

# A Service Broker for Intercloud Computing

Zur Erlangung des akademischen Grades eines  
Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Dipl.-Ing. Foued Jrad

aus Monastir

Datum der mündlichen Prüfung: 10. Juli 2014  
Erster Gutachter: Prof. Dr. Achim Streit  
Zweiter Gutachter: Priv.-Doz. Dr. Ivona Brandic



# Abstract

With the advantages of pay-per-use, easy access and on-demand resource customization, the Cloud computing concept was quickly adopted by both the industry and the academia. Over the last decade, the many Cloud infrastructures that have been built have distinguished themselves in the variety of offered services, access interfaces, costs and Service Level Agreement (SLA). On the other hand, “vendor lock-in” issues and the lack of common Cloud standards hinder the interoperability across Cloud providers. Thus, Cloud users have to manually make decisions about which Cloud to choose in order to meet their functional and non-functional service requirements while keeping the payment low. This task is clearly a burden for the users because they have to go through web pages of Cloud providers to compare their services and pricing policies. Furthermore, it is hard for them to collect and maintain all the needed information from current commercial Clouds to make accurate decisions.

The aim of the work presented in this thesis is to facilitate users’ ability to find the most suitable Cloud services by taking their functional and non-functional SLA requirements into account. A key contribution of the thesis is the design of a multi-Cloud service broker framework acting as a mediator between consumers and multiple Cloud providers to automate the service selection and deployment. The framework contains components for decision-making, monitoring and SLA management, as well as an interoperability layer to interact with heterogeneous Clouds.

In addition, the thesis addresses the multi-Cloud service selection problem by proposing a utility-based matching algorithm adopted from the economic theory which selects the Clouds, maximizing the user utility. To facilitate the matching of large-scale composite services, we introduce a hybrid utility-based genetic algorithm (HU-GA) that applies the user utility as an objective function. Furthermore, we present an ontological model to semantically describe the service requirements and Cloud characteristics.

Moreover, the thesis investigates the optimal broker-based deployment of multi-Cloud workflows while taking into account data locality, resource heterogeneity, and task dependencies. For this purpose, a two-stage multi-dimensional resource allocation scheme is proposed, where the suitable Clouds are first selected using the HU-GA matching algorithm and then the workflow tasks are distributed to the selected Cloud

resources using a data-aware scheduling policy. To optimize the Cloud-to-Cloud (Intercloud) data transfer during the workflow execution, a replica-based data management policy is introduced.

To evaluate the proposed service broker framework and the associated matching and scheduling mechanisms, a realistic simulation environment is implemented. Using simulation scenarios with real application workloads and a case study, we demonstrate from the experiment results the benefits of the utility-based matching algorithm in terms of cost and service quality. In addition, we demonstrate the efficiency of the multi-dimensional resource allocation scheme in improving the workflow execution performance and reducing the amount and costs of Intercloud data transfers.

# Zusammenfassung

Durch Vorteile wie Pay-per-use, einfache Zugriffsmethoden und On-demand maßgeschneiderte Ressourcen wurde das Cloud Computing-Konzept schnell von der Industrie und der Wissenschaft angenommen. Die im Laufe des letzten Jahrzehnts entstandenen Cloud-Infrastrukturen weisen eine Vielzahl von angebotenen Services, Zugriffsschnittstellen, Kostenmodelle sowie Service Level Agreement (SLA). Trotzdem behindern Probleme wie das "Vendor Lock-in" sowie das Fehlen von Standards die Interoperabilität zwischen den Cloud-Anbietern. Bei der Auswahl eines passenden Cloud-Anbieters bezüglich funktionaler, nicht-funktionaler sowie monetärer Anforderungen müssen Cloud-Nutzer daher manuelle Entscheidungen treffen. Diese Aufgabe stellt eine große Herausforderung für sie dar, da zunächst die Webseiten der Cloud-Anbieter durchsucht werden müssen, um angebotene Dienste und Preismodelle überhaupt vergleichen zu können. Außerdem ist es schwer, alle benötigten Informationen kommerzieller Anbieter zu sammeln und stets aktuell zu halten, um adäquate Entscheidungen zu treffen.

Die vorliegende Dissertation hat zum Ziel, die Cloud-Nutzer bei der Suche der bestgeeigneten Cloud-Ressourcen unter Erfüllung ihrer funktionellen und nicht-funktionalen Anforderungen zu unterstützen. Ein Hauptbeitrag dieser Arbeit ist die Entwicklung eines multi-Cloud Service-Broker-Frameworks, das als Vermittler zwischen Nutzern und verschiedenen Cloud-Anbietern agiert, um die Auswahl und Bereitstellung von Diensten zu automatisieren. Das Framework beinhaltet Komponenten zur Entscheidungsfindung, zum Monitoring, SLA-Management sowie eine Interoperabilitätsschicht zur Interaktion mit heterogenen Clouds.

Außerdem beschäftigt sich diese Dissertation mit dem Problem der Dienstauswahl in multi-Cloud Umgebungen. Hierzu wird ein aus der Wirtschaftstheorie übernommenes nutzenorientiertes Matching-Verfahren vorgestellt, das die geeigneten Cloud-Services unter Maximierung des Kundennutzens auswählt. Um die Kompositionen von Diensten zu erleichtern, präsentieren wir einen hybriden nutzenorientierten genetischen Algorithmus (HU-GA), der das Kundennutzen als Zielfunktion anwendet. Darüber hinaus präsentieren wir ein ontologisches Modell zur semantischen Beschreibung der SLA-Dienstanforderungen und der Cloud-Eigenschaften.

Des Weiteren untersucht diese Dissertation die optimale Broker-basierte Ausführung von multi-Cloud Workflows unter Berücksichtigung der Datenlokalität, Ressourcen-Heterogenität und Abhängigkeiten innerhalb von Tasks. Hierzu, wird ein zweistufiges mehrdimensionales Ressourcen-Allokationsverfahren vorgestellt: zunächst wer-

den geeignete Cloud-Services mit dem HU-GA Matching Algorithmus ausgewählt und anschließend die Workflow-Tasks mit einem Daten-orientierten Scheduler auf den ausgewählten Cloud-Ressourcen verteilt. Um die Cloud-zu-Cloud (Intercloud) Datenübertragung während der Workflow-Ausführung zu optimieren, wird eine Replika-basierte Daten-Management-Policy eingeführt.

Um das entwickelte Service-Broker-Framework und die dazugehörigen Matching und Scheduling Mechanismen zu evaluieren, wurde eine realistische Simulationsumgebung implementiert. Mit Hilfe von modellierten Simulationsszenarien mit realen Anwendungsworkloads und einer Fallstudie demonstrieren wir, basierend auf den Ergebnissen, die Kundenvorteile durch den Einsatz des nutzenorientierten Matching-Verfahrens hinsichtlich Kosten und Dienstqualität. Abschließend demonstrieren wir die Effizienz des mehrdimensionalen Ressourcen-Allokationsverfahrens hinsichtlich der Verbesserung der Workflow-Performanz und der Reduzierung der Menge und Kosten der Intercloud Datenübertragungen.

# Acknowledgments

There are a number of people without whom this thesis might not have been written and to whom I would like to give my sincere thanks.

First, I would like to express my deepest gratitude to my supervisor Prof. Achim Streit who gave me the opportunity to pursue my PhD and for his constructive guidance throughout this work, which showed me how research can be. I am also very thankful to Dr. Ivona Brandic for accepting to be my co-supervisor and for hosting me three months in her research group at the Vienna University of Technology. She has provided me with constructive feedback during this period and given me the support needed to pursue my research. I am particularly grateful to my colleague Dr. Jie Tao for her patience and for her support towards my research work, which helped to improve the quality of this work.

Furthermore, I would like to thank the Karlsruhe House of Young Scientists (KHYS) for funding my very productive research stay abroad at the Vienna University of Technology, which gave me the opportunity to collaborate with the nice people from the distributed systems group members especially Drazen Lucanin and Soodeh Farokhi.

I am also very thankful to Dr. Christoph M. Flath and Rico Knapper from the institute of information systems and marketing at the KIT Department of economics and management for helping me to understand the economic background needed to write a co-published paper.

I would like to thank Weiwei Chen from the University of Southern California for his contribution to this work by providing the WorkflowSim source code.

I am deeply grateful to my family for supporting me during this long journey. My daughter was born just before starting my PhD and my son was born when I was at the end stage. As they grew up, this thesis did also. This thesis is dedicated to you. Last and most importantly, I would like to thank my beloved wife. I would not have been able to go through this without her support. I am sure that it will be even greater from now on.

Foued Jrad  
Mai 2014  
Karlsruhe, Germany





# List of Publications

The contributions of this thesis have been to a degree or completely derived from work published in scientific conferences, journals and workshops. The following list presents the papers used in the composition of this thesis.

## Refereed Publications in Conference Proceedings

- **Foued Jrad**, Jie Tao and Achim Streit. SLA-based service brokering in Intercloud environments. In CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, pages 76–81, SciTePress, 2012.
- **Foued Jrad**, Jie Tao and Achim Streit. Simulation-based evaluation of an intercloud service broker. In CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization, pages 140–145, Nice, 2012 (best paper award).
- **Foued Jrad**, Jie Tao and Achim Streit. A broker-based framework for multi-Cloud workflows. In Proceedings of the 2013 international workshop on multi-Cloud applications and federated Clouds, pages 61–68, ACM, 2013.
- **Foued Jrad**, Jie Tao, Ivona Brandic and Achim Streit. Multi-dimensional resource allocation for data-intensive large-scale Cloud applications. In Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2014), pages 691–702, Barcelona, Spain, April 2014.
- Tobias Sturm, **Foued Jrad** and Achim Streit. Storage CloudSim: A simulation environment for Cloud object storage Infrastructures. In Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2014), pages 186–192, Barcelona, Spain, April 2014.
- Soodeh Farokhi, **Foued Jrad**, Ivona Brandic and Achim Streit. HS4MC: Hierarchical SLA-based service selection for multi-Cloud environments. In Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2014), pages 722–734, Barcelona, Spain, April 2014.

