally the current demand of the system and the difference between the allowed and actual usage of the system might be used to calculate a dynamic delay. Within the application server, the request may be queued again, because of the restricted size of the thread pool. However, these aspects are realized in this simulation.

Round Robin Round Robin (Figure 3.8b) introduces separate queues for different tenants. There is no more need for a queue at the application server in this scenario, as requests are directly buffered at the Request Manager. When a new request approaches the system, it is queued in the corresponding tenant's *queue*. If the Request Processor has free threads, it triggers the *Next Request Provider* to deliver a new request. The Next Request Provider then uses round robin to retrieve the next request. An empty queue for one tenant is skipped and does not block the processing of the others.

Blacklist The blacklist method (Figure 3.8c) triggers the *quota checker* for every request. It checks if the quota for this particular tenant is exceeded. The quotas available to tenants and the quotas actually used by them are maintained in the tenants metadata. If a tenant exceeds its quota, it is blacklisted. Requests from blacklisted tenants are enqueued in a separate *Backlist* Queue. When

the Request Processer requests for the next request, the Next Request Provider takes the next request from the white queue on a FIFO basis. Usually, requests from the blacklist queue are only handled if the normal queue is empty. This leads to a problem in the simulation, when a tenant is removed from the blacklist, but he has requests still pending in the blacklist queue. If requests are always pending in the white queue, blacklisted requests will never be handled. Therefore, a mechanism that slowly processes requests from the blacklist (e.g., every 30th request) was implemented.

Thread Pools The separate thread pool method (Figure 3.8d) provides a separate thread *pool* for each tenant. The limited size of these pools isolates the tenants from each other. The conceptual model includes a separate FIFO *queue* for each tenant. Every time, one of the tenant specific thread pools has an idle thread, the *Worker Controller* requests a new request from the Next Request Provider. The *Next Request Provider* selects a pending request according from the tenants thread pool.

Expected Isolation Quality One can assume that round robin provides a very good isolation, since it is a widely used scheduling approach, providing a fair distribution of a resource shared by several entities. The same is expected by using separate thread pools for each tenant.

The blacklist approach can be expected to provide a good isolation over a wide range of disruptive workloads. However, due to the blocking of the processing of one tenant and the closed workload, the arrival rate will decrease and the tenant will again be white listed. This should result in an oscillation with less effective isolation.

The delay approach can be expected to be ineffective, because of its constant delay it will only be able to maintain the isolation to a certain value of disruptive workload.

3.4.1.3. Evaluation Scenarios

In this section, the workload profiles, the performance related metrics of interest and the configuration chosen for the evaluation are discussed.

QoS-Metrics and Considered Workload The performance metric in focus was the response time. The time was measured from the moment a simulated request left a tenant to the point in time a tenant received the response. Thus, $z_t(W)$ returned the average response time for t. As a measure for the workload caused by the tenants, the number of users associated with each tenant was used.

The workload profiles were described by the users' behavior, the type of requests sent, the number of tenants in each group D and A, and the number of users associated with each tenant.

In the simulation, all users sent requests of the same type with a mean think time of 1000ms, and a standard deviation of 100ms, in a closed workload scenario. It is expected, that systems run with a high utilization (cf. Section 3.1.6). Therefore, the system was designed to serve in total 80 users. The mean service time for a request in the system without contention and was 1000ms, with a standard deviation of 150ms.

A normal scenario, and one with over-commitment have been considered. In the first scenario, the quota was set to 8 users and in the over-committed 1 to 24. In both situations, only one disruptive tenant (t_0) was expected. The number of users in the first scenario was 8 for each tenant and in the over-committed one $t_0 = 24, t_1..t_3 = 8, t_4 = 4, t_5..t_8 = 1, t_9 = 24$.

Thus, the total workload was set to a value at which the system is already at its limit of 80 users, and the disruptive tenant allocates its full quota. This is considered to be the best reference point.

For the QoS-oriented metrics, disruptive workloads also have to be defined. For t_0, 24, 40 and 251 users were chosen in the normal mode. In the over-committed scenario, the number of users were set to 40, 56 and 251. For the averaged isolation metric I_{avg} the measurements started with 8 and stopped with 248 users in the normal scenario, and 24 to 264 users in the over-committed scenario. The values were increased by a step width of 40 users. In the following, the number of users is indicated by indices added to the various isolation symbols, in order to distinguish the results.

Configuration In the chosen configuration with a standard, non-tenant aware FIFO queue as *RequestManager*, the maximum throughput was achieved at 18 requests/s with a response time of 2110ms (cf. Figure 3.9) and 38 requests are processed in parallel. Thus, the size of the thread pool was restricted to 38 threads for an optimal throughput. Without a restricted thread pool, most of the presented performance isolation methods fail, as the *RequestManager* would always forward the requests to the processor. When 80 users are simulated, a standard FIFO queue result in an average response time of 3500*ms* and 62 requests in the system, whereby 24 are queued.

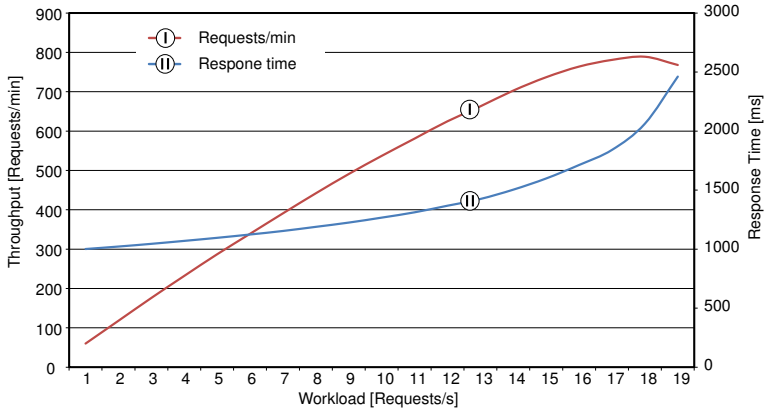


Figure 3.9.: Measurement of throughput and response time in simulation.

3.4.1.4. Evaluation Results

This section presents the results from the simulation described above and briefly comments on the observations made in the various considered scenarios. The overall assessment follows in a separate section.

Exemplary the $I_{qos_{24}}$ for the normal, non-isolated case is calculated and $I_{intFree_{251}}$ for the delay method in the over-committed scenario. In the non-isolated case, the simulation returned a response time of 3446ms at the reference workload of 8 users for t_0, and 4334ms at the disruptive workload

with 24 users for t_0. Due to the absence of isolation, the average response time for the abiding tenants is the same as for the disruptive tenants. This results in $\Delta z_{24} = \frac{4334-3446}{3446} \approx 0.258$. The relative increase of workload is $\Delta w = \frac{96-80}{80} \approx 0.2$. Consequently the isolation metric $I_{qos_{24}} = \frac{0.258}{0.2} \approx 1.29$.

In the delayed scenario with $I_{intFree_{251}}$, the point $p_{end} = 251$ and $W_{d_{ref}} = 24$. The integral describing the area under the curve of remaining abiding users $\int f(W_d)dW_d$ within the limits $[24, 251]$ was directly deduced from the measurements (cf. Figure 3.10) and has a value of 4687. $W_{a_{ref}}$ was set to 56 users in the workload definition. Thus, $W_{a_{ref}}^2/2 = 1568$ and consequently $I_{intFree_{251}} = \frac{4687-1568}{56\cdot(251-24)-1568} \approx 0.28$.

Concerning the overview of all results, the presentation begins with the QoS related metrics in Table 3.3. The value for the isolation in the non-isolated situation is almost the same in every case, since the impact on the performance is linear, because it stems from the extended length of the queue.

Approach	**Normal**				**Over-Committed**			
	$I_{QoS_{24}}$	$I_{QoS_{40}}$	$I_{QoS_{251}}$	I_{avg248}	$I_{QoS_{40}}$	$I_{QoS_{56}}$	$I_{QoS_{251}}$	I_{avg264}
Non-Isolated	1.29	1.29	1.29	1.29	1.29	1.29	1.29	1.29
Round Robin	0.00	0.00	0.00	0.00	0.02	0.02	0.06	0.03
Thread Pools	0.00	0.00	0.00	0.00	0.01	0.00	-0.01	0.00
Delay	0.32	0.59	1.22	0.9	-0.49	0.19	1.22	0.67
Black List	0.09	0.10	0.01	0.03	-0.73	-0.26	0.02	-0.03

Table 3.3.: Results of QoS based metrics.

Table 3.4 present the integral related metrics for the different approaches and workloads. The *N/A* entries stem from a very high value of $W_{d_{end}}$, which was not in the range of the evaluation. The rest of the section discusses different behaviors of the isolation methods and their impact upon the metrics, aligned with selected conspicuous measurements.

Approach	**Normal**				**Over-Committed**			
	I_{end}	I_{base}	$I_{intBase}$	$I_{intFree_{251}}$	I_{end}	I_{base}	$I_{intBase}$	$I_{intFree_{251}}$
Non-Isolated	0	0	0	0	0	0	0	0
Round Robin	N/A	1	1	1	N/A	1	1	0.99
Thread Pools	N/A	1	1	1	N/A	1	1	1
Delay	1.11	0.58	0.68	0.23	1.5	0.75	0.86	0.28
Black List	N/A	0.94	0.96	0.97	N/A	0.96	0.94	0.96

Table 3.4.: Results of workload ratio based metrics.

Round Robin This method provides a good isolation in every scenario. In the chosen system, the waiting queue for the disruptive tenant was never empty at the reference workload. Therefore,

the disruptive tenant was not able to disrupt the other tenants by increasing its workload. In cases where at the reference workload the queue of the disruptive tenant runs empty, the increased load is expected to influence the other tenants.

Separate Thread Pools In the normal mode, the size of the thread pools was set to 4 which closely corresponds with the 38 allowed threads in the system. To keep the response time for the tenants with 24 users lower than 3500ms, it was required to increase the thread pool to 17 threads. This resulted in an overloaded situation, but thanks to the reduced waiting time in the queue, a response time of 3.5s at the reference workload was still achievable. Overall, the thread pool approach showed a very good isolation.

Artificial Delay The threshold for the artificial delay was based on the number of users logged in for one tenant. Figure 3.10 presents the plot of the measurement, based on the reduction of workload for the abiding tenants. The negative values of $I_{QoS_{40}}$ and $I_{QoS_{56}}$ stem from the constant penalty, added to every request from the disruptive tenants. Therefore, a part of the disruptive tenant's resources became available for the other tenants and consequently the QoS for the abiding tenants improved. This results in negative isolation values. The isolation works within a limited range, due to the constant character of the delay. This point can be seen in Figure 3.10 when the abiding workload begins to decrease.

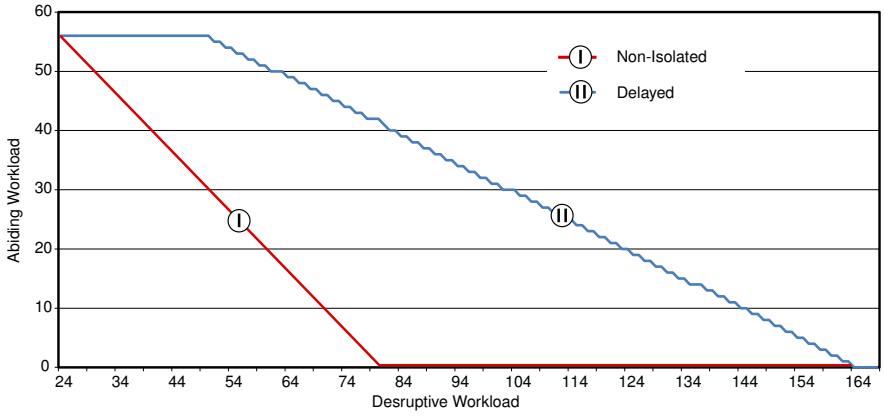


Figure 3.10.: Reduction of abiding workload while artificial delay is activated in the over-committed scenario.

Blacklist The blacklist exhibited a similar behavior as the artificial delay in the beginning. Therefore, a negative isolation was measured. The relevant metric for blacklisting the tenant is its throughput. The raw data shows, that sometimes a white tenant was blacklisted for a short while. This occurs in situations where the actual disruptive tenant becomes blacklisted and its portion of the resources become available for the other tenants. In this situation the response time and consequently the request rate of the abiding tenants improve and exceeds the quota. However, the effect is negligible with regard to the average response times.

3.4.2. Hypervisor Based Case Study

This section presents the results of a case study that investigates in virtualized hardware. By this, the applicability of the metrics in real environments was evaluated and gives some insights on the isolation capabilities of the widely used hypervisor Xen. Furthermore, it is an example how the metrics can be used by system owners, to decide for a deployment scenario.

As outlined in Section 1.1, the sharing of hardware resources by serving several operating systems on the same host, is a widely adopted technology, and the foundation for IaaS Clouds. Hence, it is one of the most common sharing scenarios. Xen [Xen, 2012] is a widely used Linux hypervisor. Thus, Xen was stressed with regard to performance isolation, by leveraging the previously described approach. More precisely, the degree of isolation was quantified for various Xen configurations and deployments, based on a black box approach, using the previously defined metrics. Therefore, several instances of the original TPC Benchmark W (TPC-W) [TPC, 2002] were deployed onto different VMs hosted by one Xen hypervisor. The case study shows the wide range of scenarios supported by the metrics and the results also allow reasoning for the isolation capabilities of IaaS Clouds running on Xen.

In the following, some details of Xen are described, followed by the scenario specific configuration and finally the results with a short discussion. More details about TPC-W were presented in Section 3.3.2.1.

3.4.2.1. Xen

Xen is one of the most common hypervisors for Linux environments. In order to configure the system, the hypervisor, and to execute administrative tasks, the first VM started in Xen (domain-0 or dom0) has special privileges. Furthermore, dom0 provides a driver abstraction for the different guest systems. The drivers in Xen are divided in two parts. The driver really accessing the hardware is installed in dom0, the guest systems (domU) drivers communicate with the dom0 to access the hardware [Gupta et al., 2006]. Consequently, dom0 can become a bottleneck for various activities. Especially I/O intensive tasks are known to produce high overhead in dom0 and thus the independent guest domains are likely to influence each other on these tasks. Such a behavior was already observed by several authors (e.g., [Huber et al., 2011, Gupta et al., 2006]. By default, the various VMs have access to all existing resources. To increase performance and isolation, it is possible to exclusively pin a core to a domain. However, this does not reduce the influence the domains have within dom0. It is worth mentioning, that dom0 usually does not host any services for the actual end user, due to its administrative role.

3.4.2.2. System Landscape

The physical landscape comprised two servers with 4x2133 Mhz and 16 GiB main memory. On both servers Xen 4.1 was installed and Suse Linux Enterprise (SLES) 11 SP2 was running on dom0 and on the guest systems. The servers were connected with a 1 Gbit Ethernet link. One server hosted

the load driver for the TPC-W benchmark in dom0. The various domains of the second server are described by the scenario specific configuration in Section 3.4.2.3.

In a first measurement it was observed, that increasing the load by more than a configuration specific maximum results in timeout exceptions or socket/file handle issues. Thus, a further increase of the load is no longer representative, because the induced demand is no longer equivalent to the load induced before this maximum was reached. Consequently, it does not represent the corresponding demand for the abiding domains anymore. Therefore, the system was tweaked in several ways. The application servers' HTTP timeouts were set to be infinite; the operating system's socket timeouts to be around six minutes and the maximum number of open TCP connections was increased to the operating systems maximum value. Besides this, one has to avoid domain internal (software) bottlenecks, because this hinders the system to increase the load for the shared hardware resources under investigation. Therefore, several measurements to find the optimum thread pool size and connection pool limits were done before the actual isolation measurements of each scenario.

3.4.2.3. Evaluation Scenarios and Expected Results

Overall, three different scenarios were investigated in this case study. In the *pinned* scenario the server hosted four guest systems (domU) and dom0. Every domU had a fixed memory allocation of 3096MiB and hosted a MySQL version 5.0 database and an LJS. The various domains were exclusively pinned to the existing cores. Thus, no competition for the same CPU resources was possible. Based on this runtime environment, four separate instances of the TPC-W bookshop application were deployed.

In the *unpinned* scenario, all domU and the dom0 were not pinned to a dedicated CPU and free to use any existing hardware resource. Xen's credit scheduler was chosen to allocate the domains to the various resources.

In addition to this, an *unpinned two-tier* scenario, which also had no fixed CPU pinning and likewise uses the Xen credit scheduler [Cherkasova et al., 2007] was investigated. However, the database and the application server in this case were deployed onto separate domains. Every domU, with an application server had a fixed memory allocation of 2024 MiB and the database domain allocated 1024 MiB. This memory setup was chosen, because of a small database volume.

Table 3.5 shows the values used to define the reference and maximum disruptive workloads for the various scenarios. The number of users at the maximum accumulated throughput of all domains is presented in the second column. The corresponding accumulated throughput, the per domain throughput and average response times are listed next. The last column shows the disruptive domains number of users, from which on the services functionality was no longer guaranteed. In the unpinned two-tier scenario, different values for the QoS based and workload ratio based metrics exist. Since in the workload ratio case, the load from the abiding tenants is reduced, and thus the overall load, the server becomes as much overloaded as in the other cases.

The highest difference in throughput for one domain, compared to the mean of all domains was around 4.5% and the highest difference of the response times around 6.5% in the pinned scenario.

Scenario	User per Installation	Throughput (Sum)	Throughput per domU	Response Time	Max. Load Disruptive
Pinned	3000	1195 req./s	299 req./s	1104 ms	15000
Unpinned	1500	721 req./s	180 req./s	842 ms	13500
Unpinned two-tier QoS	1300	617 req./s	154 req./s	833 ms	8000
Unpinned two-tier ratio	1300	617 req./s	154 req./s	833 ms	11050

Table 3.5.: Results for the scenario setup and configuration.

In the unpinned case 2.2% (one tier) and 2.7% (two-tier) difference in throughput were observed. The difference of the response times was at 8.2%(one tier) and 9.4% (two-tier).

As a consequence of the observations from Table 3.5, the maximum disruptive load was set to 15000 users for the pinned scenario, and to 13500 for the unpinned. In the unpinned two-tier scenario 11050 users were configured and the measurements had to stop for the I_{QoS} metrics at 8000 users. It is worth mentioning, that in both unpinned scenarios the maximum disruptive load was close to nine times the load of the maximum throughput for one domain.

A fixed resource allocation and a low I/O load is given in the pinned one tier deployment, and thus it is expected to be the best isolated. In the unpinned one tier scenario the I/O load is still low and thus the influence on dom0 should be low. However, the processing resources are shared and thus the unpinned one tier deployment should be the second best isolated. The unpinned two-tier deployment is expected to have the lowest isolation capabilities, since it has neither a fixed allocated processing resource nor a low I/O, which is known for weak isolation [Gupta et al., 2006, Huber et al., 2010].

3.4.2.4. Evaluation Results

This section provides an overview of the measurement results and the observed isolation metrics. Figure 3.11 combines the results for both unpinned scenarios, based on the normalized values for the abiding and disruptive load. Table 3.6 presents the QoS based metrics based on the same values of Δw. Thus, the results provide a comparable view onto both deployments.

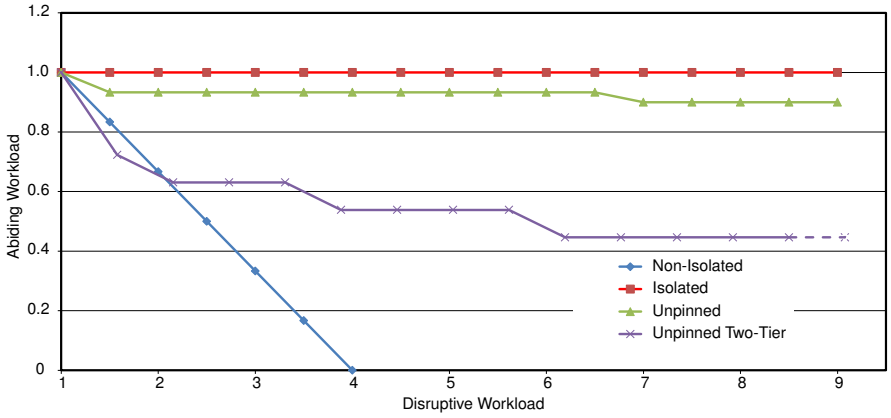


Figure 3.11.: Normalized reduction of abiding workload in the unpinned and unpinned two-tier scenario.

Table 3.6 contains the values of I_{QoS} for all scenarios investigated. The first column of Table 3.6 identifies the scenario, the second the number of users for the disruptive domain, the third column the average response time of all abiding domains followed by the results for Δw, Δz, I_{QoS} and I_{avg}. For the pinned scenario, only one measurement was done due to the very good isolation. To ensure a level playing field of comparison, Δw is equal for the relevant scenarios. The I_{avg} values were calculated based on an interpolation of the measurements of Δw at 0.33, 0.60, 1.05 in the unpinned two-tier scenario and additionally 1.47 in the unpinned scenario.

Scenario	**Disruptive load**	**Response time**	**Δw**	**Δz**	**I_{QoS}**	**I_{avg}**
Pinned	15000	1317	4.00	0.19	0.05	0.03
Unpinned	3200	927ms	0.33	0.10	0.30	0.18
	4800	942ms	0.60	0.12	0.20	
	7500	914ms	1.05	0.09	0.08	
	10000	1173ms	1.47	0.39	0.27	
Unpinned two-tier	3000	1011ms	0.33	0.21	0.64	4.20
	4400	3784ms	0.60	3.54	5.90	
	6750	4354ms	1.05	4.22	4.02	

Table 3.6.: Results of I_{QoS} in the various scenarios.

Pinned Overall, this scenario presented a nearly perfect isolation throughout the whole range. The I_{QoS} presented in Table 3.6 at a disruptive load of 15000 users was below 0.05 and the I_{avg} resulted in 0.04.

The workload ratio based metric decreased for the abiding workload only once at the last experiment done in the series. The ratio related metrics $I_{intFree15000}$ and $I_{intBase}$ result in a value short below 1. The isolation is the same as for an isolated system, except the last measurement. Therefore, it was omitted in Figure 3.11 for the sake of readability.

Unpinned For the metrics based on the QoS impact the isolation was observed at various disruptive workloads shown in Table 3.6. The response increased significantly at the first increase of disruptive load to 3200 users and at 10000 users. Accordingly, the isolation became better between 3200 users and 10000 users. This is, because of the widely constant response times by increasing load, which changed the ratio of $\Delta z/\Delta w$. In average the isolation (I_{avg}) was 0.18.

Figure 3.11 presents the total abiding workload W_a based on the disruptive tenants load increase. Similar to the I_{QoS} based results, two significant points were observed at the same position. In both cases, W_a decreased, because of an increasing response time at the abiding tenants. At a disruptive load of 13500 users (corresponds to 9 in the figure) the disruptive domain failed to successfully

handle incoming requests. Therefore, the results were no longer valid for higher disruptive loads. The overall isolation values are $I_{intFree13500} = 0.89$ and $I_{intBase} = 0.86$.

Two-tier Table 3.6 shows the various disruptive loads used to evaluate I_{QoS}. The disruptive loads were configured in a way, they result in the same Δw as in the unpinned single-tier scenarios. Due to the increasing number of timeouts and exceptions at the disruptive domain, the measurements were stopped at 6750 users. In this workload range, a continuous increasing response time was observed. Nevertheless, from 4400 users to 6750 users the isolation became better, because Δw increased more than Δz. Over the whole range of the measurements, the average isolation I_{avg} is 4.20.

Figure 3.11 presents the total abiding workload W_a, based on the disruptive users for the workload ratio based metrics. Analogously to the response times in Table 3.6, the abiding workload continuously decreases in Figure 3.11. At 1.5 in the figure, the observed isolation curve crosses the characteristic of a non-isolated system. This is, due to the selected step width for reducing the number of users in the abiding domains. At a disruptive load of 11050 users (corresponds to 8.5 in the figure), the disruptive domain failed to successfully handle incoming requests. These results were no longer valid for higher disruptive loads and are therefore illustrated using a dashed line. The overall isolation values are $I_{intFree1105} = 0.42$ and $I_{intBase} = 0.36$.

3.4.2.5. Insights on the Effectiveness of the Deployment Options

Overall, the *pinned* scenario shows the best results and the *unpinned two-tier* the worst. The selected size of the database was small enough to be mostly cached. The memory was not over-committed in the setup and the network I/O did not reach a critical point, at which the CPUs for dom0 became a bottleneck in the one tier scenarios. Therefore, the isolation was nearly perfect with pinned CPUs. In the *unpinned* scenario, the resources of the domU became shared with those for dom0, therefore the slightly increasing I/O overhead for dom0 was competing for resources and had some minor effect. The credit scheduler was not able to completely compensate this. By splitting the dom0 into an application server and database server the network I/O increased. In this setup a significant impact of the disruptive domain onto the others was observable, whereby the handling of the I/O in dom0 led to a bottleneck or requested additional processing resources from the guest domains.

When an administrator has to decide for one of the mentioned deployments, various considerations may be of importance. In a pinned setup, the overall performance and isolation is the best. However, unused resources of one domain cannot be used by other domains, thus this setup lacks in efficiency. The *unpinned* scenario overcomes this drawback, but at the expenses of performance and isolation. From a separation of concerns point of view, it is beneficial to separate database and application. However, a distributed deployment is less performant as table 3.5 shows and the isolation is the worst.

The case study showed, how the isolation metrics provide the opportunity to quantify an additional dimension in the framework of several trade-off questions a system provider has to answer.

Moreover, the results follow the assumptions that an administrator can increase isolation by hard resource allocation and deployments which reduce I/O.

3.4.3. Discussion

The discussions primary subject is the metrics. The domain independent framework was applied in both case studies. The required plugins to adapt to the system were developed without any changes to the framework itself.

For the evaluation of the metrics, the explanation concentrates on the following aspects. At first, how feasible is the metric for the target group of a system owner/provider or a developer/researcher. At second, the expressiveness of the metric in terms of the type of evidence it provides. At the third, the number of measurements required to obtain a valid value. Fourth, situations in which the metric is not meaningful.

3.4.3.1. QoS Impact

These metrics show the influence of disruptive workloads on the QoS of abiding tenants. This helps system owners to manage their systems, since it indicates the influence of disruptive workloads onto the QoS they provide, which is important for capacity planning. QoS-based metrics can prove that a system is perfectly isolated. However, they fail at ranking a system's isolation capability into the range between isolated and non-isolated. Thus, it is hard to estimate the potential of the method. In the simulation shown, it was possible to measure the system's behavior in a non-isolated case. In reality this is rarely possible, as a system owner or user might not be able to change, or even set off, the system's isolation method. A single I_{QoS} metric can be derived with only two measurements, to obtain evidence for one point of increased workload. However, to obtain some more detailed information concerning the system's isolation, more measurements are required. Therefore, I_{avg} describes the average isolation value within the upper and lower bound of interest. Nevertheless, the metric is not suitable to describe a system's behavior for different disruptive workloads onto the abiding tenants, because it cannot be set into a relation for a concrete scenario. Thus the advantage of this metric is rather in comparing different systems.

3.4.3.2. Significant Points

The metric I_{end} is unfeasible to quantify isolation in systems with good isolation. Furthermore, it is not possible to directly deduce relevant system behaviors such as response times. If the metric is given, it can help to compare two systems, regarding the maximum disruptive load they can handle. To determine I_{end}, more measurements as for QoS-based metrics are required.

I_{base} orders a system within the range of isolated and non-isolated systems for one specific point in the diagram. Nevertheless, it does not provide information about the behavior of the system before that point. It is limited to comparing the isolation behavior of the systems at one selected load level and it is inadequate to derive direct QoS-related values. The usefulness of this metric

appears to be of limited value, in contrast to the integral methods. One advantage is the evidence at a well-defined and reliable point, with measurements for only two disruptive load configurations.

3.4.3.3. Integral Metrics

$I_{intBase}$ and $I_{intFree}$ are widely comparable metrics. $I_{intBase}$ has the advantage to be measured at a predefined point. For $I_{intFree}$, the endpoint of the interval investigated must be considered to have an expressive metric. Both metrics provide good evidence of the isolation within the considered interval, ordered between the magnitudes of isolated and non-isolated systems. They lack in providing information concerning the degree of a guarantee violation. Thus the violation can be very low and acceptable or critically high in each iteration when W_a is reduced. However, in both cases, the results of the metrics are the same. This limits the value of $I_{intBase}$ and $I_{intFree}$ for system owners/providers. However, for comparison of systems and analyzing their behavior, the metrics are very useful and can be exploited by developers or researchers. Finally, on the negative side, a disadvantage of these metrics is that their measurement can be a time consuming task, compared to usual performance measurements.

3.4.3.4. Summary of Requirements and Achievements

In the previous two case studies, the metrics were applied in different environments, using the domain independent framework for the measurement of performance isolation. In this subsection, the validity/correctness of the metrics is discussed, followed by a recap of the requirements and question from Section 3.1.1 and how they were tackled by the proposed solutions.

Validity The metrics proposed are based on the impact of one tenant's workload onto, either the performance of the abiding tenants or on the amount of corresponding workload of the abiding tenants. Thus, both groups intuitively follow the Definition 2. Furthermore, in both studies, it was clear, how the various isolation and deployment approaches should be ranked with regard to their performance isolation capabilities. The results of the measurements and the derived metrics follow this ranking, and thus it is shown that the metrics worked correct.

Questions the Metrics Should Answer Three questions were identified as relevant for the metrics. To answer **Q1** and **Q2**, two separate metrics were developed. Metrics based on the integral of the workload ratios were developed together with I_{base}, to express how much potential a system's isolation method has to improve (**Q1**). The QoS impact based metrics were developed to quantify the impact of one tenant onto the others (**Q2**). All metrics can be used to compare alternative isolation methods (**Q3**).

General Requirements *Objectivity* is provided by the metrics, as they do not rely on a personal perception. However, the concrete result of a measurement highly depends on assumptions about the reference workload and experiment configuration (e.g., step width of increasing disruptive load),

which is influenced by a human. Therefore, it is important to report these things in detail, to make the result *reliable*. In general, the metrics themselves are reliable as long as the context in which they are measured is the same. This is the same, as for other commonly used metrics, e.g., response time, where the workload is needed to quantify the quality. However, especially for disruptive loads close to $\Delta W \rightarrow 0.0$ the workload ratio based metrics are not very robust. A minor modification of abiding load, due to noise, can lead to a low isolation values, since the difference between the isolated an non-isolated systems is very small (cf. Figure 3.3). QoS metrics may amplify minor changes in the observed quality since ΔW is is very small at the beginning. However, with increasing disruptive load this becomes negligible. The *economy* of the proposed metrics is a major obstacle. They require preliminary measurements to find an appropriate reference workload. Realistically, they cannot be gathered in an operative system, which requires a benchmark environment. Finally, the workload ratio based metrics require a high number of measurements. With regard to the *usefulness* they are limited to benchmarked systems. However, the questions the metrics are able to answer were motivated by examples of the daily business and the case study showed their applicability on various scenarios. Especially with their capability to entirely encapsulate the complexity in a framework, they are easily applied in different contexts, even by non-experts. Consequently, the results are important for various stakeholders, the metrics are applicable in relevant scenarios, and thus they are useful.

3.5. Conclusion

In this section, a brief summary of the results and contributions from this chapter are provided, including a tabular overview of the metrics presented. The section starts with a critical discussion.

3.5.1. Critical Discussion

Benchmark Specification Although metrics, a definition of the work, and recommendations for potential workload profiles were presented a real benchmark specification is not yet defined. A benchmark specification provides concrete run rules in addition. This is required to ensure a proper comparability between different independent measurements of systems. Examples for the sake of performance isolation are the measurement duration of each experiment, the maximum allowed standard deviation in a measurement or the creation of one single figure of merit.

However, creating a successful benchmark requires the support of a strong community. Therefore, almost all important benchmarks were initiated by a consortium of well-known stakeholders in the field investigated by the benchmark. They define, what they see as representative for their real world applications. However, researchers do not require such strict run rules, since they usually do not compare systems for economic reasons, but with a concrete research question in mind. This usually leads to goal specific metrics and setups, independent of the official benchmark. Therefore, the focus of this chapter was on the relevant research question which enables an interested party to add the technical definitions for a complete benchmark specification.

Therefore, only the generic and relevant points of the measurement process were sketched. Another point not discussed in detail, is how to treat potential negative values at the beginning of a workload ratio based measurement, caused by noise. Or isolation values within an experiment series, that reflect a system better as a perfect isolated one. This happens, if the disruptive tenant is disproportionately penalized. It can also happen that a system isolates worse than a non-isolated, e.g., when it runs in an overloaded and trashing state. How the load for the abiding tenants is reduced, in case not all have (the same degree of) guarantee violations was also not discussed. In principle, one can adjust the load individual for each tenant, or all by the same factor until all violations are solved. One of the most important aspects of a benchmark is to derive one single figure of merit, based on all the gathered metrics. However, several techniques about this topic were already discussed in the past (e.g., [Smith, 1988]) and not longer a relevant research question for the development of benchmarks.

All these issues are a question on how a benchmark will be defined and used. The proposed metrics and concepts already support these variants.

Adaptive Isolation Mechanisms Adaptive isolation mechanisms change their behavior to dynamically adapt to workload changes. This is a problem for the workload ratio based metrics, since they may change the system's behavior, when adapting the load within an experiment series. It is worth mentioning, that the metrics, benchmark and suggested load profiles are still valid and can be applied as presented. However, sometimes this does not provide an answer to the question investigated. Assume an isolation method that always adapts the behavior such, that abiding tenants exceed their guarantee by a small value. This would result in a non-isolated system. In this case, it is more appropriate to use internal knowledge about the isolation mechanism. Either it is possible to stop the dynamic adaption while doing the measurement, or one needs to derive a model of the isolation mechanism, to determine the workload ratios for a given disruptive load analytically, as if the system would not adapt.

Disruptive Load Definition The disruptive load was defined in relation to the overall load. Another possible solution might set it in relation to the load of the disruptive tenant only. However, the first has the advantage that the overall system performance is usually an already known value and often used for capacity planning. Thus, it is the potentially more common reference. Furthermore, it is independent of the reference workload. Thus, both solutions have their legitimation. The introduced concepts of the metrics, proposed load profiles and technical contributions can easily support both.

3.5.2. Summary

To evaluate a performance characteristic of a system, it is necessary to define a suitable and applicable metric, a load profile and the work. This chapter covers all these aspects for MTAs and performance isolation.

Metrics were defined to answer, how much potential an isolated system has to improve, what the impact of one tenant onto the others is, and how to compare alternative isolation methods (cf. Section 3.1). Thus, the various metrics show their advantages in different fields of application and have various semantics. The I_{QoS} and I_{avg} metrics represent the reduced QoS, based on disruptive load. They cannot provide a ranking within the range of isolated and non-isolated systems. However, for a system operator, they are helpful to estimate the impact of disruptive load onto the system. The I_{end} metric shows how many times a system is better than a non-isolated one. This information is helpful to compare different systems, if one has to decide for one. The Integral based metrics rank a system within the range of isolated/non-isolated. This knowledge is beneficial for the developer of a system, to estimate the potential for improvements. It is worth to outline again, that these metrics are not limited to the use in MTAs and can be used in shared systems fulfilling the requirements from Section 3.1.7. Table 3.7 provides a summary of the metrics and their value ranges. Additionally, suitable load profiles for the measurement were specified in Section 3.2. They depend on the maximum throughput a system can achieve.

Metric	Non-Isolated	Isolated	Max. Value	Min. Value
I_{QoS}	> 0	0	∞	$-\infty$
I_{avg}	> 0	0	∞	$-\infty$
I_{end}	0	∞	∞	-1
I_{base}	0	1	1^a	0
$I_{intBase}$	0	1	1^a	-1
$I_{intFree}$	0	1	1^a	-1^b

[a] If abiding load is allowed to increase, the maximum value is > 1 and dependent on $W_{d_{ref}}$.
[b] If $p_{end}W = d_{base}$, otherwise dependent on p_{end}. For $p_{end}W > d_{base}$ within $(-1,0)$.

Table 3.7.: Overview of the isolation metrics.

To increase the efficiency in applying the metrics to different domains and scenarios, a framework is described (cf. Section 3.2) that easily allows the adoption for various scenarios. It encapsulates the logic and algorithms for the metrics and measurement process in a technically/domain independent way. Thus it is shown, that the metrics can be technically applied to various scenarios if the framework can be used in various scenarios. That was shown by Westermann et al. [Westermann, 2013].

Both, the measurement framework and the metrics were applied in a case study using Xen, and a simulated MTA implementing simple isolation methods. In both studies, the metrics and the framework worked correct and followed the expectations (cf. 3.4.3). The case studies' results can be seen as contribution in itself. Section 3.4.1.4 already showed, that simple isolation method can achieve a certain degree of isolation, but is inefficient concerning the usage of resources or SLA awareness.

The important requirements and typical characteristics of MTAs are outlined in Section 3.3, as precondition to develop a benchmark application. The developed benchmark defined the work for

the Xen based experiments. The benchmark application itself is described independently from the isolation measurement, since it can be used for a wider range of experiments. An example is a case study, which compared the performance of an MTA with a deployment based on several application instances, using the hypervisor Xen [Krebs et al., 2013]. This shows the applicability of the developed benchmark beyond performance isolation.

The provided contributions are used in Section 6 to validate the performance isolation methods and the architecture. Thus, the applicability is further validated.

In summary, this chapter presents the contributions made to evaluate the performance isolation capabilities of MTAs. Thereby, the single contributions are not necessarily coupled to the multi-tenancy scenario, and can be used in other contexts. The chosen case studies to validate the contributions presented further insights into performance isolation methods.

4. Methods for Performance Isolation

This chapter presents solutions to ensure performance isolation in multi-tenant applications. As outlined in Section 1.3.3, several challenges have to be solved to achieve this. The most relevant is the layer discrepancy. It describes the lack of control of the application layer, while the lower layers, e.g., operating system, are not aware of the tenant concept. The workload is assumed to be interactive. The over-committed system should allow to shift resources between tenants, if they are not required by others, to maintain the guarantees whenever possible.

Consequently, the proposed isolation methods should be able to achieve the following design goals, which can be derived from the goals and challenges in Section 1.3.2.

Design Goal 1 — *Maintain Performance Isolated*
The methods have to performance isolate tenants sharing one application node. The definition of isolation follows Definition 2, including its relation to SLAs, and approaches to measure isolation presented in the Chapter 3. Furthermore, they have to reflect the fairness criteria from Definition 1. It is worth mentioning again, that high load scenarios with high response times, are of most interest (cf. Section 2.1.2.1).

Design Goal 2 — *Efficient Distribution of Resources*
Ensure, that the methods are able to efficiently use the resources provided. A strict isolation of resources would not allow leveraging workload fluctuations of tenants and thus would not allow over-commitment. Thus, the methods should provide a tenant the guaranteed service, even if he exceeds the quota, as long as it does not have a negative influence upon other tenants guarantee. This also reflects the third fairness criteria from Definition 1.

Design Goal 3 — *Low Overhead onto the Overall Performance*
Ensure that the influence upon the system's overall performance is low. A system with a bad overall performance, due to the used isolation method, can still fulfill all other requirements, but is insufficient for practical needs. Overhead includes the amount of additional resources required and the influence upon a request's processing.

Design Goal 4 — *Low Oscillation*

Avoid oscillation of the performance. Besides the actual isolation of performance, the variability of performance plays an important role for the user satisfaction. Although the performance could be isolated with regard to the average performance, it may happen that the performance changes between individual calls.

Design Goal 5 — *Short Settling Times*
Provide fast settling times, since an MTA's workload is likely to change often. Providing an isolated system and efficient sharing a long time after an unpredicted change in the workload, will lead to insufficient quality most of the time.

Design Goal 6 — *Applicability*
The solutions should have a low dependency on the surrounding environment or technologies. A method which achieves all previously outlined goals perfectly, but has many dependencies onto the infrastructure, has low chances to be widely accepted.

The first section introduces the general concept to achieve performance isolation. After that, two novel methods are introduced. One takes advantage of resource demand information to isolate resources and hence performance. The other uses a model to predict the performance of the system, combined with a fitness function for the optimization. These two methods are discussed in detail. Additionally, three methods, which are already conceptually known from similar scientific fields, are utilized and will be briefly discussed in this chapter.

To which degree the various goals are fulfilled, and the challenges were overcome, is part of the evaluation conducted in Chapter 6 and the Conclusion chapter.

4.1. Request Based Admission Control

Subsequently, a short excerpt of two potential approaches to provide performance isolation is presented, followed by a more detailed discussion of the approaches suggested in this thesis.

4.1.1. Alternative Approaches

Several solutions (cf. Section 2.3.1) try to place tenants or other entities onto a defined set of nodes, in a way their workload profiles do not interfere with others. This approach is only possible if the workload profiles for the tenants can be perfectly predicted. Thus, these solutions will not provide isolation between tenants on the same node.

To overcome the layer discrepancy, it might be possible to enhance the resource controlling layers by the tenancy concept. This means, that besides processes and threads, which are controlled by the operating system, a tenant entity would have to exist, which could be charged for the resources and thus enabling resource isolation. This would require all the other layers between the operating

system and the application to be aware of this concept, as the information must flow from the identification of the tenant to the operating system. Furthermore, this would tightly couple the application or middleware to an operating system. Consequently, the applicability would be limited and technological enhancements on various layers would be necessary.

In summary, solving the problems by the location of tenants does not provide proper isolation, while the way of enhancing the resource controlling layers has many technical dependencies.

4.1.2. Double-Staged Admission Control

Subsequently, a detailed discussion of the approach to achieve performance isolation is presented. Parts of this have already been published [Krebs and Mehta, 2013].

A negative performance influence from one tenant onto another, can only appear if requests from different tenants are handled in parallel by the system. Only in these situations, tenants compete for resources. If one tenant does not send any requests, it cannot have a negative impact upon the performance. Conclusively, if a request which would have a negative impact upon the performance of others is rejected, or delayed before it accesses the application, its impact is reduced. Thus, it can be hypothesized that controlling the requests admission to the application can isolate performance.

In Section 3.4.1.2, various scheduling mechanisms, e.g., round robin and black list were presented to decide, which request is going to be admitted next to the application. It was shown that these approaches can provide a certain degree of isolation. However, they lack in efficiency. Especially when the system becomes over-committed they cannot shift resources, due to constant priorities for each tenant. Methods like these, require a limitation of requests that are handled in parallel by the application. However, the maximum number of requests is usually limited anyway, to avoid the system from a trashing and overloaded state. Therefore, this is not a disadvantage. Automated approaches to determine the thread pool size at runtime exist (e.g., [Hellerstein et al., 2008]).

If a system is over-committed and a tenant generates the maximum allowed workload, he has to get more of the share to utilize resources not used by other tenants. Closed control loops are widely applied in different contexts to adapt a system, in case of a dynamic environment (cf. Section 2.3.2). However, in situations where one tenant suddenly changes its workload, a closed control loop needs some time to react to these changes. Furthermore, they may result in high variability of the performance due to overshooting and oscillating, in case of continuously changing conditions.

On the one hand, scheduling approaches have the benefit to react very fast to changes in the workload profile, as they do not have to wait until a control cycle is completed. On the other, they cannot dynamically adjust the priorities of the tenants to find an optimal setup. Therefore, a fast reacting scheduling mechanism is combined with a mechanism dynamically adjusting its configuration. This allows converging to optimal configurations, and additionally the impact of potential SLA violations can be limited on a short-term basis by the scheduler. Furthermore, if a tenant does not send as many requests as expected between two updates of the admission control, a work-conserving scheduler immediately prefers the other tenants and thus continues using all

resources. Thus, the system can be efficiently operated. It is worth mentioning, that such a system could be realized by a closed control loop or comparable approaches.

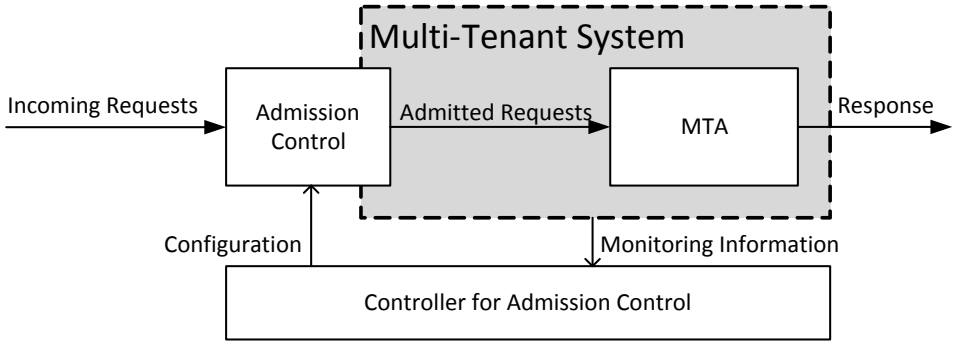


Figure 4.1.: General approach for performance isolation.

Figure 4.1 presents an overview of the idea. Incoming requests have to pass the *Admission Control*. This entity delays, rejects or directly admits a request. If a request is admitted, it is forwarded to the actual *MTA* which then generates the response. This is depicted by the leaving arrow to the right. The *Controller for Admission Control* collects any kind of operational data and updates the admission control. The concrete implementation of the admission control, the controller for the admission control and the data gathered depends on the concrete isolation method.

Dividing the approach in two parts allows to realize a low impact upon the application. Consequently, the admission control must be implemented so light weighted, that the influence upon the overall service time of a request is negligible. Furthermore, it allows to react very fast to sudden load changes by the admission control, while its configuration is optimized on the long-term. This optimization potentially comes along with high complexity. Although, the approach is not limited with regard to the type of admission control, the use of work-conserving schedulers is recommended. This ensures a sufficient utilization, even though a tenant immediately reduces load. In case a tenant suddenly increases the load, the impact is limited by the scheduler. A possible adjustment of the tenants weight is done by the controller for admission control, once enough data is gathered to react to the new situation.

4.2. Model Based Isolation

Performance models can be used to analyze the performance of systems. Some approaches use models at runtime [Kounev et al., 2010] to adapt a system to a dynamic environment at runtime. The models allow to find a suitable new configuration of the system. If this is found, the real system is reconfigured accordingly. The advantage is, that a solution can be found faster by changing and analyzing the model, instead of reconfiguring the real system and observing the results. Furthermore, once a proper configuration is found and applied, it will not change without a change in the context or the optimization goal. Therefore, the risk of unnecessary reconfigurations and oscillation is reduced. However, if the model calibration is based on information that is directly or indirectly influenced by the reconfiguration, oscillations may occur. This has to be reflected by the design of

the approach. Creating a representative model with a sufficient accuracy and an acceptable overhead is challenging. Furthermore, defining a suitable and fast optimization method matching the model is an additional challenge.

This section presents three complementary contributions. A method to estimate the performance of the MTA, a fitness function applied on the performance model to assess the potential reconfigurations and the admission controls scheduler for tenants. First of all, the general approach is presented together with a short discussion about the benefits of an analytical prediction. Then, the scheduler used for the admission control is depicted and the relevant properties are investigated. The performance prediction function is discussed afterwards, followed by a fitness function, optimizing the configuration of the scheduler.

4.2.1. General Approach

To analyze a performance model, one can either use analytic approaches or simulation based approaches. The latter ones require more time to gather the results, while they have the advantage to be able to analyze more complex/detailed scenarios and concept where no analytical approach exists yet [Brosig et al., 2015]. It is necessary to find a proper reconfiguration method applied onto the model for optimization. Actually, any mechanism that can be directly applied to the real system is applicable to the model to determine a good configuration. However, this necessarily results in several iterative steps to find a good configuration, resulting in a higher demand and longer settling times. Therefore, an analytically solvable description is more appropriate. To ensure that the complexity is manageable, a black box view onto the application is considered. The black box view encapsulates all internal aspects of the application, but not the scheduler itself, since this is the part controlled.

Figure 4.2 presents the overall approach of the solution.

The monitoring provides information about the number of active users per-tenant, the service times per tenant and arrival rates. Based on this information, the *Performance Prediction Model* can be derived. This model is completed by the *Weight Vector*, describing the share a tenant is allowed to use. A *Fitness Function* is then applied on the model to find the best possible weight vector, by using an *Optimization*. When it is found, the weights are applied on the Priority Fair Queuing (PFQ). The PFQ is always triggered when a thread becomes free at the application server and selects the next request for processing.

This general approach is not limited to the later identified fitness function, nor the specific model or scheduling mechanism. Although it was decided to use an analytical approach for the sake of this thesis, other approaches could be feasible, too. In these cases the data monitored might be different. For a better understanding, Figure 4.2 depicts the information for the method introduced later.

The subsequently presented method to predict the performance focuses on an interactive system. However, the general approach and the fitness function presented are not limited to such systems.

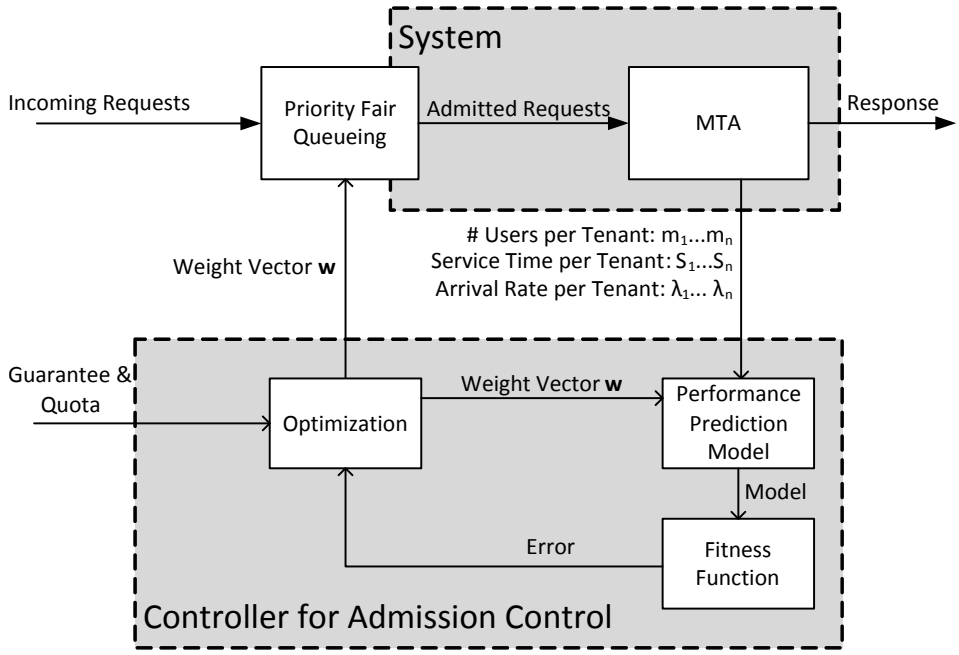


Figure 4.2.: Approach for the model based isolation.