## Refereed Publications in Journals

- **Foued Jrad**, Jie Tao, Rico Knapper, Christoph M. Flath and Achim Streit. A utility-based approach for customised Cloud service selection. International Journal of Computational Science and Engineering, 2013 (in press).

- **Foued Jrad**, Jie Tao, Ivona Brandic and Achim Streit. SLA enactment for large-scale healthcare workflows on multi-Cloud. *Future Generation Computer Systems*, 2014 (in press).

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Problem Statement . . . . .	2
1.2. Research Questions . . . . .	4
1.3. Scientific Contributions . . . . .	7
1.4. Thesis Structure . . . . .	9
<b>2. Background</b>	<b>13</b>
2.1. Cloud Computing . . . . .	13
2.2. Intercloud Computing . . . . .	15
2.3. Cloud Standards . . . . .	16
2.4. Service Level Agreement . . . . .	17
2.5. Cloud Service Brokering . . . . .	18
2.6. Composite Cloud Service . . . . .	19
2.7. Multi-Cloud Workflows . . . . .	20
2.8. Multi-Criteria Decision-Making and Auction Theory . . . . .	21
2.9. Summary . . . . .	23
<b>3. Related Work</b>	<b>25</b>
3.1. Multi-Cloud Orchestration Frameworks . . . . .	25
3.2. Matchmaking of Composite Cloud Services . . . . .	28
3.3. Multi-Cloud Workflow Frameworks . . . . .	32
3.4. Scheduling of data-intensive Workflows on Cloud . . . . .	33
3.5. Cloud Simulation Frameworks . . . . .	34
3.6. Summary . . . . .	37
<b>4. Design and Implementation of a Multi-Cloud Service Broker Framework</b>	<b>39</b>
4.1. Multi-Cloud Service Broker Architectural Design . . . . .	39
4.1.1. Cloud Service Broker . . . . .	40
4.1.2. Provider Intercloud Gateway . . . . .	42
4.1.3. Client . . . . .	42
4.2. SLA-based Service Brokering on Multi-Cloud . . . . .	42
4.2.1. SLA Discovery and Definition . . . . .	42
4.2.2. SLA Negotiation . . . . .	43
4.2.3. SLA Provisioning and Monitoring . . . . .	44
4.3. Ontological Model . . . . .	44
4.3.1. IaaS Composite Service Requester Ontology . . . . .	45

4.3.2.	IaaS Cloud Provider Ontology . . . . .	46
4.4.	Simulation-based Multi-Cloud Service Broker Framework Implementation . . . . .	46
4.4.1.	Implementation Details . . . . .	47
4.4.2.	Simple Simulation Use-Case . . . . .	51
4.5.	Summary . . . . .	53
<b>5.</b>	<b>SLA-based Matchmaking of Composite Cloud Services</b>	<b>55</b>
5.1.	Matchmaking Problem Formulation . . . . .	55
5.2.	Sieving Matching Algorithm . . . . .	58
5.2.1.	Sieving Selection Strategy . . . . .	59
5.2.2.	Sieving Implementation . . . . .	60
5.2.3.	Sieving Algorithm Time Complexity . . . . .	61
5.3.	Utility-based Matching Algorithm . . . . .	61
5.3.1.	Utility-based Selection Strategy . . . . .	62
5.3.2.	Utility-based Algorithm Implementation . . . . .	63
5.3.3.	Utility-based Algorithm Time Complexity . . . . .	65
5.4.	HU-GA Matching Algorithm . . . . .	66
5.5.	Summary . . . . .	68
<b>6.</b>	<b>Optimized Multi-Cloud Workflow Deployment</b>	<b>69</b>
6.1.	Broker-based Multi-Cloud Workflow Deployment . . . . .	69
6.2.	Multi-dimensional Resource Allocation Approach . . . . .	71
6.3.	Data Locality-Driven Scheduling . . . . .	73
6.3.1.	DAS Scheduler . . . . .	73
6.3.2.	DAT Scheduler . . . . .	75
6.4.	Replica-based Data Management . . . . .	75
6.5.	Summary . . . . .	77
<b>7.</b>	<b>Evaluation</b>	<b>79</b>
7.1.	Experimental Setup . . . . .	79
7.1.1.	Modeled Public IaaS Clouds . . . . .	79
7.1.2.	Modeled IaaS Cloud Services . . . . .	81
7.1.3.	CloudSim Setup . . . . .	81
7.1.4.	Workflow Traces . . . . .	82
7.2.	Evaluation of the Multi-Cloud Service Broker Framework . . . . .	85
7.2.1.	Match Maker and Deployment Manager Evaluation . . . . .	85
7.2.2.	Workflow Engine Evaluation . . . . .	88
7.2.3.	Results Discussion . . . . .	92
7.3.	Evaluation of the Matching Algorithms . . . . .	93
7.3.1.	Matching of Single Cloud Services . . . . .	93
7.3.2.	Matching of Composite Cloud Services . . . . .	99
7.3.3.	Case Study: CAD-aaS . . . . .	106
7.3.4.	Results Discussion . . . . .	110

7.4. Evaluation of the Multi-dimensional Resource Allocation . . . . .	111
7.4.1. Simulation Scenario . . . . .	111
7.4.2. Impact of Clustering . . . . .	113
7.4.3. Impact of Resource Allocation on Workflow Makespan . . . . .	113
7.4.4. Impact of Resource Allocation on Intercloud Data Transfer . . . . .	114
7.4.5. Cost Evaluation . . . . .	116
7.4.6. Results Discussion . . . . .	117
7.5. Summary . . . . .	118
<b>8. Conclusion and Outlook</b>	<b>119</b>
8.1. Summary . . . . .	119
8.2. Constraints and Future Work . . . . .	121
8.3. Complementary Research . . . . .	123
<b>Appendix</b>	<b>127</b>
<b>A. Appendix 1</b>	<b>129</b>
A.1. SLA Metrics of the Modeled Public Clouds . . . . .	129
<b>B. Appendix 2</b>	<b>131</b>
B.1. HU-GA Matching Algorithm Convergence . . . . .	131
B.2. Matched Datacenters . . . . .	132
<b>C. Appendix 3</b>	<b>133</b>
C.1. OCCI Monitoring Extensions . . . . .	133
<b>D. Appendix 4</b>	<b>135</b>
D.1. Sample Simulation Output . . . . .	135
<b>Acronyms</b>	<b>137</b>
<b>Bibliography</b>	<b>138</b>
<b>Curriculum Vitae</b>	<b>155</b>



# List of Figures

1.1.	Structure of the thesis and scientific contributions (SC).	10
2.1.	Cloud computing delivery (left) and deployment (right) models.	15
2.2.	Intercloud computing architectures.	16
2.3.	Service brokering concept.	18
2.4.	Cloud service brokering life cycle.	19
2.5.	Sample sequential and parallel workflow patterns presented as DAG.	20
2.6.	The Multi-Criteria Decision-Making (MCDM) process.	22
4.1.	Multi-Cloud service broker architecture.	40
4.2.	The SLA negotiation flow.	44
4.3.	The SLA provisioning and monitoring flow.	45
4.4.	Composite IaaS service requester ontology.	46
4.5.	IaaS Cloud provider ontology.	47
4.6.	Simulation environment.	48
4.7.	Simulation framework class diagram.	49
4.8.	Simulation flow.	52
5.1.	Example of an Intercloud graph for a composite multi-Cloud service.	56
5.2.	Functionality of the sieving algorithm with three SLA parameters.	60
5.3.	Functionality of the utility-based matching algorithm.	64
5.4.	Functionality of the hybrid utility-based genetic algorithm.	66
5.5.	Genetic encoding of the composite service candidates.	67
6.1.	Workflow deployment steps	70
6.2.	Multi-dimensional resource allocation scheme.	72
7.1.	A sample horizontally clustered Montage workflow.	83
7.2.	A sample vertically clustered Epigenomics workflow [96].	84
7.3.	One day broker deployment rate with different datacenter numbers and the random matching policy.	87
7.4.	One day matching rate for different matching policies with 6 datacenters.	88
7.5.	Workflow execution time with different cluster numbers $k$ .	90
7.6.	File transfer overhead for the multi-Cloud case with different cluster numbers $k$ .	91
7.7.	Total size (left) and number (right) of transferred files for the multi-Cloud case with $k=20$ .	91

List of Figures

7.8. Matching rate of the sieving and utility-based algorithms. . . . .	96
7.9. Number of covered providers by the matching algorithms. . . . .	97
7.10. Match percentage per provider after 5000 requests. . . . .	98
7.11. Average cost for 5000 requests of VM type <i>small</i> with linux OS. . . . .	98
7.12. Multi-Cloud DNA sequencing workflow deployment use case. . . . .	100
7.13. HU-GA and sieving time complexity for different VM numbers. . . . .	102
7.14. Unconstrained (U)/constrained (C) HU-GA convergence for different VM numbers. . . . .	102
7.15. Workflow makespan in minutes with HU-GA-and sieving. . . . .	103
7.16. Average aggregated SLA values for unconstrained (U)/constrained (C) HU-GA and sieving for a VM number of 50. . . . .	104
7.17. Total workflow execution cost with unconstrained (U)/constrained (C) HU-GA and sieving for different VM numbers. . . . .	106
7.18. Sample satisfaction function for the availability [52]. . . . .	107
7.19. Composite infrastructure service for a CAD-aaS standard edition [52].	109
7.20. Aggregated QoS values and cost with prospect-based and utility-based selection. . . . .	110
7.21. Workflow makespan and total data transfer time with HU-GA+DAS EU for different cluster numbers $k$ . . . . .	113
7.22. Workflow makespan with different scheduling and matching policies for $k=20$ . . . . .	114
7.23. Total amount of Intercloud transfers with different scheduling and match- ing policies for $k=20$ . . . . .	115
7.24. Percentage of data transfers with different scheduling and matching policies for $k=20$ . . . . .	115
7.25. Total execution cost with sieving (left) and HU-GA (right) matching for $k=20$ . . . . .	116
7.26. Total execution cost with one and two storage Clouds (ST) for $k=20$ . . .	117
B.1. Maximal utility values at the HU-GA convergence for different VM numbers. . . . .	131
B.2. Opt4J convergence plot with 30 VMs for constrained HU-GA sieved (left) and unsieved (right). . . . .	132



# List of Tables

2.1.	An overview of current Cloud standards. . . . .	16
2.2.	Typical functional and non-functional SLA for IaaS. . . . .	17
3.1.	A comparison of existing composite Cloud service selection frameworks with this work. . . . .	31
3.2.	A comparison of open source Cloud simulation tools. . . . .	36
4.1.	Role of the multi-Cloud broker framework components. . . . .	41
5.1.	Multi-Cloud matchmaking model. . . . .	56
5.2.	Description of the used QoS parameters in the matching. . . . .	57
5.3.	Aggregated SLA attributes for a composite service $x$ . . . . .	58
7.1.	Modeled hosts and datacenter setup. . . . .	80
7.2.	Types of the modeled virtual machines. . . . .	82
7.3.	Total number of clustered jobs with different cluster numbers $k$ . . . . .	83
7.4.	Epigenomics workflow trace characteristics. . . . .	85
7.5.	Modeled datacenters configuration. . . . .	86
7.6.	Hosts setup. . . . .	86
7.7.	Modeled synthetic VM request types . . . . .	86
7.8.	User non-functional SLA Requirements for the Montage workflow deployment. . . . .	89
7.9.	Average non-functional SLA values for the matched datacenters. . . . .	92
7.10.	Preferences of DCS customers expressed using fitting functions and relative weights. . . . .	95
7.11.	Non-functional service requirements of customers $i$ for the sieving algorithm. . . . .	95
7.12.	User QoS requirements for the multi-Cloud DNA sequencing workflow deployment. . . . .	100
7.13.	User preferences for the multi-Cloud DNA sequencing workflow deployment expressed using fitting functions and relative weights. . . . .	101
7.14.	Maximal payment for VMs, storage, and traffic for the DNA sequencing workflow deployment. . . . .	101
7.15.	CBR values with unconstrained (U) and constrained (C) HU-GA and sieving for different VM numbers. . . . .	106
7.16.	Comparison between utility and prospect-based selection. . . . .	108

*List of Tables*

7.17. Non-functional meta-SLA requirements of the CAD-aaS standard edition. . . . .	109
7.18. User non-functional SLA requirements for EU and EU-US scenarios. . .	112
7.19. User preferences for EU and EU-US scenarios expressed using fitting functions and relative weights. . . . .	112
7.20. Maximal payment for VMs, storage, and data traffic for EU and EU-US scenarios. . . . .	112
A.1. Modeled datacenters SLA metrics measured from Germany. . . . .	129
B.1. Matched datacenters with unconstrained (U) and constrained (C) HUGA for different VM numbers; ST=storage. . . . .	132

# Listings

5.1. Sieving Matching Algorithm . . . . .	60
5.2. Utility-based Matching Algorithm . . . . .	64
6.1. DAS/DAT Scheduler . . . . .	73
6.2. Function processTaskAffinity . . . . .	74
6.3. Data Management Policy . . . . .	76
C.1. Resourceinformation OCCI Mixin . . . . .	133
C.2. Offertemplate OCCI Mixin . . . . .	134
D.1. Sample Simulation Output . . . . .	135



# 1. Introduction

In the recent years, Cloud computing was introduced as a novel computing paradigm that offers computing facilities as a service, bringing us a step closer to the long-held dream of computing as the fifth utility [19, 129]. A specific feature of Cloud computing, in comparison with other computing paradigms like Grid computing [58], is that it allows the provision of on-demand, scalable storage resources and customized computing environments, using an easy, pay-as-you-go pricing model. The base for such elasticity, reliability and customization is a dynamically scalable resource pool, often virtualized [61], that contains both a large number of servers and a high storage capacity. The service provisioning in the Cloud relies on Service Level Agreement (SLA), which represents a contract, signed between the customer and the service provider, including non-functional requirements of the service specified as Quality of Service (QoS) and penalties in case of violations [45]. This new computing business model has encouraged an increasing number of users and companies to move their applications to Cloud in order to benefit from the low cost and usage flexibility. Furthermore, with the emergence of Cloud computing, many public and private Cloud infrastructures have been built to provide different services that can be classified into the following three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [154]. While IaaS mainly targets the on-demand provision of computational and storage resources, SaaS and PaaS provide on-demand access to applications and development environments, respectively, through the Internet. The current public Cloud infrastructures differ from one another in cost, offered QoS, and access interfaces, which raises the challenge of provisioning services on multiple Clouds, called multi-Cloud. The main reasons for using a multi-Cloud are the low cost and better availability, as Dana Petcu [130] said:

“The use of resources and services from multiple Clouds for reasons like high availability, cost reductions or special features is a natural evolution from in-silo Clouds.”

The research challenges addressed by multi-Cloud are aligned with the vision of global interconnected Clouds called Intercloud computing [13], much like the Internet as a network of networks. Hereby Cloud consumers should be able to freely choose and seamlessly switch between different Cloud platforms without concerns of interoperability.

## 1. Introduction

In this thesis, we investigate how users can be assisted in the task of the selection and deployment of single and composite IaaS Cloud services in a multi-Cloud environment to increase their benefits in terms of cost, quality, and performance. In particular, we examine and evaluate the frameworks and methods needed to automate the selection and deployment processes for different types of user applications, including scientific workflows. To shortly introduce the work carried out in this thesis, in Section 1.1, we present our problem statement, detailing the motivation for this work. In Section 1.2, we present the research questions that will be addressed in this thesis. Major scientific contributions of the thesis are summarized in Section 1.3. Finally, in Section 1.4 we present the organization of the remainder of the thesis.

### 1.1. Problem Statement

Due to the fast-emerging Cloud computing market over the last several years, the number of Cloud service providers has significantly increased. On the other hand, “vendor lock-in” issues and the lack of common Cloud standards hinder the interoperability across these providers, which constitutes one of the obstacles to the growth of Cloud computing [8]. Thus, today the Cloud customer is facing a challenging problem of selecting the appropriate Cloud offers that fit his needs. Therefore, standardized interfaces and intermediate services are needed to prevent monopolies of single Cloud providers. One of the promising use cases of the Intercloud vision defined by the Global Inter-Cloud Technology Forum (GICTF) [56] is the possibility of market transactions via brokers. In such a use case, a broker entity acts as a mediator between the Cloud consumer and multiple Cloud providers to support the former in selecting the provider that better meets his requirements. Another value-added broker service is the easy deployment and management of the user service, regardless of the selected provider, through a uniform interface. Frank Kenney, former research director at Gartner [65], elaborated the need for Cloud brokers, saying:

“The future of Cloud computing will be permeated with the notion of brokers negotiating relationships between providers of Cloud services and the service customers.”

We argue also that brokers are necessary to free users from the task of selecting and managing multi-Cloud services and to hide the technical details of the underlying Cloud infrastructures. However, there are still many unsolved research and technical challenges facing the design and use of Cloud brokers. In the following, we identify the major problems that need to be addressed.

#### **Problem 1**

The “vendor lock-in” problem and the lack of standardization hinder the implementation of generic multi-Cloud service brokers.

We argue that a full-functional broker should assist users in all the steps needed to manage the SLA between them and the Cloud provider candidates. These steps include service description, selection, deployment, and monitoring. However, the implementation of a full-functional multi-Cloud broker is still expected, despite research projects focusing on the development of multi-Cloud broker frameworks. In fact, most of the proposed architectures are either still visionary [17], or have only prototypical implementations [53]. These latter, like most of the existing commercial brokerage solutions [142], support specific Cloud platforms and offer limited broker functionality; thus, their current added-values are not yet attractive enough for users. One of the big issues that hinders the realization of a multi-Cloud broker is the lack of common Cloud standards and interoperability across the current heterogeneous Cloud platforms. We believe that the standardization of the technology is the key to enabling the interoperability across Clouds and to permitting their orchestration through centralized broker entities.

### **Problem 2**

The heterogeneity of Cloud services hinders the building of composite multi-Cloud services.

One of the promising features of the multi-Cloud vision is that users should be able to compose services from multiple providers to deploy complex services such as multi-tier or workflow applications. However, the heterogeneity of the currently offered Cloud services, which differ in their QoS, cost, and functionality, makes the manual composition of services a big challenge for users. Although there is a plethora of research concerning the semantic description of Cloud services [114], most of them focus on single services and are not suitable enough for describing composite services provisioned by different Cloud providers. Therefore, an extended description is needed to consider the connectivity and dependability between the single services forming a composite service. This semantic description can be used to automatically build service and provider repositories, which can be consumed by the broker for the purpose of discovery and selection.

### **Problem 3**

Due to the lack of efficient and automatic service selection methods on Cloud, users have to cope with the complexity of the decision-making.

Current commercial Cloud providers guarantee their customers only a static, non-negotiable SLA, which is usually categorized into a gold, silver, or bronze SLA. Hence, the users must manually make decisions about which Cloud to choose in order to meet their functional and non-functional service requirements while keeping the payment low. The task of manually going through the web pages of Cloud providers to compare their services and billing policies is clearly a burden for the users, especially considering the difficulty of collecting and maintaining the needed information from

## 1. Introduction

the Clouds to make accurate decisions. All these justify the need for automated SLA-based brokering policies to match user requirements with underlying Cloud policies. While Grid brokers [66] usually involve only functional SLA parameters (e.g. CPU cores, RAM size, number of servers), in Cloud computing, non-functional parameters such as cost are also crucial for the customers. In contrast to the functional parameters that often require an exact match, non-functional parameters give the matching algorithms more flexibility to make trade-off between quality and cost. Current approaches for matching Cloud services [63] concentrate on fulfilling functional requirements with minimal costs and high reliability, ignoring the support for other QoS parameters such as response time and latency. Furthermore, only few approaches support SLA-based composite services selection [32]. In the course of this thesis, our goal is to implement efficient, SLA-aware matching algorithms for automating the selection of composite Cloud services, which is known as an np-hard problem [131].

### Problem 4

Current scheduling and data management policies for workflow applications may not be efficient enough in terms of cost and performance in a multi-Cloud environment.