4.2.2. Analysis of System Aspects

Important aspects of the system, for the latter introduced model, scheduler and fitness function are introduced. Thereby, the assumptions and thus the limitations of the concrete method become visible. This does not limit the general approach which can use other models, schedulers or fitness functions.

Each tenant t_i from $t_1, \ldots, t_n$ has a sufficiently high number of independent users. All users have the same workload characteristic, but behave statistically independently from each other. Consequently, the arrival process can be assumed to a Poisson process (cf. Section 2.1.2.2).

Common queuing network multi-servers queues assume that the throughput increases linearly with the number of servers. A server running n threads does not necessarily speed up the processing and throughput by factor n. A server's capacity is limited by hardware or software bottlenecks. With increasing load, one of these bottlenecks becomes saturated and limits the throughput. However, the system's maximum throughput can increase with the number of threads, since the requests do not all utilize the same resource at once. Ideally, the server's thread pool size is configured such, that no additional requests can be served at this point to avoid trashing (cf. Section 2.1.2.1). If the demands among requests are the same, they depart one after the other, in the order they arrive at the system.

4.2.3. Priority Fair Queuing for Performance Isolation

This section introduces the admission control's scheduling mechanism for the proposed solution. Additionally, it discusses the relevant aspects of the scheduler used later to describe the system's performance behavior.

4.2.3.1. Scheduling Algorithm

Algorithm 1 implements the PFQ mechanism. If every tenant sends enough requests, to be able to allocate its guaranteed quota, PFQ realizes a WFQ. The globally accessible variables a_i counts the overall number of already admitted requests for t_i in the accounting period T. Variable p_i keeps the priorities for each tenant. Its priority is computed dynamically, each time the algorithm is triggered. On its basis, the tenant served next is selected in a WFQ manner. The variable a_i is reset to 0 every T seconds. This avoids long imbalanced durations. T is a configurable parameter. Additional symbols are defined in Table 4.1.

Algorithm 1: Algorithm for the PFQ.

Input: $J = \{j_1, \ldots, j_k\}$: Tenants indices to be considered.
$\mathbf{w} = (w_1, \ldots, w_n)$: Weight vector compliant to $\sum_{j=1}^{n} w_j = 1$ and $w_j \geq 0$.
Output: Tenants index to be serviced next or null if no requests enqueued at all.
Data: a_i: Amount of requests admitted. The initial value is 0.

```
1 if J = ∅ then
2     return null
3 for j ∈ J do
4     p_j ← a_j / w_j
5 i ← arg min_{j∈M} p_j if queue of t_i is not empty then
6     a_i++ return i
7 else
8     return selectNextTenant(J \ {i}, w)
```

Line 1 checks if there are tenants to be served. If this is not the case null is returned. This stops the recursion (cf. line 9). If tenants exist, the priority is calculated in line 4 for every tenant. Within the period T, the PFQ scheduler aligns the actual admission ratios w_i^* with their corresponding weight w_i. The ratio of t_i admitted requests is given by $w_i^* := \frac{a_i}{a}$, where $a := \sum_{i=1}^{n} a_i$. This happens in line 4, where the admission counter a_i is normalized by the weight. This is based on the assumption, that always enough requests were sent by all t_i. In line 5 the tenant with the lowest priority is selected. The priority is the number of admitted requests, normalized by the tenants weight. If its queue is not empty (line 6) a is incremented and the corresponding identifier is returned. If the queue is empty, the method is called recursively without the tenant recently selected.

Algorithm 1 terminates in worst case $\mathcal{O}(n^2)$, where n is the number of tenants. Using a sorted priority list maintained among several calls, a reduction in $\mathcal{O}(n \cdot log(n))$ is possible.

4.2.3.2. Properties of the Scheduler

The weighted fairness described by $w_i^* \geq w_i$ is guaranteed if $\lambda_i/\lambda \geq w_i$. If the request arrivals are not homogeneous, PFQ does not necessarily balance priorities within consecutive calls. If for example a tenant sends all its requests shortly before the period T ends, it can happen that not enough time is left to serve all requests within T. Furthermore, a tenant has to send enough requests to be able

to allocate its entire share given by w_i. As long as $\lambda_i/\lambda \approx w_i$ in T, this is not a problem. In case of a closed system in steady state this is the case, since the weight indirectly controls the arrival rate. PFQ is work conserving, since it always serves the tenant with the highest priority and pending requests.

The proposed scheduler does not reflect varying demands for requests. This may lead to a resource distribution different to the weights and thus would not longer be fair.

The scheduler presented may remind a bit on a deficit scheduler. However, in a usual deficit scheduler the queues are served in a round robin manner, whereby each queue is processed as long as the credit is not used. This means, that the order in which tenants are served cannot change. The primary advantage of the PFQ is, that the tenant who used less of his guarantee also has the shortest delay before served the next time.

4.2.4. Performance Prediction Approach

This section introduces the model used to predict the performance of the MTA using the PFQ. It starts with a brief disambiguation in Table 4.1. After that, the used model is introduced. Table 4.1 introduces and recaps the relevant symbols, and relations required in the subsequent section.

In the following, some insights required later. $\lambda_i > \mu_i$ is possible if at least one tenant does not fully utilize its guaranteed service rates.

Assuming a multi-threaded server could use all its capacity κ to process one single request, O_i would be the time spent in average to finish the processing of it. However, this is only a theoretical value since κ can only be utilized by parallelism and a request's processing is not speed up by the number of threads running. Thus S_i will always be higher than O_i. Except in a single tasking system where both values are the same.

The utilization U has intuitively values in $[0,1]$ since $\lambda \leq \mu$. On the long-term, it is not possible that more requests arrive the system as the system leave, since users have to wait for the response in an interactive system. Therefore $\lambda \leq \mu$ always holds for interactive systems. However, this is not necessarily true for an individual tenant. U_i is the portion a tenant uses from μ_i. Since λ_i can be greater than μ_i, $U_i > 1$ is possible.

4.2.4.1. Upper Bounds for System Performance

A method to determine the upper bounds for R_i and N_i is presented in this section. In this context, Little's Law [Little, 1961] has to cover the definition of λ based on T.

$$N_i = \frac{\lambda_i}{T} R_i. \qquad \text{(Little's Law)}$$

A single tasking server is described first and then the description is adapted for the multi-threaded approach. A *cycle* from a tenant's (t_i) perspective is the time spent between the scheduling of two requests from the same tenant. A cycle starts exactly with the selection of a request, and ends with

Symbol	Description
$t_1, \ldots, t_n$	The set of tenants currently using the system.
T	Time period after which the PFQ is reset.
$\lambda_1, \ldots, \lambda_n$	Request arrivals within T for tenants $t_1, \ldots, t_n$.
λ	The overall arrival rate. $\lambda := \sum_{i=1}^{n} \lambda_i$
$w_1, \ldots, w_n$	Weights for tenants $t_1, \ldots, t_n$ to determine the share of the server.
$\mathbf{w}$	Weight vector describing all weights $\mathbf{w} = (w_1, \ldots, w_n)$. $\sum_{j=1}^{n} w_j = 1$ and $w_j \geq 0$
R_i	Average response time for all users who are member of t_i.
S_i	The average service time of requests of t_i. The time a service spends at the server while being processed. This time does not include the queuing time.
q_i	Queue of t_i.
N_i	Average number of requests of t_i in the system, including currently queued and processed requests.
N	Amount of overall requests in the system. $N := \sum_{i=1}^{n} N_i$
κ	Maximum capacity. It defines the work the server can handle within the time period T. It is the maximum amount of work levering the speed up of all threads.
κ_i	Guaranteed capacity for t_i within the period defined by T. $\kappa_i := w_i \kappa$
ν_i	The average demand of a request of t_i. It is the average, absolute amount of κ the request demands.
ν	Average demand over all tenants. $\nu := \sum_{j=1}^{n} \lambda_j \nu_j / \lambda$
O_i	Average occupation time of requests of t_i. The time a server needs to process a request if it spent all κ onto that single request. $O_i := \nu_i T / \kappa$
μ	Service rate, defining the maximum number of requests the server can handle in T. $\mu := \frac{\kappa}{\nu}$
μ_i	Guaranteed service rate for t_i. $\mu_i := \frac{\kappa_i}{\nu_i}$
U	Utilization of the MTA defined for $\lambda \leq \mu$. $U := \frac{\lambda}{\mu}$
U_i	Utilization of the tenant t_i. $U_i := \frac{\lambda_i}{\mu_i}$
m_i	Number of users per-tenant t_i.
Z_i	Average think time of users from t_i.

Table 4.1.: Overview of definitions and symbols relevant for the performance model.

the selection of the next request from the tenant. Each cycle length is tenant specific. It depends on w_i and the system performance. The average cycle length, in a backlogged scenario, is $L_i := T/\mu_i$.

Consequently, N_i cycles have to pass from the tenant's perspective before the processing of a new incoming request in q_i can start. A new arriving request from t_i has to wait until all preceding requests in q_i are processed. That is $N_i \cdot L_i$ (cf. Section 2.1.2.2).

After that, the service requires S_i for its own service. Combining these two aspects, the average upper bound of the response time for a new request arriving to the system is

$$
\begin{aligned}
& & R_i &\leq N_i \cdot L_i + S_i \\
& & R_i &\leq \frac{N_i T}{\mu_i} + S_i \\
& \overset{\text{(Little's Law)}}{\Longleftrightarrow} & R_i &\leq \frac{\lambda_i R_i}{\mu_i} + S_i \\
\Longleftrightarrow & & R_i(1 - \frac{\lambda_i}{\mu_i}) &\leq S_i \\
\Longleftrightarrow & & R_i &\leq \frac{S_i}{1 - \frac{\lambda_i}{\mu_i}} = \frac{S_i}{1 - U_i}.
\end{aligned}
$$

The upper bound for N_i can be calculated with Little's Law

$$N_i \leq \frac{(\lambda_i/T)\, S_i}{1 - U_i}.$$

In a multi-threading scenario, requests are processed in parallel by a preemptive scheduling. Thus, the processing of a single request is not linked to a fixed period in time, but is rather blurred over time. Subsequently, the validity of the previous approach is discussed for multi-threading. The parallelism leads to an overlap of cycles. Figure 4.3 illustrates this situation.

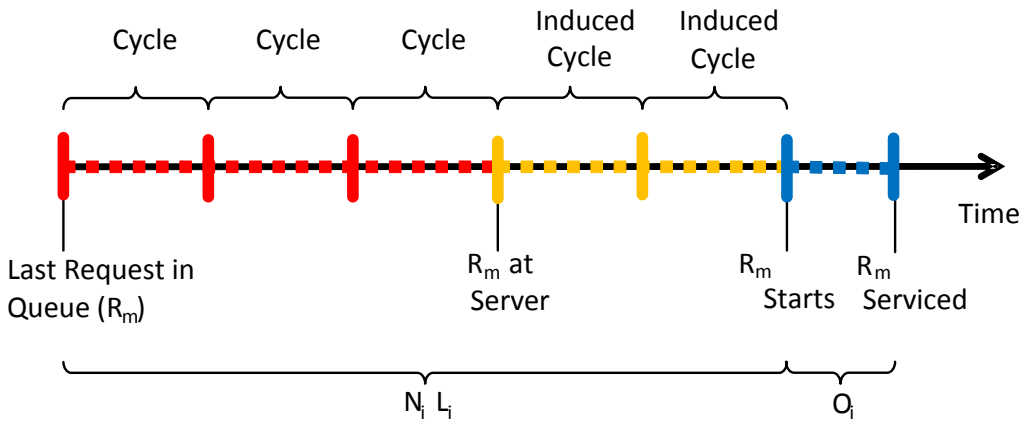


Figure 4.3.: Cycles in a multi-threaded system.

The system is expected to be in flow equilibrium in the following (cf. Section 2.1.2.2). Since several requests are serviced in parallel, additional cycles pass, while R_m is processed by the server, before the request is completely served. These cycles refer to *Induced Cycles* and they are as long as

those used to determine when a next request is removed from the queue. Thus, if the server would queue R_m internally, until all requests from t_i admitted before are finished, and would than spent κ_i for the processing, the time spent on the server would be the same.

Thus, the same approach as for the single task case can be applied.

In summary, assuming that all v_i are equal, the requests will depart the server in the order they arrived. A model assumption is, that the processing of requests served in parallel advances in cycles, similar to those waiting in the queue to become admitted. These are the induced cycles and they are as long as in q_i.

N_i still refers to the overall number of requests at the server's queue, or processing pool from the perspective of t_i. Based on this, N_i describes the sum of induced and normal cycles that must pass, before an arriving request of t_i is processed. The occupation time of the work admitted in each cycle L_i to the server, has to pass before the next request is processed. Thus, $N_i \cdot L_i$ has to pass before the new request's processing can start. When the processing of the request starts, it takes O_i until it is finished. This results in the following formula:

$$\begin{aligned} R_i &\leq N_i \cdot L_i + O_i \\ R_i &\leq \frac{N_i T}{\mu_i} + O_i \\ \overset{\text{(Little's Law)}}{\Longleftrightarrow} R_i &\leq \frac{O_i}{1-U_i}. \end{aligned}$$

An upper bound for N_i can be achieved by using Little's Law:

$$N_i \leq \frac{(\lambda_i/T)\,O_i}{1-U_i}.$$

The upper bounds are defined by

$$R_i^* := \frac{O_i}{1-U_i}, \quad N_i^* := \frac{(\lambda_i/T)\,O_i}{1-U_i}. \tag{4.1}$$

The upper bounds become sharp if all tenants have the same utilization. This is used later and therefore shown in the following.

Lemma 1

If $U_j = U_k$ for all $j,k \in \{1,\ldots,n\}$, then the upper bounds are sharp. This means $R_i = R_i^*$ and $N_i = N_i^*$ for each tenant t_i.

Proof of Lemma 1 : First it is shown that $U = U_j$ for all $j \in \{1,\ldots,n\}$. Since the premise is that $U_j = U_k$, for all $j \in \{1,\ldots,n\}$ $k \in \{1,\ldots,n\}$. There exists a factor α_k such that

$$\lambda_k = \alpha_k\,\lambda_j \text{ and } \mu_k = \alpha_k\,\mu_j.$$

Thus for all $j \in \{1,\ldots,n\}$

$$U_j = \frac{\lambda_j}{\mu_j} = \frac{(\sum_{k=1}^{n} \alpha_k)\,\lambda_j}{(\sum_{k=1}^{n} \alpha_k)\,\mu_j} = \frac{\sum_{k=1}^{n} \lambda_k}{\sum_{k=1}^{n} \mu_k} = \frac{\lambda}{\mu} = U.$$

To prove the Lemma, the following equations are applied.

1. $N_i \leq N_i^* = \frac{(\lambda_i/T)\,O_i}{1-U_i}$, $\forall i \in \{1,\ldots,n\}$,
2. $N = \frac{U}{1-U}$,
3. $U_j = U_k = U$, $\forall j,k \in \{1,\ldots,n\}$,
4. $\mu = \kappa/\nu$,
5. $\lambda\nu = \sum_{j=1}^{n} \lambda_j \nu_j$,
6. $O_i = \nu_i T/\kappa$.

The boundaries are sharp if $N_i \geq N_i^*$, since $N_i \leq N_i^*$ was already shown before. The equation $N = \sum_{j=1}^{n} N_j$ is the starting point.

$$\begin{aligned}
N_i &= N - \sum_{j\neq i} N_j \overset{1}{\geq} N - \sum_{j\neq i} N_j^* \\
&\overset{1,2}{=} \frac{U}{1-U} - \sum_{j\neq i} \frac{(\lambda_j/T)\,O_j}{1-U_j} \\
&\overset{3}{=} \frac{\lambda/\mu}{1-U_i} - \sum_{j\neq i} \frac{(\lambda_j/T)\,O_j}{1-U_i} \\
&\overset{4}{=} \frac{\lambda\,\nu/\kappa}{1-U_i} - \sum_{j\neq i} \frac{(\lambda_j/T)\,O_j}{1-U_i} \\
&\overset{5}{=} \frac{\sum_{j=1}^{n} \lambda_j\,\nu_j/\kappa}{1-U_i} - \sum_{j\neq i} \frac{(\lambda_j/T)\,O_j}{1-U_i} \\
&\overset{6}{=} \sum_{j=1}^{n} \frac{(\lambda_j/T)\,O_j}{1-U_i} - \sum_{j\neq i} \frac{(\lambda_j/T)\,O_j}{1-U_i} \\
&= \frac{(\lambda_i/T)\,O_i}{1-U_i} = N_i^*.
\end{aligned}$$

□

This section discussed how the maximum number of requests per-tenant and the corresponding response times can be calculated. However, these foundations have to be set into the context of an interactive web application. This step is discussed subsequently.

4.2.4.2. Interactive System Performance

In interactive systems, the response time R_i of t_i has and influence upon the arrival rate λ_i. This has to be reflected, to derive an estimation of the response time as a function of the weights.

The number of users of t_i is depicted by m_i. Z_i is the average think time. For each user, the processing token is either in the queue, or processed by the server, or at the user. Either it is delayed by the users think time Z_i or by the MTA's response time R_i [Menascé et al., 1994, pp. 134-136]. If the system is in steady state, the average number of requests N_i on the server side (including the queues), is described by the ratio between R_i and R_i+Z. This results in equation

$$N_i = m_i \cdot \frac{R_i}{R_i + Z_i}. \tag{4.2}$$

Accordingly, λ_i can be calculated by Equation 4.3, since the amount visits a requests makes in T, depends on the sum of the processing/waiting times spent.

$$\lambda_i = \frac{m_i \cdot T}{R_i + Z_i}. \tag{4.3}$$

With increasing R_i, more requests N_i are on the server since Z_i maintains a constant value. As a result, the probability that no request is pending for t_i on the application is low, if R_i is high. Furthermore, users in an interactive system cannot send request before they received a response. Thus λ cannot be greater than μ on the long-term. Consequently, λ_i converges to μ_i, if all R_i are high. Thus, for high tenant response times R_i, U_i tends to 1. If this is the case, there are always requests queued and thus $\lambda_i/\lambda = w_i$ (cf. Section 4.2.3).

The major concern for the isolation methods are high loaded systems (cf. Section 2.1.2.1). Due to the high load, the utilization U_j are expected to converge to 1. It is worth mentioning, that in case the utilization is imbalanced, because of low load from one tenant, an efficient mechanism shifts resources such, that the tenants with low utilization get lower weights. Consequently, their utilization converges to again fulfill Lemma 1.

Assuming that $U_j = U_k$, $\forall j,k \in \{1,\ldots,n\}$ the Lemma 1 is used as starting point

$$R_i = R_i^* = \frac{O_i}{1-U_i}.$$

This is combined with Equation (4.3), and eliminating λ_i the following equation is given:

$$
\begin{aligned}
& R_i = \frac{O_i}{1-U_i} = \frac{O_i}{1-\lambda_i/\mu_i} \overset{(4.3)}{=} \frac{O_i}{1-\frac{m_i T}{\mu_i(R_i+Z)}} \\
\iff \quad & R_i = \frac{O_i(R_i+Z)\mu_i}{(R_i+Z)\mu_i - m_i T} \\
\iff \quad & \mu_i \cdot R_i^2 + (Z\mu_i - O_i\mu_i - m_i T)\cdot R_i - O_i Z\mu_i = 0 \\
\iff \quad & R_i^2 + (Z - O_i - \frac{m_i T}{\mu_i})\cdot R_i - O_i Z = 0 \\
\iff \quad & R_i^2 + (Z - O_i - \frac{m_i \nu_i T}{w_i \kappa})\cdot R_i - O_i Z = 0 \\
\iff \quad & R_i^2 + (Z - O_i(1+\frac{m_i}{w_i}))\cdot R_i - O_i Z = 0 \\
\iff \quad & R_i = \frac{O_i}{2}\left(1+\frac{m_i}{w_i}\right) - \frac{Z}{2} \pm \sqrt{\left(\frac{O_i}{2}\left(1+\frac{m_i}{w_i}\right) - \frac{Z}{2}\right)^2 + O_i Z}.
\end{aligned}
$$

The value of the square root is always larger as the term before and negative response times are not possible. Therefore R_i is given by Equation 4.4.

$$
R_i = \frac{O_i}{2}\left(1+\frac{m_i}{w_i}\right) - \frac{Z_i}{2} + \sqrt{\left(\frac{O_i}{2}\left(1+\frac{m_i}{w_i}\right) - \frac{Z_i}{2}\right)^2 + O_i Z_i}. \tag{4.4}
$$

With the assumption of equal U_i for all tenants, R_i depends on w_i only. In situations where this is not the case, tenants with higher load utilize unused capacity from tenants with low load. Since in these situations, the response times are better as predicted, this is not a problem.

Section 4.2.6 will emphasize that a strictly convex function describing the system's response time allows a fast numerical optimization. Lemma 2 shows this for Equation (4.4). The corresponding proof can be found in the Appendix C.

Lemma 2

Consider the response time of t_i, given in Equation (4.4) as a function that maps t_i's weight $w_i \in [0,1]$ to its response time $R_i(w_i)$. Then $R_i(w_i)$ is strictly monotonically decreasing and strictly convex for $w_i \in [0,1]$. Moreover, $\lim_{w_i \to 0} R_j(w_i) = \infty$ holds.

4.2.5. Determining Required Parameters

This section shows, how the parameters required for Equatation (4.4) can be gathered.

In case of sequential request processing, S_i correlates with ν_i and thus κ directly corresponds with T. In case of parallel request processing κ is defined as 1. Using a constant κ, ν_i has to be adapted. κ and ν_i are defined from an external perspective, since the whole application is seen as black box.

A linear correlation between ν_i and S_i is assumed. Thus, ν_i is obtained by

$$\nu_i = \frac{S_i}{\sum_{j=1}^{n} \lambda_j \cdot S_j} = \frac{S_i}{\lambda \cdot S}.$$

Using this approach, ν_i is only valid for steady state arrival rates. If the arrival rate changes, ν_i may change the value. Therefore ν_i has to be continuously updated. However, due to the closed system character, the arrival rate is limited by the throughput of the MTA. Therefore, the arrival rate is expected to behave rather steady for high load scenarios.

The parameters μ_i, O_i and U_i, and the according overall parameters, can be obtained by their definitions once ν_i is known. This means, that MTAs can be characterized using ν_i, or O_i.

To characterize the whole interactive system, the think time has to be determined. If the number of users is known, R_i and λ_i are used to determine the think time Z_i. It can be calculated via:

$$Z_i = \frac{m_i T - \lambda_i R_i}{\lambda_i}.$$

If the number of users is unknown, a predetermined value for Z_i can be applied to calculate m_i. This is achieved by:

$$m_i = \frac{\lambda_i (R_i + Z_i)}{T}.$$

4.2.6. Fitness Function and Optimization

A fitness function describes how close a solution is to a given goal, by using a single figure of merit. Such functions are commonly used for automated optimizations like evolutionary algorithms (e.g., [Nelson et al., 2009]) or in the context of autonomic systems (e.g., [Walsh et al., 2004]). This section introduces a fitness function covering the **Design Goal 1** and **Design Goal 2**. This subsection has already been published [Krebs et al., 2014c].

4.2.6.1. Overview of Requirements and Goals

The proposed fitness function $f : \mathbf{w} \rightarrow \mathbb{R}$ expresses the degree of performance isolation, based on the weight vector $\mathbf{w}$, and the system function R_i. R_i uses w_i as modifiable input for the optimization. Furthermore, f includes a measure to increase the efficiency in sharing, if a tenant does not require all of its resources.

Section 4.2.4.2 describes parameters required to compute R_i. These parameters can be seen as constant within one optimization cycle. If these parameters change, a new optimization has to be triggered anyhow, since it reflects a change in the context.

The goal is to find a weight vector $\mathbf{w}$, for which the sum of the achieved fitness for all tenants is minimal. To ensure a fast and globally optimal result, R_i has to be strictly convex for w_i in $[0,1]$ and unbounded for $w \rightarrow 0$. Whereby the sum of all $w_i = 1$.

The fitness function is independent of the system function. Even a simulation could be used as a basis. Moreover, convexity is optional, but not using it would result in higher resource demands for the optimization.

4.2.6.2. Fitness Function

The proposed fitness function must contain at least two terms multiplied with each other as depicted in Equatation (4.5). $v_i : [0,1] \rightarrow \mathbb{R}_+$ describes the response time violation based on a tenants weight. This term should increase, if the response time for a tenant exceeds a guarantee. Since this term has to reflect the system function, it requires the parameter w_i. However, if a tenant exceeds its quota, the weight should decrease, because a violation of its response times should not be taken into account as much as if it is within the quota. Therefore, the penalty with regard to the weights should increase rather abruptly, when the quota is exceeded. This is covered by the term $p_i : \mathbb{R}_+ \rightarrow \mathbb{R}_+$. In the following, the details onto the two terms are considered.

$$f_i(l_i, w_i) := p_i(l_i) \cdot v_i(w_i) \tag{4.5}$$

g_i is the guaranteed response time for a tenant t_i and q_i the quota for the same tenant. The number of users defines the quota in the given situation. Due to the closed system, this also refers to the arrival rate and throughput. The penalty term does change while the weights are adapted. Thus, it is seen as constant for one optimization cycle.

First, the violation function v is defined. It expresses the degree to which a tenant's guarantee g_i is violated. Thus, it is high if $R_i(w_i) > g_i$ and close to zero if $R(w_i) \leq g_i$. Equatation 4.6 presents the chosen function.

$$v_i(w_i) := \exp\left(c_v \frac{R_i(w_i) - g_i}{g_i}\right) \tag{4.6}$$

The violation is measured relative to the guarantee. It is relatively small and below 1 if the guarantee is not violated. Due to the exponentially increasing value, the improvement of the performance, compared to tenants without violation, is very strong.

The parameter c_v is a pre-configured parameter to adjust the impact. Higher values of c_v result in a higher relevance of the response times. If c_v is tenant specific (c_v^i), one can prefer one tenant compared to those with lower values.

The penalty term p_i is investigated in the next paragraphs. The usage of the quota for a tenant t_i is given by l_i. Although the quota is defined by the number of users, it is worth to emphasize again, that it is not necessarily coupled to those. It should reduce the impact of the disruptive tenants onto the optimization, although their response times are too high. This is called penalty, which should converge to 1 if the tenant is within its quota q_i. This ensures that the guarantee term counts fully for the optimization. Vice versa, it should converge to 0 if the tenant exceeds q_i to reduce the violation terms influence. This criteria lead to the following penalty function.

$$p_i(l_i) := (1 + \exp\left(c_p \frac{l_i - q_i}{q_i}\right))^{-1} \tag{4.7}$$

The flipped sigmoid function from Equatation 4.7 weakens $v_i(w_i)$, if t_i's quota is exceeded. The value range is within $(0,1)$. It converges to 0 for $l_i \to \infty$ and to 1 for $l_i \to -\infty$. The value is exactly $1/2$ if $l_i = q_i$.

A rather abrupt transition from high to low is the result of high values for c_p. Accordingly, lower values stretch it. Thus c_p defines the slope from 1 to 0 and thus adjusts the strictness of the quota. Lower values support over-commitment, while ensuring the guarantees, as a violation from a non-disruptive tenant has a higher weight. Similar to c_v^i the parameter can be made tenant specific.

For the given approach, it is observable, that the optimization prefers tenants with few users compared to tenants with a huge number of users. This becomes visible if every parameter, except the workload/number of users is equal. This is due to the higher influence, of the same amount of changed weight, upon tenants with less users. The additional factor shown below includes this heaviness h_i of a tenant.

$$h_i := \frac{l_i \cdot n}{\sum_{j=1}^{n} l_j}$$

It sets the load of one tenant in relation to the average load of the others ($(\sum_{j=1}^{n} l_j)/n$).

The penalty and heaviness term can be seen as constant within one optimization. The terms are referred to as p_i and h_i without any arguments. Based on these insights, Equation 4.5 is redefined by Equation 4.8.

$$f_i(w_i) := h_i \cdot p_i \cdot v_i(w_i) \tag{4.8}$$

4.2.6.3. Optimization

The method's goal is to optimize the overall weights of all tenants. Therefore the function f summarizes the results for all tenants, to achieve a global function.

$$f(\mathbf{w}) := \sum_{j=1}^{n} f_j(w_j), \tag{4.9}$$

Though, the optimization goal is to find a $\mathbf{w}$ for which f becomes minimal. It is worth mentioning again, that the optimization is subjected to consider the definition of w_i. That results in the conditions: $\sum_{j=1}^{n} w_j = 1$ and $w_j \geq 0, \quad \forall j \in \{1,\ldots,n\}$

f_i ensures that tenants with high response times have fast increasing values, and thus they receive higher weights to compensate this. In cases where this leads to violations for tenants, the optimization finds a weight compensating both as good as possible. If one of these tenants exceeds its quota, it significantly receives a reduction of the violation term by p_i converging to 0.

As aforementioned, it bears advantages if f_i is strictly convex, as this allows to use numerical methods, which guarantee to compute fast the global optimum. For each convex function $\varphi : \mathbb{R} \to \mathbb{R}$ the composition $\exp \circ \varphi$ is convex. For one optimization of Equation (4.8) the terms h_i, p_i are constant positive factors. Thus, it has to be shown that $v_i(w_i)$ is strictly convex. Due to the exponential character, and the constant factor c_v, it remains to show that $(R_i(w_i) - g_i)/g_i$ is strictly convex. As Lemma 2 shows that $R_i(w_i)$ is convex in $[0,1]$, which is within the relevant boundaries, it is obvious

that the whole function is convex. Consequently, f_i is convex, since the sum of n convex functions is convex.

Another point to emphasize is, that the global minimum is within the boundaries defined by the plane of weights given by condition $\sum_{j=1}^{n} w_j = 1$. In case a point on the boundary, or corner of the plane is selected, at least one element of $\mathbf{w}$ is 0. This results in $\lim_{w_j \to 0} f_i(w_j) = \infty$ and consequently in an infinite value for f. However, for each point within the boundaries of the plane, each element of $\mathbf{w} > 0$. Thus, each result of $f_i < \infty$. Together with the proved convexity it is shown that the global minimum lies within the boundaries of the plane defined by $\sum_{j=1}^{n} w_j = 1$.

Convex optimization is broadly discussed in literature (e.g., [Boyd and Vandenberghe, 2004]) and several solutions are available.

4.2.7. Concluding Remarks

In the following, the approach is critically discussed, followed by the summary which recaps the main features of the proposed solution.

4.2.7.1. Critical Discussion

The presented method relies on a strictly convex system function for an interactive workload. However, the fitness functions used can be applied to any system function. Even if other system functions for e.g., open workload models would not be strictly convex, a numerical optimizer is likely to find good results. This means, by creating an appropriate system function other systems can be optimized. In fact, the proposed system function is a potential starting point for open workload systems, too. Another point concerns the think time of users. In case they significantly differ between tenants, it has to be reflected by an additional term similar to the heaviness. This was not presented, but does not violate the generality of the approach. Defining appropriate values for c_v and c_p significantly influences the behavior of the method and finding good values might be challenging. However, once the relevant system parameters are obtained, one can use the systems prediction function off-line to determine good settings.

4.2.7.2. Summary

This section introduced a general approach to ensure performance isolation utilizing a model to predict the performance of an MTA. The approach provides a weighted fair scheduling, to ensure a tenants guarantee within a defined period of time. This PFQ works as request admission control. The weights are adjusted by an optimization that relies on a performance prediction. The prediction can either be analytically or simulative. In the preceding sections an analytical approach was used to increase the speed of the adaptation. Thus an analytical prediction of the tenants' response times in an interactive system is possible. This model is continuously updated by the current runtime context. For the optimization of the PFQ weights, a fitness function was defined. It reflects the definition of performance isolation and the development goals for the isolation methods. The performance

prediction and fitness function can be efficiently solved by an optimization, since the resulting function of both is convex.

4.3. Resource Isolation

The amount of resources that can be used by an application influences the response time and throughput. Consequently, the control and isolation of tenant specific resource shares is an essential step to guarantee a certain tenant specific application performance. Given that sharing in the context of multi-tenancy is done at the application layer, resource usage control is a challenge which was already referred to as layer discrepancy (**Challenge 1**). Fluctuating resource requirements of tenants, be it because of variable load or changing request mixes, customized configuration or increasing amount of persistent data, further increase the complexity. These challenges were already discussed in detail in Section 1.3.3. The approach discussed subsequently ensures the isolation of resources and thus performance.

It is worth mentioning again, that the mapping of an application level SLA to a certain low level resource metric is a research question in itself [Emeakaroha et al., 2010]. Thus, this approach is independent of any application level guarantee or quota and focuses on the resource guarantees and quotas only.

This section presents three complementary contributions. First, a general approach how resources for single tenants can be controlled. Second, an evaluation of the feasibility of three existing resource demand estimation techniques is done. Third, a request based admission control to enforce resource quotas for tenants is presented. The section also includes a short discussion of enhancements, based on the basic approach.

This section has already been published to large extent [Krebs et al., 2014d].

4.3.1. General Approach

This section discusses some alternative design decisions and then introduces the proposed solution. One can differentiate between solutions where the resources used by a tenant are measured directly by a fine-grained monitoring solution, or if the demands are estimated by an RDE method. The first approach allows determining the used resources with high evidence, while it is not expected to be technically feasible. Although approaches measuring e.g., the amount of resources used by one thread between two points in time (e.g., ThreadMXBean [Oracle, 2015]) are available, and the tenant information is often mapped to the thread context, it becomes insufficient in case the request processing spans over several threads or processes. Furthermore, such an approach cannot map the overhead of the request processing, like the operating system overhead, to a particular tenant. Some sophisticated solutions (e.g., Dynatrace [Dynatrace, 2015]) allow a fine-grained monitoring of requests over several processes, or even system borders. This comes along with high technical overheads and still these methods do not provide insights into the resource demands of requests.

This yields the usage of RDE techniques. Although their accuracy is less, they can include resources used within different, technically independent threads and all layers between the application and the operating system. Assuming that the estimation provides a feasible accuracy, the resource usage is determined using RDE methods.

To enforce the isolation two solutions are possible. The controller of the admission control can derive the utilization of a resource on a tenant basis. The result can be compared with a resource guarantee for the tenants and thus weights for a WFQ like admission control can be derived. However, this approach has two significant disadvantages. First, the estimation of the demands and utilization requires some time. Thus, if the tenants' request rates or mixture of request types changes, an adaption will not occur before enough data is collected for a new accurate estimation. Second, the weight of the admission control must be adjusted, based on the observed utilization. Thus, an additional mapping is required. This constitutes another approach.

Figure 4.4 depicts the proposed solution. The MTA serves requests of different types and tenants which vary in their resource demands. The throughput $X_{t,c}$ and the service time $S_{t,c}$ for tenants t and request types $c \in C$ is monitored. It is worth mentioning, that $X_{t,c}$ and $S_{t,c}$ are measured after the actual admission control.

The application server continuously forwards samples of its utilization U_i of resource i to the *Resource Demand Estimation* component. Existing RDE approaches are applied to estimate the demand per request type and tenant $D_{t,c,i}$. The *Resource Isolation Scheduler* uses $D_{t,c,i}$ together with the number of accepted requests, to track the consumption of a resource, for each tenant, within a defined time interval. This is used to decide how the queued, or new incoming requests should be treated.

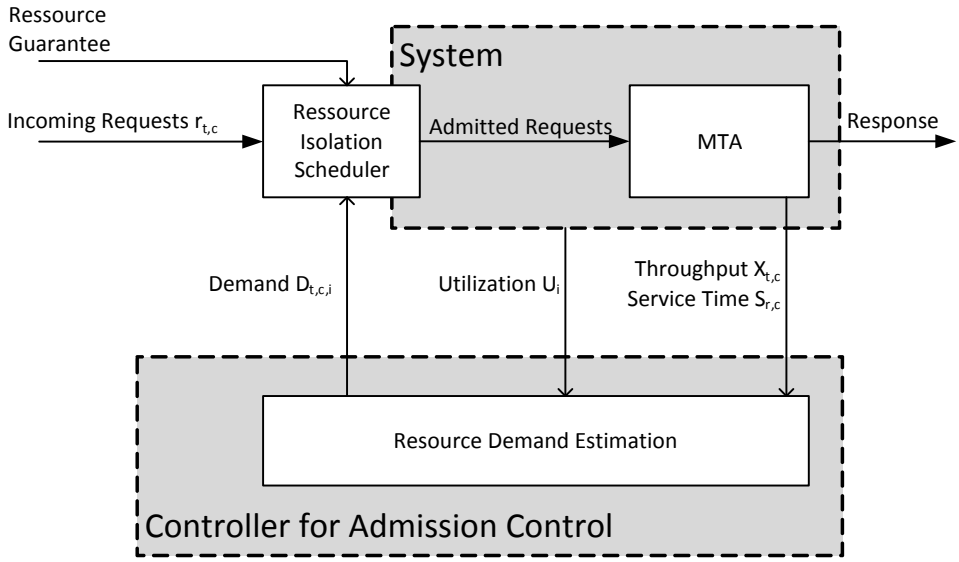


Figure 4.4.: Approach for resource isolation.

All requests admitted to the MTA are used to determine the tenants' utilization. To compute the utilization, the average over a given period of time has to be conducted. This value has to be updated very often, in short time frames of a few seconds. The estimation interval determines how often the

estimated resource demands are updated and sent to the Resource Isolation Scheduler (RIS). This can be configured depending on the workload characteristics, but it is also limited by the sampling intervals supported by the monitoring tools. The observation window is a sliding window based on the observed data, which limits the data used for resource demand estimation. It is typically in the range of a couple of minutes to a few hours. A tradeoff between the estimation accuracy and the adaptation speed for the estimation needs to be found for the observation window.

This generic approach closes the gap between the various layers by (1) connecting the monitoring information from lower layers using RDE techniques to gain knowledge about the requests impact and (2) by controlling the request flow to limit the resources used by a tenant. In the present solution, the current consumption is dynamically calculated at runtime, based on the admitted requests, within the accounting period, in which the guarantee should be assured. Though, it can react to load changes and variable compositions of request types at runtime. Changes in the resource demands, for a particular tenant, are registered and lead to a modified behavior of the admission control.

The approach connects the different layers by using monitoring information of both layers and reasoning about their relationship by using RDE techniques. The drawback of the slow RDE approaches is minimized, by dynamically calculating the consumption. In the following, this approach can react to load changes, and variable composition of request types in real time. Changes in the resource demand are updated continuously. Nevertheless, a fast estimation method which can work with a low amount of data is still recommended.

Although this approach is presented, using a concrete selection of observation parameters, orienting on the method used later, other RDE techniques rely on other information. However, at least the throughput per-tenant/request type is monitored by almost all RDEs and most also rely on the service time. Even in case of a direct measurement of resource demands, the general pattern of the approach remains.

4.3.2. Resource Isolation Scheduler

This section presents the design criteria for a concrete admission control and its controller developed with respect to the general approach from Section 4.3.1. The component ensures that the resource usage of tenants is compliant with the guaranteed resource share. Its input consists of the estimated resource demands, the resource guarantee/share per-tenant and incoming requests. The estimates are expected to be based on the request type per-tenant. The algorithm presented in the following focuses on active time sharing resources like CPUs.

4.3.2.1. General Requirements and Design Decisions

The proposed admission control maintains one FIFO request queue for each tenant. The goal is to achieve a resource consumption for each tenant, that corresponds to the preconfigured values and an efficient resource usage if one tenant does not require all resources. Efficiency in this context means, a resource should not idle, if pending requests exist and unused portions of the resource from one tenant should be available for other tenants to enable over-commitment on the resource

level. The priority on how to select requests from a set of queues can be statically or dynamically. In the proposed solution, the priorities are continuously updated, whenever the scheduler forwards a request. This allows to continuously adjust the prioritized queue, based on the actual consumption in a given accounting period. In the present case, the priority reflects the ratio of guaranteed vs. already consumed resources for a tenant.

4.3.2.2. Resource Isolation Scheduler for Multi-tenancy

This section introduces the algorithm for the scheduling of the requests. Due to the dependency of the request selected for processing, to its determined resource demand, the algorithm becomes more complex compared to Algorithm 1. Therefore, it is presented in a different granularity. Some details, which were avoided in Algorithm 1 for the sake of clarity were added in this case, since they make Algorithm 2 looking more intuitively.

Scheduling Algorithm Algorithm 2 uses predefined functions. The *dequeue* function removes and returns the first element from a FIFO queue. The *demand* function returns the resource demand of a request, based on its request type and tenant. This function uses an internal storage, where the corresponding data is available. The function is not discussed in detail, at this place.

Further input parameters are the resource guaranties per-tenant G. A guarantee $g_i | g_i \in G \wedge 0 \leq g_i \leq 1 \wedge \sum_{g \in G} g = 1$ expresses the fraction of time a tenant is allowed to use the resource under investigation. The intrinsic parameters of the admission control component used for the scheduling algorithm are the configured duration of the accounting period $t_{accountingPeriod}$, for which the used resources have to converge to the given guarantee, and a reference to the queues Q, with pending requests for each tenant. The queues are continuously updated.

The algorithm internal set P contains the triples $(priority, queue, tenant)$, used to find the next request. A lower value for the priority corresponds to a higher probability to be selected. The variable t specifies timestamps, identified by the given index and u_i represents the already used amount of a resource for a tenant i. The number of tenants is identified by n.

In line 1 the algorithm checks whether a new time interval has begun. If this is the case, $t_{lastInterval}$ is reset in line 2 to have a trigger when the next interval will finish. P is reset in line 3 and initialized with priority 0 in lines 4-6. In lines 7-10, the entry $p \in P$ is searched, which has the lowest value for the priority and is stored in the temporary variable p_{low} defined in line 10. In line 11, the current selection of p is removed from the set of all P, to update it later with a modified priority. Line 12 gets the next request from the selected queue, which is returned later as a result of the execution of this algorithm. In line 13, the consumed resources for this accounting period are re-computed based on the estimated demand for the request. The updated utilization u_t for this tenant is used to update the priority in line 14. The priority is the used resources, normalized by the guarantee. In a concrete implementation, the set P can be replaced by a priority sorted set to improve the performance.

Algorithm 2: The Resource Isolation Scheduler.

Input: $Q = \{q_1, \cdots, q_n\}$: References to request queues.
$G = \{g_1, \cdots, g_n\}$: Set of resource guarantees.
$t_{accountingPeriod}$ time interval to reset the utilization counter.
Output: Request to be processed.
Data: $t_{lastInterval} = 0$: To reset utilization counter.
$u_1 \cdots u_{|Q|}$: Used resources per-tenant.
$P = \{(p_1, q_1, t_1), \cdots (p_{|Q|}, q_{|Q|}, t_{|Q|})\}$: Priority per-tenant.

```
1  if currentTimeStamp() − t_lastInterval > t_accountingPeriod then
2      t_lastInterval ← currentTimeStamp()
3      P ← ∅
4      for i = 1 to |Q| do
5          u_i ← 0
6          P ← P ∪ {(0, q_i, t)}
7  p_low ← (∞, null, null)
8  foreach p ∈ P do
9      if p[1] ≤ p_low[1] ∧ |p[2]| ≠ ∅ then
10         p_low = p
11 P ← P \ {p_low}
12 r_return ← dequeue(p_low[2])
13 u_{p_low[3]} ← u_{p_low[3]} + demand(r_return)
14 P ← P ∪ {(u_i / (g_{p_low[3]}), q_i, p[3])}
15 return r_return
```

Properties of the Scheduler The design goals resource isolation/control and efficiency is discussed in more detail in this paragraph. The control is performed by delaying a tenant's request processing, when he already used a larger proportion of his quota as other tenants with pending requests. If all tenants have always pending requests within an accounting period, the requests are selected with the same probability during the complete period. Whereby the probability is based on their share and average demand. If a tenant does not have pending requests at the beginning of the period, other tenants are handled instead to maintain the resource utilized. However, compared to the already active tenants, a tenant that starts sending requests later, is preferred due to its priority. This again ensures the isolation aspect of the approach. In cases where a tenant already exceeded its guarantee, it is still selected, if no other tenant has pending requests. In case several tenants are exceeding their quota, the priority function selects the tenant, that exceeds its quota the least. Assuming a situation where all tenants with pending requests exceeded their quota, the tenants with higher weights are selected with higher probability, as their priority increases not as fast as for those with lower priorities. Thus the behavior is similar to Algorithm 1, but with varying sizes for the counter, which measures the number of admitted requests.

The Algorithm 2 has the worst case time complexity of $\mathcal{O}(n)$ to find the next tenant to be served. Although the priorities can be maintained in a sorted list it is not below $\mathcal{O}(n)$, since in the worst case, each queue has to be checked for containing requests. Let C be the set of all request classes of an

application, the demand function can be implemented in $\mathcal{O}(log(|C|))$. This results in $\mathcal{O}(log(|C|) \cdot n)$ for the complete processing.

The proposed scheduler guarantees a dedicated share for each tenant. It ensures a high utilization by using unused processing time of one tenant to handle requests from the others. Thus, it allows isolating resources while supporting over-commitment. Furthermore, a limited thread pool size avoids overloaded situations at the application and the impact of workload fluctuations are damped as the requests are buffered in the queues. This leads to steady monitoring data used by the RDE.

4.3.3. Resource Demand Estimation

The proposed method to isolate resources relies on the availability of accurate and timely estimates of resource demands. In the context of the proposed isolation method the resource demand specifies the processing time for a single request at a dedicated resource.

Each request belongs to a certain request type, according to a criterion depending on the application. Examples are the HTTP request's destination URL, a particular function invocation of a web service, specific database transactions or batch jobs [Kounev et al., 2011]. In case it is possible to identify request classes containing several request types with similar characteristics, the subsequent estimation methods and the previously defined RIS may work on the basis of those.

In case of MTAs, each request can be associated with a certain tenant. Given that the resource demands for a request type may vary significantly between two tenants (e.g., due to different database sizes), the estimation needs to distinguish between different tenants to identify the correct resource demands of a request type.

Assuming an application with a set of request types C, and serving a set of tenants T, it is necessary to estimate $|C| \cdot |T|$ resource demands. Previous work, such as [Rolia and Vetland, 1995, Kraft et al., 2009, Spinner, 2011], only considers the estimation in scenarios with a low number of resource demands. The maximum of 16 different demands was evaluated in [Spinner, 2011]. Therefore, the existing RDE methods have to be evaluated with regard to their behavior for a vast number of request types.

Given that the resource demands may dynamically change at system runtime, there is a need to iteratively repeat the estimation for updated values. Therefore, a fast converging resource demand estimation method is best suited for the current scenario. Further, it should come with a minimal influence upon the application performance. For this purpose, three methods which have already been successfully used in [Pacifici et al., 2008, Brosig et al., 2011, Wang et al., 2012] for the dynamic estimation of resource demands at runtime will be applied. These are linear regression, Service Demand Law (SDL) and Kalman filter. It is worth mentioning that [Wang et al., 2012] applied the Kalman filter for a multi-tenant system.

In the following section, the selected methods for RDE are introduced.

4.3.3.1. Methods for Resource Demand Estimation

In the following the three selected methods for the estimation of resource demands are discussed. *Linear regression* based approaches are used for resource demand estimation in [Rolia and Vetland, 1995, Pacifici et al., 2008, Casale et al., 2008, Zhang et al., 2007]. Two often used methods are the Linear Least Squares (LSQ), minimizing the sum of squared residual errors, and the Least Absolute Deviations (LAD). The latter minimizes the absolute value. The model for the regression is based on the Utilization Law [Denning and Buzen, 1978]:

$$U_i = \sum_{c=1}^{|C|} D_c \lambda_{c,i} + U_0$$

Let D_c be the resource demand of each request type $c \in C$ and let U_0 be the intercept capturing any processing that cannot be attributed to requests. In each monitoring interval i, the aggregated CPU utilization U_i and the average arrival rates of all request types $\lambda_1 \ldots \lambda_{|C|}$ are observed. By using an Non Negative Least Square (NNLS) regression, the coefficients $D_1 \ldots D_{|C|}$, representing the resource demands are obtained.

The second approach is based on the SDL [Jain, 1991, Denning and Buzen, 1978]. It allows computing the resource demand D_c as a function using the throughput λ_c and the utilization U_c: $D_c = \frac{U_c}{\lambda_c}$. The utilization for a particular request type U_c is the portion of the total utilization, that can be accounted to the processing of requests of type c. Monitoring tools commonly provide the total utilization U. Therefore, a measure to partition the overall utilization between the request types is required. Based on the assumption of an approximately proportional response time compared to its resource demand, a partition scheme based on weighted response times was proposed by Brosig et al. [Brosig et al., 2009]:

$$U_c = U \cdot \frac{R_c \cdot \lambda_c}{\sum_{d=1}^{|C|} R_d \cdot \lambda_d} \tag{4.10}$$

Where C is the set of all request types, R_c is again the response time of request type c and λ_c its arrival rate.

A Kalman filter [Kalman, 1960] is a stochastic filtering technique to estimate hidden states of a dynamic system, based on a series of often noisy measurements. For the given scenario, the resource demands of a system are the hidden state. A Kalman filter recursively updates its internal state used to estimate the resulting hidden parameters. This state is updated as soon as new observations become available based on the error between prediction and observation of the visible parameters. Different Kalman filter designs have been proposed for resource demand estimation [Zheng et al., 2008, Kumar et al., 2009, Wang et al., 2012]. The designs are differ in the information they rely on. In the following the Kalman filter referring to Wang et al. [Wang et al., 2012] is utilized. This filter was already applied in a multi-tenant scenario. Its internal model uses the Utilization Law [Jain, 1991].

4.3.3.2. Validation of the Methods

To identify the most suitable approach, a brief validation of the methods function was executed. The goal of this evaluation was to assess the accuracy and the convergence behavior of the considered methods for resource demand estimation in cases where a high number of resource demands need to be determined. Thus the questions answered are: Which method provides the fastest estimation? Which method provides the most accurate result? Consequently, the metrics are the time an approach needs to converge to a stable estimation and the error compared to the real demands. Additionally, the computational time was investigated.