Since the multi-Cloud research is still in its infancy, the deployment of workflows on top of composite multi-Cloud services have been addressed by few works [72, 127]. Most of these works concentrate on solving the technical challenges to running workflows on multiple Clouds without providing solutions to manage the Intercloud data transfer at runtime, which can affect costs and performance. Furthermore, today, related work on resource allocation for Cloud workflows is typically restricted to optimizing up to three objectives: cost, makespan and data locality [38]. Attempts to support other objectives (e.g. reliability, energy efficiency) [50, 148] cannot be applied directly to multi-Cloud environments. However, the support for more SLA constraints, like Cloud-to-Cloud latency and client-to-Cloud throughput, which both have high importance for data-intensive workflow applications, is still missing. In addition, their scheduling policies do not exploit the QoS of the Clouds on which the workflow is deployed nor their impact on workflow execution performance. By implementing efficient SLA-aware scheduling and data management policies, we target an optimal and cost-effective deployment of multi-Cloud workflows.

## 1.2. Research Questions

The main question underlying this thesis is:

*How can an adequate broker framework for automating and optimizing the deployment of multi-Cloud applications be realized?*



The desired broker framework should satisfy the user SLA requirements in terms of QoS and cost while assuring an acceptable application performance, thus ensuring its market success. As an example of common multi-Cloud applications serving for the validation and evaluation of the framework, we consider scientific workflows. On our journey towards a multi-Cloud service broker, the following research questions have arisen:

### **Research Question 1**

How can a generic architecture of a multi-Cloud service broker framework be designed?

In a broker-based multi-Cloud framework, a centralized broker entity should be able to interact on behalf of the user with multiple, interoperable Clouds to perform different management and monitoring tasks. In order to implement generic multi-Cloud broker architectures, a standardized abstract layer between the broker and the providers is necessary. Herewith, the broker will be able to consume the uniform monitoring and management interfaces offered by the Cloud providers by adding new service values like matchmaking and data management on top of them. On the other hand, an adequate user interface is needed to acquire the SLA requirements of different applications from the user and let him monitor the service provisioning and execution status. This research question mainly addresses Problem 1.

### **Research Question 2**

How can a composite multi-Cloud service be semantically described?

A semantic description of the composite Cloud service requests and offers is required by a broker to perform an efficient matchmaking process based on the user requirements and Cloud monitoring information. To achieve this goal, the collected Cloud providers' monitoring data and service descriptions need to be stored in an abstract manner within data repositories. Therefore, semantic techniques for describing the provider service offerings, their QoS metrics and prices, as well as the complex SLA requirements of composite services are required. These techniques should be easily adaptable and extendable to support more SLA parameters and service types. This research question is associated to Problem 2, presented in the previous section.

### **Research Question 3**

How can the best Clouds be selected while fulfilling the user's SLA requirements and maximizing their benefits?

The automated selection of Cloud services is complicated because QoS parameters and pricing information also need to be involved in the selection procedure, in addition to the functional parameters. Since the purpose of the selection is to benefit users,

## 1. Introduction

the matching scheme should select the Cloud services that result in the lowest cost and best fulfillment of the SLA. In order to address these challenges, it is necessary to automate the decision-making process in the broker by using efficient SLA-aware matching algorithms. Since the selection of composite services is more complex due to the large solution space, meta-heuristic methods need to be addressed. This research question addresses Problem 3, discussed in the previous section.

### **Research Question 4**

How can workflows be efficiently deployed on a multi-Cloud environment?

Since the QoS of the Clouds on which the workflow is deployed also heavily impacts the performance and cost of the deployment, the scheduling and data management policies should consider user SLA requirements. In order to solve this issue, scheduling and data management components should be included in the broker framework to support more high-level SLA parameters, like cost and availability. Furthermore, in order to optimize the Intercloud data transfer, data locality and replication need to be considered in the data management and scheduling policies. The impact of the matching policies currently in use on the workflow execution performance would also be interesting to study. This research question is associated with Problem 4.

### **Research Question 5**

How can the proposed multi-Cloud broker framework and the resource allocation policies be evaluated?

The evaluation of complex multi-Cloud usage scenarios and resource allocation policies on real Clouds requires different Cloud platforms with various properties in infrastructure, QoS, and cost. This constitutes a big challenge for Cloud developers and researchers. A common solution is for them to use Cloud simulation environments that permit them to conduct reproducible experiments with various scenarios without any charged costs. Alternatively, it is possible for them to model heterogeneous IaaS Cloud infrastructures with different pricing policies, QoS metrics, and service offers, including computing and storage resources. Furthermore, simulation allows them to evaluate and validate the functionality of different multi-Cloud matching and scheduling policies with different applications scenarios and user SLA constraints. Since existing simulation frameworks [20] offer only limited support to model complex multi-Cloud usage and brokering scenarios, their use requires further extensions and development work. Besides that, the simulation should be made realistic as possible in order to get more convincing results. In order to build a realistic simulation testbed for a multi-Cloud broker framework, current simulation tools need to be extended to collect the needed monitoring information and to dynamically manage the modeled Clouds. Lastly, real workload traces should be exploited by the simulation and a workflow management system should be integrated in order to model real multi-Cloud applications such as workflows.

## 1.3. Scientific Contributions

In this section we highlight our scientific contributions to the research on SLA-based service brokering for multi-Cloud, thereby concretely answering the research questions addressed in the previous section. In addition, we specify for each contribution, the references where it has been published. As our first contribution, we propose the generic architecture of a full-functional broker-based framework to assist users in deploying simple and workflow-based services on multi-Cloud. We enhanced the framework with two ontologies allowing the semantic description of the service requests and Cloud provider offerings. For the efficient matching of Cloud resources with respect to the user SLA requirements, we propose, as our third contribution, a utility-based algorithm adopted from the auction theory. In order to optimize the deployment of multi-Cloud workflows, we present a multi-dimensional resource allocation scheme based on utility-based matching and data locality-driven scheduling combined with a replica-based data management policy. For our final contribution, we implemented the broker framework into a realistic simulation environment to validate its functionality and evaluate all the proposed resource allocation approaches. Overall, the following contributions have been achieved:

### **Contribution 1**

The conceptual design and implementation of a generic multi-Cloud service broker framework.

In order to automate the deployment and management of simple and composite IaaS Cloud services in a multi-Cloud environment, we propose a generic architecture of a fully functional multi-Cloud broker framework. The main purpose of the broker is to match the user requirements to current Cloud providers and manage the service provisioning and execution on behalf of the user. In addition, the framework contains components for interacting with the underlying Clouds, delivering Cloud information, managing SLA issues, and authenticating users. The interoperability between the broker and the Cloud providers is assured using an abstract Cloud Application Programming Interface (API), which interacts with multiple provider specific gateways offering standardized monitoring and management interfaces for the broker. Furthermore, the broker supports the execution of multi-Cloud workflows by offering scheduling and data management capabilities. This contribution addresses our first research question, discussed in the previous section. It has been previously published in [90] and [89] and will be presented in Chapter 4.

### **Contribution 2**

An ontological model for semantically describing IaaS composite service requests and provider offerings.

## 1. Introduction

To store the SLA requirements and provider service offerings and to facilitate the automatic building of composite IaaS services on multi-Cloud, we present an ontological model consisting of a composite service requester ontology and a provider ontology. The former ontology allows a semantic representation of the functional and non-functional service SLA requirements, while the latter is used to describe the characteristics of the underlying Clouds including their SLA metrics, pricing information, geographical location and offered service types. This contribution addresses Research question 2 presented in the previous section, which has been published in [87] and will be presented in Chapter 4.

### **Contribution 3**

An efficient utility-based matchmaking algorithm for composite services selection on multi-Cloud.

In this thesis we propose a utility-based approach for selecting Cloud services by considering both functional and non-functional SLA requirements, including availability, response time, latency, throughput, and cost. The approach is based on an economic model adopted from the multi-attribute auction theory [9] to describe the user preferences and his payment willingness as a function of weighted, non-functional SLA attribute values. This strategy selects only the Cloud services that maximize user utility while fulfilling the service functionality. To tackle the problem of composite services selection, we make use of a graph-based mathematical model serving as input for the matching algorithm. In addition, we propose an evolutionary genetic algorithm called HU-GA, which uses the utility-based economic model as the objective function. For a comparative study, we present a simple matching algorithm called sieving, which randomly selects the Clouds that fulfill all the user SLA requirements. Using realistic, simulation-based evaluation scenarios, we demonstrate the benefit of utility-based matching compared to sieving in terms of cost and QoS. The detailed description of both matching algorithms will be elaborated in Chapter 5 and the evaluation results will be presented in Chapter 7. This contribution has been previously published in [88] and [87]. It addresses the Research question 3 from Section 1.2.

### **Contribution 4**

A multi-dimensional resource allocation scheme and data management policy for deploying multi-Cloud workflows.

In this thesis, we present a multi-dimensional resource allocation scheme to optimize the deployment of large-scale workflow applications in multi-Cloud environments. The scheme applies a two-level approach in which the target Clouds are first selected using the utility-based HU-GA matching algorithm, and then the application workloads are distributed to the selected Clouds using a data locality-driven task scheduling policy. In addition, we apply a replica-based data management policy to

reduce the data transfer between the Clouds during the execution. Using simulation-based evaluation with a real, data-intensive workflow application, we demonstrate the effectiveness of our proposed multi-dimensional scheme and data management policy in improving the workflow execution performance and reducing the amount and costs of Intercloud data transfers. In addition, we study the effect of combining different matching and scheduling policies on the workflow performance. The multi-dimensional resource allocation scheme, data-aware scheduling and data management policies will be presented in Chapter 6 and will be evaluated in Chapter 7. These contributions have been previously published in [86]. They address the aforementioned Research question 4.

### **Contribution 5**

The evaluation of the contributions using a realistic simulation framework.

To evaluate the contributions of this thesis and validate the full-functionality of the designed broker framework, we present the implementation of a simulation environment based on the CloudSim [20] simulation toolkit. In addition, to evaluate the multi-Cloud workflow deployment, we integrate WorkflowSim [24] within the simulation environment to model a workflow management system. Using the simulation environment, we are able to model a multi-Cloud infrastructure and different application scenarios for single and composite services deployment. To make the simulation more realistic, we import real workflow applications traces to the simulation and collect QoS metrics and pricing information from real Clouds to model the infrastructure characteristics. The simulation environment will be detailed in Chapter 4 and the simulation scenarios and setup will be presented in Chapter 7. These contributions have been previously published in [89] and [91]. They address Research question 5 from the previous section.