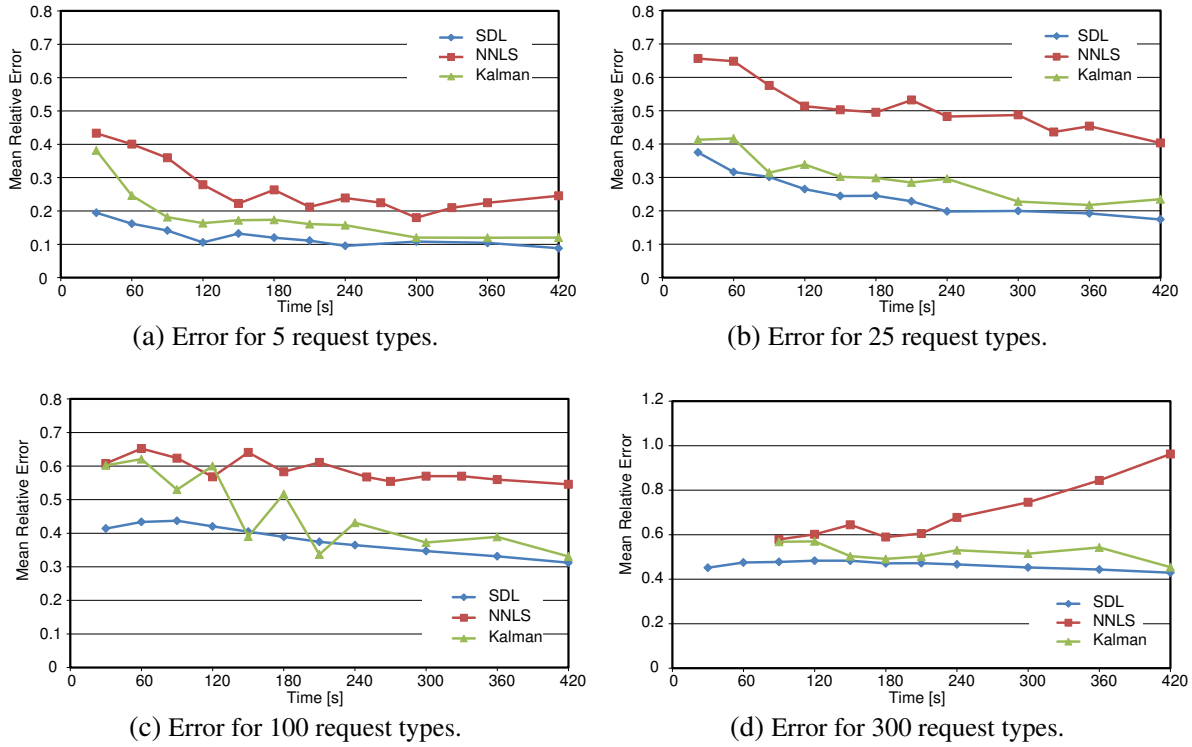
The implementations of linear regression, SDL and Kalman filter methods are based on the descriptions in [Pacifici et al., 2008, Brosig et al., 2009, Wang et al., 2012]. The observation traces were collected by using a simulation representing the real system. This allows to have well defined demands, which can be used for calculating the error of estimation.

Experiment Design The simulation that has already been successfully used in the experiment of Section 3.4.1, was used for the following experiments again. For the experiments here, only one queue with a FIFO scheduling strategy representing an application server was used. The resource demands were described by randomly generated Gaussian distributions with a mean value between 10ms-120ms and up to 10% standard deviation.

The experiments were simulated with 5 tenants and 5, 25 and 100 request types. In low load scenarios, the competition of the tenants for the resource is low. Consequently the interference is low and there is not much need to actively control the request flow. Thus, for low load scenarios the accuracy of the estimator is less important. Thus, the workload was adjusted to result in an utilization around 95%.

For SDL, the resource demands were estimated every 30 simulated seconds, using all observations from the beginning of the steady state period. In case of NNLS and Kalman filter, the length of the required samples was adjusted for each experiment, to find an optimal value. In addition, different initializations of the Kalman filter were used and the best results are presented.

Results Figure 4.5 illustrates the mean relative errors for 5, 25, 100 and 300 request types with the best fitting sample lengths and initialization parameters. In Figure 4.5a, all methods continuously increase their accuracy. The behavior for 25 types is similar. In case of 100 request types, the NNLS approach increases its accuracy slowly. Further experiments with 300 request types show that NNLS is not able to create a valid regression within a relevant observation time. Significant efforts were taken to manually find scenario specific settings for the sample length, actualization rate and initialization values for the NNLS and Kalman filter. In general, the estimates are less accurate for a high number of request types and need more time to converge. It is remarkable, that SDL always provides results better or as good as the Kalman filter and both outperform NNLS. Moreover, SDL completed the actual computation in less than $0.3ms$ for 100 request types, while



(a) Error for 5 request types.

(b) Error for 25 request types.

(c) Error for 100 request types.

(d) Error for 300 request types.

Figure 4.5.: Comparison of RDE methods.

the Kalman filter required 2*ms* and the NNLS 17*ms*. NNLS showed super linearly increasing times for the computation.

In summary, NNLS has significant weaknesses for huge type counts, Kalman filters go along with a high configuration effort and less precise results and SDL is able to provide good results in scenarios with a high number of request types.

Based on these results, the questions for this evaluation can be answered. The results let suggest that SDL converges fastest and provides the highest accuracy. SDL is identified as the most appropriate approach. It converges fast, has a low error, is of low technical complexity and has low resource requirements.

4.3.4. Towards Multiple Resources and Efficient Resource Usage

The overall approach presented so far concerns only one resource. However, assume a system with an application server and a database server. If tenants have low load, both servers are underutilized and consequently no serious performance isolation problem is expected. If the load increases, only one of both servers will result in a bottleneck, since the other will not receive enough requests, or is waiting for responses. To ensure performance isolation, it is only necessary to isolate the bottleneck resources. Checking which of the observed resources has a high utilization, can be easily realized by the Admission Control Controller (ACC). The RIS does not even have to know which resource it is isolating. The ACC simply forwards the resource demands for the bottleneck resource. The same holds not only for separated hardware nodes, but also for resources shared within one node.

If knowledge about different resources is available, it could also be an option to use this knowledge, to increase the overall utilization of resource for economic reasons. If, e.g. the database is the bottleneck and a tenant already exceeded its quota, requests that have a very low impact upon the databases resources, can still be accepted. Even though they may have a high at the application server's resources.

4.3.5. Concluding Remarks

This section provides a brief overview of critical points and their solution followed by a summary.

4.3.5.1. Critical Discussion

The SDL approach [Brosig et al., 2009] shows good results in the simulation, where only one single resource is modeled. Due to the mapping of the response times, the method lacks in effectiveness, if the request utilizes several resources. This situation leads to waiting times in resources not directly under investigation. This is negligible if only one resource has a high utilization. In scenarios with several high utilized resources, the throughput and response time for each resource has to be monitored to apply the SDL based method. An alternative RDE approach might be an option.

The solution discussed so far focused on only one resource. However, as shown in Section 4.3.4 the enhancement to support several resources is easily possible, if one clear bottleneck resource exists. If one clear bottleneck resource is not visible, due to an overall high utilization, another scheduler computing the priority based on more than one resource is required.

Furthermore, the method provided is decoupled from any guarantees at the application service level. However, for the sake of performance isolation, this is not necessary.

Finding a suitable configuration for the window length used to compute the demands might be difficult, as this is a trade-off between accuracy and fast adaptation to changes. An automated validation of the prediction results could be implemented to automatically adjust the window length.

4.3.5.2. Summary

The proposed approach allows to solve the layer discrepancy problem by providing resource information at the application layer. It is based on a resource demand estimation method to quantify the resource consumption of a tenant at runtime. This allows to control the number of requests admitted to the application for each tenant, to ensure a certain share of the bottleneck resource. Three different promising resource demand estimation methods based on Kalman filter, linear regression and the SDL were evaluated. SDL was selected and combined with the RIS.

The request admission controls RIS dynamically computes a tenant's priority. While the consumed resources of a tenant converge to a predefined share, the tenant's priority is reduced. Thus, other tenants are preferred and the resources used by a tenant are isolated.

4.4. Further Performance Isolation Methods

In this section, further isolation methods are presented. Although, they may be new in the context of multi-tenant systems, in some implementation details, or their concrete usage scenario, they are based on known techniques and therefore being discussed separately. These methods are used for comparative measurements in Chapter 6. In the following, a performance isolation method based on a Proportional Integral (PI) control loop is presented. This is followed by a simple quota enforcing mechanism and a short excerpt on a round robin based scheduling.

4.4.1. Proportional Integral Control Loop

Closed control loops were applied in the field of informatics to control computer systems [Abdelzaher et al., 2008, Zhu et al., 2009]. Lu et al. [Lu et al., 2006] is one concrete example, where a closed control loop was used to control the performance of a web server and Lin et al. [Lin et al., 2009] used them in multi-tenant systems. Thus, it is worth to discuss performance isolation methods using this concept.

The previous methods also used information observed at runtime. However, this was not a feedback in the conventional sense, since they used information providing insight about the systems characters. Whereas a feedback in this section, is related to the output of the system that should be controlled.

Parts of this section have already been published [Krebs and Mehta, 2013].

4.4.1.1. General Approach

In general, the closed control loop based methods monitor the relevant metrics for the guarantee and quota. In the concrete example, this is the throughput and the average response time. Based on this information, the controller computes the weights for a WFQ. In contrast to the model based approach, it does continuously adjust the real weights to converge to an optimum state. The model is so to say replaced by the real system. As a basis for this, an error is calculated by comparing the actual measured average response time and throughput with the guarantee and the quota. Figure 4.6 illustrates this approach.

4.4.1.2. Details on Scheduler and Controller

The scheduler uses a PFQ as described by Algorithm 1 or another WFQ approach. Similar to [Lin et al., 2009] a PI controller (cf. Section 2.1.3) is presented, which enables performance isolation in MTA.

However, compared to [Lin et al., 2009] a regulation of the overall system performance is not necessary, since a fixed thread pool size is assumed. Another difference is the control of the request admission, instead of the thread priority.

Subsequently, the controller function is presented. This function computes all new weights, based on a selected subset of tenants T_c. At first, the present approach decides between two situations.

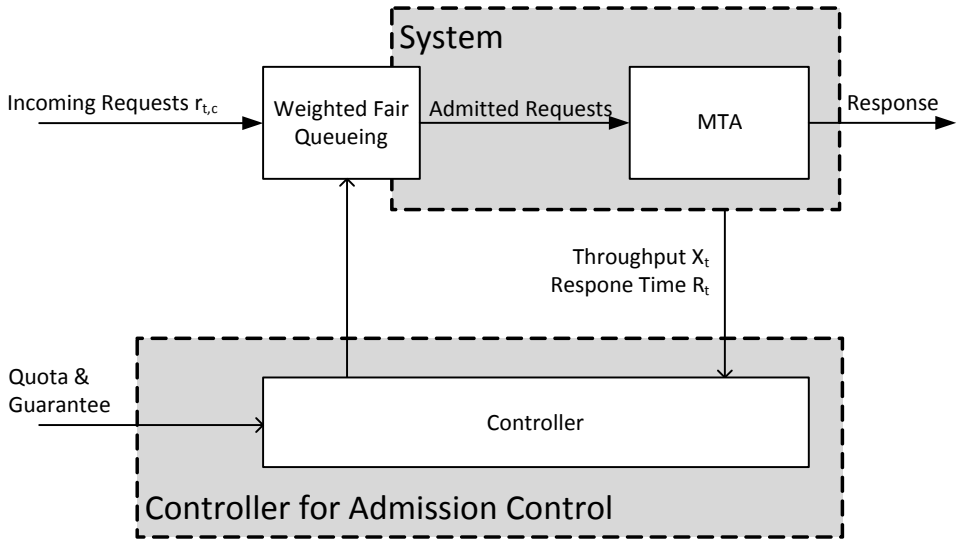


Figure 4.6.: Approach for resource isolation.

1. *Guarantee Violation:* The state in case of existing guarantee violations. This constitutes that some tenants have response times exceeding their guarantees, although they are within their quota. This yields increasing weights for these tenants. Therefore T_c contains all tenants exceeding their guarantees and being within their quota.

2. *Volunteer Increase:* The state in case tenants exceed their guarantee, while exceeding their quota and no other tenant has guarantee violations, while being below the quota. Therefore T_c contains all tenants exceeding their guarantee.

Let T be the set of all tenants, including those in T_c. Then a PI controller modifies the weights of the tenants $t_i \in T_c$ independently in the first step. The proportional part of the PI controller uses the error signal from the last period and the integral part use the average error value from the last n intervals between scheduler updates. So far, the approach follows the description of a usual PI controller. After that, the weights of all tenants in T are normalized to ensure that the sum of all $w_i = 1$.

4.4.2. Black List and Round Robin

The black list and round robin approach were already introduced in Section 3.4.1.2. For the sake of completeness, both approaches are briefly discussed in this section, in the context of the request based admission control from Section 4.1.2.

The black list, as it was presented previously, monitors the quota related information only (i.e., arrival rate/throughput). If a tenant exceeds this value for a defined duration it is black listed. This information is forwarded to the Admission Control (AC). For a black list approach, various methods for the AC are possible. Two are introduced in the following.

1. The AC maintains two queues. One for the black listed tenants, and one for all the others. Both queues are FIFO based and do not differentiate tenants internally. An arriving request is

put into the corresponding queue. As long as the queue for the white listed tenants contains requests, the black listed queue is not served.

2. The AC maintains only one FIFO queue for the white listed tenants. All incoming requests from black listed tenants are rejected. In case of an interactively closed system, the fast response will lead to higher arrival rates, as the response time does not add any further delay. Therefore, it can be beneficial to set the threshold for putting a tenant onto the black list higher than the quota, for situations where the overall load is low. This avoids the tenant from being black listed, in situations where its quota would not be exceeded, if the response times would have been higher.

Thus, a black list approach can enforce quota constraints. This has the benefit, that the size of the server thread pool does not necessarily have to be limited.

A round robin based method does not require an ACC nor the collection of any system metrics. Each tenant has its own queue, which is served continuously in a row. If one queue is empty it is skipped. This is the same as shown in Section 3.4.1.2.

4.5. Proof of Concepts

A detailed comparison of presented approaches for performance isolation is done in Chapter 6, which also discusses the achievement of the thesis' goals. This section shows that the two novel methods work as designed. Furthermore, this section provides method specific insights, which are not of general interest for all isolation methods, and therefore not included in Chapter 6. These insights help to understand how the methods work. To achieve this, both methods were experimentally analyzed.

4.5.1. Shared Aspects of the System Under Test

The following validations share some aspects concerning the deployment of the SUT. For the evaluation and implementation of the isolation method, the PIF described in Section 5.3.2 was applied. Figure 4.7 illustrates the three experiment environments used. The numbers in the lower left corners depict the artifacts for a particular deployment. The *Load Driver* emulated the users, that sent requests to the online store application (MTTPC-W). It was separated from the SUT to avoid influence. The *MTTPC-W* (cf. Section 3.3) was hosted on the LJS, which was optionally hosted within a VM. A *MySQL* version 5.1 database was used to persist the application's data and was either deployed in the same container as the *LJS*, or on a separate *Database Host*. The *VM* ran SLES11 SP2 and was hosted on the Xen hypervisor version 4.1. It hosted the LJS, the application and the database. The *LJS* executed the MTTPC-W and the admission control.

The *Admission Control* was implemented as a valve [Craig McClanahan, 2015]. A *Resource Monitor* was installed to collect the CPU utilization of the CPU pinned to the VM. The monitors delivered the required data for the RDE.

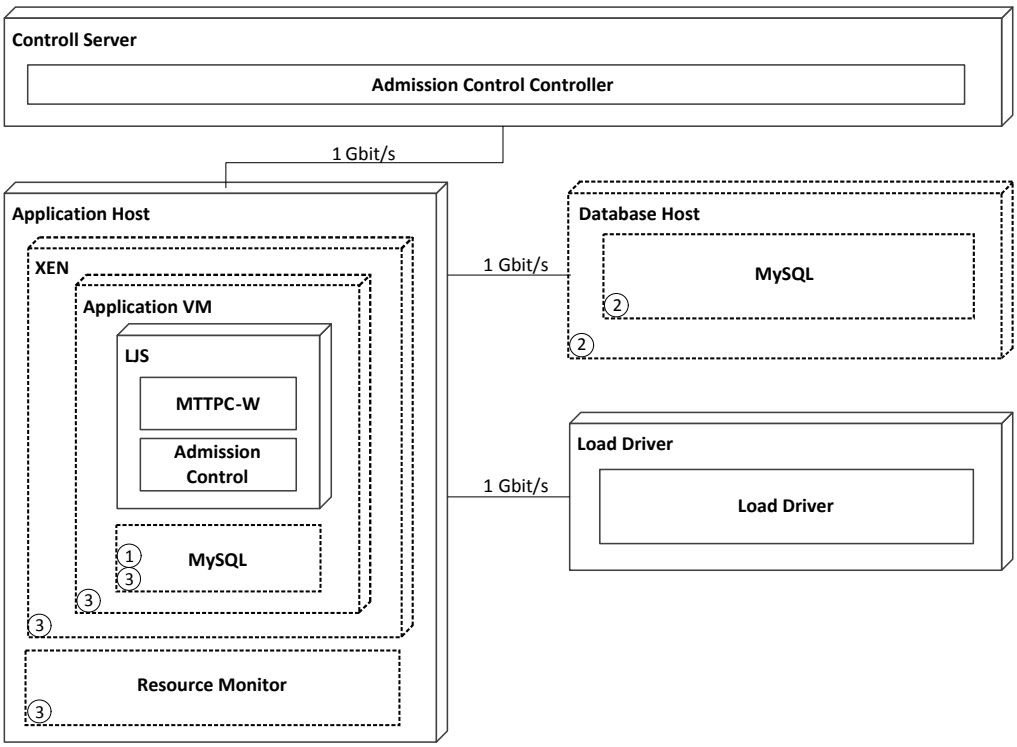


Figure 4.7.: Deployment for proof of concept.

4.5.2. Resource Isolation

This section focus on the evaluation of the resource isolation. This section has already been published to a large extent in [Krebs et al., 2014d]. The goals for the validation of the resource isolation based approach are:

1. Analysis of the SDL concerning its accuracy in a realistic scenario.
2. Analysis of the resource isolation capabilities of the approach presented.

The section starts with details on the system environment, followed by two separate sections discussing more detailed questions and outcomes for the evaluation goals. The focus hereby is on isolating the CPU.

4.5.2.1. System Environment and Configuration

Deployment 3 was chosen for the following experiment. Due to the use of virtualization, an additional layer was introduced, which makes the estimation of resource demands even more difficult. The application host, and the load driver host were equipped with four cores providing 16 x 2.13 GHz and 16 GiB memory each. They were connected with a 1 Gbit/s Ethernet LAN. The virtual machine in the application host was pinned to one CPU and 2 GiB memory. The RDE ran on a standard desktop PC.

4.5.2.2. Accuracy of the Resource Demand Estimation

In the following paragraphs, the results about the accuracy of the RDE proposed in Section 4.3.3 are discussed.

Experimental Setup The scenario consisted of eight tenants, resulting in 112 request types. The database content was randomly generated, whereby tenant 1 to tenant 6 used an equally sized small dataset and tenant 7 and tenant 8 an equally sized dataset with eight times more data. The average response time and the throughput for each request type and tenant was collected together with the overall CPU utilization. The information was aggregated periodically and sent to the RDE every ten seconds. The experiment ran 17 min. The first 3 min were used to ramp-up the emulated browsers, another 3 min were spent to warm up the system, which showed steady results from this point. In this scenario, each tenant served 250 users in order to achieve a high CPU utilization of around 95%.

Results This section answers the question how accurate the chosen approach to estimate the resource demands is. Table 4.2 depicts the demand estimates for tenant t_1, including the 95% confidence interval. In general, the observed confidence intervals reflect a high precision. The average resource demand per request for the tenants were: $t_1 = 3.2ms, t_2 = 3.3, t_3 = 3.5, t_4 = 3.5, t_5 = 3.1, t_6 = 3.4, t_7 = 5.1, t_8 = 4.6$. The obtained results show considerably higher demands for tenant 7 and tenant 8, which were caused by the larger dataset.

Requests type	Demand (ms)	Requests type	Demand (ms)
Home interaction	2.2 [2.1|2.3]	Execute	4.0 [3.9|4.1]
Admin request	3.6 [2.5|4.6]	New products	4.3 [4.1|4.6]
Admin response	7.2 [5.7|8.7]	Order display	2.6 [2.3|2.8]
Best sellers	5.8 [5.5|6.0]	Product detail	2.0 [1.9|2.1]
Buy confirm	6.5 [6.0|7.0]	Search request	1.9 [1.8|2.0]
Buy request	3.7 [3.5|4.0]	Shopping cart	3.7 [3.4|4.0]
Customer registration	2.0 [1.2|2.7]	Order inquiry	0.03[0.03|0.03]

Table 4.2.: Resource demand for MTTPC-W request types.

A cross-validation to calculate the prediction error at the same load as the training set, fits well to the requirements, because the estimation is done online at the same workload where it is applied. Therefore, samples were collected at a constant load of around 95% utilization for the training and validation. The samples were randomly partitioned into a coherent training and a validation set. The training set was used for the resource demand estimation and the remaining one to calculate the prediction error. The same process was repeated to get different training and validation sets. The observed mean error was 9%.

To get an error estimate for scenarios, where the workload suddenly increases from a low to a higher utilization, a scenario with different utilization was considered. An estimation of the requests demand at 60% utilization was chosen. The results were used for a prediction at 60% utilization which result in an error of 19% and an error of 28% for 90% utilization. According to [Menasce and Virgilio, 2000], the error rates are acceptable values. Furthermore, the used RDE underestimated the demands for all tenants. Thus the RIS still maintains a similar ratio between tenants and consequently a lower error for the resource isolation can be expected.

In Summary, the proposed RDE using the SDL provides sufficient estimations for scenarios with a high load and a high number of request types.

4.5.2.3. Isolation Capabilities

In this section, the results concerning the isolation capabilities of the resource isolation mechanism are presented. The questions to be answered are listed in the following.

Q1 Is the approach able to achieve performance isolation with regard to response times and throughput by applying resource control?

Q2 What is the estimated isolation quality?

Q3 Does the approach efficiently distribute resources to allow over-commitment?

Q4 Does the approach reflect the demands of a tenant in the request processing?

Q5 Is it possible to provide different resource guarantees to the tenants?

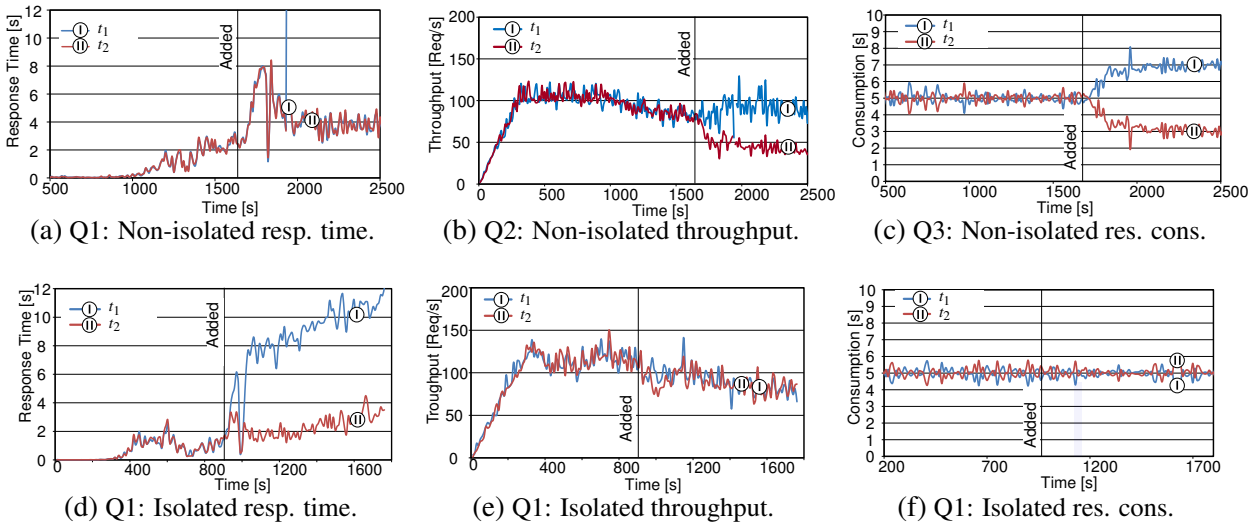
Q6 Can the chosen mechanism dynamically react to changing resource demands?

To answer these questions, 5 scenarios were conducted. In all of them the RIS accounting period was set to one second. This means, after one second the tenants resource usage counter was reset. The resource demand information was updated every 15s, the observation window for the estimation was set to 300s and the resource utilizations were updated every 10s.

Scenario and Results for Q1, Q2 To answer **Q1** and **Q2** a comparison between an isolated and a non-isolated system is required. In order to answer **Q1**, the observed response time and throughput of each tenant is needed. For **Q2** the resource consumption for each tenant has to be compared. The first experiment used a standard FIFO access mechanism without a tenant aware admission control. The experiments started with 1000 users for each tenant. After a warm-up phase, 1000 additional users were added to tenant 1.

In the second experiment, each tenant was allowed to allocate 50% of the resource, which had to be enforced by the presented isolation method. 1250 users per-tenant were used as reference workload, and the disruptive workload had 1000 users more for tenant 1.

In both situations, the database did not use indices, resulting in continuously increasing demands.



(a) Q1: Non-isolated resp. time. (b) Q2: Non-isolated throughput. (c) Q3: Non-isolated res. cons.

(d) Q1: Isolated resp. time. (e) Q1: Isolated throughput. (f) Q1: Isolated res. cons.

Figure 4.8.: Isolation capabilities of the resource isolation.

Results Q1 At the beginning of the non-isolated case (cf. Figure 4.8a) both tenants observed the same performance, because they had the same load. Once 1000 users were added at 1600s the load for tenant 1 increased. Although tenant 2 maintained a constant load its response time increased exactly as the response times for tenant 1, since the FIFO queue does not differentiate tenants. Figure 4.8b shows the complementary throughput and how it decreases for tenant 2 in response to the disruptive behavior of the other tenant. In general, a slightly increasing response time and decreasing throughput for both tenants is visible. This can be explained with an increasing database volume during the experiment.

The isolated scenario (cf. Figure 4.8d and 4.8e) started for both tenants with response times at the same level. After adding 1000 users to tenant 1 at 900s, the response time of tenant 1 started to increase. Despite the increasing database size, the performance was clearly different between the two tenants.

The mechanism provides steady response times for the tenant with steady load, while another increases its load. Thus, it enforces performance isolation.

Results Q2 When tenant 1 increases its load in a FIFO scheduled system, its resource consumption also increases (cf. Figure 4.8c) and influences tenant 2, since the consumption of it decreases. In contrast, the consumption of tenant 1 maintains the guaranteed level in the isolated case (cf. Figure 4.8f) and the consumption of tenant 2 is not influenced. In conclusion, the method has the potential to ensure an almost perfect resource isolation between individual tenants.

Scenario and Results for Q3 To answer **Q3**, the guaranteed resources were uniformly distributed between the tenants, all tenants began with 1500 users. After 500s, 1000 users were removed from tenant 1. Therefore, the request rate of tenant 1 was expected to be too low to allocate all its guaranteed resources. As a consequence, the second tenant should have been able to use these resources, in addition to its own share.

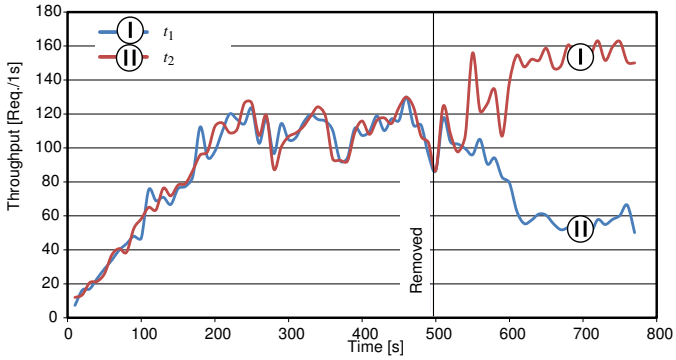
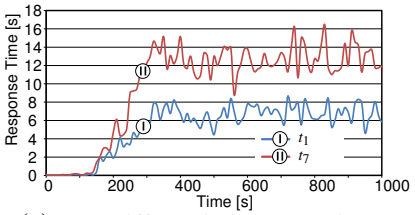
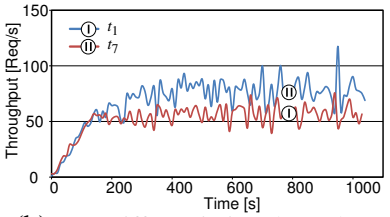


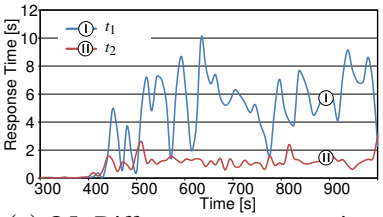
Figure 4.9.: Q3: Efficiency of the system.



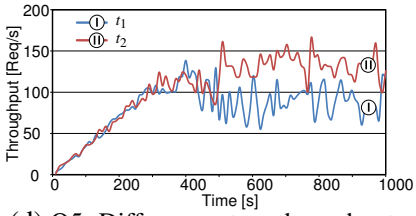
(a) Q4: Differentiation resp. time.



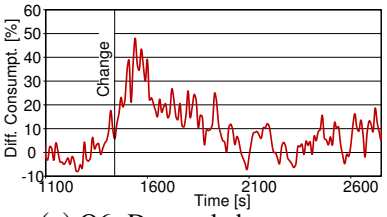
(b) Q4: Differentiation throughput.



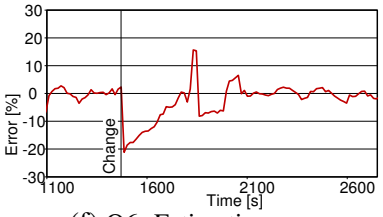
(c) Q5: Diff. guarantees resp. time.



(d) Q5: Diff. guarantees throughput.



(e) Q6: Demand change cons.



(f) Q6: Estimation error.

Figure 4.10.: Evaluation results of the resource isolation.

In the first 210s, shown in Figure 4.9, the ramp-up time of the measurement is visible. Until 500s both tenants had 1500 users and the same response time. At 500s the workload of tenant 1 was decreased as seen in the figure. As a consequence, the amount of available resources increased and tenant 1 took advantage of them. Although the resource guarantees were assigned equally to both tenants, tenant 1 consumed more resources in this case. Both tenants observed a reduced response time. Thus, the proposed mechanism ensures an efficient resource usage and allows overcommitment.

In order to answer **Q4**, tenant 1 was configured to access a small database, and tenant 7 to access a larger database. This result in different resource demands (cf. Section 4.3.3) and allows for insights about the possible negative impact, the tenant with higher resource demands has onto the other. For this experiment, the resource share was configured equally and both tenants ran 1250 users.

Figure 4.10a shows that tenant 1 with low demands received a better service quality. In case of a non-isolated system, the throughput would be the same for both and thus the ratio of the share would be 0.39 for tenant 1 and 0.61 for tenant 7, which is an error of 22% (cf. Section 4.3.3). In the given experiment, the throughput ratio was 0.58 for tenant 1 and 0.42 for tenant 7 (cf. Figure

4.10b) which resulted (cf. Section 4.3.3) in a resource share of approximately 0.48 and 0.52. This is an error of 4%.

This shows the ability of the method to treat tenants with different demands in a differentiated manner, and proves the ability of the mechanism to use the demand estimates.

Scenario and Results for Q5 To answer **Q5**, different resource shares were allocated to the tenants. Tenant 1 had 25% of the resource and tenant 2 had 75%. The number of users for both tenants was fixed to 1250.

The tenant with 75% of the resources had better response times (cf. Figure 4.3.3c) and a higher throughput (cf. Figure 4.3.3d). Thus, the proposed resource controller is able to differentiate QoS by providing different shares to each tenant.

Scenario and Results for Q6 To identify the system's ability to adapt to changing resource demands, two tenants using the system at high utilization were configured. After 1450s the resource demands for each request type of tenant 1 were increased to a higher value, by adding artificially computational overhead, simulating a change in the tenant-specific configuration. In a second experiment, the system run at a utilization of around 30% and then the load of one tenant was increased to achieve a utilization close to 100%.

Figure 4.3.3f depicts the error of the resource demand estimated in percent. At 1450s the demand was 20% underestimated and due to the sliding window approach it took some minutes to converge again to a good estimation. In total, it needed around 8 min before the estimated demands became stable again. A short overshoot is observable in the figure. A smaller observation window can speedup the adaptation at the cost of the prediction accuracy. Figure 4.3.3e depicts the difference of the resource consumption between the two tenants in percent. Once the demand increased, the difference raised up to 50%, before the estimation and system became stable at around 2100s. The experiment was repeated several times, with similar characteristics. The second experiment presented a very good resource isolation, similar to Figure 4.3.3f with a negligible error. This shows that the RDE error in various load scenarios has only little impact on the isolation and control mechanism.

Scalability Experiments with up to 8 tenants were executed. In an exemplary manner, the results from an experiment based on the scenario used to answer **Q4** and **Q5** are discussed. In this scenario, tenant 5 and tenant 6 had a larger dataset compared to the other tenants. The CPU shares of the tenants were set to (t1..t6=$\frac{2}{30}$;t7..t8=$\frac{9}{30}$). It was observed, that with a higher number of tenants, a longer observation window is beneficial. Therefore, it was set to 3s and the observation window to 7 min. The observed response times showed that the system is capable to isolate the resource usage between the tenants and to provide different QoS levels for different tenants ($t_1 = 5.0s; t_2 = 4.3s; t_3 = 4.2s; t_4 = 5.0s; t_5 = 7.1s; t_6 = 7.5s; t_7 = 0.8s; t_8 = 0.5s$).

4.5.2.4. Experiment Results and Discussion

The evaluation based on MTTPC-W showed the ability of the mechanism to isolate the CPU. In addition, over-committed systems and QoS differentiation can be handled. Thus, it successfully demonstrated:

1. The general approach how resource isolation can be achieved at the application level.
2. The SDL can be used to provide an accurate resource demand estimation, as sustained by the analysis of three resource demand estimation methods.
3. A concrete admission control implementation enforcing isolated resources.

4.5.3. Model Based Isolation

This section focuses on the evaluation of the model based isolation method, to gather insights concerning its behavior. The questions discussed are:

Q1 Do c_v and c_p influence the weights and the corresponding performance?

Q2 Does the approach provide measures to achieve performance isolation?

Q3 Does the approach efficiently distribute the weights?

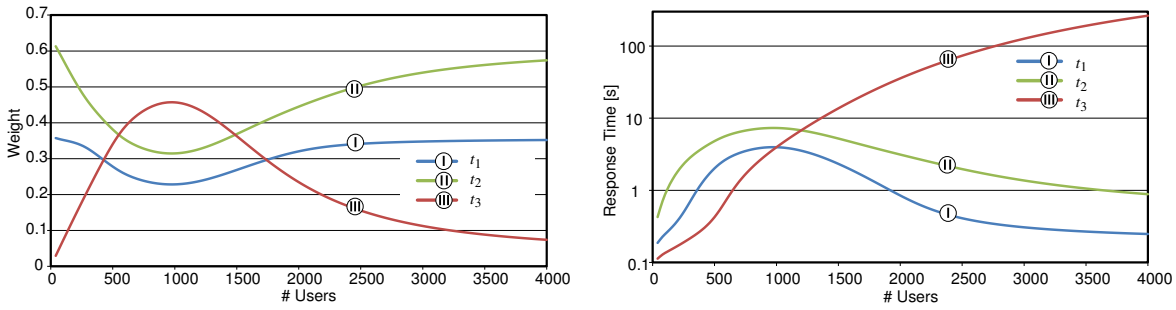
Q4 How accurate is the prediction of the system model?

The evaluation is split into two scenarios. One based on a simulation, where no live monitoring data is used to refine the model. Thus the performance predictions are based on artificially defined system characteristics. The section starts with the simulation based evaluation, primarily focusing on the investigations of the parameters influence. It is followed by a separate section using MTTTPC-W to investigate the prediction accuracy and the real isolation capabilities.

4.5.3.1. Simulation Based Experiments

This section determines, how changes in the number of users m_i, influences the weights and consequently performance. It further shows how c_v and c_p influence the observed performance. That helps to find a suitable configuration. The experiments in this section were not based on a real system, instead the performance prediction function was used. For the optimization and prediction of the performance, the algebra system Maple 10 [Monagan et al., 2005] was used.

Q1/Q3: Influence of Varying Workload Onto the Parameters In Figure 4.11a/b a plot of the weights and the corresponding response times for an increasing load is presented. Tenant 3 increased the workload from 40 to 4000 users. The other parameters maintained constant values at $m_1 = q_1 = 500, m_2 = 900, q_2 = 750, q_3 = 1000s, g_1, g_2, g_3 = 4s, c_v = 0.1, c_p = 5$ and the occupation time of $0.005s$



(a) Weight with increasing load.

(b) Response time with increasing load.

Figure 4.11.: Results of the model based isolation for increasing load.

The x-axis shows the increasing load for tenant 3. The y-axis of Figure 4.11a shows the weights as result of the penalty p_3 and heaviness h_3.

For an increasing load, the weight w_3 strongly increases, since the heaviness h_3 increases. This leads to a reduction of the weights for the other tenants.

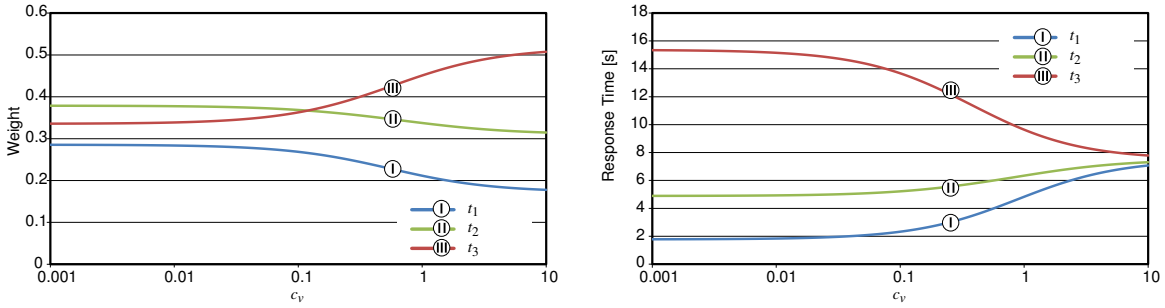
Weight w_3 increases, until the third tenant's load approaches the quota q_3. It can be seen, that the penalty p_3 becomes more important in this area. Thus, p_3 converges to 0 and consequently tenant 3 becomes more unimportant for the optimization. The more tenant 3 exceeds the quota q_3, the more severe the penalty becomes and thus the weight is reduced, although the heaviness h_3 increases linearly with the users. It implies, that the disruptive tenants positive effect of increasing heaviness looses importance in relation to the penalty p_3. Around a load equal to the guarantee, the weight continuously converges to 0, since the tenant's fitness becomes unimportant. This shows that a tenant cannot increase its weight unrestrained, by increasing its workload.

When the disruptive tenants workload is at the quota, the abiding tenants observe lower response times as before (cf. Figure 4.11b). Once a tenant exceeds its quota and becomes disruptive, the response times start to increase significantly, while the other response times are decreasing again.

Q1: Influence of Violation Factor Figure 4.12a/b depicts the weights and response times, determined by the optimization for increasing values of c_v on the x-axis. The value of c_v was varied from 0.001 to 10. All other parameters were fixed $m_1 = q_1 = 500$, $m_2 = 900$, $q_2 = 750$, $m_3 = 1500$, $q_3 = 1000$, $g_1, g_2, g_3 = 4$s, $c_p = 5$ and the occupation time was set to 0.005.

This configuration is similar to the previous example, but with changing c_v. For a low c_v, the weights of the tenants converge to values proportional to their coefficient $h_j \cdot p_j$. Contrary, for high values of c_v the term $h_j \cdot p_j$ is not significantly important.

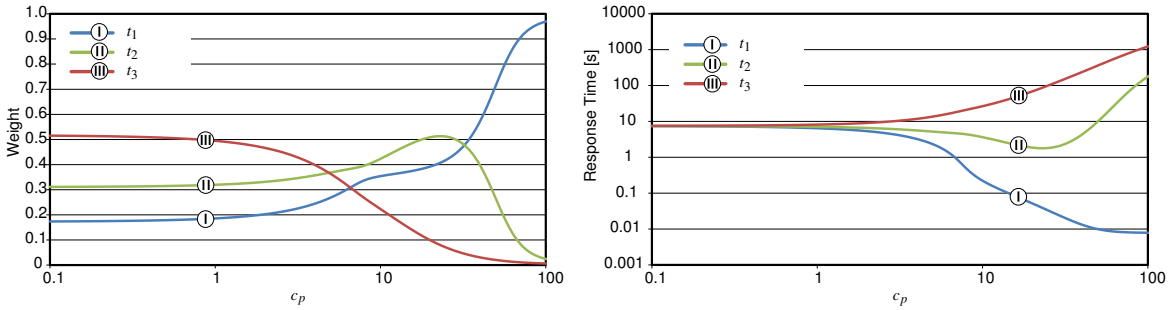
In summary, a very high or very low value for c_v is not feasible. The effects of the isolation via $h_j \cdot p_j$ and guarantee violation should be of similar impact. This ensures the mechanism to provide an efficient distribution of weights, while maintaining performance isolated. In the present scenario, values between 0.1 and 1 are suitable.



(a) Weight with increasing c_v.

(b) Response time with increasing c_v.

Figure 4.12.: Evaluation results for increasing c_v.



(a) Weight with increasing c_p.

(b) Response time with increasing c_p.

Figure 4.13.: Evaluation results for increasing c_p.

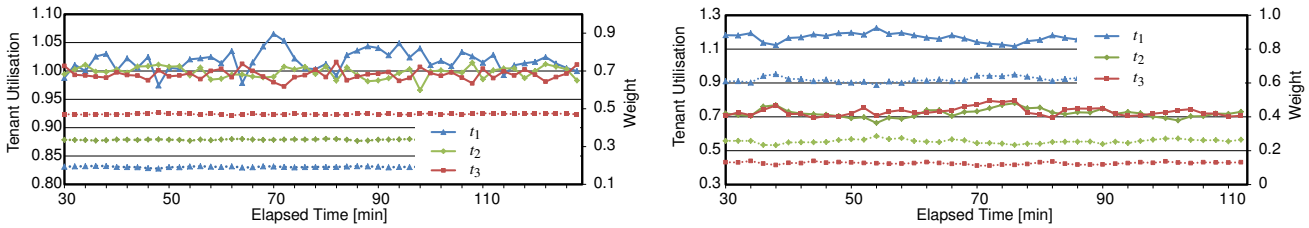
Q1: Influence of Penalty Factor Figure 4.13a/b presents the weights, and response times for an increasing c_p. c_p is varied between 0.1 and 100 while the other parameters maintained constant: $m_1 = q_1 = 500$, $m_2 = 900, q_2 = 750$, $m_3 = 1500, q_3 = 1000$, $g_1, g_2, g_3 = 4s$, $c_v = 0.1$ and the occupation time was set to 0.005

For small a c_p, the response times for all tenants converge to the same value. This is because

$$\lim_{c_p \to 0} p_j = \lim_{x \to 0} \frac{1}{1 + \exp(x)} = 1/2$$

. Thus, resulting in a penalty of approximately $1/2$ for all tenants. Consequently, no performance isolation is reflected in the optimization. If c_p is very large, the penalty function converges towards a non-continuous step function.

Thus, for large values of c_p, the disruptive tenants penalties converges to 0 and the penalty for abiding tenants to 1. This means, for very high values of c_p, the resources are divided among the tenant working within their quota only. In this situation, the disruptive tenant does not benefit from unused parts of abiding tenants quota. Consequently, moderate values for c_p are suggested. This makes tenants being penalized gradually. Values between 1 and 10 seem to be suitable for this scenario.



(a) Utilization in high load scenario.

(b) Utilization in low load scenario.

Figure 4.14.: Utilization and weights for different loads.

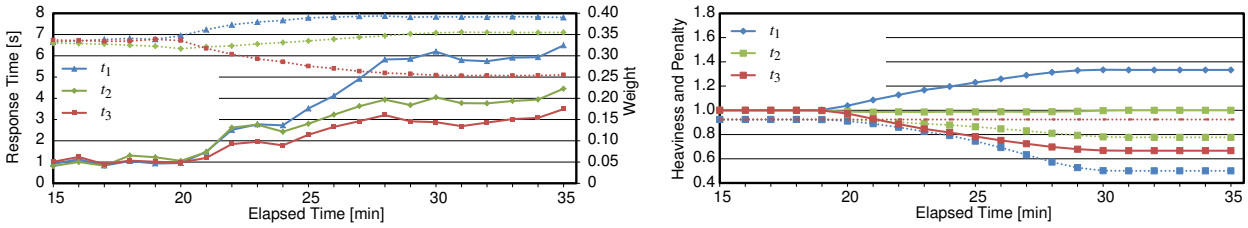
4.5.3.2. MTTPC-W Based Experiments

System Environment and Configuration For the purpose of gathering insights into the methods isolation in a real scenario, the model based isolation was implemented in a test environment. The goals were the evaluation of the prediction accuracy and the functionality of the isolation mechanism.

For the experiments, the deployment 1 and 2 from Figure 4.7 were conducted. The ACC was deployed on on 4x3.4GHz and 16GiB memory. The optimization was realized by the open source library JOptimizer [JOptimizer, 2014], which offers methods to efficiently solve convex optimization problems based on [Boyd and Vandenberghe, 2004]. The load driver ran at 2x2.4GHz and 4GiB memory. The application host at 4x3.4GHz, with totally eight hyper threads, the database host had the same hardware. Due to the nature of the isolation mechanism, it abstracts completely from the underlying technologies. To show this, the approach was also validated in a virtual environment in Chapter 6. It is worth mentioning, that the database volume varied significantly compared to the previous experiment for the model based approach.

The data of the first 15 min were discarded for each experiment, to ensure a proper warm up phase of the system. The system parameters required by the method were calculated based on moving averages of 2 min. The graphs presenting the results also plot the summarized values of two minutes. An excerpt of the findings is given in the next paragraphs.

Utilizations of the Tenants First, the operational assumption, that all tenants utilizations are approximately equal $U_i = U_j$ was validated. The first experiment in Figure 4.14a presents the observations for a setup where tenant 1 runs 200 users, tenant 2 runs 350 users and tenant 3 runs 500 users. The corresponding quota was at $q_1, q_2, q_3 = 1000$ users and thus no tenant exceeded it. In this scenario, all tenants utilizations U_i were close to 1. Therefore, the system fulfilled the preconditions for Lemma 1. In the second experiment depicted in Figure 4.14b the quotas were configured to make tenant 1 very disruptive. Tenant 1 was configured to run 400 users while the quota q_i was 300. The other tenants were configured to use low workloads of 50 users and 100 users respectively. The quota was set to $q_2, q_3 = 300$. In this configuration, the abiding tenant had unused portions of its share and these were used by tenant 3. This shows the work-conserving character. Consequently, unequal utilizations were observed, resulting in wrong predictions. It is worth mentioning again,



(a) Weight and response time.

(b) Heaviness and penalty.

Figure 4.15.: Isolation experiment with two load increasing tenants.

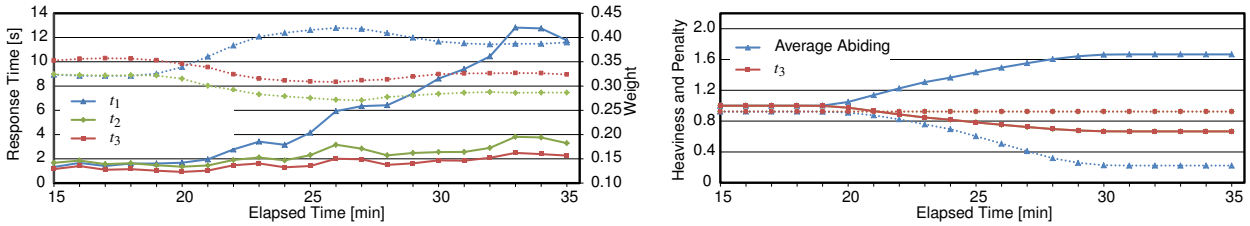
that in a real system, due to the work-conserving PFQ, this would result in a better performance of the disruptive tenant.

Q4: Prediction Accuracy For the evaluation of the prediction accuracy, the measured response time was compared with the estimated ones, provided by Equation 4.4. The mean relative error was used and three experiments were executed. In the first experiment, three tenants with workload of 50 users for each tenant was conducted. The second experiment conducted 100 users for each tenant and the last one 150 users. The quotas were kept constant at $(q_1, q_2, q_3 = 400)$. An error of 98.7% was observed in the first experiment. It is worth mentioning, that the error was equally distributed among the tenants and the estimated response times were much higher as the real ones. The second experiment showed results of around 8.5% error. In the third experiment, the prediction error was at 9.2%. The standard deviation of the error in measurement 2 and 3 was below 1%. In the last two measurements, the system was already in a state where extensive queuing arose. This illustrates the good prediction capabilities in case of the relevant, high load, scenarios.

Q2/Q3: Isolation Capabilities In Figures 4.15a/b and 4.16a/b, the performance isolation mechanism was evaluated. The computed weights w_i are shown in the a graphs with dashed lines while the measured response time is shown in solid lines. The b graphs present the penalty (dashed) and heaviness (solid) parameters.

To simulate a changing user behavior, the workload for tenant 2 and tenant 3 was increased in the first experiment. The parameters were configured as follows: $q_1, q_2, q_3 = 400$ users and $g_1, g_2, g_3 = 4$s were chosen. All tenants started with 200 users, the number of users was increased between 20 min and 30 min of the experiment for tenant 1 and tenant 2. The load was increased to 400 and 300 users respectively.

In Figure 4.16b only two lines are visible, as the two abiding tenants observed the same value. The parameters were configured as follows: $q_1, q_2, q_3 = 400$ users and different guarantees of $g_1, g_2 = 4$s, and $g_3 = 3$s. All tenants started with 200 users, after 20 min the users of tenant 1 were continually increased till 500 users in minute 30 were reached.



(a) Weight and response time.

(b) Heaviness and penalty.

Figure 4.16.: Isolation experiment with one load increasing tenant.