## **1.4. Thesis Structure**

The outline of this work is structured according to the depiction in Figure 1.1, which shows the relationship between the chapters and the scientific contributions they discuss. The remainder of this thesis is organized as follows:

- Chapter 2 presents the background information relevant to the research conducted within this thesis. It includes first a short definition for Cloud and Intercloud computing and then provides a short overview on current Cloud standards. Then, it introduces the terms SLA and service brokering in context of Cloud computing. After that, it defines composite Cloud services and multi-Cloud workflows. Finally, it gives a short overview on the multi-criteria decision-making and auction theory.

## 1. Introduction

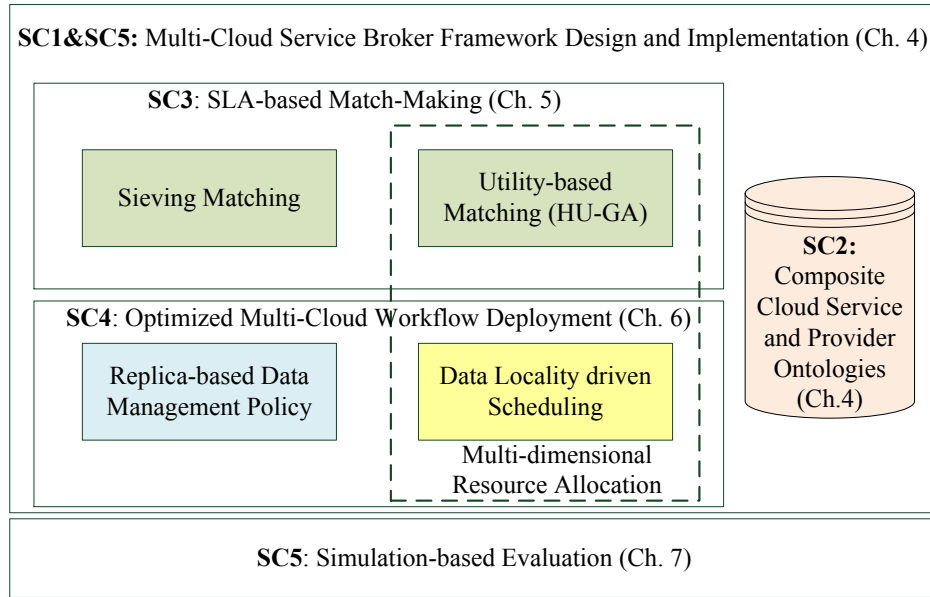


Figure 1.1.: Structure of the thesis and scientific contributions (SC).

- Chapter 3 presents the work relevant to the research topics addressed in this thesis and compares them to the contributions of therein. First, it presents the work regarding multi-Cloud orchestration frameworks and then gives an overview of existing work related to the SLA-based matchmaking of Cloud services. After that, it discusses related work on multi-Cloud workflow frameworks and on data locality-driven scheduling policies for Cloud workflows. Finally, it presents and compares the features of existing Cloud simulation frameworks.
- Chapter 4 presents the fundamental architecture for our multi-Cloud service broker framework design and details its components as discussed in Contribution 1. Secondly, it describes how the SLA-based brokering of multi-Cloud services is realized using our framework. After that, it presents the used ontologies, discussed in Contribution 2, to describe the IaaS provider offerings and composite service requests. Finally, it highlights the solved challenges to implement the simulation environment used to validate and evaluate the thesis contributions.
- Chapter 5, introduces the SLA-based utility and sieving matching algorithms as discussed in Contribution 3. First, it presents a graph-based mathematical formulation of the matchmaking problem with composite multi-Cloud services. Then, it describes the sieving algorithm and utility-based matching algorithm functionalities. Finally, it presents the HU-GA utility-based genetic matching algorithm implemented for matching large-scale service compositions.

- Chapter 6 presents the proposed multi-dimensional resource allocation scheme to optimize the deployment of multi-Cloud workflows as discussed in Contribution 4. First, it describes how the workflow deployment is performed using our multi-Cloud service broker framework. Then, it describes the functionality of the multi-dimensional resource allocation approach composed of the HUGA utility-based matching and data locality-driven scheduling (marked with a dashed-box in Figure 1.1). Finally, it details the implemented data-aware scheduling and data management policies.
- Chapter 7 discusses the simulation-based evaluations of the framework and approaches addressed in Chapters 4-6. First, it presents the simulation testbed used to conduct all of our simulation experiments. Next, it evaluates and validates the multi-Cloud service broker framework functionality. Then, it evaluates the utility-based matchmaking algorithm using several simulation scenarios with simple and composite services and shows its benefits compared to the sieving matching algorithm. After that, we use a simulated, real world case study to compare the matching efficiency of the utility-based algorithm with a prospect-based matching algorithm. Furthermore, we evaluate the introduced multi-dimensional resource allocation scheme using a real data-intensive workflow application and discuss the cost and performance benefits that this approach brings. Finally, the chapter summarizes the evaluations results and makes final conclusions about their benefits and shortcomings.
- Chapter 8 summarizes the contributions of this thesis and their constraints and presents possible extensions and future research directions.





## 2. Background

This chapter describes the background information on essential concepts necessary for the understanding of the research presented in this thesis. We first present in Sections 2.1 and 2.2 the concepts of Cloud and Intercloud computing, which constitute the target platforms addressed in this thesis. In Section 2.3, we give a short overview on current Cloud standardization approaches. Then, we discuss in Sections 2.4 and 2.5 the concepts of Service Level Agreement and Cloud service brokering, on which the thesis contributions presented in Chapter 1 are based. After that, we define in Sections 2.6 and 2.7 the terms composite Cloud services and multi-Cloud workflows, which are our target applications in this thesis. Finally, Section 2.8 gives a short introduction to multi-criteria decision-making and auction theory, on which our proposed multi-Cloud matchmaking approach is based.

### 2.1. Cloud Computing

Cloud computing extends the resource-sharing concept recently used in utility and Grid computing with a business model, where resources are provisioned as services to customers. Based on the web services [79] and virtualization technology, Cloud computing provides on-demand customized computing environments with a simple access interface to private and enterprise users. Since a precise definition for Cloud computing is difficult to find, we refer to the well-known National Institute of Standards and Technology (NIST) [111] definition:

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

Overall, Cloud has the following main characteristics [23]:

- **Resource pooling:** A multi-tenant access to shared resources.
- **Resource elasticity:** The ability to scale resources up and out as needed.
- **On-demand access:** An automatic ubiquitous access over the Internet to the resources.

## 2. Background

- **Pay-per-use:** The requested resources are charged only when used.

As shown from the left side of Figure 2.1, according to the NIST definition, Cloud services can be classified into three delivery models:

- **Infrastructure as a Service (IaaS)** provides customers with on-demand computational resources in the form of Virtual Machine (VM), storage or network. Customers can install operating systems and software packages on the machines to establish their own computing environments.
- **Platform as a Service (PaaS)** provides customers an entire hosting environment to develop their applications. The customer uses the environment without control over the operating system, hardware, or network infrastructure on which they are running. Examples of PaaS-featured Clouds are the Google App Engine [62] and Microsoft Azure [10].
- **Software as a Service (SaaS)** provides the customers a functionality of using the provider's applications that run centrally on its Cloud in the form of web services. The applications are accessible from various client devices through a thin client interface. A well-known example of SaaS is Google Doc [68], which provides a platform to produce and work with on-line documents.

Furthermore, as depicted on the right side of Figure 2.1, the NIST definition identifies four possible Cloud deployment models:

- **Public Cloud:** provides on-demand services via the Internet to the general public.
- **Private Cloud:** provides services restricted to the organization that owns and manages the infrastructure.
- **Community Cloud:** provides services to a group of organizations that have shared interests.
- **Hybrid Cloud:** provides services owned by public and private Clouds.

The understanding about Cloud computing has become more comprehensive since Amazon published its Elastic Compute Services (EC2) [5] in 2006, the first worldwide commercial computing Cloud, and its storage Cloud - the Simple Storage System (S3) [6] to allow users to rent a server and store data on Amazon's hosted computing and storage infrastructures. In addition, many commercial and research institutions are developing middleware stacks to build IaaS Clouds. Examples include Eucalyptus [118], Zimory [174], OpenNebula [145], Nimbus [98] and Openstack [122, 133].

As previously mentioned in Chapter 1, this work focuses on Cloud services offered by public IaaS Clouds. Nevertheless, the developed concepts and approaches can be easily propagated to PaaS and SaaS Clouds. Throughout this thesis, we refer to IaaS by "the Cloud provider" or "the Cloud" unless otherwise specified.

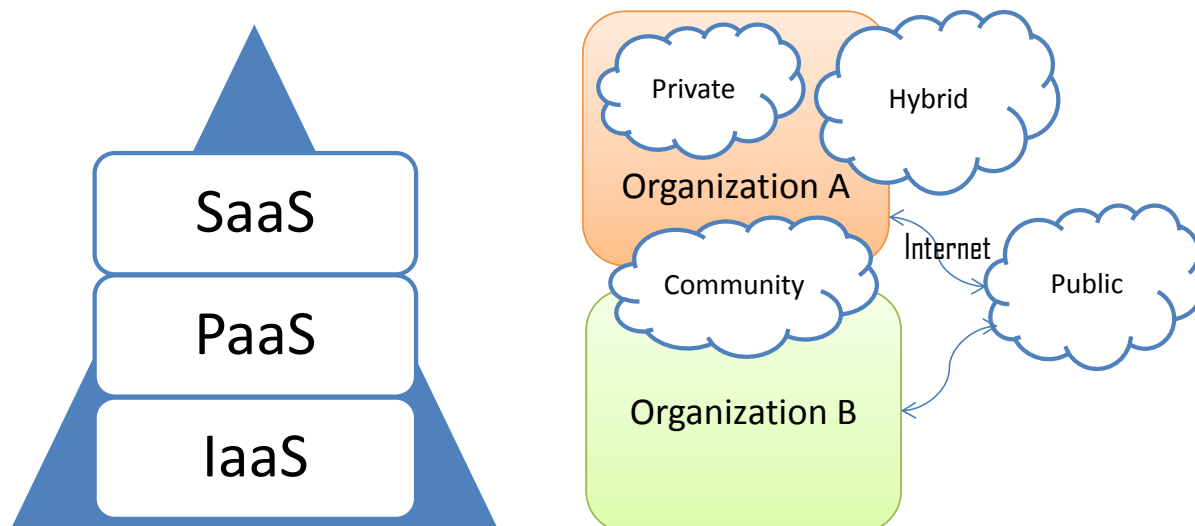


Figure 2.1.: Cloud computing delivery (left) and deployment (right) models.

## 2.2. Intercloud Computing

Intercloud [13] is a recently introduced vision of globally interconnected Clouds (Cloud of Clouds), much like the Internet as a network of networks. This vision addresses interoperability across Clouds, focusing on the use of open Cloud standards. Hereby, Cloud consumers should be able to freely choose and effortlessly switch between different Clouds. On the other hand, providers should be able to distribute their load among geographically distributed datacenters in case of workload spikes or outages in order to meet the availability agreed upon with their customers [70]. The common future use cases and functional requirements for Intercloud computing are published in a white paper by the Global Inter-Cloud Technology Forum (GICTF) [56], which is an initiative started to foster the development of Intercloud technologies in industry and academia.

Based on the strength of the relation between the participating Clouds in an Intercloud environment, we distinguish between two usage scenarios: multi-Cloud and federated Clouds. While in the former the Clouds are used independently of one another, in the latter case the Clouds establish agreements with each other in order to use the resources of the other Clouds [158]. An example of a Cloud middleware supporting Cloud federations is RESERVOIR [138]. The main difference between a multi-Cloud and a federated Cloud architecture is illustrated in Figure 2.2.

In this thesis we address multi-Cloud environments formed by independent public Clouds. However, many of the proposed concepts can easily be adapted to federated Clouds.

## 2. Background

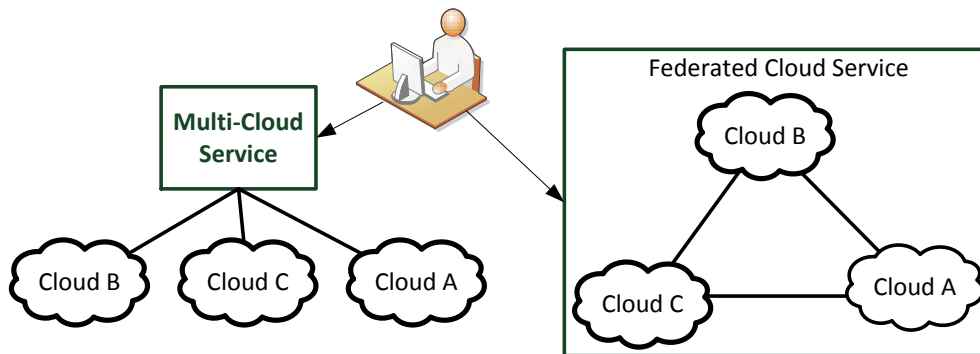


Figure 2.2.: Intercloud computing architectures.

## 2.3. Cloud Standards

Current public Clouds offer proprietary APIs to their customers to manage and monitor their Cloud services. Thus, there is strong potential for “vendor lock-in”, which hinders users’ migration from one Cloud to another. To address the interoperability and portability in Cloud, organizations like the Distributed Management Task Force (DMTF), Cloud Security Alliance (CSA), Open Grid Forum (OGF), and the Storage Networking Industry Association (SNIA) have formed working groups to determine the details of Cloud standards. A list of their important deliverables is presented in Table 2.1. The complete list can be seen under the Cloud standards wiki<sup>1</sup>.

Table 2.1.: An overview of current Cloud standards.

Name	Body	Focus	Last version
OCCI	OGF	IaaS Clouds management and monitoring	Version 1.1
OVF	DMTF	Open virtualization format for VM images	Version 2.1
CDMI	SNIA	Cloud data management interface	Version 1.0.2
CIMI	DMTF	Cloud infrastructure management interface	Version 1.0.1

As the table shows, most standards are still in the initial development stage and their use is still restricted to the research community. We believe that the actual establishment of Cloud standards in the coming years will play an important role in the fast adoption of the Intercloud computing vision.

<sup>1</sup><http://Cloud-standards.org>

## 2.4. Service Level Agreement

A Service Level Agreement (SLA) is defined as a formal contract between service providers and consumers to guarantee that the consumers' service quality expectations can be achieved [161]. This widely-used term in context of telecommunication and networking services has been adopted for Internet-based services since the start of utility computing.

As Cloud computing is based on a business model where service consumers and providers are isolated from each other (loosely coupled), SLA plays an important role for the management of service provisioning. Thus, consumers and Cloud providers should establish an agreement on the predefined service usage terms, which include the metric of each agreed SLA parameter, service cost and the penalty terms for SLA violation. The SLA parameters can be categorized into functional and non-functional parameters. While the former describe the functional requirements for the service operation, the latter assess QoS during its operation as well its usage cost. Some typical functional and non-functional SLA parameters used in context with the IaaS Cloud provisioning are provided in Table 2.2.

Table 2.2.: Typical functional and non-functional SLA for IaaS.

Functional-SLA parameter	Non-functional SLA parameter
CPU cores (number)	latency (time unit)
memory size (Gigabyte)	throughput (Mbit/s)
operating system (e.g. Linux, Windows)	availability (%)
storage size (Gigabyte)	reputation
VM image size (Gigabyte)	budget (cost unit)

According to [161], the SLA life cycle in utility computing systems has six steps, which are to discover service providers, define SLA, establish agreement, monitor SLA violation, terminate SLA, and enforce penalties for violation. The proposed approaches in this thesis address the different steps of the SLA management on Cloud, except the SLA violation and penalty enforcement, which are outside of this thesis' focus.

Current commercial Cloud services are offered with a very limited range of fixed SLAs in a best-effort manner [100]. Most Clouds usually guarantee one QoS parameter, that is, availability, by paying penalties in form of monetary discounts to their users in case of violations. Furthermore, the users must go to the corresponding provider's web page to read and accept the SLA terms. Thus, many research works, including this work, are investigating the automation and full support of the SLA management in Cloud.

## 2.5. Cloud Service Brokering

Service brokering is a business model where services are delivered to the consumer through a third party entity or company called a broker, who acts as mediator between the two parties. This concept has already been used in Grid computing to distribute computing jobs to the Grid sites and monitor their status on behalf the user [69, 66]. With the emergence of Cloud computing, service brokering has been adopted to add new business values to Cloud services. Among them is the support of the user in selecting the provider that better meets his SLA requirements. The basic interactions needed between the broker, user, and provider during the selection process are depicted in Figure 2.3.

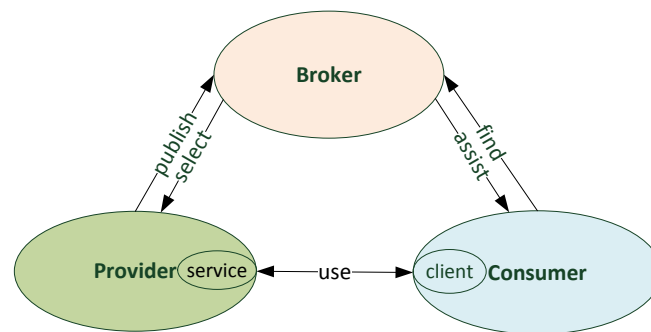


Figure 2.3.: Service brokering concept.

The market research company Gartner [65] has defined three opportunities to use a Cloud broker:

- **Cloud Service Intermediation:** Building services on top of an existing Cloud platform, such as additional security or management capabilities.
- **Cloud Service Aggregation:** Deploying customer services over multiple Cloud platforms.
- **Cloud Service Arbitrage:** Brokers supply flexibility and opportunistic choices and foster competition between Clouds.

The lack of standardization and interoperability across Cloud providers makes the deployment of Cloud service brokers on current production Clouds a challenging task. Therefore most existing commercial companies offering Cloud brokering solutions use proprietary adapters to interface the Clouds that are limited in their functionality. As depicted in Figure 2.4 the service brokering life cycle for Cloud consists of the following steps:

1. **Request Formulation:** The user defines at design time the functional and non-functional SLA requirements for the requested Cloud service.

2. **Discovery and Monitoring:** The broker discovers the candidate service offers and stores their monitored SLA metrics and pricing information in different data repositories.
3. **Matchmaking:** The broker selects the suitable Clouds for provisioning the requested service by matching the SLA requirements to the candidate computing and storage resources.
4. **Deployment:** The broker deploys the service components on the selected providers.
5. **Execution:** The service is executed and its status is continually monitored at the runtime.
6. **Termination:** The service can be terminated upon user request or by the broker (e.g., in case of repeated SLA violations).

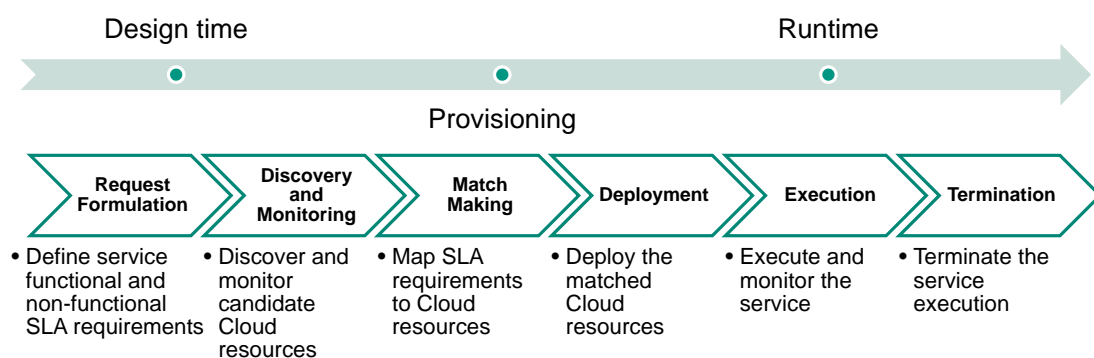


Figure 2.4.: Cloud service brokering life cycle.