4.5.3.3. Summary and Further Insights

Since all isolation measures rely on the prediction accuracy of the model, this is discussed more extensively. The difference between workload and assigned capacity is low, if the tenants utilization is equal. Vice versa, if tenants have unequal utilizations, the performance estimation is inaccurate. This situation occurs in the case of low load. However, tenants with utilizations below 1 do not use all of their available resources, they fulfill the guarantee, and other tenants with higher utilization profit from their unused capacity. Consequently, this situation is not expected to be critical with regard to performance isolation. Performance estimations also lack in accuracy when the server does not utilize its full capacity, because of wrong estimations for O_i, which are too low. This results in estimates significantly higher compared to the observed one. This implies an underloaded system. This means, that all tenants' fulfill their guarantee. Consequently, this is not critical, either. In case of relative high response times, the response time prediction is very accurate and precise. In the experiments, the error was consistently below 15%. Moreover, the prediction accuracy with deployment 2, using an external database, was tested. In this scenario, the bottleneck was the database server. Experiments with different load configuration were executed. In all scenarios, the accuracy of the prediction was below an error of 20%. Thus **Q4** is answered.

Finally, the presented data shows, performance isolation can be enforced, in accordance with configured guarantees and quotas (**Q2**). However, the performance isolation mechanism can tend to provide abiding tenants too much capacity if the disruptive tenant goes far beyond its quota. Thus providing abiding tenants a performance better than the guarantee (cf. Figure 4.16). This is a minor disadvantage in case of over-commitment and answers **Q3**. The experiment for c_p and c_v outline that too high and too low values have major disadvantages. Value of c_p around 5 and c_v around 0.5 are a first point of reference, since in this region the particular functions change the most (**Q1**).

4.6. Conclusion

This section presents a short recap of the chapter, which presented various performance isolation methods. First, the chapter defined the design goals for the isolation methods in accordance with the overall requirements and challenges from Section 1.3.3, followed by a detailed discussion of the general isolation method and several instances of it. Thereby, two are based on novel approaches.

Finally, the concept's validity was proved and insight about the methods internal behavior was collected.

The primary design goals of the methods were to ensure performance isolation, whereby an efficient distribution of the weights had to be maintained. To achieve this, a request based admission control was proposed. This admission control consists of a fast and low overhead implementation regulating the flow of requests admitted to the application. It is always triggered to provide the next request, when the application server is ready to process a new request. This admission control, asynchronously receives configuration updates requiring more intense computations. This allows a fast reaction to sudden and unpredictable changes in the workload, while improving the overall efficiency with the updates.

The novel methods consist of an adapted implementation of a WFQ based scheduling. To compute the weights for each tenant, a function to predict the response time for an interactive close workload model is used. Additionally, a fitness function using the prediction function and the weights of the tenants as input was used. An optimization finds suitable weights. The fitness function consists of one part to cover situations where tenants' guarantees are violated and a second part covering situations where tenants' exceed their quota.

Another novel approach is based on a resource isolating implementation of the admission control. This scheduler is aware of the resource demands of a particular request for the bottleneck resource. It takes advantage of this knowledge, to dynamically reduce the priority of tenants close to the guaranteed share, in order to prefer tenants not as close to their guaranteed resource usage. To estimate the resource demands, various estimation approaches were discussed and evaluated. The chosen one is based on [Brosig et al., 2009] and relies on the SDL.

Furthermore, an isolation method based on a black list, round robin and closed control loop were briefly outlined. Very similar methods were already discussed in the literature. Therefore, only a brief summary on how they can be applied for MTAs was given.

Both novel approaches were experimentally analyzed to figure out relevant characteristics. These experiments conducted insights into the resource demand estimation, and performance prediction function. Overall, both provide good isolation results. It was shown, that the resource isolation mechanism reflects the resource demands of tenants, resulting in visible performance differences. The influence of various configuration parameters, for the optimization of the model based approach, was investigated. Finally, both methods can provide performance isolation, whereby the resource isolation is not able to efficiently share the system with regard to application level guarantees, but on resource guarantees. A detailed discussion of the achieved goals is part of Chapter 6.

5. Decision Support and Architecture

The previous chapters presented measures to evaluate performance isolated systems and methods to achieve performance isolation in MTAs. As outlined in the introduction, this is not enough to enable an MTA provider creating performance isolated MTAs (cf. Section 1.2). A gap between the actual isolation methods described at a conceptual level, and how to apply them in a scenario specific real world context exists.

MTA providers neither know which characteristics of the methods are important, nor how the methods behave according to these characteristics, nor which method provides the best isolation. Isolation metrics were described in Section 3.1 and will be applied in Chapter 6 to get the required insights. However, other aspects can also be of relevance and have to be identified. Furthermore, an appropriate method to find a suitable isolation method for a particular application scenario needs to be identified. Moreover, there is no feasible documented approach how potential performance isolation methods can be implemented in an application, and how the application has to be built that performance isolation is possible. Especially not if it runs on a PaaS.

The detailed goals of this chapter are described in the following. Since the goals of this section aim at *enabling* an MTA provider to apply the presented isolation methods, they are referred to as *Enablement Goals*. The according challenges and overall goals were already described in the Section 1.3.3.

Enablement Goal 1 — *Architectural Concerns*
Analyze architectural decisions that have an impact upon the quality of a performance isolation method. Beforehand, relevant multi-tenancy specific architectural aspects have to be listed.

Enablement Goal 2 — *Information Requirements*
Identify the information required by different isolation methods. For broad applicability a classification schema of the isolation methods must be developed. Accordingly, a classification of the various kinds of information, required by the isolation methods, has to be defined.

Enablement Goal 3 — *Decission Support*
Identify a method to find the most feasible performance isolation method for a given scenario. This method has to respect non-functional requirements as well as functional relations, which exclude potential mechanisms from the candidates.

Enablement Goal 4 — *Reference Architecture*
Design a reference architecture that provides insights on how the isolation methods can be realized in an MTA. Relevant aspects are the required information and its applicability as PaaS enhancement.

In summary, this chapter provides additional knowledge, relevant for potential developers of MTAs. It identifies MTA specific architectural concerns, in which existing applications are different. Then, it is discusses, how they influence an application's performance isolation capabilities. Based on this, recommendations are formulated. The section also discusses existing isolation methods and their applicability in a systematic way. Therefore, a classification of isolation methods is developed, and the advantages and disadvantages of each method are discussed. A concrete quantification of the methods quality is presented in Chapter 6. To make this information consumable and applicable, a decision support process is identified and adapted for the given scenario. This helps to identify the best suited isolation method for a given MTA. As a foundation, relevant non-functional parameters for the isolation methods are listed. Finally, the chapter provides a discussion on how the existing methods are best integrated into existing systems, by using a reference architecture, whereby the previous insights lay the foundations.

5.1. Architectural Concerns for Performance Isolated Multi-Tenant Applications

In this section, the relevant architectural characteristics, outlined in Section 2.1.1.1, are discussed in more detail, with regard to their impact upon the performance isolation of MTAs, plus additional concerns not yet investigated. Based on these insights, recommendations are formulated to increase the isolation. The content of this section has already been published to a large extent in [Krebs et al., 2012a].

5.1.1. Architectural Concerns

In the following, high level concerns influencing the architecture of an MTA are discussed. First, details on the affinity, session stickiness and persistence concerns, which are usually transparent for the tenants, are given. For each architectural concern, examples from real-life applications are provided to show the relevance of these concerns as distinctive features of existing offerings and thus also emphasize the relevance of these concerns for MTA providers.

5.1.1.1. Request Distribution

Tenant Affinity Applications where hundreds or even thousands of application nodes serve several tens of thousands of tenants can be imagined [Schonfeld, 2009]. In such situations, the way users of one tenant are distributed, becomes a significant issue for the design of MTAs. Affinity refers to the binding of users, of the same tenant, to the application nodes. In contrast to traditional

request/response-based systems the method is using tenant specific attributes in order to route requests, and not user specific ones referring to sticky user sessions. In the following, the different types of tenant affinity are introduced and reasons for using them are explained. There are different functional reasons, why an MTA has the one or other affinity. Examples are a tenant specific state. Non-functional aspects can also be a reason. Examples are shared services (such as persistence) with limited capacity, an increasing cache hit rate, intolerable high synchronization overheads and legal restrictions.

In a *non-affine* application, a number of application nodes exist, and every incoming request can be handled by any of the nodes, without attention to the tenant from which the request origins.

In *server-affine* cases, requests originating from the same tenant are processed by the same application node, but one node can handle multiple tenants. A possible reason for such a behavior is a tenant context, that cannot be shared between nodes. Furthermore, an artificially introduced affinity can increase performance in some cases (e.g., by increasing the cache hit rate).

In *cluster-affine* situations, requests are served by a fixed subgroup of application nodes. One subgroup can serve several tenants. This scenario also implies that each application node is only part of exactly one subgroup. Keeping a tenant's context in sync among several instances, or a long logical network distance, can reduce the performance and thus limit the distribution.

Inter-cluster-affine is similar to the cluster affinity, but one application node is part of several groups. For the tenant specific context, the same arguments as for the cluster-affine behavior arises. Legal restrictions are other aspects, e.g., a server located in Germany is also located in the EU and thus part of two groups.

Session Stickiness Orthogonal to a tenant's affinity, requests are either statefull or stateless (cf. Section 2.1.1.1). Therefore, two situations can be distinguished.

1. In case of non-sticky sessions, requests are processed by all available application nodes if no tenant affinity exists.

2. In case of sticky sessions, a user specific state is bound to an application node. Therefore, all requests from one user have to be served by the same node. However, tenant affinity can bring in additional conditions for the initial binding of the user to nodes.

Examples The SAP Business ByDesign solution was developed with a server-affine behavior, due to a high amount of caching and large tenant specific contexts. Other applications distribute tenants over several nodes of the application. One example is SAP R/3 [Keller and Krueger, 2002, pp. 505-506]. However, R/3 has sticky users. Applications hosted on e.g., the Google Cloud Platform [Google, 2014] have usually neither a tenant-affinity nor sticky sessions.

5.1.1.2. Data Persistency

Multi-tenant database designs are widely discussed in the literature (cf. Section 2.1.1.1). In the following, a short summary of the different approaches is given. In a *dedicated database* system,

every tenant uses its own, completely separated database. A *dedicated table/schema* approach shares one database, which contains a separate table or schema for each tenant. In such a scenario, one achieves at least a partial sharing. The *shared table/schema* approach shares the same tables and schemas, a differentiation of the data is usually provided by adding a *tenantId* column.

Examples SAP's R/3 uses relational databases with a tenentId column [Keller and Krueger, 2002]. In contrast, Calvin [Calvin and Friedl, 2009] describes a solution using a separate database within Windows Azure. The academia describes several approaches extensively (e.g., [Wang et al., 2008, Chong et al., 2006]).

5.1.1.3. Customization

The ability to handle different tenant specific configurations regarding the UI, the functional and non-functional behavior and the services referenced, is a major differentiation for MTAs. In Koziolek's architecture [Koziolek, 2011] a separate metadata manager provides the customization information to adapt the application (cf. Section 2.1.1.1). Mietzner [Mietzner et al., 2009] created some patterns for multi-tenant services and service compositions. Based on these patterns, they built up a service-oriented system, allowing extensive modifications, including tenant specific developments.

These two papers exemplary differentiate the degree to which an MTA can be customized. A *configurable* application is one which provides tenant specific behavior or appearance, where this behavior is configured without tenant specific code enhancements. Thus, every tenant accesses the same code base.

An application allowing tenant specific *code extensions* provides the most powerful way to adapt it to customers' needs. This leads to significant technical challenges in the multi-tenant scenario. Mietzner's et al. [Mietzner et al., 2008] SOA based approach for MTAs provides a way, in which a tenant can replace or extend pieces of code, without influencing the others.

Examples Google Apps [Google, 2015] provides office applications for private use or companies with limited opportunities for customer specific customizing. Other SaaS providers such as Salesforce provide a wide range of options, including tenant specific code [Weissman and Bobrowski, 2009].

5.1.1.4. QoS Differentiation

QoS differentiation provides one tenant another service quality than another tenant. QoS differentiation is not directly related to isolation aspects (cf. Section 2.1.1.2). One can differ between input and output related service differentiation, which are not mutually exclusive. The input related differentiation promises the same behavior, regarding non-functional properties, for all tenants, by allowing a different quota. In the output case, the system allows the same amount of load for every tenant, but differs in the output related properties, like response time.

Examples Service differentiation in multi-tenant environments is a topic of interest in academia. Lin [Lin et al., 2009] provides an approach to differ response times within MTAs by using a closed control loop and an admission control. Thule [Ruehl, 2013] and Schroeter [Schroeter et al., 2012] can support additional non-functional tenant specific requirements using a tenant aware compositions of components.

5.1.2. Mutual Interferences, Dependencies and Recommendations

In this section it is discussed how various architectural decisions influence the performance isolation capabilities. This discussion is made on an abstract level and independent of the concrete performance isolation methods. Furthermore, recommendations with respect to performance isolation are given. These recommendations can be retrieved from the analysis of the architectural concerns. These recommendations are not mandatory. However, not following them leads to more sophisticated implementations of the performance isolation methods.

5.1.2.1. Affinity

In case a tenant's request does not stick to one server, the performance isolation method must coordinate its activities among several nodes to enforce the isolation. In the worst case, a tenant shares on different nodes the resources with different tenants. In case of an arbitrary kind of cluster affinity, where all application nodes share services like a database, it can happen that tenants from one cluster negatively influence the performance of the tenants in another cluster. In these situations, the required service must provide an isolation among tenants or at least clusters. Alternatively, a limitation of the number of requests going to one cluster must be enforced in addition to the tenant isolation related admission control.

Recommendation: From the performance isolation perspective, it is suggested that a server affinity makes performance isolation easier and should be preferred. If a cluster or server affinity exists, required services should not be shared among several clusters/nodes. If this is not the case, the required services should support isolation by themselves on cluster or tenant basis. In case of an inter-cluster affinity, serious efforts have to be taken to enforce isolation.

5.1.2.2. Session Stickiness

In case of a non-affine, cluster affine or inter-cluster affine system design, the user related session stickiness also becomes important. Load balancers are able to distribute incoming requests between nodes, maintaining an equal load and service qualities on the computing nodes. Consequently, the same configuration can be applied to all nodes, or even centrally on the load balancer. The challenge arisees, if the sessions are sticky. In this case, the load balancer cannot distribute the requests arbitrary. Therefore, different nodes may have different utilizations and service qualities. Therefore,

a performance isolation method must be able to globally optimize the weights, by providing each node an individual configuration.

Recommendation: It is recommended to develop applications where no tenant and no users state is held at dedicated application nodes. Instead, the state should be kept on the client or on a shared storage/cache. This is a general recommendation for scaling web applications [Fehling et al., 2014]. Since a shared storage or cache can also become a bottleneck and violate performance isolation, the recommendations from Section 5.1.2.5 have to be considered.

5.1.2.3. QoS Differentiation

Providing QoS differentiation can have an impact on the persistence layer, because some aspects can be achieved by taking advantage of database features, such as different storage capacities. Another point is to provide every tenant its single database, with different hardware settings. With respect to performance isolation, it is worth to notice, that the decision for QoS differentiation can influence the method used to ensure isolation. If for example, the QoS should be differentiated by a dynamically adjusted thread priority, isolation methods like those proposed will try to compensate this.

Recommendation: If performance related QoS differentiation is required, it should be tightly integrated with the performance isolation method. Most performance isolation methods allow to configure individual guarantees and quotas or shares per-tenant and thus already provide differentiation measures.

5.1.2.4. Customization

It is worth thinking about different databases for each tenant, if extensive modifications are offered. If, for different tenants, different execution paths are taken for the same request, this leads to different demands. This makes the performance isolation more challenging, since this has to be reflected by the isolation method in use. If tenants have separate database schemes, the performance isolation on the persistence layer can be enforced using state-of-the-art database techniques. If performance isolation cannot be ensured due to the unpredictable impact of a request onto the system, alternative sharing approaches might be feasible (e.g., [Ruehl, 2013]).

Recommendation: From the performance isolation perspective, it is best to not allow any kind of customization or code extensions. Since this is not always feasible, special care has to be taken for the selection of a performance isolation method, to ensure its support for different resource demands.

5.1.2.5. Persistence

The decision about the chosen persistence concept can influence the level of achievable performance isolation. Independently from the tenant's restrictions at the application level, one single query on the database might cause significant performance issues for all tenants. This impact is difficult to predict. Especially in cases where tenant specific code extensions are executed.

Recommendation: From the perspective of performance isolation, it would be best to provide each tenant a database that is not shared at all. However, this is not feasible in practice, as this would counteract the goal of an efficient resource usage of the MTA. Consequently, maximum query times should be defined in the case of tenant specific code extensions, a well sized persistence layer should be deployed, or other measures to performance isolate the persistence on the database have to be undertaken.

5.1.3. Summary

Performance isolation is influenced by many aspects. As discussed before, one has to make distributed control mechanisms available in non-server affine situations. Besides that, sandbox approaches to isolate code extensions are required. Without sandbox, a tenant can easily deploy code affecting other tenant's performance. Another issue is, the database which needs to provide adequate mechanisms to ensure performance isolation or a proper overall performance. The same holds for other required services.

5.2. Selection of Performance Isolation Method

Concrete performance isolation methods are presented in Chapter 4. Additional approaches, similar to the presented concepts in Chapter 4, can be realized. For instance, an approach using a system performance function to model an open workload (cf. Section 4.2) can be expected to share most characteristics with one using a closed workload system function. This section first introduces a classification based on the discussed solutions from Chapter 4. The methods requirements and benefits are discussed on this class level. To make the gained knowledge easier accessible, a short decision process is discussed, in which the outcomes of the discussion can be made consumable for MTA providers.

5.2.1. Classification Schema for Existing Methods

In this section introduces the classification. Parts of this section have already been published in [Krebs and Loesch, 2014].

The classes depicted in Figure 5.1 are primarily motivated by the information and knowledge required by them. Location oriented approaches place tenants onto various application nodes with regard to SLAs. Resource control mechanisms enhance the resource accessing mechanisms (e.g.,

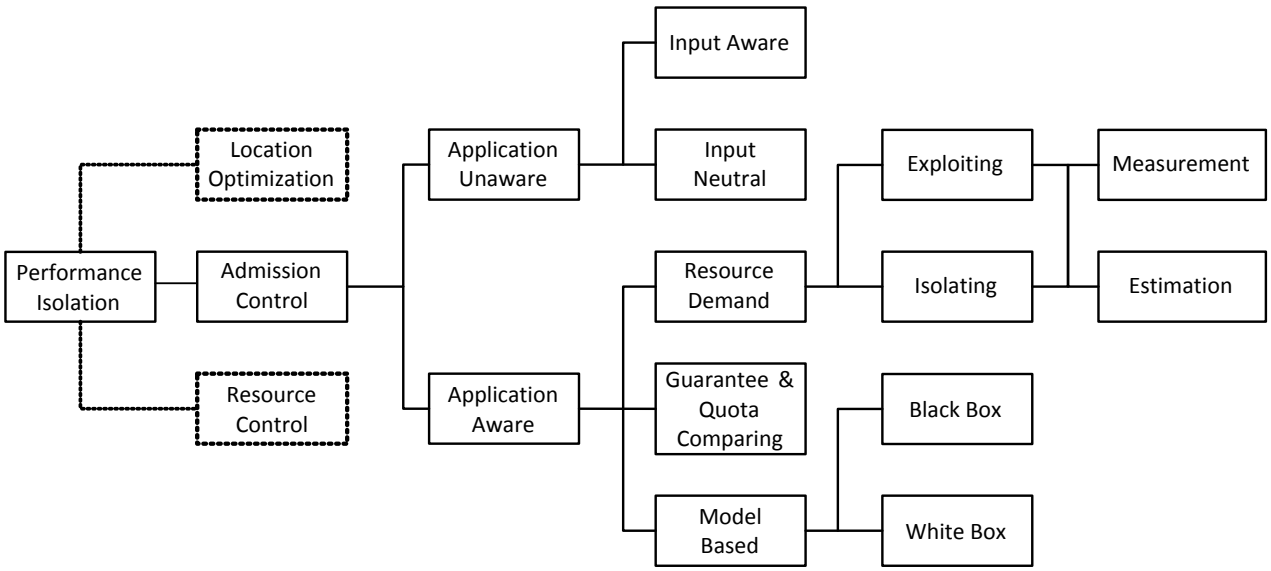


Figure 5.1.: Classification of Isolation Methods

OS Schedulers) by the tenant concept. Allocation driven and resource control methods were identified as insufficient (cf. Section 2.3 and Section 4.1.1). Since the thesis focuses on admission control based methods the other trunks are not presented in more detail.

5.2.1.1. Application Unaware Methods

The methods in this category do not use any information about the application's state, nor performance observed. However, they may rely on information about the input of a system.

Input Neutral approaches are not aware of any information. The admission control mechanism has a static behavior, and thus constant priorities per-tenant for the decision which request is allowed to pass to the MTA. They may indirectly use the guarantees or quotas to initially derive an appropriate configuration of the admission control, but no change of the method's behavior at runtime appears. Different kinds of input neutral approaches are discussed in Section 4.4.2. Furthermore, these approaches are widely applied in related fields, where a static scheduling is used (cf. Section 2.2.2). An example is a round robin based scheduling, with a separate FIFO queue for each tenant.

These approaches may provide a very good isolation with short settling times. However, since they do not have any feedback, it can easily happen that one tenant observes a bad performance, because it sends plenty of requests, although others have only a few requests pending. Thus, the method has a rather low efficiency and can hardly support over-commitment (cf. Section 3.4.1.2).

Input Aware methods make decisions using information describing the input of the system. Thus, they do not have any information from the systems processing or responses. A possible metric to influence the admission controls behavior can be the request arrival rate, or the number of users using the system. Based on this information, a request is either delayed or refused. Such approaches simply enforce a quota. In Section 4.4.2 and Section 3.4.1.2, the black list method was introduced that realizes such a behavior. It is worth mentioning, that these methods do not leverage any knowl-

edge about the internal structure, nor the observable behavior of the application. This implies, that based on the incoming information the method cannot infer the impact on other tenants.

The benefit of such approaches is their rather simple implementation and the opportunity to react dynamically to load changes. The combination of random load and a hardly predictable system behavior makes them insufficient with regard to resource usage, since they cannot use any feedback from the system.

5.2.1.2. Application Aware Methods

These methods take advantage from knowledge that is either related to the output of the system or influence of input parameters upon the system's internal state. It can be assumed, that application aware methods are implicitly input aware. Thus, they may be seen as an enhancement of these.

Resource Demand Based Approaches Approaches, such as the presented one in Section 4.3, control the resource consumption of a tenant to ensure application level performance guarantees. The methods do this by a resource aware admission control.

One subclass of approaches is *Isolating* resources. In this case, the admission control limits the requests admitted for a tenant, based on the resources it already used. Such methods enforce a certain share, based on a priority based AC. Whereby the priority is computed based on the difference of the guaranteed and used resources (cf. Section 4.3).

Another subclass *Exploits* the knowledge of resource demands a particular request type has, but does not directly ensure the isolation of a resource. In these scenarios, the knowledge is only used to decide, which request types have to be rejected for a particular tenant, in case performance problems appear. Wang et al. [Wang et al., 2012] describes such a method.

In general, the mapping of application level SLAs, such as response time guarantees to low level resource requirements is not covered, and is a research topic in itself. The root cause of a low performance isolation, is the inability to control resources. It can be expected, that these approaches outperform the others if the request demands vary strong among the tenants.

Measurement vs. Estimation For determining a tenant's resource demand, two approaches are possible: Measurements and Resource Demand Estimations.

Measurements on a fine-grained basis, initiated by the application level, may allow to directly gather the resources consumed by a tenant. This is possible, if supported by the operating system or platform used. If such monitoring functionality is available, it often comes along with a significant overhead or lacks monitoring capabilities among different processes. However, the accuracy is better.

Methods based on *Resource Demand Estimation* determine the resource demand on a tenant and/or request type basis, by using information that can be measured from outside, without special fine-grained monitoring measures. Most RDE approaches use the resource utilization, throughput and response time. Such a method was presented in Section 4.3.

Guarantee and Quota Comparing Isolation Methods The previously discussed approaches have disadvantages in over-committed scenarios, when shares have to be adapted, to fulfill a disruptive tenants guarantee in case of over-commitment on the application level guarantees.

A disruptive tenant should receive a performance fulfilling its guarantee, if other tenants have resources left. The tenants' quality can be increased by reducing the quality of tenants observing a quality better as the guarantee. Runtime information must be considered to adjust the priorities according to dynamically occurring load changes. The interesting parameters are the guarantee and the quota and their corresponding observed qualities. It is worth mentioning, that the internal structure of the application is not known by these approaches. Usually, closed control loops are considered for such scenarios. The approaches come with more complexity, but potentially more efficient usage of resources. Lin et al. [Lin et al., 2009] or Section 4.4.1 depict examples.

Model Based Approaches Model based approaches use performance models to forecast the impact of a change in the context, or the application's configuration, upon the observed performance. To create such a model, structural knowledge of the application is required. Moreover, model specific monitoring information, to calibrate the model at runtime, are needed. In case the internal application aspects are modeled, resource demand estimations are often required. In case of *Black Box* oriented models, information observable from outside the application are sufficient (cf. Section 4.2). They usually model less details of the system. Analytical solutions are often black box oriented. In Section 4.2, a concrete example was introduced, which black boxed the application. *White Box* based approaches describe more details, using internal application information. Thy usually apply simulation oriented solutions to solve situation not yet covered by analytic approaches. However, simulation based solvers require much more time (cf. Section 2.1.2.2). Thus, this class does not directly utilize the measured guarantee or performance related metrics, although it tries to optimize them.

Performance models and their usage in self aware/adaptive systems are of interest in the respective research area (e.g., [Kounev et al., 2010]). The model based approaches benefit from adapting the system without manipulating the real system, while doing the optimization. This increases the adaptation speed, reduces performance oscillations and allows to even proactively react to future incidents. The drawback is the complexity of the model and the abstraction of reality, which can lead to wrong adaptations in case of erroneous predictions.

5.2.2. Informational Requirements

The Table 5.1 depicts an overview of the information required by each isolation class. In the following, a brief explanation is given. *Input* is related to information coming from a tenant into the system. For example, the arrival rate, or the number of users. The *output* is the information related to the response to the tenants, e.g., the response time. The *extrinsic* information is related to metrics that can be measured from outside the MTA. Note that this does not include the AC, while *output* does. An example is the time a request is actively processed by the MTA. The extrinsic information

is differentiated on per-request or a per-tenant basis. *Intrinsic* information includes aspects that can only be measured by having access to the MTA and its underlying stack. Examples are resource utilizations, or fine-grained monitoring on how long a request is processed by different nodes or resources. Actually, *Resource Demands* are part of the intrinsic information. However, due to the challenges in measuring resource demands on a tenant basis (cf. Section 4.3.1), this is listed as a separate point. It can further be differentiated between a tenant based or request type based measurement. A profound *system understanding* is necessary to build a reliable performance model of the MTA. This point is not directly related to information, that have to be gathered at runtime, while all the previous are.

	Input Neutral	**Input Aware**	**Quota and Guarantee**	**Res. Isol. (RDE)**	**Res. Isol. (Measured)**	**Res. Expl. (RDE)**	**Res. Expl. (Measured)**	**Model (White Box)**	**Model (Black Box)**
Quota	-	√	√	-	-	(√)	(√)	√	√
Guarantee	-	-	√	√	√	√	√	√	√
Input	-	√	√	-	-	(√)	(√)	√	√
Output	-	-	√	-	-	(√)	(√)	(√)	(√)
Intrinsic	-	-	-	√	-	√	√	-	(√)
Extrinsic (Ten.)	-	-	-	√	-	√	-	√	(√)
Extrinsic (Req.)	-	-	-	(√)	-	√	-	(√)	(√)
Res. Demand (Ten.)	-	-	-	-	√	-	√	-	(√)
Res. Demand (Req.)	-	-	-	-	(√)	-	√	-	(√)
System Understanding	-	-	-	-	-	-	-	√	√

Table 5.1.: Overview of informational requirements for various methods.

In case of the resource isolation methods, it is possible that request are treated by the average resource demand per tenant. Therefore, the request based information is optional, although the method proposed in this thesis requires it. A resource exploiting mechanism necessarily requires request type specific information. Independently on how the demand is gathered, it also needs information about the system, in order to find the bottleneck resource. This allows to reject only the disturbing requests. Depending on the trigger to start the resource exploiting mechanism, additional information can be required. As previously mentioned, black box methods usually assume a higher degree of abstraction. It is not expected, that black box models will reflect various intrinsic information. The overall predictions of the model might be calibrated with the output information of the system. White box approaches rather use intrinsic and resource demand information. However, both models require a certain system understanding.

5.2.3. Selecting an Isolation Method

This subsection describes how an MTA provider can find the most suitable performance isolation method for a given scenario. According to [Baker et al., 2001] a general decision making process consists of 8 steps, which are outlined in the following.

1. Definition of the problem.
2. Determination of requirements the solution must meet.
3. Establishment of goals.
4. Identification of alternatives.
5. Development of evaluation criteria.
6. Selection of decision making tool.
7. Application of the tool to determine a solution.
8. Validate the solution of the tool.

This is a very general process and can be specialized for the particular scenario of performance isolation, already pre-answering recurring points.

The problem is that an MTA is not performance isolated (1). Therefore, the requirements a solution must fulfill have to be defined (2). Goals are usually related to the identification of alternatives. It defines what an alternative has to do, while the requirements define how they do it (3). Alternative solutions were already identified by the classification in Section 5.2.1 (4). The fifth step (5) defines the evaluation criteria to ensure the fulfillment of the requirements. The sixth step (6) has to identify an appropriate tool/method that helps to compare the different alternatives. This tool is applied (7) and finally it is implemented and validated if the result fulfills the requirements. In the following the focus is on the steps 1-6.

For an inexperienced stakeholder in the field of performance isolation, several questions arise. He does not know which requirements are important differentiators for the isolation methods, and does not know how to assess the various methods, with regard to these requirements and do not know, which decision making tool is appropriate for the existing problem.

Subsequently, these relevant questions are discussed to enable an MTA provider selecting an appropriate isolation method.

5.2.3.1. Solution Requirements and Goals

The general goal of the selection process is to find a method, which increases performance isolation according to the requirements from the MTA provider. The goal can be more detailed in some situations. An example is, that the solution should be able to enforce an application level guarantee, such as response time. Or the isolation method has to reflect varying resource demands.

Concerning the requirements it is convincing that the metrics identified in Chapter 3 are potential candidates. That means, the solution should provide good isolation, short settling times and a low oscillation. Moreover, the isolation methods are subject to efficiently distribute resources to support over-commitment. Consequently, this should also be reflected.

Non-Functional Properties Besides the already mentioned performance isolation related aspect, further non-functional properties are of importance for some MTA providers. Several publications list potential candidates [Sommerville, 2006, McCall, 1977, Boehm et al., 1976, Davis, 1993, Dromey, 1995, ISO/IEC, 2011b]. However, only Pors et al. [Pors et al., 2013] identified the most relevant non-functional properties of interest in the context of multi-tenancy. They identified 21 properties. Not all of these properties are relevant in the context of the performance isolation methods. Subsequently, the irrelevant properties are listed.

Irrelevant Aspects *Deployment time* is the time to get an existing and complete software running on a system. However, this strongly depends on the organization and the target system. It can further be assumed, that this is almost similar for most of the isolation methods and thus a weak property to differentiate isolation methods. *Flexibility* is the degree to which a system fulfills functional and non-functional requirements, specific to an individual tenant. However, although some approaches may enable the MTA to provide different service qualities, this is not a requirement for the approach itself. If it has to support this, it should be defined as a goal. *Variability* refers to the "degree to which the system can support customized solutions and tenant-dependent configurations, extension and evolution" [Pors et al., 2013]. Following the previous argumentation, it is not relevant either. *Diverse* SLAs are irrelevant, since it is assumed that the isolation method itself is not part of any SLA. The application is likely to be subject to a certain SLA, which is ensured by the admission control. However, this is not relevant for the selection of the isolation method since these properties are already covered by the performance isolation quality metrics. Additionally, the methods were not developed with QoS differentiation in mind. *Monitoring* is important to realize certain isolation methods. However, the monitoring functionality of the approach itself is less important. If it is important to keep records of the method's behavior, it should be fairly easy for each approach to do so. *Security, authorization and authentication* aspects belong to the concrete realization of a method. Therefore, these aspects should not be a criteria for the selection of a method. If such aspects are of relevance, they have to be realized, and they can be implemented in each of the methods.

Merged Relevant Aspects As the results in Section 6.1.3.2 and Section 6.1.4 outline, it is not necessary to discuss *throughput, supported number of tenants/end users* separately. Therefore, these three parameters identified as important by [Pors et al., 2013] are referred to as *capacity* in the following. Furthermore, the resources used by the methods and thus the performance, is negligibly different. Consequently, this is also covered by capacity. The *software complexity* is closely related to aspects such as *maintainability* and *development costs*. Therefore, software complexity is not

discussed separately, to avoid an overlap and thus double counting. Originally, development costs were not considered by Pors et al. although they are of high importance to argue for or against MTA (cf. [Momm and Krebs, 2011]). Since performance isolation methods will be introduced to increase the economic success, a trade-off between expected revenue increase and costs of their development has to be made. *Recoverability and availability* aspects are subsumed to the *reliability* of the isolation method.

Considered Requirements for the solution Based on the previous statements, the following criteria were selected to find a suitable performance isolation method in the set of alternatives: Isolation capabilities, settling time, oscillation, efficiency, capacity, reliability, maintainability, development costs and operating costs.

Goals Based on the characteristics of the various isolation method classes (cf. Section 5.2.1), the following goal a solution has to achieve can be obtained: *The method should isolate the performance of the tenants.* This means, that all classes identified in Section 5.1 are potential candidates.

5.2.3.2. Decision Making

The way a particular alternative of the isolation methods is evaluated concerning the requirements, is an interplay with the decision making tool and technical constraints, limiting the usage of some methods.

Process Steps To find an appropriate isolation method for a given MTA, the subsequently present process is proposed. The *preparation* phase compares existing isolation methods concerning the identified selection criteria, described by the requirements. This is an independent and asynchronous activity. However, it can be done with a dedicated application type in mind, to be more representative. The comparison can be based on concrete implementations, or on the class level. Furthermore, a set of rules to define, in which situations a particular method cannot be applied is created. The created artifacts are used later, to find a suitable solution for a concrete MTA, and can be reused for other scenarios. The preparation process requires a performance isolation specialist, because detailed knowledge about the isolation methods is needed.

Once the documents exist, the concrete selection process can start. Hereby the criteria/requirements are ranked according to the importance for the given application scenario. The set of rules is then applied, to see if the best ranked method can be applied. If this is not the case, the next best methods are checked against the rules, until an appropriate method is found. However, if no satisfying solution can be found, it is an option to modify the application, to fulfill the requirements of another isolation method. It is worth mentioning that this is an optional step and may require detailed knowledge about the isolation methods. Finally, the method can be applied. A decision method that allows to quantify the difference between two alternative performance isolation methods can help to justify potential modifications. For the *selection* process, detailed knowledge about

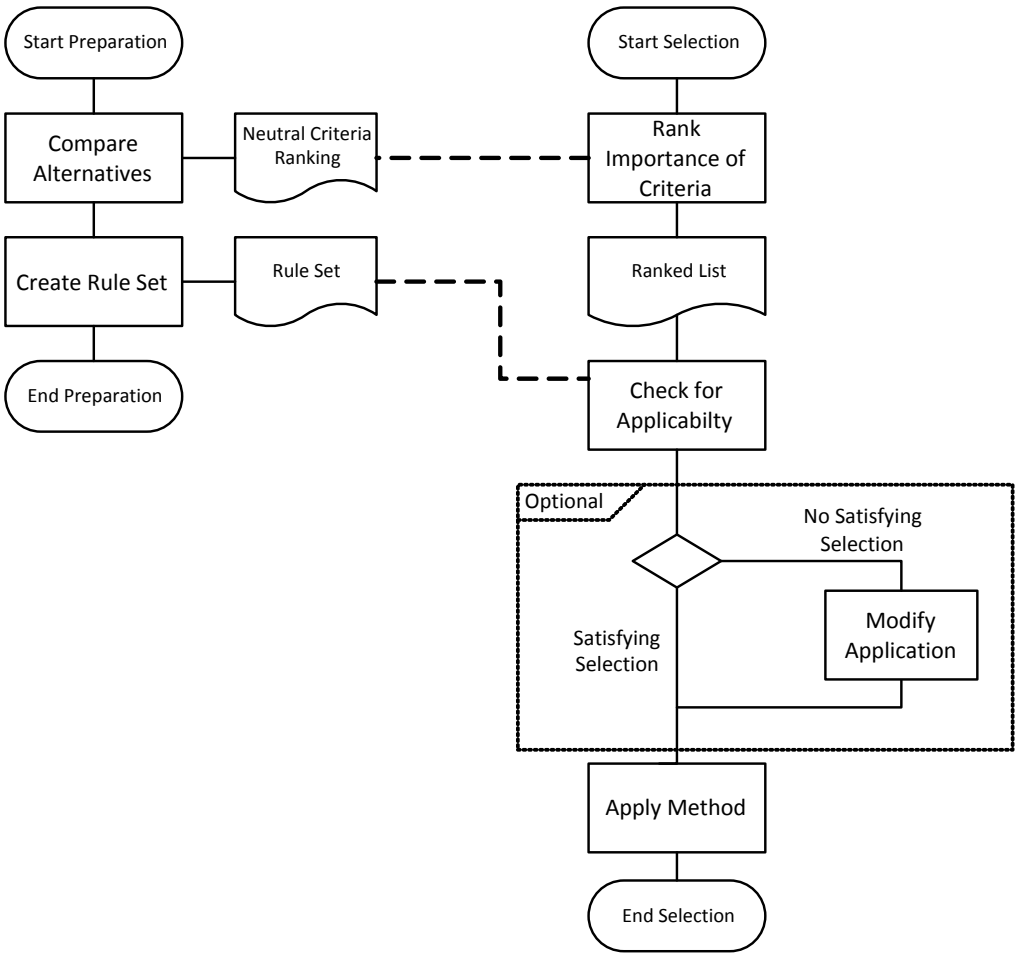


Figure 5.2.: Selection process to find the most suitable isolation method.

performance isolation methods is not necessarily required. Consequently, the expert knowledge is reusable in different contexts.

Ranking of Isolation Methods The selection of a proper decision making tool or method is crucial to receive a proper ranking of the isolation methods. Therefore, it is reasonable to systematically look for a decision support method, fitting to the given task. Multiple Criteria Decision Making (MCDM) tools/methods are used in situations where multiple criteria are present. They can be branched in two groups. Multiple Attribute Decision Making (MADM) address problems where the alternatives are a finite set. Whereas Multiple Objective Decision Making (MODM) assume the alternatives to be infinite. This is possible, e.g., if a value in a continuous value range has to be determined. In the given scenario, the decision space is discrete, containing the isolation methods, and therefore MADM is discussed in more detail.

A decision making method should fulfill a few requirements to be used for the selection of a performance isolation method. It should be possible to compare the alternative methods with regard to each criteria. However, for a concrete application, different criteria are of different importance. This should be reflected by the method. Ideally, a comparison of the isolation method can be reused for several applications. To allow a comparison of different results, a ranked list of isolation methods, with an indication of the difference between two positions should be the output.

More than 70 MADMs exist [Koen, 2008, Figueira et al., 2005]. Sen et al. and Fülöp et al. [Sen and Yang, 2011, Fülöp, 2005] distinguish between: elementary-, Multi-Attribute Utility Theory (MAUT)- and outranking methods.

Elementary methods do hardly allow criteria weighting. This means that all criteria used for the comparison have the same weight. Furthermore, if one criterion is evaluated bad, it cannot be compensated by another. In the context of performance isolation, various requirements may have different importance. Consequently, elementary methods do not seem to be appropriate. This is similar for most outranking methods, which discard alternatives.

MAUT methods usually support trade-offs between different criteria and produce a ranked result list. Furthermore, they allow the elicitation of weights and can handle qualitative information. A relative weighting to express the preference of one alternative to another alternative concerning a criteria is an appropriate option for the present task. This allows to easily compare the identified criteria. AHP [Saaty, 2000] is the concrete method used in the following. It is a widely used solution with a large support on existing tools.

The execution of AHP can be split in two parts. The first part compares the alternatives with regard to the criteria, while the second one provides a relative weighting of the criteria. This allows the application in the context of the proposed process.

Rules Based on the classes and the discussed drawbacks, as well as the information requirements, at least three rules can be derived. These rules can be used to strike out inapplicable results from the AHP. For concrete methods, additional rules might appear. Isolation methods provide a more or less efficient distribution of resources and thus support over-commitment in different quality. A rule for over-commitment support might be possible. However, this aspect is already covered by the efficiency criteria in the AHP.

Scenario Situations	Consequence
SLA Guarantees Exists	Only isolation methods that are aware of guarantees/quotas and the respective metrics at runtime can enforce isolation. This includes the guarantee & quota class and the model based approaches. Other methods are less efficient and/or have more often isolation violations.
Missing Monitoring Information	According to Table 5.1, each method has dedicated information requirements. A system must provide the required information.
Variable Res. Demands	If resource demands vary among tenants, non-resource demand based methods will not be able to provide proper isolation.
Unknown Request Types	If incoming requests cannot be mapped to a certain type, with significantly different resource demand at the admission control, a resource exploiting approach cannot be realized.

Table 5.2.: Set of application rules for existing isolation classes.

5.3. Reference Architecture

This section answers how the performance isolation methods can be realized. Therefore, two scenarios are considered.

1. In the motivating example (cf. Section 1.2) an MTA is hosted on a PaaS. Therefore, the platform provider has to provide an environment for the SaaS developer, in which he can deploy the application-specific isolation algorithms. A development of such features, solely based on traditional PaaS runtime environments, is not possible, because the necessary access to runtime performance information and the requests flow is limited.

2. In case the MTA is not deployed within a PaaS, it is still required to find an architecture that helps to realize the methods.

The reference architecture presented is primarily designed to be used as an enhancement for a PaaS, to support performance isolated MTAs. Nevertheless, it can also be used for the second scenario. The focus hereby is on the isolation relevant aspects only. In this section, an analysis of the best position reflecting the information required by the admission control is evaluated. After that, the architecture is presented, and finally important decisions are outlined.

The content of this section has already been published to large extent [Loesch and Krebs, 2014, Loesch and Krebs, 2013, Krebs et al., 2014a].

5.3.1. Position of the Admission Control

In this subsection, the information available on different positions in a load balanced MTA is described. The information available is then elaborated with regard to the scenarios for which performance isolation is possible. Thus, this section discusses the information required for distributed scenarios, whereas Section 5.2.2 discussed the information for a concrete method, without considering distributed scenarios. The recommendations from Section 5.1.2 discussed on an abstract level which kind of affinity is preferred. However, if the recommendations is not followed, the subsequent insights become more important.

5.3.1.1. Possible Positions

Depending on the position of the request admission control, in a cluster, it gathers different data. Figure 5.3 depicts three possible positions for a request based admission control in a load balanced cluster.

Position 1 *In front of the load balancer* the admission control has access to requests of all tenants and can gather all application level quota and guarantee information on the global level, on a tenant and request basis. This is important, since the isolation is measured on the global level from

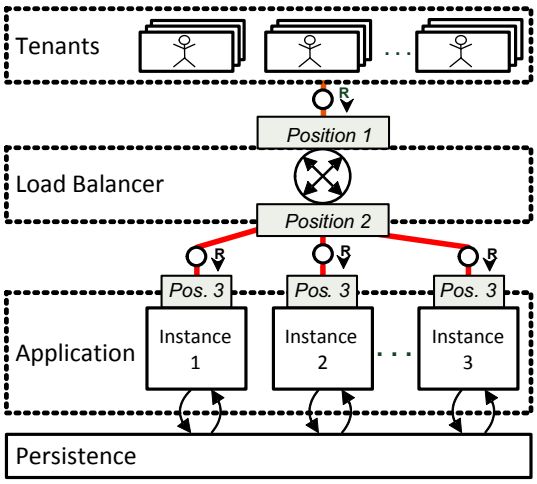


Figure 5.3.: Positions to enforce performance isolation.

the user's perspective. The load balancer decides, for each request, at which node, it has to be processed. Thus, at the position before the load balancer, no information about the request distribution is available.

Position 2 *Directly after or included into the load balancer* the admission control retrieves a superset of the information from Position 1, since it has access to the request distribution. Thus it is known, which node is responsible for which request, and still an overall view is possible. Again, if internal knowledge about the state of processing nodes is required, additional communication overhead is expected.

Position 3 *In front of the application*, the admission control has no global information. Instead, node specific performance information is available and access to the processing node's internal state is possible, with low overhead. If the information of all nodes is shared, the information is a superset of Position 2.

5.3.1.2. Comparison of Different Positions

The kind of tenant affinity and session stickiness are important decisions concerning the application's scalability. A load balancer's goal is to homogeneously distribute load equally among all available application nodes. Thus, the accumulation of requests from one tenant onto a single application node is avoided. In the following, different positions are discussed, with regard to isolation methods relying on tenant specific input and output related performance measurements.

Position 1 In front of the load balancer, the type of tenant affinity is important.

Server-affine: Performance isolation is not possible. It is possible to observe performance qualities, like increasing response times. It is known, that one tenant is always served by one node. However, it is not known, which tenants share the same node. This makes it hard to separate ten-

ants, and thus more complicated to identify the tenant responsible for the problem. Sticky sessions are not important in this case.

Non-affine: For non-sticky sessions, isolation can be achieved. Access to global information is available and thus it can be reasoned for the tenants overall achievement of guarantees and quota. This allows to performance isolate tenants ,by a request admission, in front of the load balancer. If sticky sessions are used, requests are processed by an unknown node. Thus, requests are not longer uniformly distributed. Hence, the most aggressive tenant is not necessarily responsible for a potentially bad performance of another. Although initial requests are distributed homogeneously, the system has a certain risk that long running session lead to an unbalanced system.

Cluster-Affine: A proper performance isolation is not possible and the system behaves similar to a sticky session and non-affine setup. The fundamental problem is, the missing allocation information or the missing uniform distribution of requests.

Position 2 Directly after, or included in the load balancer, the same characteristic appears for all kinds of affinities. *Server-Affine, Non-Affine, Cluster-Affine:* Performance isolation is possible. The load balancer keeps state to ensure a certain tenant affinity and session stickiness. The information available includes the information from the two other positions (except the node internal state). This allows to increase performance isolation.

Position 3 In front of the application, different types of tenant affinity have to be considered. *Server-affine:* Performance isolation is possible. Information from other nodes does not provide any benefits, as all requests of one tenant are processed by the same nodes. Thus, all information is directly accessible. Since requests are bound to a node, the session behavior is irrelevant.

Non-affine: Performance isolation requires further information. The load balancer distributes requests of each tenant homogeneously. However, the overall information for each tenant is not available. Thus, it cannot be decided, whether a tenant exceeds its guarantee or quota on a global level. A particular session stickiness does not influence this.

If the processing capacity of each node is equal, the sessions are not sticky, the total number of nodes is known and the requests are load balanced, the global metrics can be determined by projecting one nodes results onto the others.

Cluster-affine: Performance isolation without further information is not possible. The situation is comparable to the non-affine case with less nodes.

Missing information about other node's performance raises a challenge. Session stickiness is again not influencing this. However, isolation is possible, if all processing nodes have the same capacity and requests are homogeneously load balanced.

5.3.1.3. Implications

The session stickiness is not always relevant. In a non-affine setup, with session stickiness, a centrally managed admission control, using the global information of all requests and the node specific

performance is required. Position 1 and position 3 do not offer isolation, because of missing information about the requests on each individual node. Position 2 can realize isolation for all situations. Table 5.3 summarizes the findings from above. The first column defines the tenant affinity and the second column the session stickiness. The other columns depict for each position the capability to maintain performance isolated.

Tenant Affinity	Sticky Session	Pos. 1	Pos. 2	Pos. 3
affine	no	-	√	√
	yes	-	√	√
non-affine	no	√	√	√
	yes	-	√	-
cluster-affine	no	-	√	√
	yes	-	√	-

Table 5.3.: Positions of admission control and possibility to achieve performance isolation.

This section discussed the required information for distributed scenarios and methods reflecting quotas and guarantees, or being input aware only. For model based approaches, the data to calibrate the model is required. If resource demand based approaches are applied, a deployment at the application node has more advantageous, since resource information and other intrinsic data are more likely to be available, and with a better accuracy. However, if resource usage of tenants should be globally optimized, a communication between the separate admission controls is required. Consequently, if data is transferred between nodes, position 3 has the highest potential to realize isolation methods.

If the recommendations for the development of performance isolated MTA is followed (cf. Section 5.1.2), a load balancer would be able to maintain the utilization of all nodes the same. Thus an isolation would be possible on each position.

5.3.2. Performance Isolation Framework

This subsection prospects a framework used to implement performance isolation methods. The requirements are elicited, an overview of the general concept and a brief discussion on the details of the components is presented.

5.3.2.1. Requirements Elicitation

The discussion is separated into concrete requirements concerning the functionality, the scope of supported isolation methods and non-functional aspects.

Functionality Enforcing *performance isolation* is the primary goal. In addition, some tenants are willing to pay more for performance as others. Hence, the solution may provide *QoS differenti-*

ation, although it was not the primary goal of the thesis. *Over-commitment* increases the economic efficiency of SaaS offerings. If every tenant is using the full quota and thus the system runs in an overloaded situation, the framework must keep a valid state. Since workloads from tenants are characterized by fluctuations, this must be covered by the solution.

Isolation Method Categories Several classes for performance isolation methods are introduced in Section 5.2.1. All have application scenario specific pros and cons, all methods have specific information requirements. The proposed architecture should be able to provide the information for all classes. However, a concrete implementation of the architecture might focus on some classes only.

Generic Solution The system should support various performance isolation methods, since the preferred approach, strongly depends on the application deployed on the platform. Thus, the application-specific isolation algorithm, has to be decoupled from the generic parts, that are common for all algorithms. This includes technical details like data transfer, and domain specific reused functionality. The proposed solution should be portable. This means, it should not rely on a particular PaaS, middleware or operating system features

Scalability Issues In MTAs, horizontal scaling is widely used. A load balancer represents the endpoint for the tenants and forwards requests to the nodes [Koziolek, 2011, Koziolek, 2010]. Therefore, the architecture should provide measures to cover various kinds of affinity and session stickiness. Some applications may support elasticity and thus dynamically add or remove nodes. It is not the goal of the framework to enable elasticity. However, if the application is elastic, the framework should be able to work with this dynamic behavior.