As previously mentioned in Chapter 1, we propose in this thesis a broker framework that acts as mediator between the Cloud user and multiple interoperable Cloud providers and supports the above-described brokering steps.

## 2.6. Composite Cloud Service

A composite service is defined as follows [128]:

“Complex (or composite) services typically involve the assembly and invocation of many pre-existing services possibly found in diverse enterprises to complete a multi-step business interaction.”

According to the above definition, and compared to single Cloud services, which provide a simple function, composite or complex Cloud services combine multiple services in order to generate an added value [16]. The composition of services is a widely used technique in the context of web service to implement business workflows

## 2. Background

(e.g., payment processing services). A composite Cloud service, which is composed of single services provisioned by different Clouds, is called a composite multi-Cloud service.

The operation and management of composite services in multi-Cloud environments is very challenging, because the component services operate in a highly variable environment and their QoS changes very frequently. In addition, there are no common interfaces to access the component services belonging to different Clouds in a uniform way. All of these raise the need for dynamic automated service composition techniques. In this thesis, we address the selection and deployment of composite Cloud services in multi-Cloud environments while taking SLA user requirements into account.

## 2.7. Multi-Cloud Workflows

Workflow is a technology that allows users to split a complex problem into smaller parts that can be solved using a single computing unit, like a computing node of a cluster system. According to [94], workflows are:

“coarse-grained parallel applications that consist of a series of computational tasks logically connected by data- and control-flow dependencies”

Over the last decade, many scientific communities in the field of astronomy, gravitational physics, computational biology, climate modeling, and life-sciences have successfully used workflow technology to carry out large-scale experiments [126] on Grid like Datagrid and Medigrid [110].

A simple workflow can be represented in the form of a Directed Acyclic Graph (DAG), where nodes represent the single computing tasks and the edges represent the data flow between them. Figure 2.5 depicts a simple DAG workflow consisting of sequential and parallel tasks. In this example, tasks are executed according to the alphabetical order of the letters assigned to their names. For example, task C cannot be executed only if tasks A and B are finished, whereas tasks D and E can be executed in parallel after task C is finished.

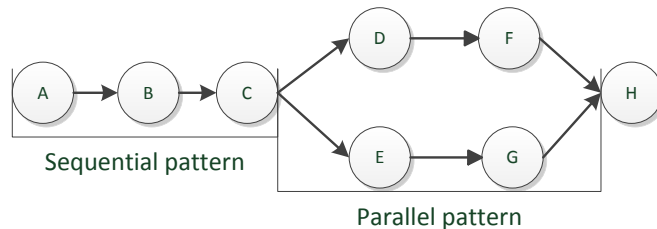


Figure 2.5.: Sample sequential and parallel workflow patterns presented as DAG.



The automated execution of workflows requires the use of a middleware called Workflow Management System (WfMS), such as Pegasus [39] or Gridbus WfMS [69]. The WfMS takes the workflow description as input and coordinates the execution of workflow tasks with respect to their order. The mapping of workflow tasks to compute resources and the inter-task data transfer are performed using different scheduling and data management policies [166].

On the one hand, the use of Cloud as a computing infrastructure to run workflows brings many technological benefits to the workflow users, such as on-demand provisioning, elasticity, provenance, and reproducibility [37]. On the other hand, it also presents them with new challenges, like resource heterogeneity and “vendor lock-in”, which hinder the adoption of workflows to Cloud. In addition, other factors such as the economic and the QoS considerations become crucial with the execution of workflows on the Cloud. Furthermore, data management and security issues need to be addressed before the migration to Cloud can be implemented.

In this thesis, we address the scheduling and deployment of multi-Cloud workflow applications, focusing on scientific workflows. Multi-Cloud workflows are a particular type of Cloud workflow applications running on top of composite multi-Cloud services, where the workflow tasks are executed across multiple Clouds.

## 2.8. Multi-Criteria Decision-Making and Auction Theory

The Multi-Criteria Decision-Making (MCDM)[168] theory is a research field belonging to operations research concerned with solving decision problems involving multiple, and often conflicting, criteria. Based on the goal of the decision maker, decision problems are classified into choosing, ranking and sorting problems. In the literature there are three basic methods for solving MCDM problems: Multi-Attribute Utility Theory (MAUT), outranking, and the Analytic Hierarchy Process (AHP). These methods differ in the amount of input required from the decision maker and in their used objective functions, however their input and output parameters are similar. The basic input parameters (i.e., criteria and alternatives) and the required inputs from the decision maker to perform a MCDM process are depicted in Figure 2.6.

In MAUT [99], the preferences of the decision maker for or against each criterion are quantified using utility functions and their weight values. The decision problem is then simplified into a single objective function by aggregating the weighted utility functions of the decision criteria. Thus, the best solution should maximize this objective function.

The outranking MCDM method [54] checks the degree of dominance of one alternative over another by comparing its performance against all decision criteria. Unlike

## 2. Background

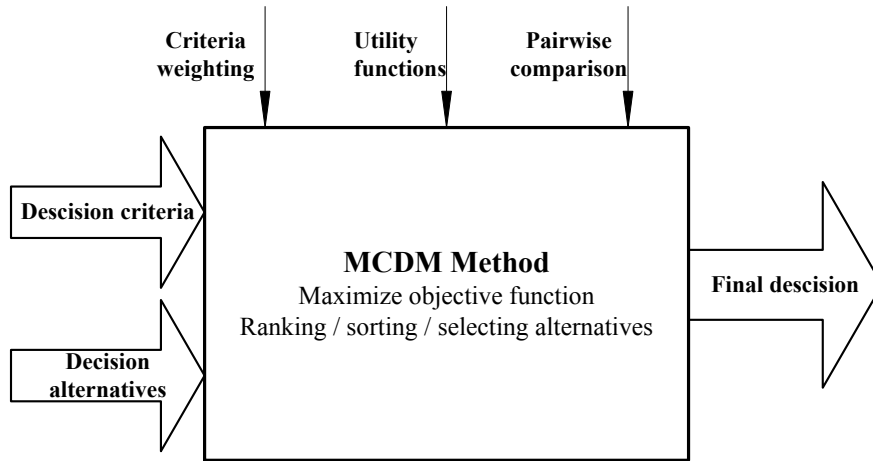


Figure 2.6.: The Multi-Criteria Decision-Making (MCDM) process.

the other methods, the scaling and weighting of criteria is not required for the decision process. Since the outranking method is in some cases unable to find a non-dominant alternative, it is usually combined with other MCDM methods as a pre-step to reduce the number of alternatives.

The AHP [139] methods solve decision problems by arranging the criteria and their alternatives into a hierarchy structure. Unlike MAUT, AHP is based on pairwise comparisons of decision criteria (using a numerical scale ranging from 1 (equal) to 9 (extremely important)) rather than using utility and weighting functions [63]. The unique features of AHP compared to other MCDM methods are the capabilities to support the interdependence among criteria and to check inconsistencies in the solutions found.

The auction theory is a sub field of the game theory, widely applied in economy, that deals with market auctions, assuming the presence of asymmetric information among the participants. These latter, bidders, auctioneers, and vendors, follow different strategies to handle their revenues. In contrast to classical auction types, such as English auctions [22], where only a single criterion (cost) is addressed, a multi-attribute auction involves other criteria, such as the quality of the goods and the reputation of the sellers. Many multi-attribute auction approaches adopt MAUT methods to rank the bids based on the additive weighting of different criteria [15]. Other approaches use a quasi-linear utility function to manage the cost quality trade-off [9]. In these approaches, the buyers assign relative weights and scoring functions to each of the auction attributes in addition to their payment willingness. The bids that maximize their revenues are then selected. In this thesis, we apply a similar method to match Cloud services offered with different QoS and price policies.

## **2.9. Summary**

This chapter presented the background information on Cloud computing, Intercloud computing, SLA, and service brokering, which are helpful for readers who may not be familiar with some of the material needed to follow this thesis. Firstly, the concepts of Cloud and Intercloud Computing were presented. After that, we discussed some state-of-the-art Cloud standardization approaches. Then, the terms SLA and service brokering were introduced in the context of Cloud. After defining composite services, the chapter presented the challenges related to the execution of multi-Cloud workflows on top of composite Cloud services, since they are the type of Cloud applications addressed in this thesis. Finally, we briefly introduced the methods for multi-criteria decision-making and the multi-attribute auction theory, which are the main ideas behind our proposed matchmaking approach on multi-Cloud.



## 3. Related Work

In this chapter we present related work relevant to the research topics addressed in this thesis. In Section 3.1 we present related work regarding multi-Cloud orchestration frameworks. An overview of existing work related to the matchmaking of Cloud services is provided in Section 3.2. After that, we discuss in Section 3.3 related work on multi-Cloud workflow frameworks. In Section 3.4 we present related work on data-driven scheduling policies for Cloud workflows. Finally, Section 3.5 presents existing Cloud simulation frameworks. Many parts of this chapter are based on [89, 88, 91, 86, 87].

### 3.1. Multi-Cloud Orchestration Frameworks

The research on multi-Cloud orchestration is still in its infancy stage. In fact, most of the research conducted in this field has been undertaken by academic research projects and early adopters from the industry. A taxonomy of the existing Intercloud architectures and their brokering features with current research challenges is provided in [71]. In the following we list the main previous works in this area.

The CloudBus research project [17] provides a visionary architecture for market-oriented Cloud computing. The three key components of this architecture are a Cloud broker, a market maker and an InterCloud. The Cloud broker schedules applications on behalf of the user by specifying the desired QoS requirements, whereas the market maker acts as a mediator bringing together Cloud providers and customers. It aggregates infrastructure demands from the Cloud broker and matches them against the available resources published by the Cloud providers. The InterCloud [18] provides a scalable, federated computing environment composed of heterogeneous, interconnected Clouds, enabling Intercloud resource sharing. The development of the CloudBus architectural framework is still ongoing. However, the initial experimental results using Aneka [21] and Amazon EC2-based Clouds demonstrated that the market-oriented CloudBus architecture and the proposed federation approach bring benefits to user's application performance in optimizing the cost and execution time. Although the focus of this project is market-oriented, federated Clouds, some components of the proposed multi-Cloud service broker framework in this thesis are inspired from the CloudBus architecture.