Performance Overhead and Application Scenario The focus is on interactive web applications and consequently low performance overhead onto the MTA is expected. Furthermore, the general overhead, including the communication should be low.

5.3.2.2. Overall Concept

The load may be unequally distributed among the application nodes. Thus, the admission control has to be specific for each node in a cluster. The discussion in Section 5.3.1 outlines that a central controlling instance of the request processing is needed to support all kinds of affinity. Hence, the proposed PIF architecture splits the functionality in two parts (cf. Figure 5.4). First, the application node specific request admission control (*Execution Point*), which can be a separate proxy, or part of the runtime containers request processing pipeline. Its behavior follows a specific *Admission Control Strategy*. Second, the *Performance Isolation Framework Core (PIF Core)*, which contains the application specific isolation algorithm. It periodically sends a *Policy* to the strategy.

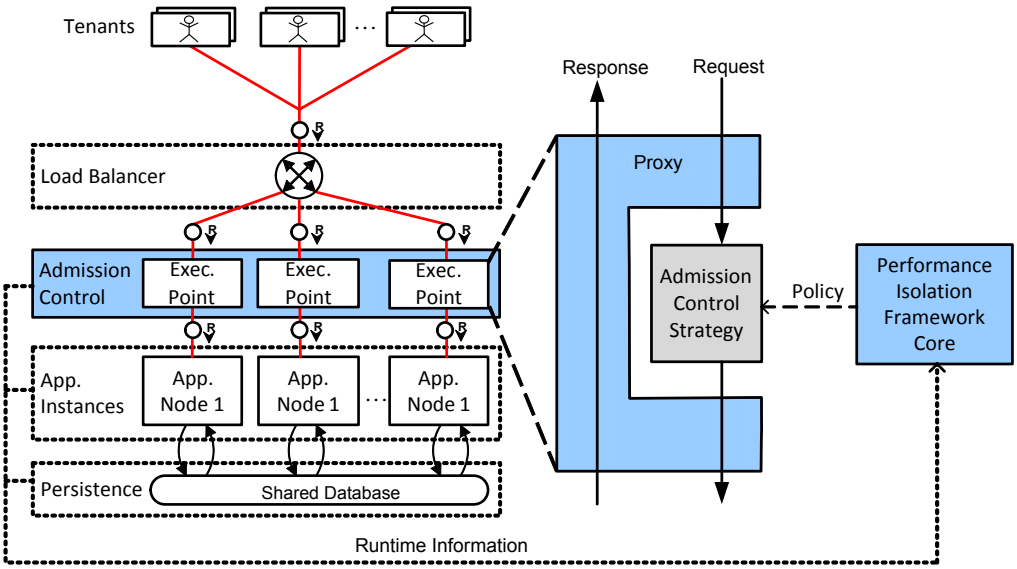


Figure 5.4.: Conceptual overview of the Architecture.

An *Admission Control Strategy* is a parametrized algorithm/scheduler, defining how to admit, reject or delay a request based on the tenant it originates from. Thus, it realizes the request scheduling. Examples of strategies are the schedulers presented in Section 4.2.3 and 4.3.2.2.

Policies adjust the strategy. They are fragments containing the new configuration, exchanged between the PIF core and the execution point. An example is the definition of weights per-tenant, for a weighted round robin admission control.

Structural Overview An overview covering all entities in the system is depicted in Figure 5.5. It consists of seven major entities:

A *Tenant* with several users send requests that are distributed by the *Load Balancer*. Thereby, potential tenant or session affinities are considered. The *Execution Point* handles the request admission, based on an exchangeable strategy. The *Performance Isolation Framework Core* tracks state of data required for the generation of policies. Furthermore, it creates the policies for the *Execution Point*.

The *Application Instances* run the MTA's business logic and process requests. Monitoring information is collected from required services such as a *(Shared) Database* and the application nodes. An *Elasticity Manager* in cloud environments usually realizes horizontal scalability. In the context of the framework, it is also responsible to start the respective execution point.

Detailed Component Description In Figure 5.5 layers can be identified. The persistence stores the monitored data. The next layer, consisting of two parts (dashed rectangles), preprocesses the data when triggered by the *Information Collector*. The information collector combines the information and provides a single view on the system. The policy controller ensures the communication with the clientside execution points and the activities executable their. More details onto the components depicted in Figure 5.5 follow.

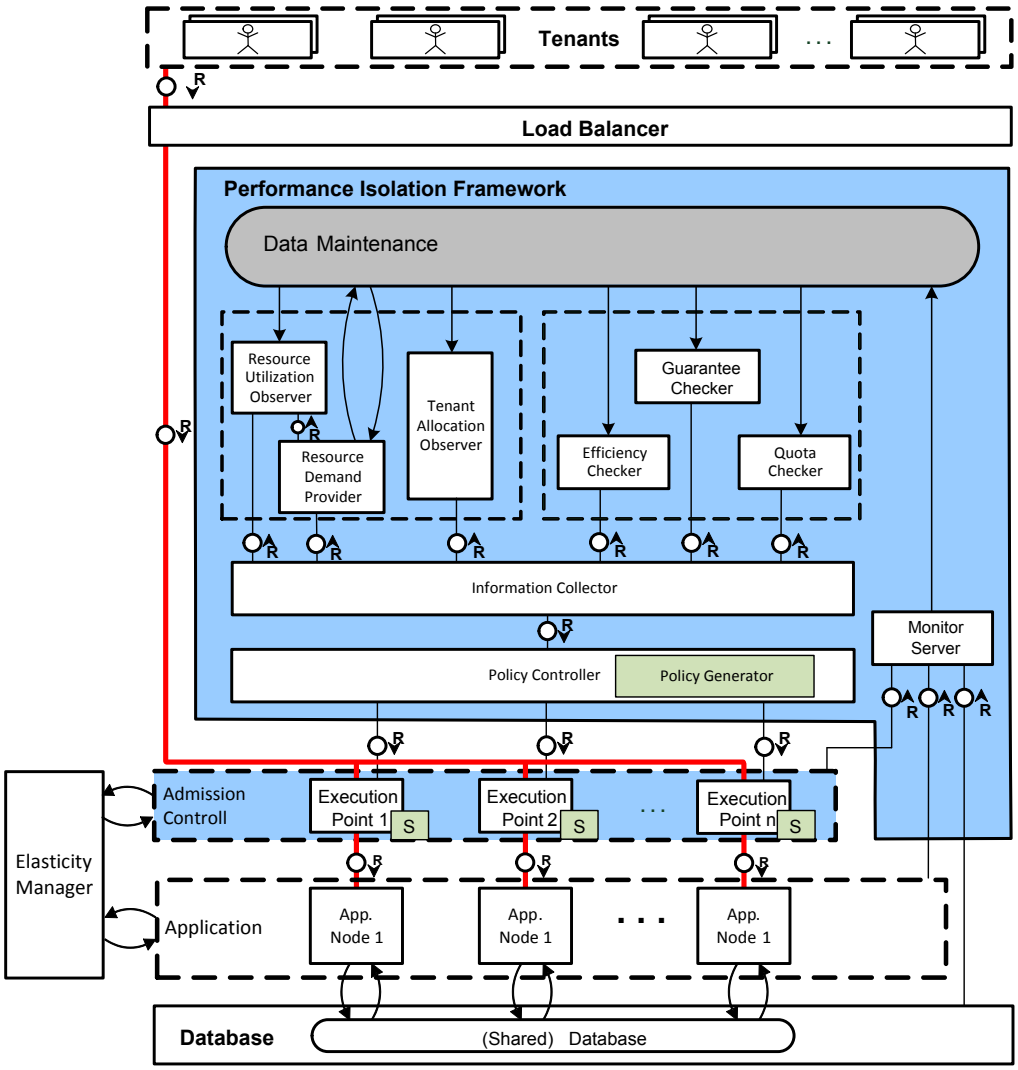


Figure 5.5.: Architecture of the Performance Isolation Framework

Execution Point The component is responsible for admitting, rejecting or delaying incoming requests in order to guarantee isolated performance. In case of static approaches, the policy is updated only once at the beginning, based on the existing SLA, affinity and allocation of tenants onto nodes. At start-up/shutdown, this component registers/unregisters itself at the *Policy Controller*. The concrete strategies (*S*) running as plugin, can be changed at runtime. This allows to fundamentally change the behavior. In case of DoS attacks, for example, another strategy might be beneficial as in normal situations.

Policy Controller/Generator The *Policy Controller* generates the policy for the strategy, executed by the execution point, and keeps the state of them to establish the communication. This includes the registration process of execution points. The *Policy Generator* is an application scenario specific plug-in component. The *Information Collector* periodically triggers the *Policy Controller*. This generates new policies, with the help of the policy generator, for each *Execution Point*. An individual policy is required, as affinities or variable performance of the application nodes can lead to different performance.

The created policies are strategy specific and thus tightly coupled to each other. One strategy might require a delay per-tenant, while another needs a set of priorities. Therefore, the interface between the *Execution Point Strategy* and the *Policy Controller* defines meta information (e.g., tenant IDs) and a generic part used by the policy data.

Monitor Server The *Monitor Server* receives data from different probes at runtime and persists it within the *Data Maintenance*. The various probes push aggregated data in defined intervals to the monitor server. The time frames for the aggregation can be configured. The two most important probes are listed below.

Execution Point Probe: Reports the overall response times, the processing time of requests after admission, discarded requests and throughput at least on a tenants granularity. It is possible to configure the monitoring based on request type and the aggregation interval.

Application Server Probe & (Shared) Database Probe: The resource demand is usually computed at runtime. Therefore, the utilization of the respective resources is measured. The required granularity depends on the configuration and used isolation method. This allows the implementation of resource isolation mechanisms.

Information Collector The *Information Collector* iteratively collects the necessary information for a policy generation. The information is forwarded to the policy controller in a coherent view. Policy update intervals, and the call sequence of the components are managed by this component.

Guarantee Checker, Aggressiveness Rater and Efficiency Checker The *Guarantee Checker* determines the compliance of the guarantee per-tenant and application node, as well as the overall compliance. The tenant- and node-individual evaluation is needed due to the aforementioned differences, caused by affinity and stickiness. By default, the guarantee checker computes the ratio between the guarantee and the observed quality, additionally it provides the absolute values.

The *Aggressiveness Rater* component reports the aggressiveness for all tenants. It uses the difference between the quota and the induced load. To optimize the policies, the distribution of the requests among all application nodes is also provided.

The *Efficiency Checker* calculates the efficiency of the current policy. One possible metric for efficiency is the ratio between the current throughput and the maximum throughput a system can achieve. In context of over-commitment, the differences of the abiding tenants to their guarantee is more appropriate. This enables algorithms to increase the throughput for a tenant, which already exceeds its quota, to keep the overall resource utilization in a good state.

The calculation rules for all three components, discussed in this section, are realized as a plugin, and can easily be replaced by an application scenario specific variant. Furthermore, additional outlier filters can be added. All the calculations are done on an application node basis.

Tenant Allocation Observer In case of an affine behavior, tenants that mutually influence each other are identified. It also identifies which application nodes are shared with each other.

Resource Utilization Observer and Resource Demand Estimator The *Resource Utilization Observer* provides the average resource utilization for a relevant time frame, for a dedicated resource. If a particular resource utilization can not be directly measured, this component provides estimates. The utilization per-tenant is either derived by statistical methods and stored in the *Data Maintenance* or directly measured and already persisted by the *Monitor* in the same table. When the component is called, to deliver the corresponding data, it selects the information stored in the *Data Maintenance* and returns it. The information is required by the most RDE methods and for the resource exploiting isolation methods.

The *Resource Demand Estimator* estimates the resource demands. This is based on the tenant or request type and is subject to the chosen isolation method. Whereby the same request type of different tenants is considered separately. This component works asynchronously. When it is triggered, it returns the precomputed data. In case the demands are directly measured, they are stored in the data maintenance and returned instead of the estimates.

Elasticity Manager The elasticity manager usually already exists and must be adapted to start/-modify an execution point for each added application node.

5.3.2.3. Isolation Method Related Plugins

A concrete performance isolation method consists of (1) a strategy, and (2) a strategy-specific isolation algorithm, referred to as *Policy Generator*. Both are realized as plug-ins and hence the proposed architecture allows to realize performance isolation, with regard to specific needs, by implementing the two interfaces. The framework periodically starts the generation of a policy for a certain execution point, based on the preprocessed information. The strategy has to implement two methods. The first is called to add an incoming request. The transferred request object is framework-specific and provides metadata (e.g., tenant ID, request type). The second method returns a request for processing. It is called when a server thread becomes free. Moreover, the add method can directly forward a request to the application server not waiting for the get call, or immediately reject a request. To identify a certain request type, the implementation of an additional interface is required. It defines only one method, with the request object as parameter, returning a unique identifier for the request type. Furthermore, it can add additional information, such as relevant application specific metadata. An example is a request that belongs to the context of a larger number of data to be transferred. Depending on the MTA, such requests might be admitted with preference.

5.3.3. Relevant Design Decisions

The various design and trade-off decisions taken to fulfill the requirements elaborated in Section 5.3.2.1 are discussed in the following.

5.3.3.1. Isolation Capabilities

High Degree of Isolation vs. Performance Overhead A policy, configuring the execution point for a fixed time frame, results in a lower accuracy, since the workload may change in the meantime. Contrarily, the overhead to make complex computations for each request is not acceptable. Such approaches may make sense, if long running jobs have to be scheduled. However, the focus within the thesis is on interactive scenarios. Therefore, the execution point implements a fast working strategy to control the requests, and complex computations are done asynchronously, in the policy generator.

Data aggregation in the monitoring processes further reduces the accuracy and adoption speed. However, due to the enormous number of requests, there is no adequate alternative. It is worth mentioning, that monitoring intervals and policy update intervals can be adjusted, to increase the adaptation speed.

QoS Differentiation & Over-Commitment The policy generator receives all relevant information. To ensure this, the classification of the approach and the corresponding informational requirements were identified Section 5.2.2. Additionally, the information required in a load balanced system was investigated. Consequently, the policy generator can realize the isolation methods and has the information to efficiently provide over-committed systems.

5.3.3.2. Generic Solution

Suitability for Various Isolation Methods The proposed architecture provides components and measures to deliver all information required by the mechanisms identified in Section 5.2.1.

Plugin Mechanism for Isolation Algorithm The policy controller abstracts the technical details concerning the communication, registering and unregistered of an execution point. Thus, it keeps state of the execution points. The information collector collects all relevant information and forwards it to the policy controller. Thus, the policy generator can easily be implemented and replaced by an alternative implementation, as it is decoupled from technical details and the actual preprocessing of the required information. If needed, individual implementations for the SLA/efficiency checker and the aggressiveness rater are possible.

Portability Between Different System Environments Not all load balancers support enhancements to control/delay requests based on strategies, hence the are implemented in the execution points. The execution points are deployed together with the MTA instance. Furthermore, no reason to use OS dependent interfaces exists. Even the RDE methods are independent of the concrete technique used to measure the utilization of resources.

5.3.3.3. Scalability Issues

The elasticity manager has to ensure that for each new application node a corresponding execution point is instantiated. The new execution point is automatically registered at the policy controller. Therefore, policies are generated from that point on for the execution point. In general, the amount of policies increases linearly with the number of application nodes and the data to be processed grows linearly with the amount or tenants. Furthermore, the framework can deal with different loads on the application nodes, and thus it supports different types of affinity. The allocation observer identifies the relevant nodes and tenants, which are responsible for a potential isolation incident. Thus, the policy generator can focus on the relevant entities to solve the issue.

Another decision was to put the execution point close to the application server. This hinders the execution point from being a bottleneck, in case a high number of requests arrive and queuing is required.

5.4. Conclusion

This chapter outlined relevant aspects, when it comes to the implementation of a performance isolation method. A critical discussion of the proposed solutions is presented before a summary concludes this chapter.

5.4.1. Critical Discussion

Three positions for a request based admission control in a load balanced cluster are discussed in Section 5.3.1. This discussion is based on the distribution of the throughputs and response times per tenant and node. If an isolation method does not rely on this information, the results may not be useful for it. A short outline of the impact on other relevant isolation classes was given in Section (5.2.1).

Only AHP was considered in detail for the decision making process. Other methods might also be applicable. The existence of other decision making approaches does not falsify the general approach. AHP itself is not a contribution, but one very feasible approach fulfilling all requirements.

AHP has a rank reversals issue. This is a phenomena in several decision making methods. Rank reversal means that adding a new alternative can change the ranking of the old alternatives in relation to their previous rank. For example, a new alternative is ranked on position three and the rank of position 1 and 2 changes. However, this also happens in human decision making, since the amount of references increases and thus the amount of information.

Only a limited set of criteria, relevant for the selection of performance isolation methods was identified. This might not be feasible for all situations. If a criteria is of importance, which was not yet covered, this information can be forwarded to the performance isolation expert who updates the criteria ranking. AHP allows to easily add new criteria without readjusting previous settings.

5.4.2. Summary

An overview and a discussion of various architectural concerns in MTA and how they interact with performance isolation was given. Based on these insights, design recommendations for MTA were defined.

Furthermore, existing performance isolation methods were classified and their specific advantages and disadvantages were outlined. A comparison of their information requirements, helping developers of multi-tenant applications to find an appropriate method was also provided.

After that, relevant requirements for MTAs, which are of interest for the performance isolation method, were identified and a method and process using AHP to select an appropriate isolation method was proposed.

The best place for a performance isolation component, levering request admission control, was analyzed from an information centric point of view. While the classification schema and its information requirements focused on the methods, this investigation focused on a deployment question, when it comes to an implementation in a load balanced environment. It was shown that the positions before the load balancer, or directly before the applications have disadvantages. However, the use of information about a processing node's state allows to increase the performance isolation quality and this is best possible when the component is located at the respective processing node.

Finally, a reference architecture was prospected that enhances PaaSs or non-PaaS based MTAs to enable application developers realizing tailored performance isolation mechanisms.

6. Evaluation

This chapter summarizes validations from previous chapters and adds additional evaluations of performance isolation methods, to identify the strength and weaknesses of them. Especially the latter provides important insights for MTA providers to identify a suitable isolation method for a given scenario (cf. Section 5.2.3.2).

Chapter 1 defines the general goal of the thesis as "This thesis aims at providing performance isolation mechanisms, a methodology to quantify performance isolation and insights based on these methods that support MTA providers realizing performance isolated MTAs.". Consequently, the validations an evaluations in this thesis focus on the three major parts, listed in the following.

Evaluation Goal 1 — *Quality of the Isolation Methods*
It has to be shown, that performance isolation between tenants in an interactive web based application can be increased, by the admission control of incoming requests using the proposed methods. In addition, a comparison of performance isolation methods, with regard to the effectiveness of the isolation, the efficiency of the share distribution, settling times, oscillation and performance related overheads are relevant.

Evaluation Goal 2 — *Method to Quantify Performance Isolation*
Quantifying the degree of isolation is essential for the evaluation of the methods and for the MTA stakeholders. Thus, the metrics and measurement related contributions have to be tested, concerning their functionality and applicability.

Evaluation Goal 3 — *Applicability of the Performance Isolation Methods*
Another focus for the validation is whether the developed framework can be applied in different contexts and if it is able to realize the isolation methods of the different classes defined in Section 5.2.1. Moreover, it has to be shown that the selected decision support method and process can help increasing the performance isolation.

To systematically validate the contributions, the GQM approach is used [van Solingen et al., 2002, van Solingen and Berghout, 2001]. The separate chapters already provided insights and validations of distinct contributions. The measurement environment was already used by the validation of the methods in Chapter 4. The metrics are used in the experiments in this chapter. Therefore, a brief evaluation was already done in Chapter 3. For the isolation methods, it made sense to analyze their

behavior and the underlying hypotheses together with the method to support the understanding of the detailed outcomes. However, for a global comparison or isolation methods and a summary of the results, this dedicated section is more appropriate.

The remaining chapter is organized as follows. At first, the isolation methods are compared to each other. The results of the metrics and measurement related contributions are summarized at second, since they rely on some insights from the first part of the evaluation. The last section focuses on the enabling of the MTA developer and includes a case study.

6.1. Isolation Methods

In Chapter 4 two novel methods to realize performance isolation are presented together with known patterns from related fields. In this section, the evaluation goals are outlined and motivated together with the related questions and metrics. Experiments are conducted to answer the relevant questions if not already done by the previous chapters. The summary of the results comprises all insights from this and the previous chapters to cover all results with respect to the GQM plan.

6.1.1. Goal, Questions and Metrics

The first goal **G1** is to evaluate the isolation methods capabilities to efficiently isolate a system in terms of performance. Whereby efficiency is interpreted in two ways. First, with regard to the ability to provide over-commitment. That means, a disruptive tenant should be able to use resources from abiding tenants, as long as the abiding QoS is not beyond the guarantee (cf. Section 1.3.2). Second, the overhead of the methods. To achieve this goal, the following questions (Q) will be answered by using the subsequently listed metrics (M).

Q1.1 How good is the isolation provided by the performance isolation methods?

- **M1.1.1** Isolation (I_{base}, $I_{intBase}$, $I_{intFree}$ I_{QoS}) for increasing disruptive load.
- **M1.1.2** Settling time, after a load increase by the disruptive tenant.
- **M1.1.3** Oscillation, while the workload is not changed.
- **M1.1.4** Percentage of time, in which a guarantee violation appeared, for a replayed real workload.

Q1.2 How efficiently can the performance isolation methods support over-commitment and the distribution of the resources?

- **M1.2.1** Gap between observed performance and the guarantee for abiding tenants, while another tenant continuously increases its load.
- **M1.2.2** Settling time, after a decreasing load of the abiding tenants.

Q1.3 What is the overhead of the performance isolation methods and how well do they scale?

- **M1.3.1** Impact upon the response time and throughput.

M1.3.2 Impact upon the utilization of the server, hosting the core parts of the isolation framework.

In addition to **Q1.1** to **Q1.4**, some method specific questions are of relevance to understand in detail how they utilize their knowledge about the system. For the resource isolation methods presented in the thesis, the goal (**G2**) is to gain insights on how accurate, these methods can leverage the resource demand knowledge to ensure resource isolation. For the model based approach, the goal (**G3**) is to gain insights on how accurate the method can maintain isolation and an efficient distribution of weights.

Q2.1 How accurate is the resource demand estimation?

M2.1.1 Difference between predicted utilization and real utilization.

M2.1.2 Resource demand estimation reflects tenant individual demands.

Q2.2 Can the method enforce performance isolation, by using the resource demand knowledge?

M2.2.1 Influence of one tenant upon the performance of another while both have the same resource demands per request.

M2.2.2 Allocated resources for each tenant, while both have the same demands per request.

M2.2.3 Difference of performance between two tenants with the same load, while the resource demand is different.

Q2.3 Can the method provide different resource guarantees to individual tenants?

M2.3.1 Difference of performance between two tenants with the same resource demands per request, while the resource quota is different.

Q2.4 Can the approach adapt to a changing demand at runtime?

M2.3.1 Time to provide a proper isolation, after the demand for one tenant has changed.

Q2.5 How accurate is the isolation of the method?

M2.5.1 Influence of the disruptive tenant upon the abiding.

Q3.1 Can the mode model provide accurate results?

M3.2.2 Difference between predicted and observed response times for tenants.

M3.2.3 Difference between predicted and observed throughput for tenants.

Q3.2 Do the fitness functions configurable parameters influence the isolation and efficiency capabilities?

M3.2.1 Change in response times/throughputs for the tenants for various tuning parameters.

Q3.3 How accurate is the isolation of the method?

M3.3.1 Influence of the disruptive tenant upon the abiding.

G2 and **G3** are answered by the experiments in the respective sections of Chapter 4. Therefore, the focuses is on the first goal, which is the comparison among all the methods. However, the outcomes from the previous section are summarized again in this section, to answer the respective questions.

6.1.2. Multi-Tenant Application Based Experiments

Subsequently, the system landscape for the evaluation of the isolation methods is presented, followed by the experiments and workloads. After that, the results of the experiments are discussed. This section primarily focuses on **Q1.1** and **Q1.2**

6.1.2.1. System Landscape

Hypervisors can have a significant impact upon the performance behavior observed by an application hosted within a virtual machine [Huber et al., 2010, Huber et al., 2011] and may obfuscate typical characteristics. However, virtualization is a widely used technique, and at least 70% of x86 server workloads are virtualized with increasing numbers [Bittman et al., 2014]. One example is the HCP (cf. Section 1.2). Consequently, an environment where the application runs in a hardware virtualized setup is more realistic. To apply results from a non-virtualized measurement, additional knowledge about the hypervisor is required. Predicting the performance of applications running in a virtualized environment, where the characteristics of the hypervisor and the applications have not been evaluated together, is a research question in itself. Furthermore, it is convincing that the differences between two different hypervisors are less, compared to the difference to a non-virtualized system. Consequently, using virtualization is seen as beneficial for the experiments answering **G1**.

The environment presented subsequently is motivated by the relevant aspects of the HCP. The methods were evaluated from the perspective of an MTA provider, who hosts the application on the platform. Assuming the platform services to be isolated from other applications and tenants, the relevant parts for the experiment environment were the runtime container (LJS) and a relational database system. Figure 6.1 presents the detailed deployment, based on the motivating example from Section 1.2.

As workload generator, the multi-tenant version of the TPC-W benchmark (MTTPC-W) was used. (cf. Section 3.3). The experimental setup comprised five physical servers: The *Load Server* run the *Load Driver*, which emulated the users. The *Application Server* hosted the MTTPC-W application together with the *Isolation Valve* within a *LJS*. The isolation valve leveraged the valve concept [Craig McClanahan, 2015] to implement the execution point. Thus, it enforced the admission control. The *LJS* was running within the *Application VM* on the SAP JRE 1.7, and the *Database Server* served as the persistence layer. The MySQL server, running within the *Database VM* used the *Storage* server to save the database files. The core part of the *Performance Isolation Framework* was hosted on a separate node.

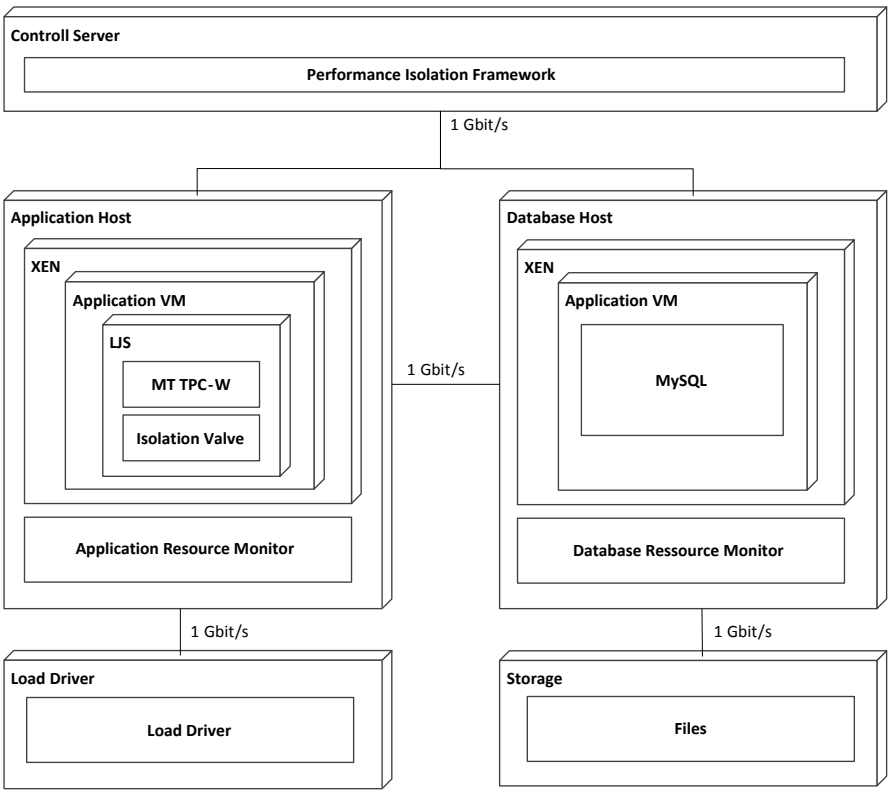


Figure 6.1.: Deployment for comparing the various isolation methods.

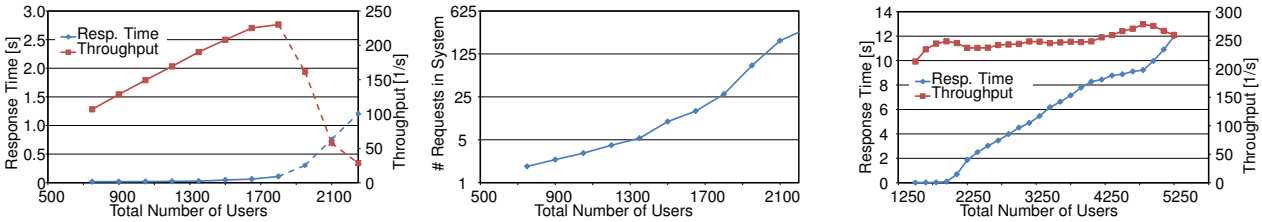
The *Application Server*, *Database Server* and *Storage Server* had the same characteristics. In particular, each of them had a processing power of 8x1.6 GHz, and a main memory of 16 GiB. The *Load Server* was a 4x1.6 GHz server. Each node was connected by a 1 Gbit/s Ethernet LAN to the network. The network was isolated from other traffic. SUSE Enterprise 11 SP2 was used as operating system at all places. A MySQL version 5.1 database was used and Xen 4.1 was the hypervisor. The VMs allocated 2 GiB memory, with one VCPU pinned to one physical CPU. The dom0 was pinned to another physical CPU.

6.1.2.2. Conducted Experiments

In this section, the general performance behavior of the system is investigated. Based on these results, the detailed load profiles for the experiments were derived.

System characteristics To gather the system characteristics, three tenants (t_1, t_2, t_3) were configured. The workload was equally distributed among all tenants and was stepwise increased. In each step, the users were uniformly added within a duration of five minutes. After that, a waiting period of 15 min was maintained. The duration for gathering the data was dynamically adjusted to ensure a representative result. The response time and the throughput were collected as average value for a five seconds window, aggregating the data among all request types and tenants.

The x-axis for all subfigures in Figure 6.2 depicts the total number of users. Figure 6.2a shows the throughput and response times for the SUT, while the load has been increased, and an unlimited size of the server's thread pools was configured. The diagram shows a typical behavior. The throughput



(a) Response time and throughput without request admission.

(b) Amount of parallel requests in the system.

(c) Response time and throughput with request admission.

Figure 6.2.: System characteristics with and without limited thread pool.

increased linearly, until the beginning of a slight plateau became present around 1650 users. The maximum throughput was observed at 1800 users, with 230 requests/s. With more load, the throughput reduced significantly. In parallel, the response times started to increase above 1800 users. The dotted line shows, that in this area the measurements were no longer reliable. From this point on, a high portion of requests had not been able to successfully establish a connection, and the results were not longer within the defined confidence intervals. Since the measurements in this area were already beyond the point of interest for configuring the system, no further investigations about the detailed behavior of the system had been done. Before that point, all experiments showed results, according to the expectations, with a low fluctuation.

Figure 6.2b presents the average number of requests served in parallel by the MTTPC-W application. The system had been in flow equilibrium while the measurements were made. Based on [Little, 1961], the known response times, and the throughput/arrival rate, the number of requests served in parallel were calculated. The y-axis is printed base 5 logarithmic scale.

Based on these insights, the optimal size for the application server's thread pool was determined. For the given scenario, 27 threads turned out to be optimal. However, due to the fact that this point has been very close to an overloaded state and trashing system behavior, a safety margin of 10% was chosen, resulting in 24 threads.

The observed system characteristics for a setup with limited thread pool size is depicted in Figure 6.2c. As the graph shows, the system did not enter an overloaded state. At 1850 users the observed throughput was at 248 requests/s. The increased throughput stems from the limitation of the maximum parallel requests. The throughput maintained a rather steady value.

Isolation Mechanisms Configuration The response times and throughput for the evaluation were collected in five second intervals. To avoid a high overhead, only the aggregated values for this time frame were logged. Besides the scenario dependent configuration for the quotas and guarantees, the isolation methods configuration was the following. Average resource utilization information was sent from the monitors to the PIF core every two seconds. The request type/tenant dependent average throughput, response times, processing times for application server and database server, were reported every two seconds by a probe, integrated in the valve. The update periods for the strategy, and the window size for averaging the guarantee and quota relevant performance metrics were set to 30s. The resource demand estimator was configured to use data from the last

seven minutes for the estimation, which showed good results in the experiments from Section 4.5.2. For the computation of the users' average think times, a two minutes timeframe was configured.

The violation parameter c_v for the model based optimization was set to 0.8 and the penalty parameter c_p to 2.6. The proportional gain for the closed control loop based isolation was set to 0.175 and the integral gain to 0.06.

Abiding tenants can be black listed if the resources from the black listed tenant are allocated by them, since this increases their throughput. Therefore, the black lists threshold was set to $1.3 \cdot quota$. Furthermore, a disruptive tenant had to exceed three periods in a row, i.e., 90s its quota, before it was black listed. This was configured to smooth out short peaks in the request rate. The method specific configurations were selected by doing several experiments to find a good isolation. This was done at 50% of the maximum disruptive load.

In the following, several workload configurations are presented. The mapping of workloads to questions is done within the concrete experiment description, following this section.

Workload I This workload profile was primarily used to determine the isolation and oscillation of the different methods. Thus, it focused on **M1.1.1**, **M1.1.3** and **M1.2.1**. Section 3.1.6 emphasizes the benefits of a reference workload close to the maximum throughput. In the characterization of the system, with a limited thread pool size, the throughput maintained a rather steady value around 240 requests/s when exceeding 1800 users. Consequently, any point above 1800 users is an appropriate reference workload.

The classical doctrine in the field of usability engineering says that a response time, including rendering, below 0.1s is not recognized by a user. With response times higher than one second, the users flow of thought is already interrupted. In case of more than ten seconds a user looses attention [Miller, 1968, Card et al., 1991, Nielsen, 1994]. However, this does not finally answer which response time is still acceptable for situations of high load. According to [Jupiterresearch and Akamai, 2006, Young and Smith, 2006] three to four seconds are the time at which web users start to leave the web page. Thus, the authors recommend to maintain response times below four seconds, including rendering. As reference workload and guarantee, a response time of three seconds was selected for the experiments. This is below the four seconds, to have some buffer for rendering. Three seconds correspond to a reference workload of 850 users per-tenant and a throughput/arrival rate of 80 request/s per-tenant.

Instead starting every tenant with the same number of users, the following distribution was chosen $t_1 = 500, t_2 = 850, t_3 = 1200$. Tenant 1 and t_2 was seen as abiding, while t_3 was the disruptive tenant. Using this setup, an efficient isolation method should be able to maintain the QoS for all tenants at the reference load. Selecting the loads this way, it is possible to gain insights into the behavior of the methods for the most relevant situations. The first situation is an abiding tenant, who is exactly at its guarantee/quota, and thus requires very good isolation. The second situation is defined by a tenant, far below its quota. Thus its unused resources should be used by the others, if the method is efficient.

For the isolation related experiment series, the load for t_3 was continually increased by 100 users. Each load configuration was maintained for a period of 30 min. The first ten minutes of each period were omitted for the calculation of the isolation, to avoid warm-up effects influencing the results.

Workload II This workload definition was used to quantify the settling time (**M1.1.2** and **M1.2.2**). Increasing the load by 100 users was not enough to clearly identify the events for the time measurement (cf. Section 3.1.5.1), due to relatively high noise, compared to the increasing workload. However, a similar workload as in the previous workload definition was considered. For experiments investigating increasing load, the start workload was $t_1 = t_2 = t_3 = 850$ users. This workload was kept for 120 min, before the users for t_3 were increased by a factor of three, resulting in 2550 users. The users were added uniformly within 180s. For load decreasing experiments, the starting load was $t_1 = t_2 = 850$ users and $t_3 = 2550$ users. After a minimum waiting time of 120 min, the load for t_1 and t_2 was reduced to 200 users. Users were removed, when the request token was at the load driver, emulating the think time. Thus, no open connections awaiting an outstanding response were interrupted. Each experiment was repeated five times.

Workload III This workload was used to evaluate the isolation methods in a dynamic load context (**M1.1.4** and **M1.2.2**). The scenario describes a dynamic load behavior by using monitoring information from a real application. In this case, the RuneScape dataset version 1.0 [Marzolla et al., 2012] was used. It contains the amount of RuneScape users from 2011-05-04 to 2011-11-22, with a resolution of two minutes. The dataset stems from a closed workload scenario and lists the number of users, and not the average arrival rates as other popular workload traces such as the FIFA 98 [Arlitt and Jin, 2000] or the wikipedia [wikimedia, 2015] logs. Consequently, it can be easily applied for the MTTTPC-W. Furthermore, it stems from a cloud based application, has a high resolution, contains data from a time frame of six months and is based on an interactive web application. Moreover, it is publicly available. The major drawback is, that it does not stem from a multi-tenant application. However, publicly available, representative and technically usable workload load traces from MTAs are not available.

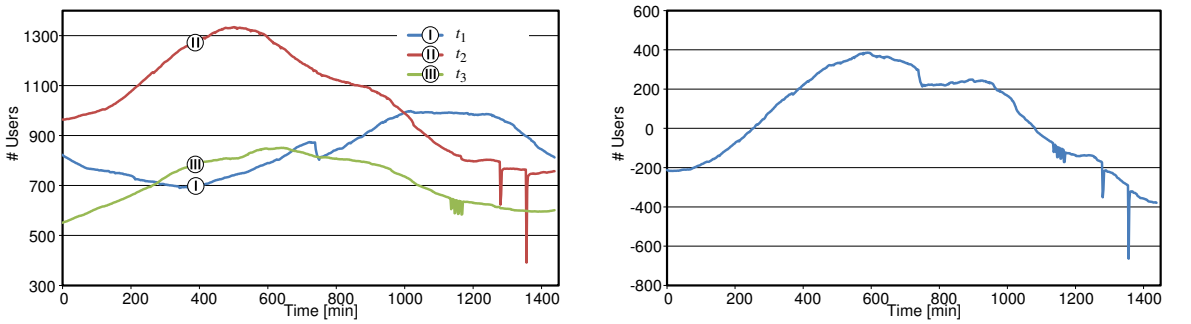
A multi-tenant application provides each tenant a similar functionality. Thus, it is assumed that tenants will use it with similar characteristics, and it can be assumed, that the load profiles are comparable. Most potentially, they vary in intensity and the time when the load is generated.

For this workload configuration, three 24h slices were used from the RuneScape trace. Every sample was multiplied by a slice individual factor, to adapt the number of users to the SUT. Table 6.1 characterizes these data sets.

Figure 6.3a depicts the load behavior of the tenants, while Figure 6.3b presents the overall load. The x-axis shows in both cases the time in minutes, for which the number of users is depicted on the y-axis of Figure 6.3a. In Figure 6.3b the y-axis depicts the difference between the overall number of users using the system, and the amount at which an efficient method would fulfill the guarantee for all tenants.

Tenant	Start	End	Factor	Max Users	Min Users	Average Users
t_1	2011-5-10 06:00:01	2011-5-11 05:58:01	0.0039753	997	681	842
t_2	2011-5-15 12:00:01	2011-5-10 11:58:01	0.0053899	1334	391	1051
t_3	2011-5-26 12:00:01	2011-5-10 11:58:01	0.0041461	851	550	716

Table 6.1.: Slices used to simulate a real workload.



(a) Users per-tenant based on the time.

(b) Difference between actual and maximum workload.

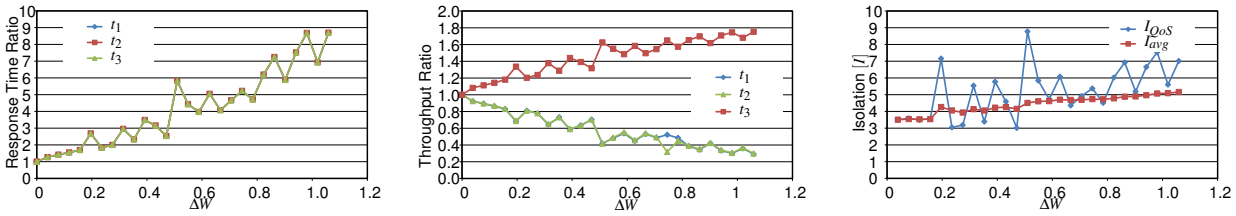
Figure 6.3.: Traces of real load profiles.

The resolution of two minutes is still a bit coarse-grained. Therefore, the data was linearly interpolated and discretized to 20s intervals.

6.1.2.3. Resulting Influence Between Tenants

The subsequent results are based on *Workload I*. The figures present the average response time and throughput for each tenant, together with the achieved I_{QoS}. Thus, it contributes to **M1.1.1**. To emphasize the relevant characteristics, the plotted data is normalized by the performance at the reference workload. This allows to immediately see the relative impact of the increasing load onto different tenants. The x-axis of each subfigure depicts the relative increase of the overall workload induced by the disruptive tenant. The y-axis depicts the relative influence on the throughput and response time. Subfigure (c) shows the I_{QoS} for each ΔW and its average value I_{avg}. I_{avg} is the arithmetic mean of all measured I_{QoS} between the reference load and the current load configuration. The I_{QoS} refers to the influence of the disruptive tenant upon the response time of the abiding.

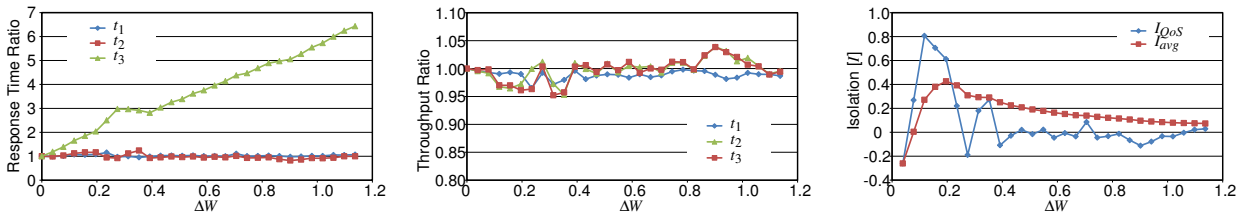
Non-Isolated System Figure 6.4 shows the results for a non-isolated system with a limited thread pool as reference for further discussions. The response times of all tenants increased the same way, since all requests were treated the same way. Consequently, the results of the throughput show the decreasing value for the abiding tenants. The relative decrease of the abiding tenants was almost the same. The I_{QoS} converged from a value short above 3.5 to a value around 5.2.



(a) Response time normalized. (b) Throughput normalized. (c) Isolation based on QoS impact.

Figure 6.4.: Measurement results for a non-isolated system.

Round Robin The round robin method's results in Figure 6.5 show similar characteristics as in the simulation from Section 3.4.1.4. The response times for the abiding tenants maintained a close to constant value. The response times for the disruptive tenant increased linearly. The throughput had negligible fluctuations below 5%. Consequently, the isolation was very good. After a low peak at the beginning, I_{QoS} and I_{avg} converged to a value around 0. On the assumption that the response times of the disruptive tenants keep a ratio around 1, it is convincing that round robin achieves a perfect isolation.

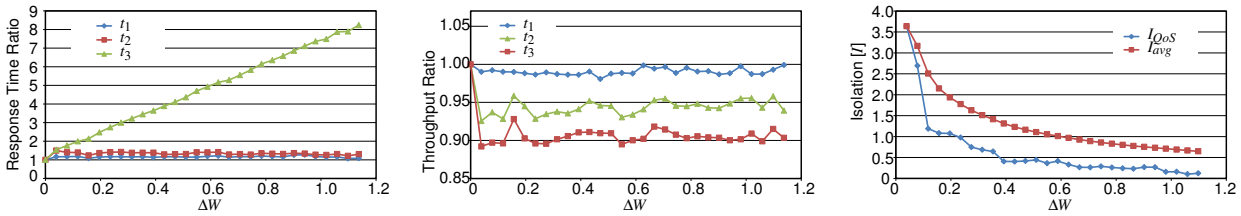


(a) Response time normalized. (b) Throughput normalized. (c) Isolation based on QoS impact.

Figure 6.5.: Measurement results for round robin.

Resource Isolation The resource isolation method is presented in Figure 6.6. Similar to round robin, the response times for the disruptive tenant increased linearly. The response times for the abiding tenants maintained a rather steady value. However, between the first and second measurement, a minor impact upon the throughput and response time of the abiding tenants can be observed in the figure. The performance for t_1 became negligible worse, while the performance of t_2 decreased by around 5%. The throughput of the abiding tenant, reduced by 10%. Consequently, the overall performance of the system became a bit less at the beginning of the experiment. However, the general insights are still valid: The method obviously provides a very good isolation, as the observed parameters maintained a constant value for most of the experiment time. The I_{QoS} and I_{avg} isolation value converges to 0. Without the disturbance at the beginning, the same behavior as for the round robin can be expected.

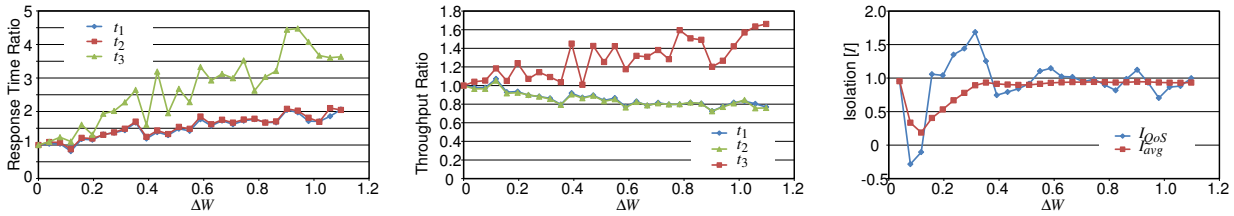
Closed Control Loop The closed control loop in Figure 6.7 shows increasing response times and decreasing throughputs for the abiding tenants. For the abiding tenants, the relative change in response time and throughput was very similar. The change for the disruptive tenant was steeper



(a) Response time normalized. (b) Throughput normalized. (c) Isolation based on QoS impact.

Figure 6.6.: Measurement results for resource isolation.

than those of the abiding tenants. This shows, that the closed control loop provides isolation to a certain degree. The response times and throughput curves characteristics can be interpreted as being linear. The curves for the disruptive tenant show fluctuations. However, there is still a linear trend visible. This leads to isolation values worse as those observed before. I_{avg} became steady around 0.91 and can be expected to maintain this value due to the linear behavior.

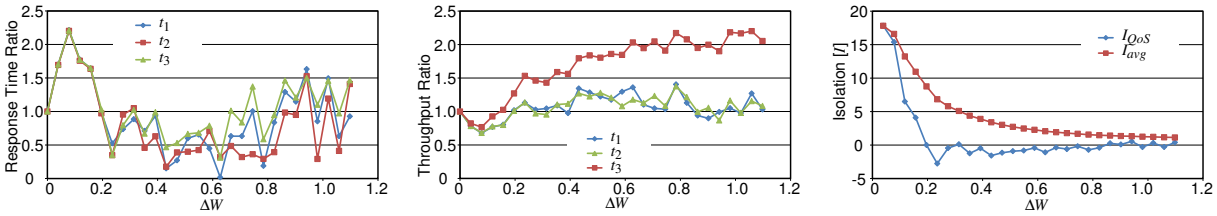


(a) Response time normalized. (b) Throughput normalized. (c) Isolation based on QoS impact.

Figure 6.7.: Measurement results for closed control loop.

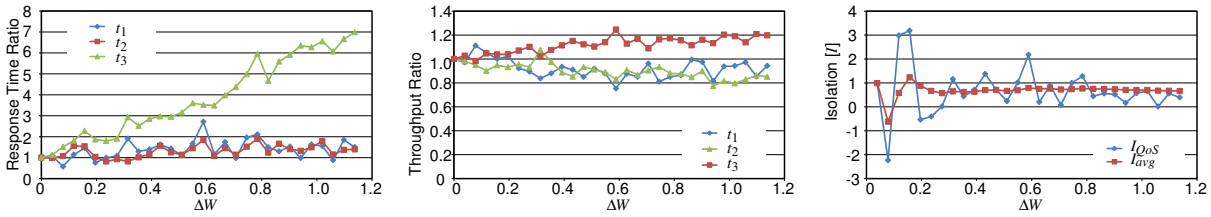
Black List The black lists response times in Figure 6.8 vary between a relative value of 2.2 and 0.25. Although the response time of the disruptive tenant did not increase significantly, one has to keep in mind, that not all requests from this tenant have been serviced. Therefore, a high amount of them became immediately answered without a valid response, but short response times. The same holds for the throughput. The diagram shows an increasing throughput for the disruptive tenant, although the throughput for the abiding ones keeps constant, or even increases for $\Delta W = [0.2, 0.8]$. Accordingly, the isolation of this approach converged to a value around 0. Especially at the beginning of the experiment series, the response times behaved very similar for all tenants. This is because the quota was configured with some buffer. Thus, the system behaved like a non-isolated at the beginning, and all tenants observed the same system behavior for low load increase.

Model Figure 6.9 presents the measurement results of the model based approach. The configuration of the optimization algorithm leads to slightly increasing response times for abiding tenants, which became flatter in the second half. The response times for the disruptive tenant increased significantly. The throughput for the abiding tenants decreased slowly, while the throughput for the disruptive one increased. However, this value also became more stable in the second half. The isolation value was close to 0.9.



(a) Response time normalized. (b) Throughput normalized. (c) Isolation based on QoS impact.

Figure 6.8.: Measurement results for black list.



(a) Response time normalized. (b) Throughput normalized. (c) Isolation based on QoS impact.

Figure 6.9.: Measurement results for the model based isolation method.

6.1.2.4. Resulting Influence Based on Guarantee

The following results are based on *Workload II* and help to answer **M1.1.1**. The subsequent Figures 6.10a/b present the achieved isolation, in relation to the guarantee, for each isolation method. Figure 6.10a depicts on the x-axis the relative increase of the overall workload ΔW, originated by the disruptive tenant. The y-axis depicts the isolation I_{QoS}, whereby the reference for the guarantee is 3s. The figure shows negative values for some approaches, and mostly a significant change of the isolation at the beginning. In the previous experiment series, the reference value was the observed response time at the reference workload. Consequently, the response time increased with higher load and the difference to the reference value was usually positive. When comparing the observed quality, with a reference not gathered by a measurement, there can be a high difference at the beginning. However, a constant difference converges $I_{QoS} \rightarrow 0$, due to the increasing workload. Another aspect is, that some methods increase their restrictions onto the disruptive tenant with growing load, and therefore need some time before adapting.

Figure 6.10b depicts the quartiles of the achieved isolation I_{QoS} for $\Delta W > 30\%$. Above that value, the most methods already converged to a rather steady value. The x-axis depicts the method for the corresponding box plot, while the y-axis shows the achieved isolation value.

The following paragraphs discuss the results of the various methods investigated.

Non-Isolated The workload conducted for the experiments was configured such, that every tenant could be served with a response time of 3s. The non-isolated system does not influence the processing of the requests. Consequently, all tenants observed the same response times during the experiment series. The observed response times at the beginning of the experiment series was short

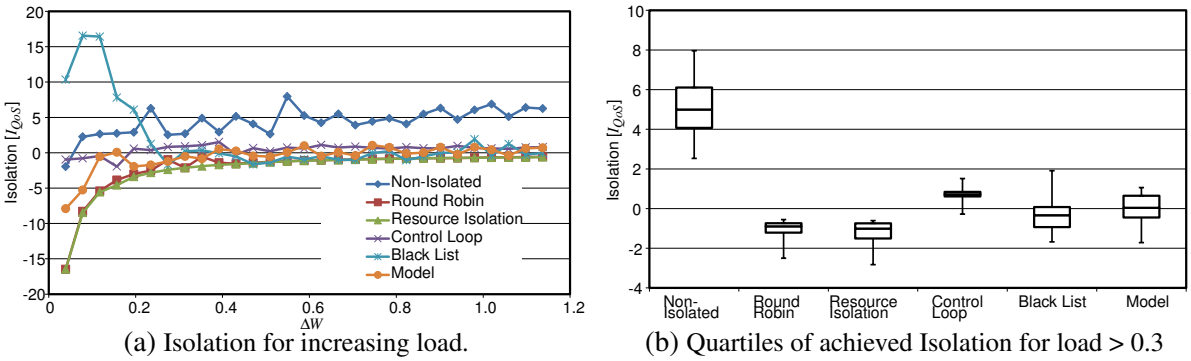
below the guaranteed 3s, which resulted in an isolation value short below 0. The response times increased for all tenants, and thus I_{QoS} increased to a value around 5.

Round Robin and Resource Isolation Round robin is a static approach. Therefore, the response times for the abiding tenants did not change at all. Although the resource isolation method uses knowledge about the system, it does not actively observe the achieved response times. Therefore, both methods provide a similar behavior as long as the requests resource demands are the same. This was the case in the given scenario. The response times were too short for t_1, and thus the average response time of the abiding tenants. This resulted in a negative isolation, which means that the quality observed was better as the guarantee. The difference maintained a steady value, while the workload increased. Consequently, the isolation slowly converged to 0. Although the data has a clear tendency, the corresponding box plot helps to estimate the quality at higher loads. In summary, the round robin and resource isolation methods will always maintain a negative value. Both describe the lower border line of the measurements, since they are not adaptive with regard to the guarantees and observed performance.

Closed Control Loop The closed control loop immediately started with an I_{QoS} short below 0. This emphasizes its ability to shift resources from the abiding tenants to the disruptive. However, while the load increased, the response times increased slightly for the abiding tenants. As a result, the negative isolation value increased from around -0.15 to a value around 0.9. These values are very similar to those observed in Figure 6.7, because the observed quality at the reference workload was already close to that one guaranteed.

Black List The black list begins with an I_{QoS} around 15. Then it converged fast to a value close to 0. This is similar to the observations in Figure 6.8. At the beginning, the disruptive tenant did not exceed the guarantee, and therefore it was not black listed. Therefore, the response times were very high for all tenants. However, in this figure, it becomes clear, that ones the disruptive tenant is black listed, the observed response times seem to be close to the guarantee. In average, this resulted in an isolation I_{QoS} of -0.1. The second and third quartile is between -1.1 and 0.1. The overall range of results is between -1.9 and 2.

Model The model based approach started with I_{QoS} at -8.3, converging fast to 0 at $\Delta w = 0.35$. This behavior is based on the utility function, optimizing the weights. At the beginning, the disruptive tenant did not exceed its guarantee so much. Therefore, there was not much need to shift priorities from the abiding tenants to the disruptive. The process of shifting resources became stronger with increasing load. Nevertheless, the isolation was maintained with regard to the guarantee. Additionally, it can be seen, that the model based approach converges much faster and closer to 0 as the resource control and round robin method. This shows an advantage of an active optimization. The model based method ended up with an average isolation of $I_{QoS} = 0$, while the second and third quartile was within 0.7 and -0.3. The first and fourth quartile showed a trend to the better



(a) Isolation for increasing load.

(b) Quartiles of achieved Isolation for load > 0.3

Figure 6.10.: I_{QoS} based on the guarantee.

isolation. Although the isolation quality of the model and closed control loop look very similar in the Figure 6.7 and Figure 6.9, they are different. The model based isolation method started with lower response times for the abiding tenants, which increased rather fast, due to the optimization, as it can be seen in Figure 6.10a. Therefore, the previous results are different, compared to those in this analysis.

6.1.2.5. Resulting Isolation Based on Workload Ratios

The results described in the following are based *Workload I* and contribute to **M1.1.1**. The used metrics are based on the workload ratios (cf. Section 3.1.4). Only the control loop based approach would have significantly decreasing abiding workload, if the guarantee is considered (cf. results from Section 6.1.2.4). Consequently, the response time observed at the reference workload was considered as the reference.

Figure 6.11 shows the results for the integral based metric over the whole range of ΔW analytically derived (cf. Section 3.5.1). The size of the y-axis was limited to emphasize the differences within the relevant areas. At low disruptive loads, negative values occurred for the blacklist isolation method and the resource isolation. The negative values origin from minor fluctuations in the response times, resulting in isolation values even worse than non-isolated systems. Minor fluctuations at the beginning have a very strong impact upon the workload ratio based metrics. This is caused by the small difference between the two curves for a low ΔW (cf. Figure 3.3). However, the values achieved a positive value fairly fast.

Table 6.2 contains the results for I_{base} (cf. Section 3.1.4.1). W_{base} would correspond to $\Delta W \approx 0.529$. This relevant point was not covered by a direct measurement. Therefore, the results for $W_{a_{base}}$ were calculated by a linear interpolation of the next lower $\Delta W \approx 0.509$ and next higher $\Delta W \approx 5.490$. Due to the high isolation qualities, the I_{end} metric was not considered in the evaluation. While adjusting the amount of abiding workload, situations appeared, in which the abiding load could have been larger as the one of the reference measurement. This was primarily due to variance in the measurements. For these situations, the abiding load from W_{ref} was considered for the calculation of the isolation.

At a glance, all methods finally achieved a value for $I_{intFree} > 0.5$ and $I_{base} > 0.6$. Subsequently the result for each approach is discussed on an individual basis.

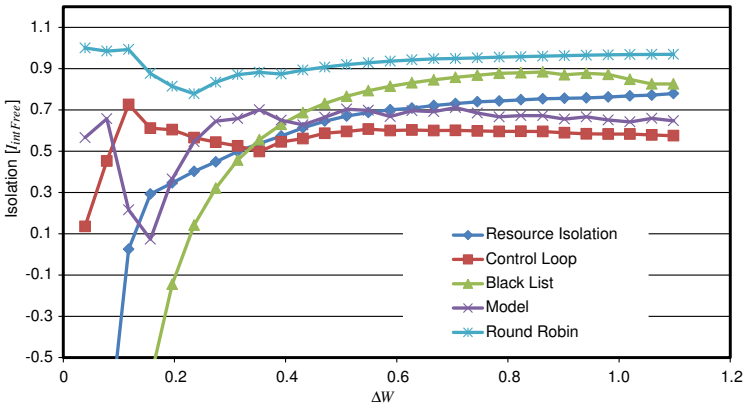


Figure 6.11.: $I_{intFree}$ based on the observed response time at the reference workload.

Resource Isolation	Control Loop	Black List	Model	Round Robin
0.8	0.65	0.98	0.78	0.97

Table 6.2.: Evaluation results for the I_{base} metric.

Resource Isolation The resource isolation started with a negative value of -2.0 for $I_{intFree}$ and increased fast to a positive value. It became steady at a level short above 0.7. The result for $I_{base} = 0.8$, which was above the integrals ratio at this point. This shows, that the negative values at the beginning have a high impact upon the overall result. With regard to $I_{intFree}$ at the highest ΔW, the method is the third best.

Closed Control Loop The closed control loop started with a low isolation value of 0.13, reached a short top of 0.72 at $\Delta W \approx 0.12$ and than became steady around 0.6. The result of I_{base} is similar to the final value 0.57 for $I_{intFree}$. With regard to the overall isolation capabilities, the closed control loop is performing worst in the given scenario. Which corresponds to the observations in Section 6.1.2.4.

Black List The black list started with a value of -3.31 and converged as slowest of all methods. The best isolation was observed at $\Delta W \approx 0.86$ with a value of 0.88. The value for $I_{base} \approx 0.98$ and had the highest difference to $I_{intFree}$. This stems from the high negative values of the method at the beginning. This shows, that the method can provide good isolation for high disruptive loads. For the given configuration, it had a weak isolation for low ΔW. The reason was the configured safety margin. The end value for $I_{intFree} \approx 0.83$ and the second best.

Model With an $I_{intFree} \approx 0.57$, it is the second best method for the first measurement. In the experiments conducted, the isolation worsened at the beginning, before it started to increase again and maintaining a value of approximately 0.65 around $\Delta W \approx 0.25$. The final value was 0.65. The observed $I_{base} \approx 0.78$ was also better, as the overall value of $I_{intFree}$. With regard to I_{base}, the method has a value comparable with the resource isolation. The closed control loop is outperformed by 20%. The low $I_{intFree}$ stemmed from the problems for a low ΔW. This can be explained by the chosen fitness function configuration.

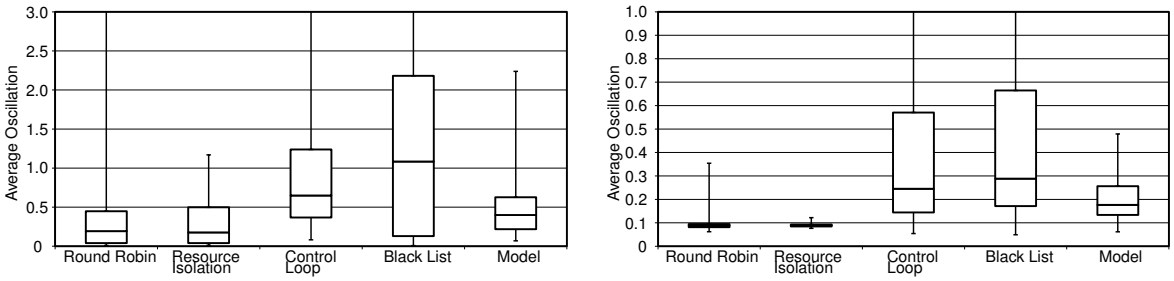
Round Robin Starting with an isolation value of 1, round robin roughly maintained the isolation quality, except a minor decrease between $\Delta W \in [0.1, 0.2]$. The $I_{intFree}$ at the end of the measurement was approximately 0.97. Overall, the $I_{intFree}$ was always better as for other methods. The I_{base} value was 0.97. This is only 0.01 lower as the black list's value.

6.1.2.6. Resulting Oscillation

The following results are based on *Workload I*. For the evaluation of the oscillation, the considerations from Section 3.1.5.2 come to bear. Thus, this section contributes to **M1.1.3**. Figure 6.12 depicts the oscillation over the whole range of measurements of all tenants. As denominator, the arithmetic mean was conducted (cf. Section 3.1.5.2). For Figure 6.12a, the response time was investigated, while Figure 6.12b presents the details on the throughput. As expected, the results are similar in both cases. The y-axis is limited, to emphasize the relevant values range.

The round robin and the resource isolation method had the lowest dynamic in their behavior. Thus, the oscillation is considered to be low, which is also shown in the diagrams. Since these two methods do not much adapt their preferences for one tenant, their value can be seen as the borderline. The closed control loop and the black list had a strong variability. The response time's median for the closed control loop was 0.65 and for the throughput 0.25. For the black list, the corresponding values were 1.08 and 0.28. Additionally, the general trend argues against the black list, which exceeds the maximum value of the closed control loop by 307% for the response times. The model based isolation method also provides a low oscillation over a wide range, and is clearly better than the closed control loop and the black list approach. Nevertheless, it still has more fluctuations than the round robin, and the resource isolation mechanism.

For the black list, the oscillation of the response times reached a median of 1.08. This means, that a high portion of the observed response times perceived by the tenants significantly varied. For an abiding tenant with a guaranteed response time of 3s, this results in a variable time between 1.5s and 4.5s for 50% of the requests. Additional 25% exceeded 4.5s. Thus, several requests are above an acceptable value of 4s [Jupiterresearch and Akamai, 2006], although the average response time was good. This shows the importance of the metric.



(a) Oscillation of response time.

(b) Oscillation of throughput

Figure 6.12.: Oscillation of response time and throughput.

6.1.2.7. Resulting Performance Gap for Abiding Tenants

The following results are based on *Workload I* and provide insight on **M1.2.1**. The guaranteed response time for the tenants was set to 3s. Tenants generating less load than their quota, would have faster response times when using completely isolated resources. However, to fully use the benefits of shared systems, the isolation methods should foster the reduction of a tenant's share, if it requires less resources. This allows to maintain the guarantee of tenants exceeding their quota at the same time. This capability was one of the mentioned design goals (cf. Section 4).

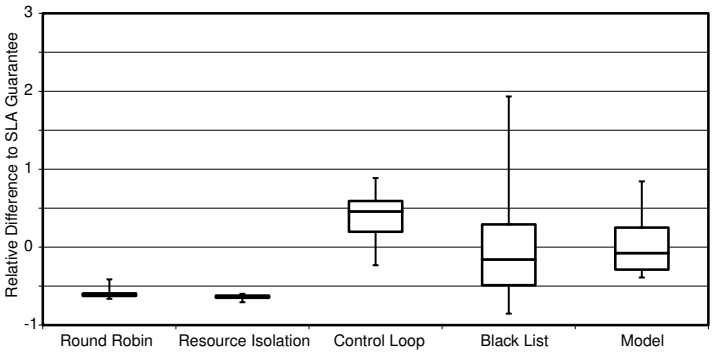


Figure 6.13.: Relative difference of the average response from the abiding tenants to the guaranteed one.

The relative, average difference, of all response times, from the abiding tenants' guarantee was calculated. The box plots in Figure 6.13 summarize the results.

Round robin and the resource isolation method are not able to reflect any response time oriented guarantee. Consequently, both showed a widely comparable behavior. As a result of the low oscillation, these values were also very stable. Their median was at -0.61 for the round robin, and -0.64 for the resource isolation. Thus, resources were spent for the abiding tenants, although their response times were very good. The closed control loop based method had a median of 0.45. The positive value stems from the bad isolation of the method. It shifts resources from the abiding tenants to the disruptive, but was not able to ensure a proper isolation. The black lists median was at -0.16 and the mean with 0.06 noticeably higher. The model based approach with a median of -0.08 and a mean of 0.05 has a negligible small difference. However, the blacklist achieves this with a much higher deviation and some outliers.

6.1.2.8. Resulting Settling Times

The following results on settling times (cf. Section 3.1.5.1) are based on the *Workload II*. Thus, this section provides the results for **M1.1.2** and **M1.2.2**. The average value, to identify the end event was computed from the samples collected between $n + 60s$ and $n + 180s$. For the non-isolated case, it does not make sense to measure the settling time. Round robin and the resource isolation method do not utilize a direct feedback concerning the tenants' performance. Consequently, the admission control keeps its behavior independent of the load. Therefore, the settling times can be assumed to be 0. Table 6.3 summarizes the other isolation methods. The results of the increasing and decreasing load scenarios are depicted.

	Control Loop	**Black List**	**Model**
Increasing	130s	128s	52s
Decreasing	206	13s	43s

Table 6.3.: Settling times for the isolation methods.

The model based isolation method performs best. The settling times were less than 1 minute. Followed by the closed control loop with 130s in the increasing, and 206s in the decreasing scenario. For both methods, the measurements delivered repeatable results. The fast reaction of the model based approach is in line with the expectation. The optimization adapts the weights every 30s, if the systems underlying characteristics have changed. However, once the system is adapted, no reconfiguration is made. This is different for the closed control loop, which needs several iterations.

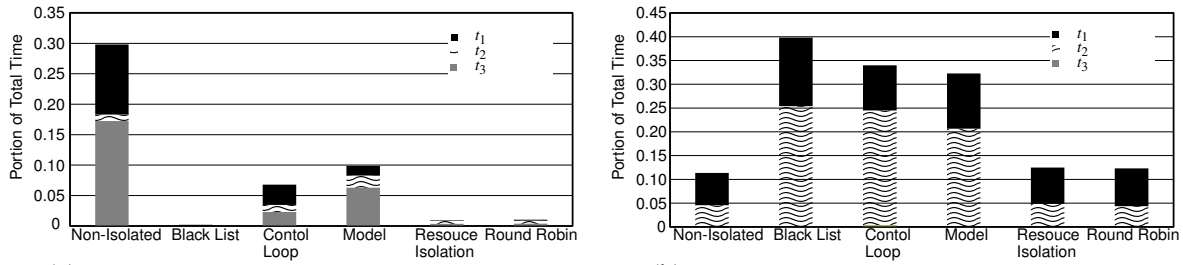
In case of the black list method, the results had a high variance. For the decreasing scenario, no significant pattern stands out of the noise. In view of the chosen workload, and isolation method, this is comprehensible. Even with a reduction of the load from the abiding tenants, the disruptive tenant is not able to use these resources. The high number of users always leads to request rates exceeding the quota, and therefore the disruptive tenant has been black listed always. Therefore, the results for the black list should be treated with caution.

6.1.2.9. Results Based on Dynamic Load

The subsequent results are based on the dynamic load described by the *Workload III*. That delivers the results for **M1.1.4** and **M1.1.2**. Due to the continually changing load, it is not possible to determine the settling times and the oscillation. The same holds for the isolation metrics. Therefore, the amount of time is observed, in which:

1. A tenant has less users as allowed, but observes response times longer than the guarantee. This measures the isolation capabilities.

2. A tenant exceeds its quota, but has faster or equal response times as the guarantee defines. This expresses the methods capabilities to efficiently share the system.



(a) Guarantee violation without quota violation.

(b) Achieved guarantee despite quota violation.

Figure 6.14.: Results for the dynamic load scenario.

Figure 6.14a/b depict the results. The bar diagram in Figure 6.14a, shows the fraction of time in which situation 1 appeared. As a basis, the mean of 5s intervals is used. The system's load originated from three tenants. The total amount of time three tenants spend on the system, is three times the time a single tenant uses it. This results in 3x24h for a complete day. Consequently, the reference value for the computation of the bar's overall size is 72 hours. If one is interested in the value for one single tenant, the bar size has to be multiplied by three. The same holds for Figure 6.14b. It presents the fraction of time where situation 2 appeared.

The black list, resource isolation and round robin provide all an almost perfect isolation. Compared to a non-isolated system, the two approaches, providing an active adaptation of scheduling weights, are still significantly better. The closed control loop based approach outperforms the model based approach.

The advantages of the control loop and model based approach become visible in Figure 6.14b. Both outperform the resource isolation and round robin. It is worth mentioning, that the value of the black list method is not highly representative. Having a detailed look into the data shows, that t_2 is black listed almost the whole time. This resulted in fast response times, but without any useful content.

For the experiment, the configuration of the isolation methods was the same as for the first workload description. The used configuration for the model based approach (c_v, c_p) needs a certain degree of guarantee violation before the optimization becomes a significant counterpart (cf. Section 6.1.2.3). The change of load in this experiment was much less as in the previous. Therefore, the model based approach performs worse as the closed control loop, which already showed good results for low violation scenarios before.

In general, one can observe that t_2 is seldom the reason for guarantee violations. The rationale behind is its load behavior. Tenant 2 exceeds its quota almost all time. Therefore, it cannot be responsible for any guarantee violations. Tenant 3 is almost the whole duration below the quota, while t_1 exceeds it sometimes. This is clearly reflected in the results depicted in Figure 6.14a. In Figure 6.14b t_3 never appears. This is due to the fact that t_3 never exceeds its quota and thus cannot benefit from the implemented dynamic shift of resources. Accordingly, t_2 who exceeds quota the most time, benefits the most.

6.1.3. Core Components Overhead

Subsequently, the system landscape for the evaluation of the methods impact upon the performance of the application is presented (**M1.3.1**), together with the corresponding results.

6.1.3.1. Experiment Setup

As outlined in Section 5.3, the functionality of several components is required to run a concrete implementation of an isolation method. The PIF implementation calls only the relevant components for a given method. Thus, the functionality provided by the framework for a particular method has to be seen as part of a method, because others methods do not require it. Therefore, it does not make sense to evaluate the performance of the policy generator isolated.

The following parameters are of relevance: the number of tenants, the number of request types and the number of application nodes. The information sent by the diverse monitors is aggregated and thus the number of requests is not important.

To generate the relevant data, in a realistic scenario, the probes were mocked to send artificial monitoring information. Thus, it was possible to simulate the number of tenants, requests and application servers. The admission control was emulated as well.

The framework's configuration was the same as described in Section 6.1.2.2. The framework was executed within the SAP JRE 1.7 on a SLES11 SP2 installation. The hardware was configured to use 1x1.6GHz CPU and 16GiB memory. A MySQL version 5.1 database provided persistence for the isolation framework. It was deployed together with the framework on the same server. The emulated parts of the load generation were deployed on a separate host.

In a first experiment series, two parameters were modified. The number of tenants was varied between 1 and 101. The number of request types was varied between 1 and 101 for each tenant. The number of tenants and request types was increased by a step width of 10. Each load configuration was held for 20 min. The first 10 min were the warm-up phase, for which no data was gathered. To prove the linearly increasing demands, for an increasing number of application nodes, a second experiment series was conducted.

6.1.3.2. Results

The closed control loop, the resource isolation, black list and the model based method were evaluated. Since the round robin method not necessarily requires the core parts of the framework, it was not subject to the experiments. During the observation time, the total processing time of the java and MySQL process was observed. The accumulated value was divided by the number of executed strategy generations, which were triggered every 30s.

For the closed control loop, black list and the model based method, a linear relationship between the number of request types m, the number of tenants n and the processing time was observed. A function of the form $f_{procTime} : c_1 + m \cdot n \cdot c_2$ predicts the real observed values with an accuracy between 4.9% and 5.0% error for all three approaches. Thus c_1 and c_2 contain enough information to

compare these methods. The resource isolation based method did not show such a linear characteristic and is discussed separately. For the black list method $c_1 = 24$ ms;$c_2 = 0.5066$ was observed, for the control loop $c_1 = 27$ ms;$c_2 = 0.5059$, for the model $c_1 = 22$ ms;$c_2 = 0.5113$. The results show, that the approaches are not significantly different. The constant values are negligible. For 101 tenants and 101 request types this result in 5188 ms for the control loop, which is the fastest. With 5238 ms the model based isolation method is the slowest. In fact, the differences are not significant. However, the resource isolation based solution has $c_1 = 26; c_2 = 0.6043$, which shows a clear difference. Compared to the average $c_2 = 0.5079$ from the other isolation methods, this is $\approx 19\%$ worse. Moreover, the error of $f_{procTime}$ was with 15.8% significantly higher. Looking at Figure 6.15 one can see that the results for 101 tenants and request types are significantly higher as those of the other approaches. Furthermore, a light, super linear characteristic of each curve is visible.

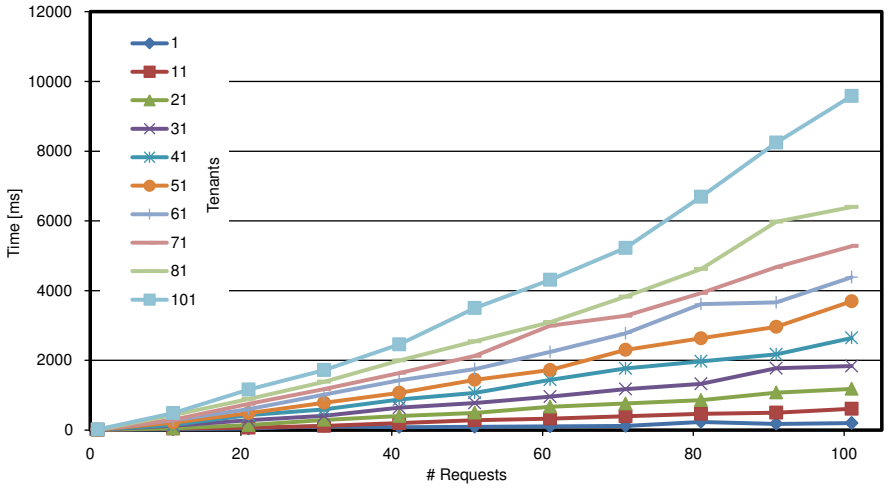


Figure 6.15.: Processing time of resource isolation method for different number of tenants.

Concerning the number of application nodes, random samples showed the expected relationship of a linear increase by the number of application servers. It is worth mentioning, that the policies were optimized using only the data of the server they were thought for. Thus, no overall optimization was triggered. The memory allocated by the java virtual machine varied between 250MiB and 320MiB, while the database server allocated around 700MiB. A relation to the induced load was not observable.

In summary, except the resource isolation method, all approaches have almost the same performance and scale linearly. Within the observed boundaries, the resource isolation method performs 19% worse.

6.1.4. Admission Control Strategy Overhead

Subsequently, the system landscape and experiments for the evaluation of the impact upon the request processing (**M1.3.1**) is presented. Furthermore, the corresponding results are discussed.

6.1.4.1. Experimental Setup

The expected overhead of the admission control's strategy is low. Therefore, a significant difference between the application induced load is hardly observable. Moreover, the time required by the actual admission control strategy should be analyzed independent of other aspects, which are commonly used technologies and often integrated into the system anyway. These aspects can be the overhead induced by monitoring, the limitation of the thread pool size, and adding a valve to the system. Therefore, a setup was conducted that captured only the overhead of the admission control strategy. To achieve this, the strategy was instantiated outside an application runtime container, by a mocked proxy/execution point. The mocked execution point realized the following tasks. It generated request objects with a random selection of request types and tenants. Furthermore, it provided all necessary information like a suitable policy (e.g., providing a random selection of resource demands) and iteratively added and removed requests to/from the strategy. The experiments were executed on a SAP JRE 1.7, installed on SLES11 SP2. The hardware was configured to use 1x1.6GHz CPU and 16GiB memory.

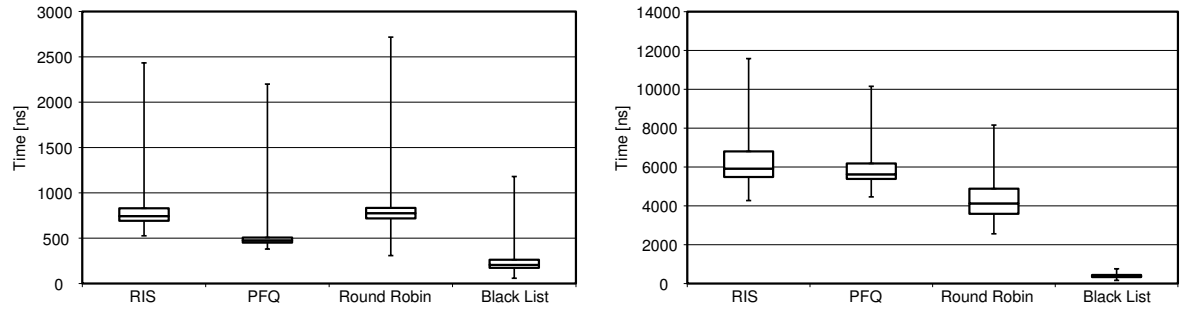
Three parameters were modified in the experiment series. The number of tenants was varied between 1-501. The number of request types between 1-501 for each tenant, whereby the resource demand was different for each request type. The total number of requests enqueued varied between the minimum value, given by the number of tenants of a particular experiment, and 10001. The number of requests was homogeneously distributed among all tenants. The number of tenants was increased by a step width of 50, the number of enqueued requests by 500 each step and the request types by 50 per step. In case of the black list strategy, 5% of the tenants were black listed. All requests were added to the strategy in advance. After a waiting time of 2 min, requests were removed and added iteratively, with a 5 ms delay between each call. The time difference was measured using the java *System.nanoTime()* method. The measurement was stopped after 2000 samples for each method.

6.1.4.2. Results

The results gathered did not show any significant change based on the varied parameters. Therefore, a discussion about the influence of the various parameters is discarded. Instead, a box plot (cf. Figure 6.16a/b) that summarizes the results gathered among all configurations is present.

Figure 6.16a shows the quartiles for the time to add a request to the strategy. The median of the RIS, PFQ and round robin are comparable. The black list queue performs best. The difference in the median between the black list (205 ns) and the round robin (776 ns) is the largest. Nevertheless, the processing time spent for the task of adding a request to the strategy is negligible for interactive web applications.

Figure 6.16b depicts the quartiles for the time required to retrieve the next request from the strategy. Again, the RIS, PFQ and round robin show similar results. However, in the measurements conducted, the results of the round robin are roughly 30% better, comparing the medians. The black list requires only 6% of the time the RIS needs, and 9% of the time the round robin needs. However,



(a) Time to add an isolated request.

(b) Time to get an isolated request.

Figure 6.16.: Resource consumption of the strategy.

with a maximum median delay of 5918 ns, even the slowest of the three approaches should not significantly influence the overall response times.

In the scenario from Section 6.1.2, the system was configured to provide a throughput of 240 requests/s at 24 threads. Thus a request was processed by the MTA in $1/10s$. For each processed request, 6661 ns were spent in the worst case. This result in $\approx 0.007\%$ of the overall time for processing.

In summary, an observable difference between the isolation strategy's performance exists. However, they all perform fast and the impact upon the overall performance is negligible for almost all web applications.

6.1.5. Concluding the Isolation Methods Experiments

In this subsection, the results from the previous experiments are summarized and interpreted to answer the evaluation questions. This section does not recapitulate all the details from before, but focuses on the key messages using only the most important results.

6.1.5.1. Outcomes

Isolation Qualities The results shown in the Sections 6.1.2.3, 6.1.2.4, 6.1.2.5, 6.1.2.9 help to answer **Q1.1**, which concerns the quality of the isolation with regard to metric **M1.1.1** and **M1.1.4**. The Section 6.1.2.8 answers the question concerning the settling time (**M1.1.2**) while Section 6.1.2.6 discusses the oscillation of a particular approach, which refers to **M1.1.3**.

Table 6.4 shows the overall results gathered, to answer the questions about the isolation capabilities. For an overview of the value ranges and the corresponding interpretation of the isolation metrics please refer to Section 3.5.2. Where relevant, the disruptive load of the measurement and type of the performance reference values is added.

The results show, that all methods have always a better isolation as a non-isolated system. The resource isolation, round robin and black list converge to a perfect isolation with increasing disruptive load, while the model based approach and the closed control loop maintain an $I_{QoS} \approx 1$, when compared to the reference loads response time.

Metric	Non-Isolated	Round Robin	Resource Isol.	Control Loop	Black List	Model
I_{avg} ($\Delta W = 1.1$/reference)	5.78	0.07	0.65	0.91	1.13	0.90
I_{QoS} ($\Delta W = 1.1$/reference)	7.27	0.03	0.12	1	0.08	1.02
I_{avg} ($\Delta W = 1.1$/guarantee)	4.39	-2.14	-2.23	0.46	1.75	-0.53[a]
I_{QoS} ($\Delta W = 1.1$/guarantee)	6.25	-0.57	-0.61	0.8	-0.05	0.09
$I_{intFree}$ ($\Delta W = 1.1$/reference)	0.00	0.97	0.78	0.57	0.83	0.65
I_{base}	0.00	0.97	0.80	0.65	0.98	0.78
Violations (Dynamic Load)	0.29	0.01	0.01	0.07	0.00	0.10
Settling Time [s]	N/A	N/A	N/A	130	128[b]	52
Oscillation	N/A	0.19	0.18	0.65	1.08	0.40

[a] Average for $\Delta W > 0.2 \approx 0.0$.
[b] Not necessarily representative (cf. Section 6.1.2.8).

Table 6.4.: Overview of the results for the isolation capabilities of various methods.

The metrics, using the guaranteed response time as a reference, also show the ability to outperform a non-isolated system. In these experiments, round robin and the resource isolation have a negative isolation. The black list has a large difference between I_{QoS} and I_{avg}, which stems from the high errors at the beginning.

A difference between the isolation values calculated for the guarantee and for the observed response times at the reference workloads exits. For the round robin and the resource isolation this is higher as for the methods to actively adapting the admission control. This is a sign that the closed control loop, black list and the model based isolation have an influence upon the observed response times at the reference workload.

In general one can observe, that the approaches with fixed scheduling weights provide a better isolation. Furthermore, they do not rely on a direct feedback from the system and therefore do not have any settling times when the load changes. In the group of methods, where the admission control is directly adapted by the asynchronous calls, the model based approach is the fastest and it has the lowest oscillation within this group. Round robin and the resource isolation method have the lowest oscillation values among all.

The results gathered from the experiments show the isolation capabilities of the methods. Thus, they answer the relevant question concerning the quality of the isolation. In general, the round robin and resource isolation provide the best isolation, followed by the other approaches sharing similar capabilities. If the tenants have significantly different request type ratios, or different demands per request type, the resource isolating approach is advantageous as shown by the results from Section 4.5.2.

Efficiency of Isolation Methods **Q1.2** is answered by the experiments in Section 6.1.2.7, which tackle the metric **M1.2.1** and Section 6.1.2.8 for **M.1.2.2**. The round robin's and the resource

isolation's average abiding tenant response times are far below the guarantee. Thus, resources that can be used by the disruptive tenants are not used. The methods which actively control the scheduling weights, and being aware of guarantee and/or quota, perform better. The closed control loop exceeds the guarantee for the abiding tenants by around 45%, whereby the black list is 16% below, and the model based approach almost perfectly achieves the optimum. For situations where the load decreases, and thus resources can be rescheduled to achieve a better distribution, the closed control loop performs worst. It has a delay of more than 3 min. The model based approach achieves a value of 43s. The black lists settling time was measured with 13s. However, the black list cannot really leverage resources which become dynamically free from other tenants. This is caused by its limited view onto the input information only.

In summary, the model based approach provides the most efficient solution, followed by the closed control loop, black list and the other approaches.

Scalability The investigations of the scalability are subject to **Q1.3**. The results from Section 6.1.4 and Section 6.1.3.2 can answer this question. The black list performs better than the others, while the others have all the same impact. Overall, the methods impact upon the performance of the application server is negligible for every strategy in use. The time spent to process them, is independent of the number of tenants, request types or amount or requests. This is different for the policy generation. Except the resource isolation based method, all approaches scale linearly with the number of tenants, request types and application nodes. Furthermore, all scale by the same linear member. The resource isolation method scales slightly super linear, performing 19% worse for the configurations investigated.

Thus, all approaches are equally good, except the resource isolation which provides less scalability.

Using Resource Demand Information The evaluation goal **G2** concerns the accuracy of the resource isolation method as presented in Section 4.3. The actual results for the metrics investigated are presented in Section 4.5.2. Herewith, a brief summary of the results is given.

Q2.1 focuses on the preconditions and technical foundation for the resource isolation method, namely the quality of the resource demand estimation. The presented resource demand estimation shows a high accuracy for the relevant, high load situations, where one major bottleneck exists. Tenants, with higher demands, are also estimated with higher demands by the method (cf. Section 4.5.2.2).

Concerning **Q2.2**, it is shown that resource isolation, and consequently performance isolation, is achieved. The isolation is maintained, if tenants have the same or different demands per request. The ability to provide a service quality differentiation (**Q2.3**), is also discussed in the respective section. The method is able to adapt to changing resource demands per request type (**Q2.4**). The time it takes depends on the chosen configuration of the resource demand estimator. However, a time frame of several minutes is much more likely as a duration measured in seconds.

The method provides an acceptable degree of isolation (**Q2.5**) for scenarios where the database is deployed on a separate node (cf. Section 6.1.2) and in scenarios where the database is deployed together with the application (cf. Section 4.5.2.1). However, application level SLAs are not reflected.

Using a Model to Optimize Scheduling Weights The **G3** of the evaluation is to gain insights on how accurate the model based isolation method can maintain isolation and an efficient distribution of shares. The metrics and results for the questions are presented in Section 4.5.3, together with the explanation of the method. This paragraph gives a short overview of the results.

The basis for the optimization is the model. Thus the accuracy **Q3.1** of it is a relevant metric. Section 4.5.3 shows that for high load scenarios accurate results for the response times and the throughput can be obtained. The error margin is consistently below 20%. According to [Menasce and Virgilio, 2000] these are acceptable values.

Q3.2 is answered by the plots in Section 4.5.3.1. The optimal value of the configuration parameter, depends on the given system characteristics and scenario. However, a rough indication saying that c_v should be between 0.1 and 1.0 and c_p between 1 and 10 is possible.

The method showed an acceptable isolation (**Q3.3**) for scenarios using a virtual environment (cf. Section 6.1.2). The same holds for non-virtualized environments, with a database deployed together with the application server or separately (cf. Section 4.5.3.2). It can reflect application level oriented SLAs. However, depending on the configuration parameters a trade-off between over-commitment and isolation must be found.

6.1.5.2. Critical Discussion

This subsection introduces some threats that might concern the validity of the experiments presented in the previous sections.

Limited Amount of Experiment Configuration In the experiments, only a few tenants were used. If a high number of tenants are deployed on the system, the differences in the observed performance are less for each tenant. Thus, it is more complex to analyze the relevant patterns. Furthermore, although an MTA might be subscribed by several tenants, it is likely that only few produce a significant load. In general, the number of potential configurations for an experiment depends on many factors. The two most relevant are the number of tenants, and the amount of workload each of them is generating. It is not possible to test all combinations. However, the general insights gathered in the experiments are representative for other configurations as well. A round robin and black list will provide the same characteristics independent of the number of tenants or configurations. It can be assumed that it will always provide a very good isolation. The capabilities of the resource isolation methods depend on the way the resource demands are determined. If a resource demand estimation is used, the quality of the isolation will be limited for a high number of tenants or request types. An estimation of these limitations is possible by the experiments conducted in Section 4.5.2.2. If the resource demands can be exactly determined, the

isolation behavior is the same for a high number of tenants and request types. The model based approaches optimization always selects the global optimum (cf. Section 4.2.6.3), independent of the number of tenants. The general capabilities of the closed control loop to maintain a system isolated is neither affected by the number of tenants, nor the distribution of workloads. However, if the number of tenants increase, the share they get from a system is less. Thus, the reactions of the isolation method or the system onto the same absolute size of changing load per-tenant is stronger. Therefore, oscillation and settling times may increase.

Selection of Workloads The experiments to measure the isolation in the evaluation chapter conducted only one load configuration. As outlined in Section 6.1.2.2, this configuration covered the relevant aspects. Furthermore, in Section 4.5 additional experiments with other configurations were described. Additionally, the dynamic load scenario also reflected a numerous amount of workload combinations. All experiments showed consistent results concerning the overall conclusions.

Configuration of Isolation Methods The configuration of the methods was based on the observations of only one load profile. Better results might be possible, if the configuration is optimized for the whole measurement space. This might be possible for an artificially benchmarked system, where all relevant load combinations are known. Evaluating the weights for every possible situation in a realistic scenario is not possible. It is reasonable that MTA providers will optimize their configuration for a selected subset of situations. Consequently, the methods were configured with the same quality as it is expected from an MTA provider.

6.1.5.3. Summary of the Isolation Methods Evaluation

Several experiments in different system environments were executed, in order to evaluate the performance isolation methods. Some experiments rather focus on the internal behavior and functionality of the novel methods. These experiment results were primarily part of Section 4.5. The experiment results of an overall comparison were presented in this chapter. For evaluating the isolation capabilities, the MTTPC-W was used and the metrics from Chapter 3 were applied. To understand the characteristics of the methods, a representative MTA was emulated, in an environment motivated by a real world example. The results of these measurements were extensively discussed. Within the summary section, the results from all relevant experiments were collected and interpreted to find a ranking of the methods. Furthermore, it discussed the validity and representativeness of the experiments.

All novel methods, developed in the thesis, can increase performance isolation. They do it by either reflecting the resources used by a tenant, or using an analytical model together with a utility function. These methods outperform simple black list or round robin methods by lower oscillation or a more efficient sharing of the resources. The latter is required to enable over-commitment. However, in case a very reliable isolation is required, a round robin based request scheduling is still a relevant option. Moreover, it was shown that the analytic description, used in the model

based isolation method, provides good predictions of the performance. Furthermore, the resource demand estimation approach, based on the service demand law, can provide appropriate estimations for MTAs.

A significant difference between the methods was visible with respect to the isolation capabilities, efficiency of sharing the system, and maintaining a performance with low oscillation and stable values.

It is possible to cluster the isolation methods in two large groups. Those using admission controls, where the policies directly influence the admission of a tenants request (i.e., black list, model based, closed control loop), and those which do not have a policy at all, or providing information used within the strategy, to dynamically adjust the priority (i.e., resource isolation, round robin). The latter have the lowest oscillation, while the model based, closed control loop and black list behave in ascending order. However, the first group is able to share the system more efficient, and thus enables over-commitment, considering application level guarantees. Both approaches from the second group do not provide this, or only if one tenant does not have any request in the queue. The resource isolation and round robin do not face the problem of settling times. However, this can hardly be seen as an advantage, since this means they do not adapt at all. They outperform the other methods concerning the isolation. More details can be found in the previous Section 6.1.5.1.

It is worth to compare these results with the experiments from Section 3.4.2, which evaluated the performance isolation capabilities of the hypervisor Xen. In the experiments of the admission control based isolation, the database was deployed separately. Due to the multi-tenancy, the tenants were able to dynamically use resources not used by other tenants. The most similar Xen deployment to be compared with, is the unpinned scenario where the database was deployed in a separate domain. In this scenario, the achieved isolation was $I_{avg} \approx 4.20$. The methods described in this thesis were able to outperform this. Using the reference workloads observed performance, the method with the lowest isolation achieved an $I_{avg} \approx 1.13$ (cf. Section 6.4). It is the same case for $I_{base} \approx 0.36$ vs. $I_{base} \approx 0.65$ and similarly for the other metrics. As a matter of fairness, however, it has to be mentioned that in case of the hypervisor, the workload of all abiding tenants had already been close to their maximum. In the multi-tenancy scenario, one tenant was far below its maximum. Since the metrics used for comparison reflect the measured performance quality at the reference workload, this should not weigh much.

6.2. Measurement of Isolation

In Chapter 3, metrics and tools to quantify the performance isolation of MTA are introduced. These foundations laid the ground for the measurements in Section 6.1. They are essential for the evaluation of the methods and for MTA stakeholders. The contributions for the metrics and measurement were already validated in Section 3.4.

This chapter summarizes the results, and combines them with insights gained in Section 6.1. At first, a complete list of relevant goals, derived questions and metrics is provided. A separate section links the metrics to the corresponding experiments from the previous sections and outlines

the relevant aspects. Finally, this section is concluded. The previous validations did not outline the experiment goals with respect to the overall thesis goals. This is done here and set in the correct context. The oscillation metric and the settling time are not investigated in deep detail, since similar metrics were already described in other contexts. Nevertheless, at the end a brief summary about their applicability in the field of performance isolation is presented.

6.2.1. Goal, Questions and Metrics

Goal (**G4**) aims at evaluating the isolation metrics capability to answer the questions they were defined for in Section 3.1.1. Thus the questions for the evaluation are listed below.

Q4.1 Can the metrics describe isolation in relation to a perfect isolated system and a non-isolated system?

- **M4.1.1** Amount of scenarios a metric is applied to describe a system in relation to an isolated or non-isolated system.

Q4.2 Can the metrics express the impact one tenant has onto another?

- **M4.2.1** Prediction error of the performance observed by the abiding tenants.
- **M4.2.2** Amount of scenarios the metric is applied, to describe the impact.

Q4.3 Can the metrics be used to compare various isolation methods?

- **M4.3.1** The metric corresponds to the isolation of a system.
- **M4.3.2** Amount of scenarios the metrics is applied, to compare systems.

Since the number of scenarios, where a metric was applied to, is investigated, the applicability of the metrics is reflected. It is also worth to investigate the soft properties a metric should fulfill (**G5**) (cf. Section 3.1.1). These were defined in accordance with [Hoecker et al., 1984]. Based on that, the subsequent listed questions can be derived.

Q5.1 Are the metrics objective?

- **M5.1.1** Metrics do not depend on a subjective assessment.

Q5.2 Is the metric reliable?

- **M5.2.1** Metric provides the same results when measured multiple times.
- **M5.2.2** Metric provides a stable value for different disruptive loads.

Q5.3 Is the usage of the metrics economically?

- **M5.3.1** It is possible to derive the metric automated.
- **M5.3.2** Amount of domains in which the metrics can be applied.
- **M5.3.3** Time required to obtain a value for the metric.

Q5.3 Is the metric useful?

M5.3.1 Amount of cases, where the metric provided useful insights.

The sixth evaluation goal (**G6**) aims at evaluating the applicability of the tools for automated measurement (cf. Section 3.2) and the developed benchmark application (MTTPC-W) (cf. Section 3.3). By doing so, it is also shown that the metrics calculation can be independent of a concrete scenario. It also shows, that the knowledge on how to calculate the metrics can be encapsulated, to make the metrics available for users without detailed knowledge of them.

Q6.1 Is the measurement framework applicable?

M6.1.1 Amount of scenarios, it is used.

M6.1.2 Efforts to adapt it to a new scenario.

Q6.2 Does the MTTPC-W provide the basis to gather system's behavior insights?

M6.2.1 Amount of scenarios, where it is used to successfully evaluate a system's behavior.

6.2.2. Reflection on Experiments and Results

This section briefly summarizes the results by answering the various questions. For more details and insights see Chapter 3.

6.2.2.1. Quantifying Isolation

The metrics were applied in a case study, evaluating the performance isolation capabilities of a hypervisor (Section 3.4.2), for three different deployments. They were applied in a simulation based environment with four different isolation mechanisms (Section 3.4.1) and in a comparison of six performance isolation methods. In accordance with **M4.1.1**, the metrics based on the workload ratios were successfully used in all scenarios. Together with the design rationales of the metric, this shows that they can be used to describe a system's isolation in relation to a non-isolated and isolated system (**Q4.1**).

All these scenarios also applied the metrics based on the QoS impact and thus answer **M4.2.2**. When the I_{avg} for a ΔW of 1.1 is applied to predict the response times of the abiding tenants in the experiments from Section 6.1, an overall median error of 24.2% is observed (**M4.2.1**). The accuracy highly depends on the scenario observed. Figure 6.17 shows the quartiles of the prediction errors for different scenarios. Although in case of the non-isolated system and the black list, the median error is very high, and for the other approaches respectively low, it is convincing that this metric allows to indicate the potential impact. Thus the metrics can express the impact one tenant has onto another (**Q4.2**). The isolation value for a concrete ΔW represents the impact even more accurate.

As outlined by the experiments in Section 3.4.3.4, the metrics are able to rank existing systems in correct order, according to their isolation capabilities (**M4.3.1**). Additionally, both case studies and the experiments within the evaluation chapter successfully applied the metrics, to compare various

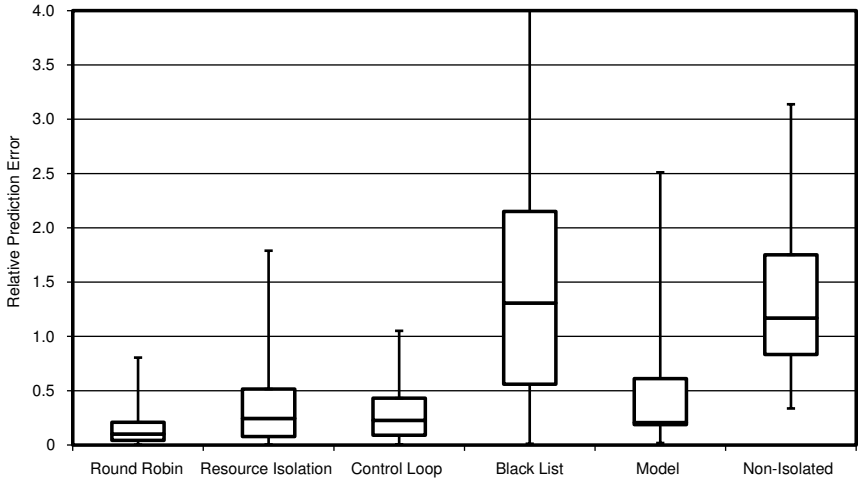


Figure 6.17.: Prediction accuracy of the metrics.

isolation methods, in different system environments (**M4.3.2**). Consequently, the metrics can be used to compare various isolation methods (**Q4.3**).

I_{end} was seldom used in the scenarios considered, since the observed isolation was rather good. Therefore, the use of I_{end} seems to be limited to scenarios with low isolation.

6.2.2.2. Applicability of the Metrics

The isolation metrics have a clear definition, but how the metric is applied to a particular situation can influence the results. For example, the selected reference workload or the step width to adjust the load have a significant influence. Nevertheless, this is true for almost all performance related metrics. The isolation metrics can be automatically calculated, as shown by the solution developed in Section 3.2. This means, that the metrics are not influenced by a human observer, and thus they are objective (**Q5.1**).

If the measurement process receives exactly the same results from the SUT in two different experiment series, the metrics derived will provide the same isolation value (**M5.2.1**). However, as the measurement of performance is usually subject to random processes, it is unlikely to observe exactly the same results two times. It can be observed, that the isolation metrics tend to show a stronger change in value for low disruptive loads. This can be explained by the fact that minor, possibly even random, changes in the observed QoS have a stronger influence at the beginning. This is especially true for the metrics not considering an average value, consisting of measurements at different ΔW. For low disruptive loads, the integral based metrics have a very small difference between the two reference bounds. This makes these approaches less robust concerning changes in the underling performance metrics. However, once the disruptive load exceeded a certain value, the isolation value becomes more steady (**M5.2.2**). Overall, the metric can be seen as reliable and reflecting reality with regard to **Q5.2**.

In the context of this thesis, the metrics were already successfully applied in a simulated environment, a hypervisor virtualized system and for a benchmark application (**M5.3.2**). The metrics

were used in these scenarios by using the measurement framework (cf. Section 3.2). This shows, that the knowledge required to compute the metrics can be encapsulated. The framework automatically adapts the workload, and thus reduces human efforts. Furthermore, it is based on the SoPeCo, and leverage its adapter mechanism. This allows to technically easily apply the metrics to other domains (**M5.3.1**). The capability to easily integrate other technical environments into the SoPeCo was already demonstrated by Westermann (e.g., [Westermann, 2013]).

The time it takes to calculate the metrics, depends on the number of experiments, and the time required to obtain a reliable result for each experiment. The latter one depends on the SUT, the first one depends on the experiment series configuration. For the workload ratio based metrics, a simple approach requires $\mathcal{O}(n \cdot m)$ load configuration in the worst case. Here n refers to the maximum of steps for the increasing load and m refers to the maximum steps possible to adjust the abiding load. Obviously, one can improve this by using a binary search to adjust the abiding load, resulting in $\mathcal{O}(n \cdot log(m))$. The I_{QoS} based metric requires only one optional experiment at the reference workload. If the influence should be calculated on an artificially defined guarantee, no additional measurement is required. Thus $\mathcal{O}(1)$ is possible. If an average value along a range of disruptive loads is investigated, the complexity is $\mathcal{O}(n)$. Additionally, it can be necessary to find a good reference workload, which is a task not directly related to the metrics. Although the workload ratio based metrics may require a high number of experiments (**M5.3.3**), the opportunity to automate the measurement, and the wide field of applicability makes these metrics economically usable (**Q5.3**).

The metrics were used as a basis to compare the performance isolation methods developed. Further usage scenarios investigated were based on the hypervisor Xen and a simulated MTA. In all these contexts they provided useful insights (**M5.3.1/Q5.3**)

6.2.2.3. Technical Contributions Concerning the Metrics

The rather technical contributions directly related to the metrics make them even more feasible for an MTA provider. The measurement framework was applied in almost all cases of the thesis, where the isolation was considered (**M6.1.1**). Thereby, the adaption for a concrete scenario was of low overhead. Only the technical adapter for the load driver and monitors had to be implemented. For sure, the efforts are much lower than adapting the load profiles manually (**M6.1.2**). Thus, the measurement framework is applicable (**Q6.1**) and it is shown that the knowledge to measure the isolation can be encapsulated.

The MTTPC-W was applied to evaluate the potential performance related benefits of a multi-tenant system in a related paper [Krebs et al., 2013]. It was also used in this chapter, to compare various isolation methods (cf. Section 6.1.2) and in several other experiments (cf. Section 4.5) focusing on isolation method specific aspects (**M6.2.1**). Consequently, the MTTPC-W already provided several times the basis to gather insights into a system's behavior (**Q6.2**).

6.2.3. Concluding the Isolation Measurement Validation

This section presented the summary of all questions concerning the isolation metrics discussed in this thesis. Thereby, it was shown, how the metrics and the technical contributions were successfully applied. This way, the experiments described in the thesis, can be seen as case study showing the effectiveness and feasibility of the contributions made.

In summary, the metrics can be used to describe a system's isolation in relation to an isolated and non-isolated system. This expresses how much potential they have to improve. They allow to investigate the impact a disruptive tenant has, and they can be used to compare various solutions. The metrics are objective, reliable, economical and useful. The rather technical contributions show, how the metrics can be used by an automated approach. Only I_{end} lacks in practical applicability if a good isolation is provided.

6.3. Performance Isolated Document Service

A "...case study is an empirical method aimed at investigating contemporary phenomena in their context." [Runeson et al., 2012]. In the present case study, the context is defined by the SAP HANA Cloud Document Service and the investigated phenomena are the contributions to enable a developer, to realize performance isolated applications.

The SAP HANA Cloud Document Service provides the opportunity, for applications deployed on the HCP, to persist unstructured documents (cf. Section 1.2). At first, the goals of the case study are outlined. Then a brief overview of the service is provided, followed by a problem statement. After that, the environmental setup used for the case study is explained. In the further course of the section, the decision process from Section 5.2.3 is used to identify the most suitable isolation method. This method was realized and validated against the previously identified problem scenario.

The technical details and performance related observations presented within this case study are based on a test environment.

6.3.1. Goal and Questions

In this section, the goals of the case study are outlined. It has to be differentiated between the technical goal, which is to performance isolate the Document Service, and the goals related to the phenomena under investigation. The primary focus of the second is on the contributions concerning Chapter 5. The isolation methods were already validated by the previous sections of this chapter.

The first case study's goal **CG1** is to show the relevance of the architectural concerns. Furthermore, it should be investigated, if the process to select a certain isolation method can be applied on a real world application, which is the case study's goal **CG2**. Finally, the applicability of the architecture and the selected method is **CG3**. Due to the goals and questions characteristics, a separation into metrics seems not feasible. The questions listed subsequently are answered directly later in this section.

CQ1.1 Do the identified architectural concerns appear in the Document Service?

CQ2.1 Is it possible to initialize AHP with appropriate data, to compare the isolation methods?

CQ2.2 Can a developer apply the defined process, to find a suitable isolation method?

CQ3.1 Can the architecture and the selected isolation method be applied to the Document Service?

CQ3.2 Does the implemented isolation mechanism increase the isolation for the given problem scenario?

6.3.2. SAP HANA Cloud Document Service

The SAP HANA Cloud Document Service is a content repository for binary objects. This data can be structured or unstructured. An application served by the HCP accesses, the Document Service via a client library, provided by the runtime container. However, the Document Service provides a standard protocol for the communication, also used by the library. This protocol is the Content Management Interoperability Services (CMIS) [OASIS, 2012] standard from OASIS.

6.3.2.1. Technical Details

Figure 6.18 depicts a simplified view on the system under investigation. Several accounts host different applications (cf. Section 1.2). Each *Application* is used by one or several *Tenants*. The load onto the Document Service is load balanced by a simple round robin scheduling. The nodes of the actual service require a *Relational Database* system, storing meta information for documents. A separate *Storage* contains the documents, whereby the same storage system is shared among tenants.

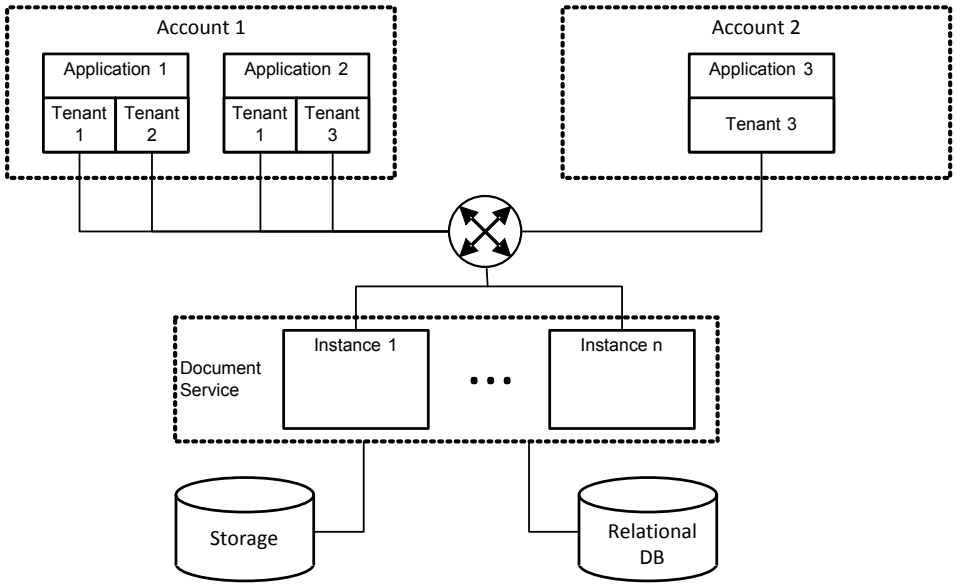


Figure 6.18.: Simplified logical view onto the Document Service.

Figure 6.19 depicts an abstracted view onto the dynamic behavior. Usually, when a user of a tenant calls a function of the application, the application first opens a session to communicate with the Document Service. Then it browses some metadata, such as traversing a folder structure, before

it uploads, deletes or downloads a document. This happens in a sequential order for each processed user request. Such sessions are bound to one particular node of the Document Service but load balanced when initiated. This binding does not rely on the application, nor the account, and not on the tenant. Thus, there is a session stickiness per users application call, but no affinity on any other entity. This may lead to unbalanced load scenarios. Furthermore, there is a thinking delay between consecutive application calls, but usually a negligible delay between Document Service calls, within the processing of one user request. Thus, the requests arrive in bulks. It is worth to emphasize that this reflects a closed workload concerning the requests arriving at the Document Service. The request's originating tenant is encoded in each HTTP request's header, sent to the Document Service.

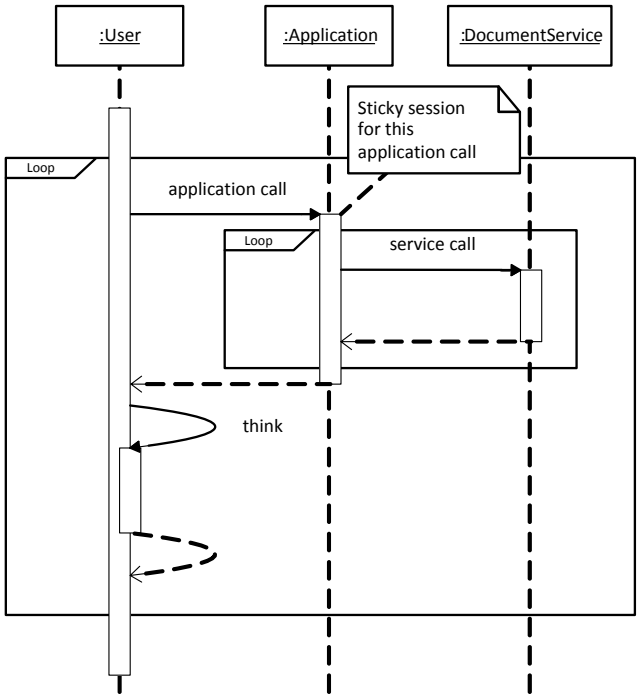


Figure 6.19.: Call sequence of the Document Service.

Additionally, the account information and application related identifiers are embedded.

6.3.2.2. Problem Statement

From the Document Service's perspective, several potential performance isolation objectives appear. An isolation method could focus on accounts, the applications or the tenants. However, isolating accounts or applications would not lead to performance isolation among tenants. Assume a very document centric application, its primary concern would be to get the Document Service calls performance isolated. In the context of the case study, the Document Service aims at maintaining performance isolated on the tenants level. This supports a document centric application the most.

An example of a document centric application is a file hosting application. The following scenario for such an application is considered in the case study. Users browse their folders and upload new content once a while, but usually download single files shared with other users. From time to

time, users download a vast amount of data, e.g., their whole repository or a folder containing many files. In such a scenario, the file hosting application opens one session, traverses the folders and downloads one file after the other from the Document Service. The service calls are independent of previous results in this use case. Therefore, parallelism is used to increase the download speed within one session. However, the maximum number of threads used is limited by the application. It is easily imaginable that in these cases, the load onto the Document Service node, serving this particular user's requests, suddenly increases. Consequently, a negative impact upon the other tenants appears.

Figure 6.20 depicts an example from the test environment, where a download of a large amount of files was made. The average response times are displayed as normalized values. They were normalized by the arithmetic mean between 0s to 650s of the depicted observation time. Around 650s the download started, and around 1750s it finishes. This is clearly visible by the significantly increasing response times of the Document Services respective node.

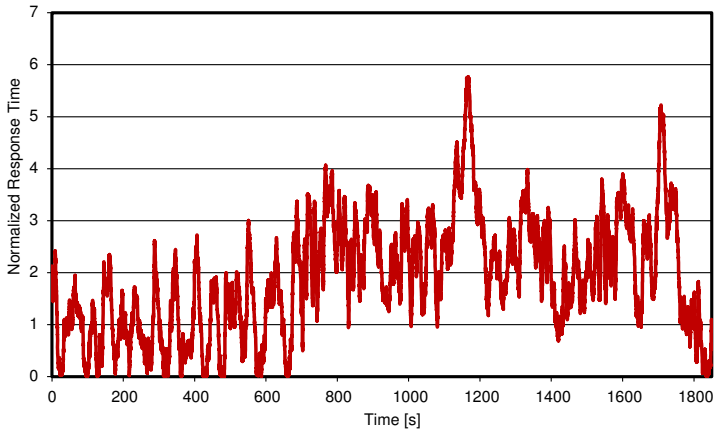


Figure 6.20.: Increasing response times in a non-isolated Document Service.

6.3.3. System Setup and Load

This section introduces the system setup of the case study, and how the load was modeled to ensure a realistic load behavior.

6.3.3.1. Landscape

Figure 6.21 depicts the deployment of the Document Service, including the PIF.

In the experiments conducted, the two nodes of the Document Service application instance, are hosted on two separated servers. Both nodes run within an *LJS* instance, in a virtual environment. *Storage 1* is used to store the documents and is mounted via Network File System (NFS). The *MongoDB* is hosted on a separate server. The Database Management System (DBMS) is running in a virtual environment, referencing a storage system via NFS. The *Load Servlets* emulates, together with *JMeter*, realistic call scenarios. The load servlets represent a PaaS hosted application using the Document Service. However, to have more flexibility in generating various load profiles, the

load servlets provide only one atomic operation per call. Examples of atomic operations are: The download of one single document, loading metadata from one object, or loading a list of all elements in a folder. Thus, the JMeter model creates the call sequences to represent an application and the users. The only missing component in this scenario is the load balancer. However, since the load balancer realizes a round robin for each session, it does not reflect the actual load of the servers. Therefore, each of the two load servlet applications is bound to exactly one Document Service node.

The *Performance Isolation Framework* core is connected with the Document Services *Isolation Valve* and its *Application Resource Monitor*, which collects the required information. The valve [Craig McClanahan, 2015] realizes the execution point, using a particular strategy for the request scheduling.

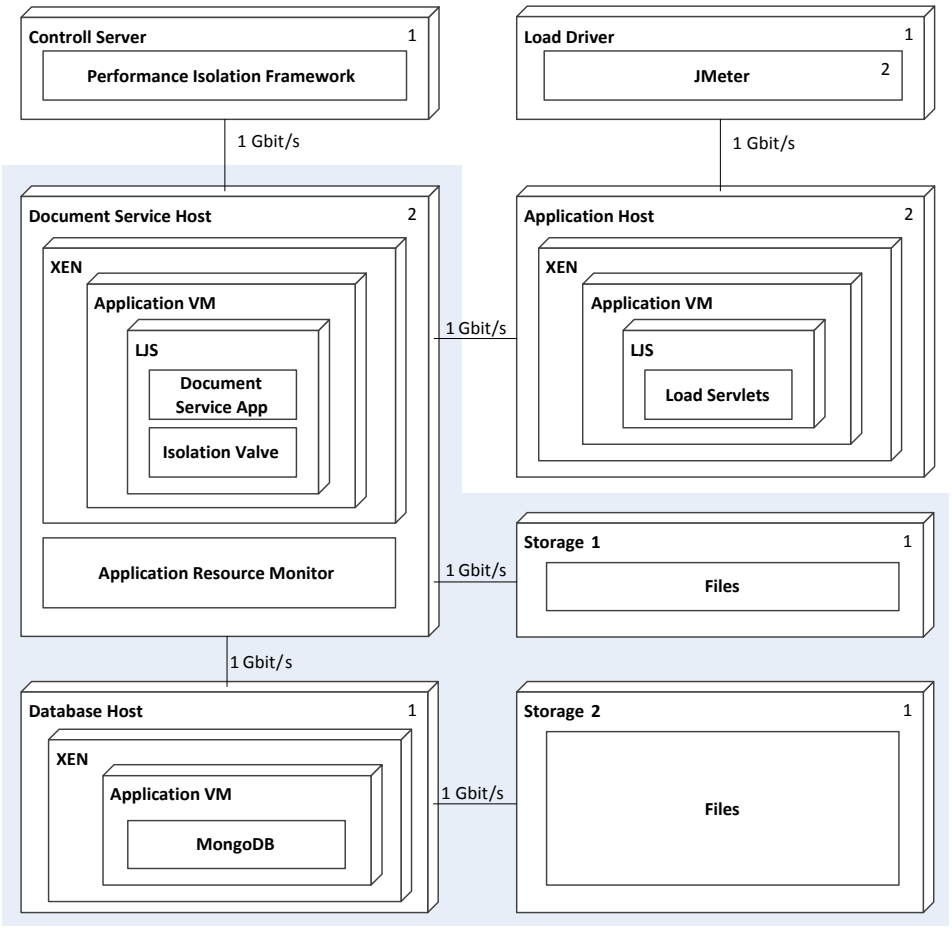


Figure 6.21.: Simplified view onto the deployed test landscape.

6.3.3.2. Representative Workload Generation

Service Calls and Files In the Appendix A.1, a limited view on an analysis of the state dependent call probability for a particular atomic Document Service call is presented as Markovian model (cf. Section 2.1.2.1). The data is based on monitored system traces. The data reflect the call behavior within one bulk, arriving the Document Service. The first table in Appendix A.1 omits negligibly seldom calls and was chosen such, that it includes the 15 most often called transitions.

The second table displays all the services, with an overall call probability of more than 5% of the cases. The second table, was used to create the JMeter script. Moreover, the average number of requests within such a bulk varies. This was modeled by an additional transition going into a sleep state, waiting until the new bulk starts.

10000 files with sizes between 1KiB and 1MiB plus 350 files with sizes between 1MiB and 30 MiB and 1000 folders were created for each tenant.

Time Behavior Another important aspect is the timing of calls. Within a bulk, the next call is immediately sent after the response arrived. However, the time span between two bulks needs some additional investigations. In Appendix A.2, the observed probability density of think time between two bulks is shown. Some clear outliers were removed. Moreover, a prediction function assuming a Poisson process is added to the diagram. The prediction error for each interval was weighted with the number of requests sent in this interval. In total, a request is predicted with an error of 57%. This is due to the long tail in the observations.

Download Behavior Considering the problem statement, the disruptive tenant starts a bulk download. This downloads files without pausing or sending other requests in significant amount in between. Moreover, this task is parallelized by starting several threads. However, the number of additional threads started for the download, is still negligible low compared to the overall number of end users using the Document Service. In relation to the overall number of users per node, the downloading user's number of threads is below 0.5%.

6.3.4. Application of the Selection Process

This section applies the selection process defined in Section 5.2, to find a suitable isolation method for the SUT. For the application of AHP, the tool Open Decision Maker [Bender et al., 2013] was used.

6.3.4.1. Preparation

Five isolation methods were evaluated in detail in Section 6.1, and additionally a method based on artificial delay in Section 3.4.1.2. Further potential isolation methods were classified, but not presented in detail in Chapter 5. As foundation for the criteria comparison, the measurement results of this thesis were considered. For the non-implemented methods, the behavior was estimated by a conceptual analysis based on Section 5.2.1. For example, a simulation model based approach will clearly be able to solve the oscillation problem as the analytical model does. However, it will for sure have longer settling times, will have much higher implementation efforts and will be more complicated to maintain in case of a white box model.

Some of the identified criteria from Section 5.2.3.1 are difficult to predict. In particular, this includes, reliability, maintainability, development and operating costs. An adequate estimation method of these aspects, for the isolation methods, was not considered to be in the scope the thesis.

These questions are a research topic in themselves. These aspects were estimated by two, and part wise by three persons guesses. All had a detailed understanding of the isolation methods conducted and had been working on this particular topic for at least 7 months. The experts were graduated, or about to graduate in the field of computer science. In general, one can consider approaches requiring more information or the core parts of the PIF to be more complex. Thus, they are less reliable, have higher development costs and higher operational costs.

Due to the large extent of comparisons, an excerpt of the alternative comparison (cf. Section 5.2) is provided in tabular form in Appendix B.

6.3.4.2. Selecting an Isolation Method

The data from the previous step was used as a foundation for selecting a concrete isolation method. Members of the Document Service's development team provided a ranking of the criteria importance. Based on their weighting, the ranking in Table 6.5 results.

Method	Value
Input Neutral (Round Robin)	22.49%.
Input Aware (Black List)	20.78%.
Resource Exploiting (Measurement)	11.16%.
Resource Isolating (Measurement)	10.95%.
Model Based (Analytical)	10.54%.
Guarantee & Quota	9.92%.
Resource Isolating (Estimation)	5.25%.
Resource Exploiting (Estimation)	5.24%.
Model Based (Simulation)	3.67%.

Table 6.5.: Ranking of isolation methods for case study.

Applying the rules from Section 5.2.3.2, most of the solutions had to be discarded for the Document Service.

The Document Service has variable resource demands per request, depending on whether it requests meta information or downloads a file. The latter one also varies in the file size. Furthermore, when the considered problem scenario appears, the average demand for a request of the disruptive tenant is significantly higher, since the proportion of requests downloading a file increases. For the system considered in the case study, the Document Services Application CPU was identified as the bottleneck. In case of a bandwidth issue, the resource demand would have been directly measurable by the data transferred by a request. However, in this case it is not possible. Therefore, the measurement based approaches are not of relevance. Model based and guarantee based methods have to be discarded, since no service level guarantee is provided by the Document Service, and they do not reflect resource demands.

For the particular case study system, the CPU demand correlates with the size of data transferred. Thus, the average demand was calculated based on this information. Moreover, it was easily possible to receive the according monitoring information on short time intervals. Consequently, a resource isolation method based on resource demand estimations was selected for the outlined scenario.

6.3.5. Experiments

In this section, the experiment results are outlined. Additionally, to the resource isolation method, a round robin based isolation method was implemented for comparison. Additionally, a scenario was assumed where one tenant accesses files of larger sizes, compared to the other tenants. Thus, the goal **G1** was to evaluate the performance isolation of both methods and **G2** to evaluate, if the rules defined are helpful. These goals allow to derive the following questions.

1. What is the influence of the disruptive tenant upon the observed response times of the other tenants?
2. How much is the difference of the observed response times between the two isolation methods?

6.3.5.1. Results

Figure 6.22 exemplary shows the results for the round robin based isolation method. It shows the average response time in a sliding window of 300 requests. For the three abiding tenants, the average value of all three is depicted. At 550s the disruptive tenant started its download. From this point in time on, the disruptive tenant had higher response times as the abiding. Due to the high noise, it is hard to identify clear patterns.

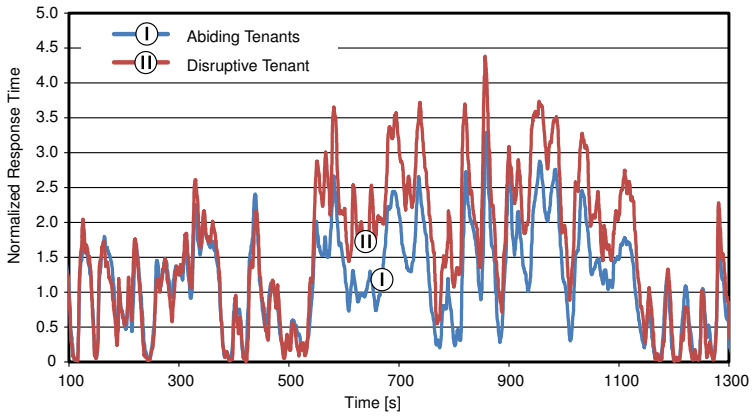
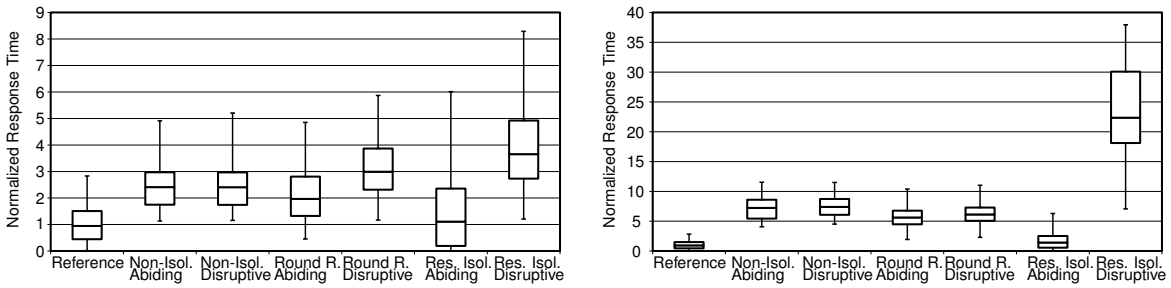


Figure 6.22.: Response times for a round robin isolated Document Service.

Therefore, Figure 6.23 depicts a comparison of the observed response times for the abiding and disruptive tenants in a box plot. For the download scenario in Figure 6.23a and the scenario with different file sizes in Figure 6.23b. All figures show the response times, normalized by the mean of



(a) Downloading a large set of files.

(b) Accessing different sized files.

Figure 6.23.: Relative response times of abiding and disruptive tenants in different isolation scenarios.

the response times in a non-isolated system, without disruptive load. The upper and lower whisker show the 95% and 5% percentile to filter outliers.

In both scenarios, the non-isolated system has for both tenants the same response times. The round robin provides a certain isolation in the downloading use case, which become visible when comparing the median of the abiding tenants with the disruptive tenants' median. This is primarily due to the increasing request arrival rate, caused by the parallelism without think times. In the scenario with different file sizes, but no change in the request arrival rate, the difference is negligible. In Plot 6.23b the round robin scenario provides a slightly better performance as the non-isolated case. However, as the differences between the abiding and disruptive tenant is the same for both methods, the root cause is a general performance fluctuation of the system. Only the resource isolation based method is able to guarantee a satisfying performance isolation. In both scenarios, the abiding tenants' performance is close to the reference workload performance, and the disruptive tenants' response times are convincingly higher.

There was no influence of the disruptive tenant upon the performance on other nodes visible. Therefore, the results from the other node are not presented.

6.3.6. Related Insights

This section briefly discusses insights related to the case study and relevant to answer the questions related to it.

6.3.7. Application of the Architecture

For the case study, the PIF was applied. The existing framework, used in the previous experiments, was modified by a minor modification concerning the communication between the execution point and the core parts of the framework. A second modification belonged to the monitoring, which had to collect the size of the files transferred. This was not foreseen as a potential parameter of interest. The resource isolation strategy from the previous experiments was reused without modifications. The resource demand estimation component was reimplemented, to support the new estimation method based on transferred file size. Despite the minor modifications in the framework's components, the overall architecture was not modified for this scenario.

6.3.7.1. Usefulness of Selection Method

It is worth thinking about the usefulness of the selection approach, because of two reasons. First, the AHP approach might always result in the same ranking, and can therefore be skipped. Second, the rules to be applied might always leave only one method, which then has to be used independently from the AHP selection.

Adjusting the weighting of the criteria such that efficiency gains more importance, leads to a completely different result, preferring the model based approach, followed by the guarantee & quota based methods. Changing any other criteria weight, also results in rank changes. Usually the input neutral methods outperform the others. However, changing ranks between the other methods are observed more often. The best ranked method is not always applicable. Following this insight, rank changes are always of relevance. This implies that AHP based ranking is important.

The rules strike out particular options. However, it does not mean that the process always ends up with just one solution. Even more interesting, if a non-supported method seems much better than the first directly applicable, it can be an option to modify the system, or to adapt the requirements. Assume a system, where the resource demand aware methods receive much better ranking than the guarantee & quota based methods and additionally, SLAs are defined and resource demands do not vary much among tenants. In this case, an adjustment of the requirements for the isolation methods can be the solution. The provider can still provide SLAs for the tenant, but he may use a non-SLA aware resource isolation, because of its benefits. Another example is a situation where, based on the rules, the analytical model and the guarantee & quota based methods are applicable. This happens easily if resource demands can be ignored. In this case an additional ranking is helpful.

It is worth mentioning, that the rules are defined on the level of the classes. If the alternatives of various concrete implementations within one class exist, a ranking method becomes even more important.

6.3.8. Concluding the Document Service Case Study

In this section, a brief summary and critical discussion of the case study is presented.

6.3.8.1. Critical Discussion

Workload Model A trade off between efforts, representativeness, a feasible degree of abstraction and legal aspects have to be considered. Consequently, the workload model did not consider all transitions and transactions observed in the traces. The modeled timing behavior differed from the observations made in the productive landscape. The file sizes, as well as their meta-information, was not based on the productive system environment, since access to this information is limited.

The data selected, to create the workload model and the transition probabilities covered 88% of all transactions running on the service. The most significant lacking are the upload operation and the query operations as defined by CMIS. However, the appearance of an upload operation was negligible, in the traces analyzed. The query operation might cause long running operations at the

Document Service. However, modeling representative query operations consumes a vast amount of time. Furthermore, it was assumed, that their impact upon the performance of the bottleneck resource is small compared to a download activity. One indicator is that queries may have many round trips to the database and waiting for its response. At this time, the application CPU, and thus the bottleneck, is not utilized.

Another point to discuss is that only four tenants were considered. In a productive environment, much more tenants can be expected. The system used in the case study allows to estimate the demands very accurate, even for a high number of tenants. Additionally, although a high number of tenants can be registered in a system, usually only few of them produce significant load at once. It is convincing, that the selected isolation method will also be able to deal with higher amounts of tenants.

Distributed approach The Document Service was provided by two nodes in the present example. Although different utilizations of the nodes can be expected, the solution ensures performance isolation on an individual node basis only. Section 5.3.1 outlined, that this can lead to disadvantages concerning the overall efficient distribution of weights. This is true in this case as well. However, in the following it will be outlined why this weighs not much in the given scenario, where resources are isolated.

Consider a situation with two nodes A and B that provide the service with a round robin based load balancing for each session. In case one user, of one tenant starts to download a large amount of data, the load increases on node A. A distributed isolation method might now shift resources from node B to A, if the disruptive tenant does not use its entirely guaranteed resources on node B. This happens only, if the disruptive tenants load is low, except that load created by one single downloading user. Consequently, it would provide the other tenants more share on node B, by removing resources of them on node A. However, since the global load of the tenant with a disruptive download has to be rather low, the additional resource might not be helpful. The reason is, that the maximum number of threads for the downloading user is limited, and therefore additional resources cannot be utilized. Furthermore, since the load balancer is not aware of the modified node guarantees. It would not adapt the balancing and thus the modifications cannot be utilized. In the case where one tenant access larger files, the requests are distributed among the nodes and thus a local optimization is sufficient.

Note that this is not the case, if the number of users significantly changes in a multi-tenant system with sticky, long running user sessions.

6.3.8.2. Summary

In this section, a case study based on the HCP was presented. First, the section outlined the relevant goals of the case study. After that, the technical details of the Document Service were presented and the workload was characterized. Furthermore, a realistic problem statement was explained. Based on this knowledge, the selection process for a performance isolation method presented in Section 5.2

was applied and a resource isolation method was suggested. For comparative reasons, a round robin based isolation method was implemented. Both methods were applied, using the insights, architecture and the PIF discussed in Section 5.3. Finally, the measurements clearly showed that the isolation method selected by the process, provides a very good performance isolation, while round robin fails in providing sufficient isolation.

The relevance of the architectural concerns, especially referring to session stickiness and affinity, play a major role in the case study and the previous discussion helped to identify important aspects related to performance isolation. Customer based customizing did not appear. However, the different sized data volumes reflect realistic problem. Furthermore, the Document Service uses a shared table schema (**CQ1.1**). It was possible to initialize the AHP process, with useful information (**CQ2.1**). Furthermore, the results were used by the development team of the Document Service to find a proper isolation method for their scenario (**CQ2.2**). Two isolation methods were realized, using the reference architecture from Section 5.3. The isolation method selected by the AHP, and the proposed process achieved a very good isolation in the case study (**CQ3.1**, **CQ3.2**). The round robin isolation method selected for comparison failed to provide sufficient isolation (**CQ2.2**).

6.4. Concluding the Overall Evaluation Results

This chapter was divided into three sections. Each summarized the evaluation results and conducted a critical discussion. In this section, a brief overall summary and a critical discussion on an overall level is conducted. For the evaluation details, please refer to the corresponding subsection.

6.4.1. Critical Discussion

Amount of Environments In most of the experiments, only the LJS was conducted as the runtime container for the measurements and proof of concepts. However, the LJS is used in a public available, productive cloud offering and thus it is a representative runtime container. Furthermore, it was deployed in a variety of system setups. In virtual an non-virtual environments, with separated or integrated databases, and separated storage or local storage. Additionally, the general applicability of the performance isolation framework was also under investigation in a simulation based environment. Although it was not part of the extensively discussed environments within this thesis, further related experiments were made. Thereby, a standard tomcat server and an execution point implementation as part of a stand alone proxy [Krebs et al., 2014a], completely decoupled from any runtime container dependent technology, were implemented. These implementations also provided a sufficient isolation.

Valve as Execution Point In the experiments presented in this chapter, only valve based execution points were considered (cf. [Craig McClanahan, 2015]). However, as in the previous paragraph mentioned, other implementations based on a stand alone proxy implementation were also used successfully. Furthermore, similar concepts to filter incoming requests are also known from other

containers. Overall, this is a technical issue without impact upon the general applicability of the contributions presented. Another issue not discussed so far is that using a valve like concept may result in a high number of open sessions, since for each queued request a session is kept open. This may lead to situations where performance isolation becomes violated, due to an insufficient number of connections on the lower layers. However, in the experiments conducted with a realistic configuration, this specific limit was not reached. Additionally, adding an upper limit for the queue length and rejecting further arriving requests is only a minor modification.

Amount of Case Studies Only one case study was conducted to evaluate the contributions from Chapter 5, since realistic case studies are hard to find and time consuming to setup. However, the case study validated almost all aspects from Chapter 5, by walking through the whole process identified in the motivation (cf. Section 1.2). The single contributions of the Chapter 5, e.g., architecture and the resulting framework, were also applied in other contexts. This shows the applicability in various contexts. Moreover, it was possible to easily integrate the isolation methods into the existing framework. Furthermore, the classification, informational requirements, architectural concerns and the analysis of relevant requirements were derived by valid arguments and a conceptual analysis of the existing methods. An empirical study on applying AHP is not necessary. The method is widely accepted and thus already proved to support decision finding processes. That AHP is an appropriate solution to select performance isolation methods, can be concluded from the analysis of requirements in Section 5.2.3.

6.4.2. Summary

This chapter validated the contributions of the thesis in numerous experiments and case studies. Overall, functional validations and case studies were provided in Chapter 4 and Chapter 3. Moreover, this chapter provided comparative measurements of five isolation methods, with regard to isolation, settling times, oscillation and overheads. Furthermore, by applying the contributions from Chapter 3, the metrics were again validated against their applicability and usefulness. Moreover, a case study based on the SAP Hana Cloud Document Service was conducted to evaluate the contributions from Chapter 5. The main findings of all these aspects are outlined in the following:

1. A Request based admission control, with a fast reacting and asynchronously updated work-conserving scheduler can increase performance isolation.

2. There are significant differences in the various quality aspects of isolation methods from different classes.

3. The two novel performance isolation methods, were are able to provide performance isolation to a certain degree. Furthermore, they outperform comparable approaches by either the isolation quality, in case of variable resource demands, by lower oscillation or efficiency when over-commitment is used.

4. In general, the overhead onto the application is negligible for all methods. Overall, the methods provide a good scalability.

5. The analyzed performance isolation methods can be clustered in two groups. Those where the policies directly influence the admission of a tenant's request, and those which do not have a policy at all, or using the information internally, to adjust the priorities based on input data. The latter have the lowest oscillation, and the first group is able to share the system more efficient, enabling over-commitment, while reflecting application level performance guarantees.

6. Results from experiments utilizing hypervisors can be compared with the results from the proposed request based admission control. For scenarios, providing similar properties, both achieve a similar isolation.

7. The isolation metrics describe a system's isolation, referring to the questions arising for various stakeholders. They are also applicable in cases where the system is a black box.

8. The metrics computation can be automated, and the developed benchmark application allows to evaluate multi-tenant environments.

9. The information gathered by the experiments, and discussed as enabling knowledge can be used to find an appropriate isolation method for a given problem scenario.

7. Conclusion

This chapter starts by summarizing the thesis. Additionally, a critical discussion section focusing on general aspects of the whole thesis and its limitations follows. An outlook on potential future research directions and concluding remarks finalize the thesis.

7.1. Summary

First, a brief summary of the content of the various chapters in this thesis is given, followed by a summary of the research questions answers. This indirectly reflects the contributions made. For more detailed summaries, the reader may refer to the respective summary sections in the previous chapters.

7.1.1. Recapture of the Chapters

The *Introduction* motivated the need for multi-tenant applications to increase economic efficiency, followed by a discussion of the relevance of performance in general and, conclusively, the need for performance isolation. Thereby the understanding of performance isolation, which briefly means that a disruptive tenant has no negative impact upon the performance observed by other tenants, was discussed. Based on this, the goals of the thesis, the challenges, and the research questions were derived. In the *Foundations and State-of-the-Art* section, the novelty of the topic for MTA was emphasized. In most related approaches, the information about the isolated entity is transported to the resource controlling layers or the entities are allocated according to their SLAs to different resource pools. This is insufficient for MTAs, as it would mean introducing new operating system concepts and forwarding the tenant information to several resources distributed among the whole stack. An SLA-aware allocation requires knowledge about future workload, which is not the case here. The related work considering MTAs rather focuses on service differentiation. The *Measurement of Isolation* section presented metrics based on the impact one tenant's workload has upon the performance of another. Thereby, common requirements for metrics were respected. Furthermore, methods to gather the metrics were introduced. These metrics were validated in different contexts. In general, the *Methods for Performance Isolation* section described a request based admission control, which is dynamically reconfigured at runtime to reflect the observed workload scenario. One proposed approach is based on a black box model of the MTA, and one utilizes resource demand estimations of single requests. Additionally, the section showed a proof of concept for these approaches on depicted scenarios. The section *Decision Support and Architecture* provided relevant insights that are needed to realize the isolation methods. Therefore, the informational requirements

of various isolation methods were outlined and relevant non-functional aspects were collected. A process using AHP was recommended to find the best isolation method for a given problem scenario. Furthermore, implementation recommendations were given in the form of a framework that can realize several isolation methods. The *Evaluation* compared the isolation methods and identified their strengths and weaknesses. It clearly showed the advantages of the advertised methods, when resource demands of tenants are different or MTAs are over-committed. The knowledge was used to apply an isolation method on a real world application, to solve a realistic problem. Finally, this chapter summarizes the results.

7.1.2. Answering the Research Questions

This section summarizes the answers for the research questions from Section 1.3.4.

RQ 1 — *What is an appropriate approach to provide performance iIsolation in multi-tenant applications?*

A request admission control is able to limit the mutual influence of tenants on several layers. By using work-conserving schedulers, it is possible to provide a good utilization of the system. The automatic reconfiguration of the admission control's behavior ensures best performance for tenants, while considering guarantees. A framework can be used to implement application-specific isolation methods.

RQ 2 — *What are appropriate metrics to quantify the level of performance isolation a system provides?*

The measure of the impact of one tenant's changing workload upon the performance observed by another tenant can be used in two ways to quantify isolation. Either one sets the change in quality in relation to the increasing load, or one can identify workload combinations of disruptive and abiding tenants, for which the quality of the abiding tenants maintains the same value for all disruptive loads. The relation between the two workloads can then be used as a basis for a metric. Besides that, metrics covering the dynamic behavior are required: one covering the time a method needs to settle to a stable configuration and one describing how the performance changes, although the load is kept constant.

RQ 3 — *What are the relevant characteristics of MTAs and the isolation methods to select and realize a feasible isolation method for a given scenario?*

For MTA, the session/tenant affinity and the tenant's freedom in customizing, which may result in different demands per request, are most relevant. Besides the quality of the isolation methods, the informational requirements are of importance to select an isolation method that is supported by a certain scenario.

RQ 4 — *What is the best isolation method for a particular scenario?*

In general, non-feedback based approaches provide a better isolation with lower complexity if the resource demands among all tenants' requests are the same. Some feedback based methods can provide better isolation if resource demands for tenants are different. Other feedback based methods also provide a more efficient request handling in case of over-commitment at application level SLAs.

7.2. Critical Discussion, Future Research and Limitations

At the end of each chapter, a critical discussion of the chosen approach was conducted. Therefore, this section focuses on some aspects of global interest that were not yet covered.

Additional Isolation Methods Potential solutions transferring the tenant concept to the resource access layers, or based on the placement of tenants, were not evaluated in detail. These approaches might be appropriate for certain scenarios. The drawbacks for interactive web applications were discussed in Section 4.1.1.

While methods based on allocating tenants onto dedicated nodes cannot provide proper isolation, resource control methods might do. The primary reason against resource control methods is that the complexity and coupling between the application and the underlying layers would make the practical use insufficient. However, from a research point of view, it could be interesting to investigate this topic in the future to either prove or falsify this assumption.

Other approaches could be based on thread pool and thread priority management per tenant. Hereby, it is worth mentioning that limiting the thread pool sizes on a tenant basis is similar to the tenant individual request admission control. The latter can easily realize a work-conserving mechanism without the risk of overload.

For the future work, it would be interesting to identify certain scenarios in which isolation methods not using an admission control are beneficial and how they can be realized. It could also be worth developing methods that use workload forecasts on a short-term basis to reduce settling times. Combining placement driven approaches with the request based admission control methods from this thesis might also be advantageous.

Considering Malicious Workload Scenarios Malicious workload relates to, e.g., manipulated requests or DDoS attacks, executed by one tenant with bad intention. The work in this thesis excludes malicious attacks that aim at reducing the performance of the system, since this is related to security research. However, if a tenant is identified to generate malicious requests, the admission control might add a black list approach rejecting all requests from this tenant. Thus, the overall concept is still feasible. In a first proof of concept, not discussed in the thesis, it was already shown that the isolation method can be dynamically changed at runtime to handle malicious loads. This mechanism has been successfully applied in situations where a tenant tries to send many requests to catch all connections on the operating system level. This would result in a denial of service, although the requests are queued.

Future work may validate this approach in more detail, and could identify existing mechanisms or new ones from the security domain to decide when to switch to a new admission control strategy.

Non-Isolated Applications It was assumed that application instances and application nodes do not influence each other. However, as shown in several publications, application instances installed on different virtual machines on the same host are likely to influence each other. Nevertheless, if performance is important for the providers, a deployment of the applications can be optimized for low inference. Moreover, the isolation methods presented do not necessarily rely on a constant performance of the application. Thus, the isolation among tenants is still better isolated, and consequently the tenants' performance is more stable than those without the contributions made in this thesis.

It could be interesting to evaluate the influence of variable performance of an application node upon the isolation mechanisms. Additionally, applying the proposed isolation methods in a cross application scenario potentially increases isolation among application instances' performance.

Cluster Deployments MTAs might be deployed on more than one server, and thus consist of several application nodes. The isolation methods considered are optimized for one node so far. However, following the design recommendations for MTAs leads to systems where the isolation can be maintained by the proposed isolation methods. Although several nodes may exist, tenants are usually not distributed among them. It is rather common to have a server affinity. If tenants produce much workload, the need for approaches distributing the application among several nodes increases. However, this would also mean that the economic benefits of MTA are reduced, which is also a reason that server affinity is not uncommon. Even if an application does not follow the design recommendations, the isolation is better with the contributions made in the present thesis than without. Furthermore, the admission control could be realized at the load balancer, providing less efficiency but still ensuring proper isolation.

Although the most relevant scenarios are already covered, the development of distributed isolation methods could be beneficial for selected scenarios and thus worth being discussed in future. The foundation for such work is already provided. Existing mechanisms from the field of virtualization describe how an all-embracing configured resource guarantee can be enforced by adjusting the shares on single nodes [Gulati et al., 2012]. These mechanisms do not answer how to maintain performance isolation within one multi-tenant application node, but provide solutions to make the isolation methods cluster enabled. The model based isolation method can be enhanced for cluster scenarios by combining the independent server functions and using an adjusted optimization space, covering the weights for each tenant, on each server. Therefore, the most relevant research-related challenges are covered by the thesis.

Implementation of Additional Approaches Although the classification schema in Section 5.2.1 defined measurement-based resource demand and simulation-based methods, these were not evaluated by the experiment in this work. It can easily be assumed that resource isolation works

with higher accuracy if the demands can be measured. In a proof of concept not discussed in the thesis, the Queuing Petri Net Modeling Environment [Kounev and Dutz, 2009] has been enhanced to simulate the admission controls of the isolation methods. The model predicted the performance of individual tenants, with high accuracy. However, the simulation scales worsened with higher numbers of tenants and runs into unsustainable simulation times for the sake of performance isolation when workloads change fast. Therefore, it can be assumed that only very few situations can benefit from a simulation.

Since a simulation can include application internal characteristics, it could be beneficial for situations that cannot be covered by the analytical black box model proposed. Therefore, investigations to speed up the simulation would be beneficial.

Tenant-Specific Presentation Quality Dynamically adapted presentation quality is one approach used to reduce a server's load when facing high workloads. It could also be an option to reduce the functional qualities of an application for disruptive tenants to decrease their influence upon others tenants; e.g., checking the liquidity of a credit card is an optional task in a payment process. It could be delayed to a point with low loads. Such concepts might be considered in future work.

Automated configuration The proper configuration of the presented isolation methods is a time consuming task. In future, algorithms might be applied, which automatically detect suitable settings and reconfigure the isolation methods at runtime. This may include the monitoring and update intervals, as well as method-specific parameters.

7.3. Potential Further Applications

Since the presented isolation methods can control performance on a tenant basis, it is also possible to provide different tenants *individual performance qualities*. Both isolation methods presented in detail already support this idea, although it was not the primary goal of the thesis. This also lays the foundations for *tenant-individual elasticity*. Since the share a tenant is allowed to use can be controlled, additional resources added to the application can be explicitly guaranteed to the tenant who pays for them.

The focus of the contributions in this thesis is on performance isolation of individual tenants in an MTA. Nevertheless, several contributions and approaches are also beneficial in other fields. Relevant to other fields are the isolation metrics. The metrics are defined independently from any technical domain or concrete quality metric that has to be isolated. Thus, the *metrics are likely to be applied in other contexts*, where resources are shared among several entities. Examples are active network components, like switches or routers, storage systems or fine-grained schedulers on the operating system level. The case studies already showed how the metrics can be applied to study the Xen hypervisor. Besides performance related metrics, *other quality aspects* might be applicable as long as they can be quantified adequately.

Resources are also shared outside IT related topics. For instance, call centers are often shared by different clients. They may use them for, e.g., first level support, or to handle ordering processes. In such situations, the call center providers might be interested in sharing their resource fairly among the clients. Imagine a client starts an advertising campaign, or a bug in a product suddenly arises with an update. These are situations where isolation methods might be considered. However, since the metrics were rather developed for benchmarked environments, it might be difficult to apply them. If the request arrivals of one client maintain a steady value, the QoS impact-based metrics are an option. Besides the metrics, isolation methods could be applied. Depending on the situation, the demand between clients might vary significantly. In the given example it could be measured, and thus demand-based approaches might be applicable. By using an open workload model, it would also be possible to apply a model-based isolation method. It is worth mentioning that the isolation methods proposed might also be applicable to *other fields in the IT context*. Among others, I/O schedulers could benefit from the ideas, since they rather consider a QoS differentiation and have to handle a hidden internal state of the device.

Since resources are isolated between tenants sharing one application instance, this allows the creation of new business models. Usually prices are paid based on application level quantities, such as the number of business transactions or number of requests. If resources can be mapped to requests, and isolation of resources can be achieved, the pricing might relate to resources.

The reference architecture was developed to fulfill performance isolation related requirements. However, similar information is required for self-aware systems in general. Thus, the *performance isolation framework might also be used in other contexts*. One option could be to automatically adapt the thread pool sizes to optimize the overall performance.

The insights from the *side contributions* are also relevant outside the performance isolation context. Resource demand estimations are required when, e.g., performance models are generated, or as basis for capacity planning. The benchmarking application can be used when an MTA supporting middleware should be studied.

7.4. Concluding Remarks

Providing sustainable performance for tenants, while sharing one application instance, is an essential enabler for high quality SLA guarantees in SaaS environments. Within the present work, methods are provided to establish reliable performance in multi-tenant application instances. The thesis provides performance isolation methods based on request admission control. The experiments proved the capabilities of the methods to increase isolation by maintaining a high efficiency in the real world based application scenarios. Metrics and methods are provided to quantify the isolation quality of an approach. The metrics were successfully applied in several experiments, considering a variety of technical domains. Moreover, important characteristics of isolation methods and multi-tenant applications are described, and design recommendations for developers of multi-tenant applications are given. Moreover, an evaluation of several isolation methods and the development of a decision-making process helps to find the best isolation method. Overall, this thesis makes nine

important contributions directly related to the goals, and three side contributions tackling the wider field of shared systems and multi-tenancy. More than ten systems were deployed or simulated, each used for several experiment series to show the validity and applicability of the contributions. Furthermore, several contributions can also be applied in fields other than tenant based performance isolation.

Acronyms and Abbreviations

AC Admission Control

ACC Admission Control Controller

AHP Analytical Hierarchical Process

CMIS Content Management Interoperability Services

CPU Central Processing Unit

DB Database

DBMS Database Management System

EB Emulated Browser

FIFO First In First Out

GQM Goal Question Metric

HCP SAP Hana Cloud Platform

IaaS Infrastructure as a Service

LAD Least Absolute Deviations

LJS Lean Java Server

LSQ Linear Least Squares

MADM Multiple Attribute Decision Making

MAUT Multi-Attribute Utility Theory

MCDM Multiple Criteria Decision Making

MIMO Multiple Input Multiple Output

MODM Multiple Objective Decision Making

MTA Multi-Tenant Application

MTTQR Mean Time To Quality Repair

NFS Network File System

NNLS Non Negative Least Square

PaaS Platform as a Service

PASTA Poisson Arrivals See Time Averages

PFQ Priority Fair Queuing

PI Proportional Integral

PID Proportional Integral Differential

PIF Performance Isolation Framework

QN Queuing Network

RAM Random Access Memory

RDBMS Relational Database Management System

RDE Resource Demand Estimation

RIS Resource Isolation Scheduler

SaaS Software as a Service

SDL Service Demand Law

SDN Software Defined Networks

SISO Single Input Single Output

SLA Service Level Agreement

SoPeCo Software Performance Cockpit

SPE Software Performance Engineering

SUT System Under Test

VM Virtual Machine

WFQ Weighted Fair Queuing

Glossary

Alternative In decision making processes: One of the options from which the solution is selected.

Application Instance Deployed and running application serving requests. Could exist of several nodes.

Availability Degree to which a system is in a specified operable state within a given period in time.

Cluster Set of nodes providing a service together.

Criteria In decision making processes: The characteristics of the alternatives used to identify the best alternative for achieving a goal.

dom A VM in the Xen context.

dom0 Privileged VM in Xen allowing direct access to hardware and controlling of other doms.

Java Runtime Environment Abstract definition of computing system to execute java byte code. Several compatible implementations exist.

Java Programming language.

Lean Java Server Application server of the SAP HCP.

Maintainability Degree to which a system can be maintained.

MapReduce Programming paradigm for the processing of large datas ets using parallelism and distributed algorithms.

Application Node One instance of an application that is part of a cluster.

Reliability A system's ability to correctly perform its required functions under stated conditions.

SOA Principle for designing and developing inoperable software services.

UI User Interface

VM A virtual machine adds a virtualization layer between hardware and operating system. Often used to operate several virtual machines on the same hardware.

Xen Software to enable the operation of VMs.

List of Figures

List of Tables

Bibliography

[Abdelzaher et al., 2008] Abdelzaher, T., Diao, Y., Hellerstein, J. L., Lu, C., and Zhu, X. (2008). Introduction to control theory and its application to computing systems. In *Performance Modeling and Engineering*, pages 185–215. Springer.

[Aiken, 2011] Aiken, L. (2011). Why Multi-Tenancy is Key to Successful and Sustainable Software-as-a-Service (SaaS). *Cloudbook*, 2(1).

[Ali-Eldin et al., 2012] Ali-Eldin, A., Tordsson, J., and Elmroth, E. (2012). An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS)*, pages 204–212.

[Amazon, 2014] Amazon (2014). Running databases on aws. Technical report, amazon corporation. last accessed: 19. Nov 2014.

[Andrikopoulos et al., 2013a] Andrikopoulos, V., Chabanoles, N., Exertier, F., Jimenez, R., Krebs, R., Pelletier, B., and Riggs, S. (2013a). Immigrant PaaS Technologies: Scientific and Technical Report D7.1.3. Technical report, 4CaaSt - Seventh Framework Programme (FP7/2007-2013), grant agreement no 258862.

[Andrikopoulos et al., 2013b] Andrikopoulos, V., Chabanoles, N., Exertier, F., Krebs, R., Jimenez-Peris, R., Pelletier, B., and Riggs, S. (2013b). Immigrant PaaS Technologies: Scientific and Technical Report D7.2.3. Technical report, 4CaaSt - Seventh Framework Programme (FP7/2007-2013), grant agreement no 258862.

[Andrikopoulos et al., 2013c] Andrikopoulos, V., Chabanoles, N., Exertier, F., Krebs, R., Pelletier, B., Porcher, G., and Jimenez-Peris, R. (2013c). Immigrant PaaS Technologies: Scientific and Technical Report D7.3.3. Technical report, 4CaaSt - Seventh Framework Programme (FP7/2007-2013), grant agreement no 258862.

[Arlitt and Jin, 2000] Arlitt, M. and Jin, T. (2000). A workload characterization study of the 1998 World Cup Web site. *Network, IEEE*, 14(3):30–37.

[Armbrust et al., 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

[Aulbach et al., 2008] Aulbach, S., Grust, T., Jacobs, D., Kemper, A., and Rittinger, J. (2008). Multi-tenant Databases for Software As a Service: Schema-mapping Techniques. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1195–1206, New York, NY, USA. ACM.

[Baker et al., 2001] Baker, D., Bridges, D., Hunter, R., Johnson, G., Krupa, J., and Murphy, J.and Sorenson, K. (2001). Guidebook to decision-making methods. Technical Report WSRC-IM-2002-00002, United States Department of Energy.

[Bender et al., 2013] Bender, Blocher, Rossmehl, and Rotter (2013). Open decision maker v1.0.1. `http://sourceforge.net/projects/opendecisionmak`.

[Bezemer and Zaidman, 2010] Bezemer, C.-P. and Zaidman, A. (2010). Multi-tenant SaaS Applications: Maintenance Dream or Nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 88–92, New York, NY, USA. ACM.

[Binnig et al., 2009] Binnig, C., Kossmann, D., Kraska, T., and Loesing, S. (2009). How is the weather tomorrow? In *Proceedings of the Second International Workshop on Testing Database Systems - DBTest '09*, DBTest '09, page 1, New York, New York, USA. ACM Press.

[bitcurrent, 2011] bitcurrent (2011). BITCURRENT CLOUD COMPUTING SURVEY 2011.

[Bittman et al., 2014] Bittman, T., Margevicius, M., and Dawson, P. (2014). Magic quadrant for x86 server virtualization infrastructure. Technical report, Gartner. Gartner Reference ID: G00262673.

[Boehm et al., 1976] Boehm, B. W., Brown, J. R., and Lipow, M. (1976). Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, pages 592–605, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Bolch et al., 1998] Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. S. (1998). *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, New York, NY, USA.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Berichte über verteilte messysteme. Cambridge University Press.

[Braden et al., 1997] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. (1997). RFC 2205: Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification. Technical report, IETF.

[Brataas, 2014] Brataas, G. (2014). CloudScale: Design support Deliverable D1.2. FP7-ICT-2011-8-317704.

[Brey and Lamers, 2009] Brey, T. and Lamers, L. (2009). Using Virtualization to Improve Data Center Efficiency.

[Brosig et al., 2011] Brosig, F., Huber, N., and Kounev, S. (2011). Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems. In *26th IEEE/ACM International Conference On Automated Software Engineering (ASE 2011)*.

[Brosig et al., 2009] Brosig, F., Kounev, S., and Krogmann, K. (2009). Automated extraction of palladio component models from running enterprise Java applications. In *VALUETOOLS 09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST.

[Brosig et al., 2015] Brosig, F., Meier, P., Becker, S., Koziolek, A., Koziolek, H., and Kounev, S. (2015). Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based Architectures. *IEEE Transactions on Software Engineering (TSE)*, 41(2):157–175.

[Cain et al., 2001] Cain, H. W., Rajwar, R., Marden, M., and Lipasti, M. H. (2001). An Architectural Evaluation of Java TPC-W. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 229–240, Monterrey, Mexico.

[Calvin and Friedl, 2009] Calvin, P. and Friedl, S. (2009). Lessons Leraned: Building Multitenant Applications with the Windows Azure Platform. recording: `http://www.microsoftpdc.com/2009/SVC33`.

[Card et al., 1991] Card, S. K., Robertson, G. G., and Mackinlay, J. D. (1991). The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 181–186. ACM.

[Casale et al., 2008] Casale, G., Cremonesi, P., and Turrin, R. (2008). Robust Workload Estimation in Queueing Network Performance Models. *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, pages 183–187.

[Chandra et al., 2006] Chandra, A., Pradhan, P., Tewari, R., Sahu, S., and Shenoy, P. (2006). An observation-based approach towards self-managing web servers. *Computer Communications*, 29(8):1174–1188.

[Cherkasova et al., 2007] Cherkasova, L., Gupta, D., and Vahdat, A. (2007). Comparison of the three CPU schedulers in Xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51.

[Chong et al., 2006] Chong, F., Carraro, G., and Wolter, R. (2006). Multi-Tenant Data Architecture. `http://msdn.microsoft.com/en-us/library/aa479086.aspx`. last accessed: 10. March 2015.

[Cooper et al., 2010] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA. ACM.

[Courtois and Woodside, 2000] Courtois, M. and Woodside, M. (2000). Using Regression Splines for Software Performance Analysis. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 105–114, New York, NY, USA. ACM.

[Craig McClanahan, 2015] Craig McClanahan, Gunnar Rjnning, P. D. (2015). Interface Valve - Apache Tomcat 7.0.59. `https://tomcat.apache.org/tomcat-7.0-doc/api/org/apache/catalina/Valve.html`. last accessed: 1. March 2015.

[Das et al., 2011] Das, S., Nishimura, S., Agrawal, D., and El Abbadi, A. (2011). Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud Using Live Data Migration. *Proc. VLDB Endow.*, 4(8):494–505.

[Davies et al., 2014] Davies, J., Thompson, E., Herschel, G., Maoz, M., Desisto, R. P., Sarner, A., Collins, K., Kraus, D., Correia, J. M., Sullivan, P. J., and Elliot, B. (2014). The Gartner CRM Vendor Guide. Gartner report, Gartner. Gartner Reference ID: G00261534.

[Davis, 1993] Davis, A. M. (1993). *Software Requirements: Objects, Functions, and States*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Denning and Buzen, 1978] Denning, P. J. and Buzen, J. P. (1978). The Operational Analysis of Queueing Network Models. *ACM Computing Surveyes*, 10(3):225–261.

[Diao et al., 2002a] Diao, Y., Gandhi, N., Hellerstein, J., Parekh, S., and Tilbury, D. (2002a). Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. In *IEEE/IFIP Network Operations and Management Symposium, 2002.*, pages 219–234.

[Diao et al., 2002b] Diao, Y., Hellerstein, J. L., and Parekh, S. (2002b). Optimizing Quality of Service Using Fuzzy Control. In *DSOM '02: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 42–53, London, UK. Springer-Verlag.

[Dirlewanger, 1994] Dirlewanger, W. (1994). *Messung und Bewertung der DV-Leistung: auf Basis der Norm DIN 66273*. Hüthig.

[Dory et al., 2011] Dory, T., Mejías, B., Van Roy, P., and Tran, N.-L. (2011). Measuring elasticity for cloud databases. In *The Second International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2011*, pages 154–160.

[Dromey, 1995] Dromey, R. G. (1995). A Model for Software Product Quality. *IEEE Trans. Softw. Eng.*, 21(2):146–162.

[Dynatrace, 2015] Dynatrace (2015). Website: Dynatrace. `http://www.dynatrace.com`. last accessed: 15. March 2015.

[Eclipse, 2013] Eclipse (2013). Solutions Guide for EclipseLink Release 2.5.

[Elmasri and Navathe, 1999] Elmasri, R. A. and Navathe, S. B. (1999). *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

[Emeakaroha et al., 2010] Emeakaroha, V. C., Brandic, I., Maurer, M., and Dustdar, S. (2010). Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In *High Performance Computing and Simulation (HPCS) 2010*.

[Fehling et al., 2010] Fehling, C., Leymann, F., and Mietzner, R. (2010). A Framework for Optimized Distribution of Tenants in Cloud Applications. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing, CLOUD 2010*, pages 252–259. IEEE Computer Society.

[Fehling et al., 2014] Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014). *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Publishing Company, Incorporated.

[Figueira et al., 2005] Figueira, J., Greco, S., and Ehrgott, M. (2005). *Multiple criteria decision analysis: state of the art surveys*, volume 78 of *International Series in Operations Research & Management Science*. Springer Verlag.

[Fülöp, 2005] Fülöp, J. (2005). Introduction to Decision Making Methods. Technical report, Hungarian Academy of Sciences.

[Galante and Bona, 2012] Galante, G. and Bona, L. C. (2012). A survey on cloud computing elasticity. In *Proceedings of the IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 263–270. IEEE Computer Society.

[Gao et al., 2011] Gao, B., An, W. H., Sun, X., Wang, Z. H., Fan, L., Guo, C. J., and Sun, W. (2011). A Non-intrusive Multi-tenant Database Software for Large Scale SaaS Application. In *Proceedings of the 2011 IEEE 8th International Conference on e-Business Engineering*, ICEBE '11, pages 324–328, Washington, DC, USA. IEEE Computer Society.

[Georges and Eeckhout, 2010] Georges, A. and Eeckhout, L. (2010). Performance Metrics for Consolidated Servers. In *HPCVirt 2010*.

[Glanz, 2012] Glanz, J. (2012). The Cloud Factories: Power, Pollution and the Internet.

[Glass et al., 2002] Glass, R. L., Vessey, I., and Ramesh, V. (2002). Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44(8):491–506.

[Google, 2014] Google (2014). Implementing Multitenancy Using Namespaces. https://cloud.google.com/appengine/docs/java/multitenancy/multitenancy. last accessed: 25. Nov 2014.

[Google, 2015] Google (2015). Google Apps for Work. https://www.google.com/work/apps/business. last accessed: 16. March 2015.

[Grzech et al., 2009] Grzech, A., Swikatek, P., and Rygielski, P. (2009). Improving QoS Guaranties via Adaptive Packet Scheduling. In *Proceedings of the 16th Polish Teletraffic Symposium (PTS 2009)*, pages 53–56. Technical University of Łódź, Technical University of Łódź. Young Author Best Paper Award.

[Gulati et al., 2010] Gulati, A., Merchant, A., and Varman, P. J. (2010). mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–7, Berkeley, CA, USA. USENIX Association.

[Gulati et al., 2012] Gulati, A., Shanmuganathan, G., Holler, A., Waldspurger, C., Ji, M., and Zhu, X. (2012). VMware Distributed Resource Management: Design, Implementation, and Lessons Learned. *VMware Technical Journal*, 1(1):45–64.

[Guo et al., 2007] Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., and Gao, B. (2007). A Framework for Native Multi-Tenancy Application Development and Management. In *The 9th IEEE International Conference on ECommerce Technology and the 4th IEEE International Conference on Enterprise Computing ECommerce and EServices 2007 CECEEE 2007*, pages 551–558. IEEE Computer Society.

[Gupta et al., 2006] Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A. (2006). Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 342–362, New York, NY, USA. Springer-Verlag New York, Inc.

[Hellerstein et al., 2002] Hellerstein, J., Diao, Y., and Parekh, S. (2002). A first-principles approach to constructing transfer functions for admission control in computing systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 3.

[Hellerstein et al., 2004] Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[Hellerstein et al., 2008] Hellerstein, J. L., Morrison, V., and Eilebrecht, E. (2008). Optimizing Concurrency Levels in the .NET ThreadPool: A Case Study of Controller Design and Implementation. In *Proceedings of the FeBID Workshop 2008*.

[Herbst et al., 2013] Herbst, N. R., Kounev, S., and Reussner, R. (2013). Elasticity in Cloud Computing: What it is, and What it is Not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*. USENIX.

[Herbst et al., 2015] Herbst, N. R., Kounev, S., Weber, A., and Groenda, H. (2015). BUNGEE: An Elasticity Benchmark for Self-Adaptive IaaS Cloud Environments. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*.

[Herndon et al., 2006] Herndon, B., Smith, P., Roderick, L., Zamost, E., Anderson, J., Makhija, V., Herndon, B., Smith, P., Zamost, E., and Anderson, J. (2006). VMmark: A Scalable Benchmark for Virtualized Systems. Technical report, VMware.

[Hoecker et al., 1984] Hoecker, H., Itzfeld, W.-D., Schmidt, M., and Timm, M. (1984). *Comparative descriptions of software qualtiy measures*. Gesellschaft fÃijr Mathematik und Datenverarbeitung.

[Huber et al., 2010] Huber, N., Von Quast, M., Brosig, F., and Kounev, S. (2010). Analysis of the performance-influencing factors of virtualization platforms. In *On the Move to Meaningful Internet Systems, OTM 2010*, pages 811–828. Springer.

[Huber et al., 2011] Huber, N., von Quast, M., Hauck, M., and Kounev, S. (2011). Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. In *CLOSER*, pages 563–573.

[Hung and Danson, 2013] Hung, J. and Danson, N. (2013). From Slow To Fast: Improving Performance On Intuit Web Sites By Up To 5x.

[IBM, 2010] IBM (2010). Dispelling the vapor around cloud computing. Technical report, IBM Financial Services.

[IEEE, 1998] IEEE (1998). IEEE Std 1061-1998. *IEEE Standard for a Software Quality Metrics Methodology*.

[IEEE, 2006] IEEE (2006). IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks. *IEEE Std 802.1Q-2005 (Incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)*.

[Internap, 2014] Internap (2014). Cloud Landscape Report: Price & Performance. Technical report, Internap.

[Iosup et al., 2011a] Iosup, A., Ostermann, S., Yigitbasi, M., Prodan, R., Fahringer, T., and Epema, D. (2011a). Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945.

[Iosup et al., 2011b] Iosup, A., Yigitbasi, N., and Epema, D. (2011b). On the Performance Variability of Production Cloud Services. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM*, pages 104–113.

[Islam et al., 2012] Islam, S., Lee, K., Fekete, A., and Liu, A. (2012). How a consumer can measure elasticity for cloud platforms. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*, page 85, New York, New York, USA. ACM Press.

[ISO/IEC, 2011a] ISO/IEC (2011a). ISO/IEC 14756 - Measurement and rating of performance of computer-based software systems. Technical report, International Organization for Standardization/International Electrotechnical Commission.

[ISO/IEC, 2011b] ISO/IEC (2011b). ISO/IEC 25010 - Systems and software Quality Requirements and Evaluation (SQuaRE). Technical report, International Organization for Standardization/International Electrotechnical Commission.

[Iyer et al., 2001] Iyer, R., Tewari, V., and Kant, K. (2001). Overload control mechanisms for web servers. In *Performance and QoS of Next Generation Networking*, pages 225–244. Springer.

[Jackson, 2011] Jackson, K. L. (2011). The Economic Benefit of Cloud Computing.

[Jacobs and Aulbach, 2007] Jacobs, D. and Aulbach, S. (2007). Ruminations on Multi-Tenant Databases. In *BTW*, volume 103, pages 514–521.

[Jain, 1991] Jain, R. (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley.

[Janert, 2013] Janert, P. K. (2013). *Feedback Control for Computer Systems*. O'Reilly Media, Inc.

[Jex, 1998] Jex, S. M. (1998). *Stress and job performance: Theory, research, and implications for managerial practice.* Sage Publications Ltd.

[JOptimizer, 2014] JOptimizer (2014). JOptimizer Version 3.4.0. `http://www.joptimizer.com`.

[Jupiterresearch and Akamai, 2006] Jupiterresearch and Akamai (2006). RETAIL WEB SITE PERFORMANCE - Consumer Reaction to a Poor Online Shopping Experience. Technical report, Jupiterresearch.

[Kabbedijk et al., 2015] Kabbedijk, J., Bezemer, C., Jansen, S., and Zaidman, A. (2015). Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective. *Journal of Systems and Software*, 100:139–148.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82:35–45.

[Karl-Dieter and Oliver, 2012] Karl-Dieter, T. and Oliver, R. (2012). *Keine Panik vor Regelungstechnik*. Springer.

[Karlsson et al., 2005] Karlsson, M., Karamanolis, C., and Zhu, X. (2005). Triage: Performance differentiation for storage systems using adaptive control. *Transactions on Storage*, 1(4):457–480.

[Keller and Krueger, 2002] Keller, H. and Krueger, S. (2002). *ABAP Objects*. PEARSON EDUCATION LIMITED.

[Koen, 2008] Koen, R. (2008). *Aspects of MCDA classification and sorting methods*. dissertation, University of South Africa (UNISA).

[Koh et al., 2007] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C. (2007). An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 200–209.

[Kounev, 2011] Kounev, S. (2011). Performance Engineering of Enterprise Software Systems. - 7 Benchmarking. Lecture Notes: Karlsruhe Institute of Technology.

[Kounev et al., 2011] Kounev, S., Bender, K., Brosig, F., Huber, N., and Okamoto, R. (2011). Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics. In *4th International ICST Conference on Simulation Tools and Techniques*, pages 27–36, Brussels, Belgium, Belgium. ICST.

[Kounev et al., 2010] Kounev, S., Brosig, F., Huber, N., and Reussner, R. (2010). Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems. In *IEEE International Conference on Services Computing (SCC)*, pages 621–624.

[Kounev and Dutz, 2009] Kounev, S. and Dutz, C. (2009). QPME - A Performance Modeling Tool Based on Queueing Petri Nets. *ACM SIGMETRICS Performance Evaluation Review (PER), Special Issue on Tools for Computer Performance Modeling and Reliability Analysis*, 36(4):46–51.

[Koziolek, 2010] Koziolek, H. (2010). Towards an Architectural Style for Multi-tenant Software Applications. In *Proceedings on Software Engineering (SE'10)*, volume 159 of *LNI*, pages 81–92. GI.

[Koziolek, 2011] Koziolek, H. (2011). The SPOSAD Architectural Style for Multi-tenant Software Applications. In *Proceedings of the 9th Working IEEE/IFIP Conf. on Software Architecture (WICSA'11)*, pages 320–327. IEEE.

[Kraft et al., 2009] Kraft, S., Pacheco-Sanchez, S., Casale, G., and Dawson, S. (2009). Estimating Service Resource Consumption from Response Time Measurements. In *Proceedings of the*

Fourth International ICST Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '09, pages 48:1–48:10. ICST.

[Krebs and Loesch, 2014] Krebs, R. and Loesch, M. (2014). Comparison of Request Admission Based Performance Isolation Approaches in Multi-Tenant SaaS Applications. In *Proceedings of 4th International Conference On Cloud Computing And Services Science (CLOSER)*. SciTePress.

[Krebs et al., 2014a] Krebs, R., Loesch, M., and Kounev, S. (2014a). Platform-as-a-Service Architecture for Performance Isolated Multi-Tenant Applications. In *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE Cloud 2014)*. IEEE.

[Krebs and Mehta, 2013] Krebs, R. and Mehta, A. (2013). A Feedback Controlled Scheduler for Performance Isolation in Multi-tenant Applications. In *Proceedings of the 3rd IEEE International Conference on Cloud and Green Computing (CGC 2013)*.

[Krebs et al., 2012a] Krebs, R., Momm, C., and Kounev, S. (2012a). Architectural Concerns in Multi-Tenant SaaS Applications. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*. SciTePress.

[Krebs et al., 2012b] Krebs, R., Momm, C., and Kounev, S. (2012b). Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In Buhnova, B. and Vallecillo, A., editors, *Proceedings of the 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012)*, pages 91–100, New York, USA. ACM Press.

[Krebs et al., 2014b] Krebs, R., Momm, C., and Kounev, S. (2014b). Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. *Elsevier Science of Computer Programming Journal (SciCo)*, Vol. 90, Part B:116–134.

[Krebs et al., 2014c] Krebs, R., Schneider, P., and Herbst, N. (2014c). Optimization Method for Request Admission Control to Guarantee Performance Isolation. In *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM.

[Krebs et al., 2014d] Krebs, R., Spinner, S., Ahmed, N., and Kounev, S. (2014d). Resource Usage Control In Multi-Tenant Applications. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014)*. IEEE/ACM.

[Krebs et al., 2013] Krebs, R., Wert, A., and Kounev, S. (2013). Multi-Tenancy Performance Benchmark for Web Application Platforms. In *Proceedings of the 13th International Conference on Web Engineering (ICWE 2013)*. Aalborg University, Denmark, Springer-Verlag.

[Kumar et al., 2009] Kumar, D., Tantawi, A., and Zhang, L. (2009). Real-Time Performance Modeling for Adaptive Software Systems. *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools*.

[Kupperberg et al., 2011] Kupperberg, M., Herbst, N., Kistowski, J., and Reussner, R. (2011). Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms. Technical report, Karlsruhe Institute of Technology.

[Lang et al., 2012] Lang, W., Shankar, S., Patel, J. M., and Kalhan, A. (2012). Towards multi-tenant performance SLOs. In *28th International Conference on Data Engineering (ICDE)*, pages 702–713. IEEE.

[L'Ecuyer and Buist, 2005] L'Ecuyer, P. and Buist, E. (2005). Simulation in java with SSJ. In *Proceedings of the 37th conference on Winter simulation*, WSC '05, pages 611–620. Winter Simulation Conference.

[Lenk et al., 2009] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. (2009). What's Inside the Cloud? An Architectural Map of the Cloud Landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 23–31, Washington, DC, USA. IEEE Computer Society.

[Li et al., 2008] Li, X. H., Liu, T., Li, Y., and Chen, Y. (2008). SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment. In *6th International Conference on Service-Oriented Computing - ICSOC*, pages 649–663.

[Lin et al., 2009] Lin, H. L. H., Sun, K. S. K., Zhao, S. Z. S., and Han, Y. H. Y. (2009). Feedback-Control-Based Performance Regulation for Multi-Tenant Applications. In *15th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 134–141. IEEE.

[Little, 1961] Little, J. D. (1961). A Proof for the Queuing Formula: L= Îż W. *Operations Research*, 9(3):383–387.

[Liu et al., 2007] Liu, X., Zhu, X., and Padala, P. (2007). Optimal multivariate control for differentiated services on a shared hosting platform. In *46th IEEE Conference on Decision and Control*, pages 3792–3799.

[Loesch and Krebs, 2013] Loesch, M. and Krebs, R. (2013). Conceptual Approach for Performance Isolation in Multi-Tenant Systems. In *Proceedings of the 3rd International Conference on Cloud Computing and Service Science (CLOSER 2013)*. RWTH Aachen, Germany, SciTePress.

[Loesch and Krebs, 2014] Loesch, M. and Krebs, R. (2014). Locations for Performance Ensuring Admission Control in Load Balanced Multi-Tenant Systems. In Springer, editor, *Communications in Computer and Information Science - Third International Conference, CLOSER 2013, Aachen, Germany, May 8-10, 2013, Selected Papers*.

[Lu et al., 2006] Lu, C., Lu, Y., Abdelzaher, T. F., Stankovic, J. A., and Son, S. H. (2006). Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):1014–1027.

[Marco et al., 2012] Marco, V., Henrique, M., Kai, S., and Samuel, K. (2012). *Resilience Benchmarking*, pages 283–301. Springer Berlin Heidelberg.

[Marzolla et al., 2012] Marzolla, M., Ferretti, S., and D'Angelo, G. (2012). Dynamic Resource Provisioning for Cloud-based Gaming Infrastructures. *Comput. Entertain.*, 10(3):4:1–4:20.

[Matthews et al., 2007] Matthews, J. N., Hu, W., Hapuarachchi, M., Deshane, T., Dimatos, D., Hamilton, G., McCabe, M., and Owens, J. (2007). Quantifying the Performance Isolation Properties of Virtualization Systems. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, New York, NY, USA. ACM.

[McCall, 1977] McCall, J. (1977). *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*. General Electric.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. Technical Report Special Publication 800-145, NIST.

[Menascé et al., 1994] Menascé, D. A., Almeida, V. A. F., and Dowdy, L. W. (1994). *Capacity Planning and Performance Modeling: From Mainframes to Client-server Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Menasce et al., 2004] Menasce, D. A., Dowdy, L. W., and Almeida, V. A. (2004). *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Menasce and Virgilio, 2000] Menasce, D. A. and Virgilio, A. F. A. (2000). *Scaling for E Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[Mietzner et al., 2008] Mietzner, R., Leymann, F., and Papazoglou, M. P. (2008). Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns. In *2008 Third International Conference on Internet and Web Applications and Services*, pages 156–161. IEEE.

[Mietzner et al., 2009] Mietzner, R., Unger, T., Titze, R., and Leymann, F. (2009). Combining Different Multi-tenancy Patterns in Service-Oriented Applications. *2009 IEEE International Enterprise Distributed Object Computing Conference*, pages 131–140.

[Miller, 1968] Miller, R. B. (1968). Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277. ACM.

[Momm and Krebs, 2011] Momm, C. and Krebs, R. (2011). A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings. In Reussner, R. and Pretschner,

Alexander amd Jähnichen, S., editors, *Proceedings of the Software Engineering 2011 – Workshopband (ESoSyM-2011)*, pages 139–150, Bonn-Buschdorf, Germany. Fachgruppe OOSE der Gesellschaft für Informatik und ihrer Arbeitskreise, Bonner Köllen Verlag.

[Monagan et al., 2005] Monagan, M. B., Geddes, K. O., Heal, K. M., Labahn, G., Vorkoetter, S. M., McCarron, J., and DeMarco, P. (2005). *Maple 10 Programming Guide*. Maplesoft, Waterloo ON, Canada.

[Moy, 1994] Moy, J. (1994). RFC 1583: OSPF Version 2. `http://www.ietf.org/rfc/rfc1583.txt`.

[Musabbir et al., 2013] Musabbir, S., Krishnamurthy, D., and Casale, G. (2013). RPO: Runtime web server optimization under simultaneous multithreading. In *IFIP/IEEE International Symposium onIntegrated Network Management (IM 2013)*, pages 85–92.

[Natis, 2012] Natis, Y. (2012). Gartner Reference Model for Elasticity and Multitenancy. Gartner report, Gartner.

[Nelson et al., 2009] Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness Functions in Evolutionary Robotics: A Survey and Analysis. *Robot. Auton. Syst.*, 57(4):345–370.

[Nichols et al., 1998] Nichols, K., Blake, S., Baker, F., and Black, D. (1998). RFC 2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Technical report, IETF.

[Nielsen, 1994] Nielsen, J. (1994). *Usability engineering*. Elsevier.

[Noorshams, 2014] Noorshams, O.-Q. (2014). *Modeling and Prediction of I/O Performance in Virtualized Environments*. PhD thesis, Karlsruher Institute of Technology. defendet, not yet published.

[Noorshams et al., 2013] Noorshams, Q., Kounev, S., and Reussner, R. (2013). Experimental evaluation of the performance-influencing factors of virtualized storage systems. In *Computer Performance Engineering*, pages 63–79. Springer.

[OASIS, 2012] OASIS (2012). Content Management Interoperability Services (CMIS) Version 1.1. Technical report, OASIS Committee.

[ONF, 2013] ONF (2013). OpenFlow Switch Specification Version 1.4.0. Technical report, Open Network Foundation.

[Oracle, 2015] Oracle (2015). Java SE 6.0 API: Interface ThreadMXBean. `http://docs.oracle.com/javase/6/docs/api/java/lang/management/ThreadMXBean.html`. last accessed: 06. March 2015.

[Pacifici et al., 2008] Pacifici, G., Segmuller, W., Spreitzer, M., and Tantawi, A. (2008). CPU demand for web serving: Measurement analysis and dynamic estimation. *Performance Evaluation*, 65(6-7):531–553.

[Padala et al., 2009] Padala, P., Hou, K.-Y., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., and Merchant, A. (2009). Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 13–26, New York, NY, USA. ACM.

[Pors et al., 2013] Pors, M., Blom, L., Kabbedijk, J., and Jansen, S. (2013). Sharing is Caring - A Decision Support Model for Multi-Tenant Architectures.

[Qin and Wang, 2005] Qin, W. and Wang, Q. (2005). Feedback performance control for computer systems: an LPV approach. In *Proceedings of the 2005 American Control Conference*.

[Rolia et al., 2010] Rolia, J., Kalbasi, A., Krishnamurthy, D., and Dawson, S. (2010). Resource demand modeling for multi-tier services. In *WOSP/SIPEW 10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*. ACM.

[Rolia and Vetland, 1995] Rolia, J. and Vetland, V. (1995). *Parameter estimation for performance models of distributed application systems*, page 54. IBM Press.

[Ross, 2006] Ross, S. M. (2006). *Introduction to Probability Models, Ninth Edition*. Academic Press, Inc., Orlando, FL, USA.

[Ruehl, 2013] Ruehl, S. (2013). *Mixed Tenancy - A hybrid Approach between Single and Mulit-Tenancy*. PhD thesis, Technical University Clausthal.

[Runeson et al., 2012] Runeson, P., Host, M., Rainer, A., and Regnell, B. (2012). *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Blackwell.

[Saaty, 2000] Saaty, T. (2000). *Fundamentals of Decision Making and Priority Theory With the Analytic Hierarchy Process*. AHP series. RWS Publications.

[SAP SE, 2014] SAP SE (2014). Multitenant Applications. `https://help.hana.ondemand.com/help/frameset.htm?54a76156cd114e5d928642b8dde47b91.html`. last accessed: 25. Nov. 2014.

[Schad et al., 2010] Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. (2010). Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *Proceedings of the VLDB Endowment*, 3(1-2):460–471.

[Schneider, 2014] Schneider, P. (2014). Performance Isolation Controller for Multi-Tenant Applications. Master's thesis, Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany. Advisor: Rouven Krebs.

[Schonfeld, 2009] Schonfeld, E. (2009). The Efficient Cloud: All Of Salesforce Runs On Only 1,000 Servers. http://techcrunch.com/2009/03/23/the-efficient-cloud-all-of-salesforce-runs-on-only-1000-servers.

[Schroeter, 2013] Schroeter, J. (2013). *Feature-based configuration management of reconfigurable cloud applications*. PhD thesis, Technische Universitaet Dresden.

[Schroeter et al., 2012] Schroeter, J., Cech, S., and Götz, S. (2012). Towards Modeling a Variable Architecture for Multi-Tenant SaaS-Applications. In *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*, pages 111–120.

[Schuller, 2009] Schuller, S. (2009). What if Salesforce.com were not multi-tenant? http://www.saasblogs.com/business/what-if-salesforcecom-werent-multi-tenant. last accessed: 23. November 2014.

[Schurman and Brutlag, 2009] Schurman, E. and Brutlag, J. (2009). The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search.

[Sen and Yang, 2011] Sen, P. and Yang, J.-B. (2011). *Multiple Criteria Decision Support in Engineering Design*. Springer.

[Shing Wai Chan, 2013] Shing Wai Chan, R. M. (2013). Java Servlet Specification Version 3.1. Technical report, ORACLE.

[Shivam et al., 2008] Shivam, P., Marupadi, V., Chase, J., Subramaniam, T., and Babu, S. (2008). Cutting Corners: Workbench Automation for Server Benchmarking. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 241–254, Berkeley, CA, USA. USENIX Association.

[Shue et al., 2012] Shue, D., Freedman, M. J., and Shaikh, A. (2012). Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 349–362, Berkeley, CA, USA. USENIX Association.

[Smith and Williams, 1993] Smith, C. U. and Williams, L. G. (1993). Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives. *IEEE Trans. Softw. Eng.*, 19(7):720–741.

[Smith, 2011] Smith, D. (2011). Hype Cycle for Cloud Computing, 2011. Technical report, Gartner. Gartner Reference ID: G00214915.

[Smith, 1988] Smith, J. E. (1988). Characterizing Computer Performance with a Single Number. *Communications of the ACM*, 31(10):1202–1206.

[Sommerville, 2006] Sommerville, I. (2006). *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Sousa and Machado, 2012] Sousa, F. R. and Machado, J. C. (2012). Towards Elastic Multi-Tenant Database Replication with Quality of Service. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 168–175, Washington, DC, USA. IEEE Computer Society.

[SPEC, 2012] SPEC (2012). SPECjEnterprise2010 version 1.03. `https://www.spec.org/jEnterprise2010`. last accessed: 2. April 2015.

[SPEC, 2013] SPEC (2013). SPECvirt. `http://www.spec.org/benchmarks.html#virtual`. last accessed: 02. April 2015.

[SPEC, 2015] SPEC (2015). Standard Performance Evaluation Corporation Glossary. `https://www.spec.org/spec/glossary`. last accessed: 14. February 2015.

[Spinner, 2011] Spinner, S. (2011). Evaluating Approaches to Resource Demand estimation. Master's thesis, Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany.

[Stallings, 2008] Stallings, W. (2008). *Operating Systems: Internals and Design Principles*. Prentice Hall Press, Upper Saddle River, NJ, USA, 6th edition. Part 4: Scheduling.

[Strauch et al., 2013] Strauch, S., Andrikopoulos, V., Gomez Saez, S., and Leymann, F. (2013). ESB MT: A Multi-tenant Aware Enterprise Service Bus. *International Journal of Next-Generation Computing*, 4(3):230–249.

[Suleiman and Venugopal, 2013] Suleiman, B. and Venugopal, S. (2013). Modeling Performance of Elasticity Rules for Cloud-Based Applications. In *17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 201–206.

[Tanenbaum and Wetherall, 2011] Tanenbaum, A. S. and Wetherall, D. J. (2011). *Computer Networks*. Prentice Hall, 5th edition.

[TPC, 2002] TPC (2002). TPC BENCHMARK W. Technical report, Transaction Processing Performance Council.

[van Solingen et al., 2002] van Solingen, R., Basili, V., Caldiera, G., and Rombach, H. D. (2002). *Goal Question Metric (GQM) Approach*. John Wiley and Sons, Inc.

[van Solingen and Berghout, 2001] van Solingen, R. and Berghout, E. (2001). Integrating Goal-Oriented Measurement in Industrial Software Engineering: Industrial Experiences with and Additions to the Goal/Question/Metric Method (GQM). In *IEEE METRICS*, page 246.

[VMWare, 2012] VMWare (2012). Whitepaper: Implementing Cloud Solutions Within Your Existing IT Environment.

[von Kistowski et al., 2015] von Kistowski, J., Arnold, J. A., Huppler, K., Lange, K.-D., Henning, J. L., and Cao, P. (2015). How to Build a Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*, ICPE '15, New York, NY, USA. ACM.

[Walraven et al., 2012] Walraven, S., Monheim, T., Truyen, E., and Joosen, W. (2012). Towards Performance Isolation in Multi-tenant SaaS Applications. In *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, MW4NG '12, New York, NY, USA. ACM.

[Walsh et al., 2004] Walsh, W. E., Tesauro, G., Kephart, J. O., and Das, R. (2004). Utility Functions in Autonomic Systems. In *International Conferance on Autonomic Computing*, pages 70–77.

[Wang et al., 2012] Wang, W., Huang, X., Qin, X., Zhang, W., Wei, J., and Zhong, H. (2012). Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 439–446. IEEE.

[Wang et al., 2008] Wang, Z. H., Guo, C. J., Gao, B., Sun, W., Zhang, Z., and An, W. H. (2008). A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing. In *IEEE International Conference on e-Business Engineering*, pages 94 –101.

[Weissman and Bobrowski, 2009] Weissman, C. D. and Bobrowski, S. (2009). The design of the force. com multitenant internet application development platform. In *SIGMOD Conference*, pages 889–896.

[Welsh and Culler, 2003] Welsh, M. and Culler, D. (2003). Adaptive Overload Control for Busy Internet Servers. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 4–4, Berkeley, CA, USA. USENIX Association.

[Wert et al., 2012] Wert, A., Happe, J., and DennisWestermann (2012). Integrating software performance curves with the palladio component model. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pages 283–286. ACM.

[Westermann, 2013] Westermann, D. (2013). *Deriving Goal-oriented Performance Models by Systematic Experimentation*. PhD thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

[Westermann et al., 2010] Westermann, D., Happe, J., Hauck, M., and Heupel, C. (2010). The Performance Cockpit Approach: A Framework for Systematic Performance Evaluations. In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*, pages 31–38. IEEE Computer Society.

[Westermann and Momm, 2010] Westermann, D. and Momm, C. (2010). Using software performance curves for dependable and cost-efficient service hosting. In *Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems*, QUASOSS '10, pages 3:1–3:6, New York, NY, USA. ACM.

[wikimedia, 2015] wikimedia (2015). Page view statistics for Wikimedia projects. `http://dumps.wikimedia.org/other/pagecounts-raw`. last accessed: 09. March 2015.

[Wilder, 2012] Wilder, B. (2012). *Cloud Architecture Patterns*. O'Reilly & Associates.

[Wolff, 1982] Wolff, R. W. (1982). Poisson Arrivals See Time Averages. *Operations Research*, 30(2):223–231.

[Xen, 2012] Xen (2012). Xen.org. `http://xen.org`.

[Xen Project, 2015] Xen Project (2015). Credit scheduler. `http://wiki.xenproject.org/wiki/Credit_Scheduler`. last accessed: 16. February 2015.

[Xi et al., 2011] Xi, S., Wilson, J., Lu, C., and Gill, C. (2011). Rt-xen: towards real-time hypervisor scheduling in xen. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 39–48. IEEE.

[Yaish and Goyal, 2013] Yaish, H. and Goyal, M. (2013). Multi-tenant Database Access Control. In *Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering*, CSE '13, pages 870–877, Washington, DC, USA. IEEE Computer Society.

[Young and Smith, 2006] Young, J. and Smith, S. (2006). Akamai and JupiterResearch Identify '4 Seconds' as the New Threshold of Acceptability for Retail Web Page Response Times. `http://www.akamai.com/html/about/press/releases/2006/press_110606.html`. last accessed: 02. February 2015.

[Zeng et al., 2013] Zeng, L., Wang, Y., Shi, W., and Feng, D. (2013). An Improved Xen Credit Scheduler for I/O Latency-Sensitive Applications on Multicores. In *Proceedings of the 2013 International Conference on Cloud Computing and Big Data*, CLOUDCOM-ASIA '13, pages 267–274, Washington, DC, USA. IEEE Computer Society.

[Zhang et al., 2007] Zhang, Q., Cherkasova, L., and Smirni, E. (2007). *A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications*, page 27ff. IEEE Computer Society.

[Zhang et al., 2010] Zhang, Y., Wang, Z., Gao, B., Guo, C., Sun, W., and Li, X. (2010). An effective heuristic for on-line tenant placement problem in saas. *Web Services, IEEE International Conference on*, 0:425–432.

[Zheng et al., 2008] Zheng, T., Woodside, M., and Litoiu, M. (2008). Performance Model Estimation and Tracking Using Optimal Filters. *IEEE Transactions on Software Engineering*, 34(3):391?406.

[Zhu et al., 2009] Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., and Shin, K. (2009). What does control theory bring to systems research? *ACM SIGOPS Operating Systems Review*, 43(1):62.

[Zimmermann, 1980] Zimmermann, H. (1980). OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection. *Communications, IEEE Transactions on*, 28(4):425–432.

A. Call Characteristics of Document Service

A.1. Analysis of Call Probability

Top 15 Transitions for Document Service									
	1	2	3	4	5	6	7	8	9
StartState	0,030	0,008	0,804	0,008	0,137	0,004	0,002	0,000	0,005
1: getChildren()	0,543	0,001	0,450	0,000	0,001	0,002	0,003	0,000	0,000
2: getContentStream()	0,067	0,760	0,135	0,000	0,000	0,000	0,000	0,000	0,038
3: getObject()	0,373	0,001	0,378	0,038	0,002	0,013	0,014	0,001	0,177
4: getObjectByPath()	0,000	0,000	0,922	0,056	0,002	0,018	0,000	0,003	0,000
5: getRepositoryInfos()	0,013	0,000	0,567	0,006	0,014	0,133	0,265	0,000	0,001
6: getTypeDefinition()	0,073	0,000	0,448	0,012	0,243	0,178	0,038	0,000	0,001
7: query()	0,039	0,000	0,819	0,000	0,110	0,015	0,013	0,000	0,003
8: setContentStream()	0,047	0,000	0,860	0,070	0,000	0,000	0,000	0,023	0,000
9: getObjectParents()	0,001	0,000	0,978	0,000	0,000	0,000	0,000	0,000	0,003
Average	0,119	0,077	0,636	0,019	0,051	0,036	0,034	0,003	0,023

Services With an Overall Call Propability > 0,005				
	1	2	3	4
StartState	0,031	0,008	0,821	0,140
1: getChildren()	0,546	0,001	0,453	0,001
2: getContentStream()	0,070	0,790	0,140	0,000
3: getObject()	0,495	0,001	0,501	0,002
4: getRepositoryInfos()	0,022	0,000	0,954	0,024

Figure A.1.: Limited view onto the call probability of the Document Service.

A.2. Timing Behavior of Calls

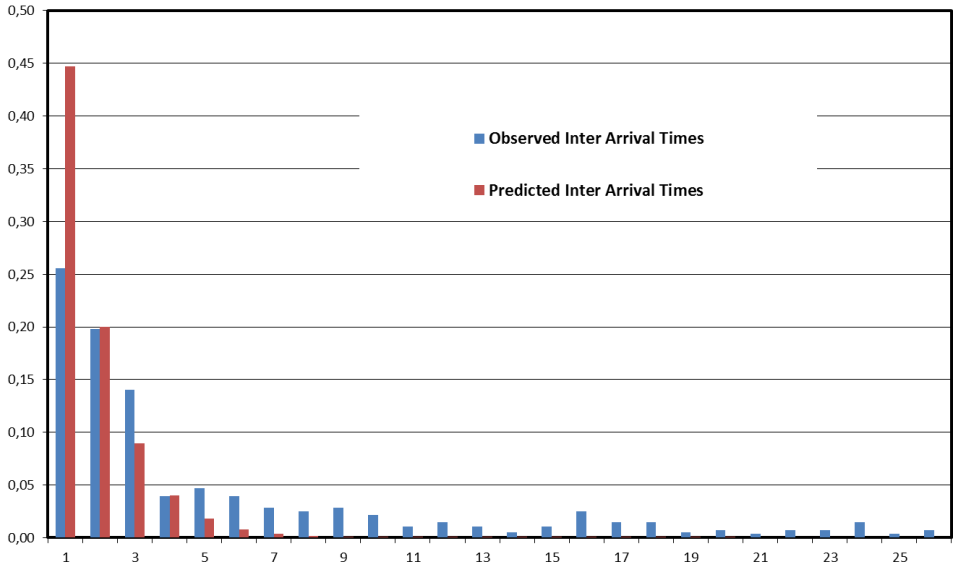


Figure A.2.: Filtered probability density of bulk inter arrival times.

The data was normalized. The real time span between two bulks of the same user are described by multiplying the x-axis value, with a constant factor.

B. AHP Weighting Matrix

	Guarantee & Quota	Input Aware (Black List)	Input Neutral (Round Robin)	Model Based (Analytical)	Model Based (Simulation)	Resource Exploiting (Estimation)	Resource Exploiting (Measurement)	Resource Isolating (Estimation)	Resource Isolating (Measurement)
Guarantee & Quota	1	0,33	0,33	2,00	5,00	3,00	1,00	3,00	1,00
Input Aware (Black List)	3,00	1	1,00	3,00	7,00	5,00	3,00	5,00	3,00
Input Neutral (Round Robin)	3,00	1,00	1	3,00	7,00	5,00	3,00	5,00	3,00
Model Based (Analytical)	0,50	0,33	0,33	1	5,00	3,00	1,00	3,00	1,00
Model Based (Simulation)	0,20	0,14	0,14	0,20	1	0,33	0,17	0,33	0,17
Resource Exploiting (Estimation)	0,33	0,20	0,20	0,33	3,00	1	0,33	1,00	0,33
Resource Exploiting (Measurement)	1,00	0,33	0,33	1,00	6,00	3,00	1	3,00	1,00
Resource Isolating (Estimation)	0,33	0,20	0,20	0,33	3,00	1,00	0,33	1	0,33
Resource Isolating (Measurement)	1,00	0,33	0,33	1,00	6,00	3,00	1,00	3,00	1

	Guarantee & Quota	Input Aware (Black List)	Input Neutral (Round Robin)	Model Based (Analytical)	Model Based (Simulation)	Resource Exploiting (Estimation)	Resource Exploiting (Measurement)	Resource Isolating (Estimation)	Resource Isolating (Measurement)
Guarantee & Quota	1	0,20	0,20	0,20	0,33	0,33	0,20	0,33	0,20
Input Aware (Black List)	5,00	1	1,00	1,00	3,00	3,00	1,00	3,00	1,00
Input Neutral (Round Robin)	5,00	1,00	1	1,00	3,00	3,00	1,00	3,00	1,00
Model Based (Analytical)	5,00	1,00	1,00	1	3,00	3,00	1,00	3,00	1,00
Model Based (Simulation)	3,00	0,33	0,33	0,33	1	1,00	0,33	1,00	0,33
Resource Exploiting (Estimation)	3,00	0,33	0,33	0,33	1,00	1	0,33	1,00	0,33
Resource Exploiting (Measurement)	5,00	1,00	1,00	1,00	3,00	3,00	1	3,00	1,00
Resource Isolating (Estimation)	3,00	0,33	0,33	0,33	1,00	1,00	0,33	1	0,33
Resource Isolating (Measurement)	5,00	1,00	1,00	1,00	3,00	3,00	1,00	3,00	1

Figure B.1.: Upper table: isolation. Lower table: oscillation. Tables copied from created PDF file using [Bender et al., 2013]

C. Proof of Convexity for System Function

In the following a proof of Lemma 2 is shown. The proof is a minor modified version of [Schneider, 2014].

Proof : For the sake of simplicity, the system function from Equation 4.4 is shortened by subsuming parameters into constants. This results in

$$R_i(w_i) = \frac{a_i}{w_i} + b_i + \sqrt{\left(\frac{a_i}{w_i} + b_i\right)^2 + c_i},$$

with $a_i := O_i m_i/2$, $b_i := (O_i - Z)/2$ and $c_i := O_i Z$, whereat $a_i, c_i > 0$. The derivative of $R_i(w_i)$ is then given by

$$\frac{d}{dw_i} R_i(w_i) = -\frac{a_i}{w_i^2}\left(1 + \frac{\frac{a_i}{w_i} + b_i}{\sqrt{(\frac{a_i}{w_i} + b_i)^2 + c_i}}\right).$$

The fraction inside the bracket is strictly bigger than -1 as shown in the following.

Obviously it holds that

$$\frac{\frac{a_i}{w_i} + b_i}{\sqrt{(\frac{a_i}{w_i} + b_i)^2 + c_i}} > -1.$$

since $c_i > 0$ in a given system which ensures that

$$|\frac{a_i}{w_i} + b_i| < \sqrt{(\frac{a_i}{w_i} + b_i)^2 + c_i}$$

Thus

$$\frac{d}{dw_i} R_i(w_i) = \underbrace{-\frac{a_i}{w_i^2}}_{<0\ \forall w_i > 0} \underbrace{\left(1 + \frac{\frac{a_i}{w_i} + b_i}{\sqrt{(\frac{a_i}{w_i} + b_i)^2 + c_i}}\right)}_{>0} < 0.$$

That proofs that R_i is strictly monotonically decreasing. In the following, the convexity of $R_i(w_i)$ is proved for $w_i \in [0,1]$. For this reason, $\frac{d^2}{d^2 w_i} R_i(w_i) > 0$ is shown which and thus strict

convexity is proved. The second derivative $\frac{d^2}{d^2 w_i} R_i(w_i)$ is given by

$$\underbrace{\frac{2a_i}{w_i^3}}_{>0} \underbrace{\left(1 + \frac{\frac{a_i}{w_i} + b_i}{\sqrt{(\frac{a_i}{w_i} + b_i)^2 + c_i}}\right)}_{>0 \text{ (shown before)}} + \underbrace{\frac{a_i^2}{w_i^4 \sqrt{(\frac{a_i}{w_i} + b_i)^2 + c_i}}}_{>0, \text{ since } c_i > 0} \underbrace{\left(1 - \frac{(\frac{a_i}{w_i} + b_i)^2}{\underbrace{(\frac{a_i}{w_i} + b_i)^2 + c_i}_{<1, \text{ since } c_i > 0}}\right)}_{>0} > 0.$$

Obviously $\lim_{w \to 0} R_j(w) = \infty$ holds. Note that therefore $R_i(w_i)$, as well as its derivatives have singularities at $w_i = 0$. By defining $-\infty < c < \infty$ for each $c \in \mathbb{R}$, the result of this lemma is conserved. □