### 3. Related Work

Theilmann et al. [75] presented a flexible framework for multi-level SLA management within Clouds developed in context of the SLA@SOI EU project [144]. The core framework consists of a business manager and an SLA manager. The business manager controls relations between customers and providers, whereas the SLA manager deals with SLA related issues, including negotiation, provisioning, and monitoring. Besides the core framework, a domain-specific service manager provides management functionalities for the SLA manager by interfacing the native provisioning system. The main contribution of the SLA@SOI framework is that the service quality can be predicted and enforced at runtime through automated SLA management. However, the implemented SLA framework lacks the support for service selection, which is mainly addressed in this thesis.

Metsch et al. [149] implemented a prototype broker architecture by integrating the above described core SLA@SOI framework and the RESERVOIR framework, which offers an open platform for federated Cloud computing. In their presented architecture, the core SLA@SOI framework acts as an SLA-based broker, whereas the RESERVOIR enabled sites act as candidate Cloud providers for the broker. The interoperability between the two Cloud frameworks is achieved by implementing a standardized SLA@SOI service manager interface using the Open Cloud Computing Interface (OCCI). The main contribution from their implementation was to validate the use of Cloud standards as enabling technology for Cloud brokering. This result inspired us in this thesis to use OCCI as standard Cloud API for managing heterogeneous IaaS Clouds.

The EU funded OPTIMIS project [53] developed a toolkit based on agent technology to optimize the full service life cycle in Cloud. The flexible OPTIMIS architecture supports multi-Cloud as well as Cloud federations. One of its key components is a deployment engine, acting as a service broker that allows a decision to be made based upon business aspects like trust, cost and risk. The toolkit has been validated with real application use cases. In addition, the initial simulation experiments conducted with real workload traces proved the benefits from the use of cost and risk aspects as elasticity policies in the decision-making. A major drawback of the OPTIMIS architecture in terms of interoperability is that the participating Cloud providers need to develop and maintain multiple vendor-specific OPTIMIS adapters to benefit from the entire toolkit.

Another EU funded project is mOSAIC [115] which aims to simplify the development and deployment of multi-Cloud applications. It proposes an agent-based framework composed of a Cloud agency, which maintains the best resources configuration that satisfies the application SLA and cost requirements, and a platform-independent programming model for developers called mOSAIC API. One of the shortcomings of this approach is that the developers need to rebuild their applications to integrate the mOSAIC API before using the framework.

Kertesz et al. [101] investigated the use of autonomic computing principles for resource management and SLA enforcement in Cloud environments. They proposed

an SLA-based Service Virtualization (SSV) architecture, which is built on three main components: a meta-negotiator responsible for SLA agreement negotiations, a meta-broker for selecting the proper execution environment and an automatic service deployer for service virtualization and on-demand deployment. The proposed service virtualization architecture has been validated in a simulation environment based on CloudSim using a real biochemical application as a case study. The simulation results showed performance gains in terms of execution time from the SSV architecture compared to a less heterogeneous Grid meta-brokering solution.

The recently started MODAClouds, an EU funded research project [7], aims to deliver model-driven design tools and a runtime environment to simplify the development and deployment of applications on multi-Clouds with guaranteed QoS. The MODAClouds architecture features a decision support system that supports the costs and risks assessment at design time and a monitoring system to guarantee the quality assurance at runtime. As the project is still in the initial phase, no evaluation results have been published yet.

As the first industry-driven project, the TM Forum Cloud service broker catalyst [57] explored the role of a value-added service broker by demonstrating a proof of concept for a trusted, transparent Cloud management platform. In contrast to our work, their main purpose of using a broker was to enable Cloudbursting capability for applications using a hybrid Cloud delivery model with respect to the SLA requirements.

Besides the above-mentioned research projects, some companies have started to provide commercial Cloud brokering solutions as ready PaaS and SaaS products to Small and Medium Enterprises (SME) customers. Their solutions are mainly implemented to facilitate the aggregation and mediation of customer services, which are deployed on different, public IaaS Cloud providers. Herewith they solved vendor lock-in issues by providing integrated management and monitoring interfaces. In fact, the customers do not need to use a multi-Cloud library (e.g., jclouds [84]) or write specific adapters to access multiple Clouds at the same time. Two examples of the companies providing such solutions are RightScale [137] and Jamcracker [83]. Nevertheless, few companies offer additional brokering features like automated SLA-based multi-Cloud management and Cloud service selection. Among those companies we refer, for example, to SensibleCloud [142] and CloudSwitch [29].

In summary, the above discussed multi-Cloud orchestration frameworks such as [17, 101, 149] are mostly based on centralized broker components, which are acting as mediators between the user and providers. The proposed multi-Cloud broker framework, presented in Chapter 4, acquired some ideas from these previous works. However, we have designed a high-level, generic architecture by integrating several state of the art technologies and standards. First, our solution combines all the brokering features included in the previous works, like SLA management, service deployment, and monitoring and SLA-aware resource matchmaking. Secondly, we provide an abstraction layer to hide the technical details of Cloud providers by using current Cloud

### 3. Related Work

standards. Finally, we support the deployment of scientific multi-Cloud workflow applications.

## 3.2. Matchmaking of Composite Cloud Services

The brokering policies currently in use for matching the user requirements to the underlying Cloud services directly influence the quality of the service at runtime, as well as the service usage cost. While Grid brokers usually involve only functional SLA parameters in the process of seeking an appropriate computing element for a submitted job, in Cloud computing, non-functional parameters, including cost, are also crucial for customers. In contrast to the functional parameters that often require an exact matching, non-functional parameters give the matching algorithms more flexibility to enable trade-off between quality and cost.

A lot of early work on SLA-based service matching was done in the context of web services as described in [42]. A wide range of different methods has been proposed for determining suitable web service compositions for users. Most of these approaches rely on artificial intelligence planning algorithms and apply backward chaining to derive suitable compositions from a certain goal. Such an approach is either for stateful [107] or for stateless services [143]. Often such approaches are based on formal descriptions of service functionality using the SAWSDL, OWL-S and WSMO description languages. Composition that includes additional information about the temporal behavior of a service is presented in [11]. All these approaches consider service functionality as the only composition criteria and largely disregard other non-functional, particularly business-related, service properties used for service differentiation (e.g., QoS or prices).

Several approaches are proposed to solve QoS-based web service composition problems, as described in [4, 156, 2]. A detailed evaluation of these techniques is presented in [140]. Most of these approaches like in [169, 12] are based on linear programming methods, which are not applicable for large-scale service composition problems as in Cloud computing. Besides, these approaches do not provide declarative representations of service offers and requests as required in a Cloud scenario. An approach for an ontology-based representation of configurable web service requests and offers with complex pricing and utility-based preference functions is presented in [106]. Although this approach supports multiple service configurations and features in a service selection algorithm, it is restricted to the selection of single services and is not applicable for service networks. Nevertheless, this approach serves as basis for our proposed SLA-based matchmaking algorithm, presented in Chapter 5, which adopts the efficient utility-based selection algorithm to match composite Cloud services.

There is preliminary work in the field of QoS-based service selection in Cloud environments. A broad overview of the commonly used selection methodologies with



some related research issues is given in [33]. Clark et al. [26] proposed a priority-based service selection algorithm based on agent technologies. In their approach, a broker agent determines whether the requirements in the service request can be matched for each of the registered service offers. In case several offers fit the requirements, the selected candidates are sorted by the specified priority in the user request. For example, if cost has a high priority, the provider with the lowest price is proposed to the user as the best candidate for the service deployment. Their current algorithm implementation supports three functional SLA attributes (OS type, CPU and storage size) and two non-functional SLA attributes (price and availability). This proposed priority-based approach is suitable for matching single services and its use for matching composite services requires major modifications.

Modica and Tomarchio [114] used the Semantic Web Rule Language (SWRL) to map between requests and service offers. The mapping process takes as input a request and a provider ontology that describe the service requirements and the offered service features, respectively. The best offer is then selected from the semantically-mapped offers by ranking them against a semantic affinity metric. One of the shortcomings of this approach is the large overhead in designing and maintaining the ontologies and the difficulty of writing semantic rules for requests with complex SLA requirements. A similar semantic-based selection method based on SPARQL was proposed by Nizami et al. [117] to rank service offers based on cost, reputation, reliability and security with the goal of finding the cheapest provider. Both of these approaches are restricted to the selection of single services. Redl et al. [136] apply feedback-oriented machine learning methods to automatically match the Cloud provider SLA specifications and select the best-fitting service for the user. However, their approach supports only functional SLA and is not suitable for composite services.

In [82], the authors investigated the use of auction and incentive-based schemes in a multi-Cloud service broker. The latter leverages the dynamic pricing policies of the Cloud service providers to select the offers maximizing the user utility, which is calculated based on the additive weighting of cost, trust, and reputation indexes. In their evaluation, conducted with single services, they showed the revenue benefit for Cloud providers from their matching schemes but not yet the user profit.

In the context of the SMICloud project, Garg et al. [63] use AHP decision-making methods to rank candidate Cloud service offerings. Although they showed the cost and QoS matching effectiveness of the AHP-based ranking, as the project is still in the initial phase, the evaluation is done only with simple Cloud services. Juan-Verdejo and Baars [92] proposed a framework based also on AHP to support the partial migration of business intelligence applications to Cloud infrastructures by taking into account business and economic considerations. However, their ongoing work still lacks evaluation.

Menzel and Ranjan [112] presented a framework called CloudGenius to select the best combination of a VM image and a Cloud infrastructure service to support Web

### 3. Related Work

server migrations to the Cloud. Their presented evaluation results with Amazon-based services showed the time complexity of the algorithm, but not yet the impact of the matching on the QoS. Although AHP-based selection methods are effective on the ranking of Cloud services characterized by conflicting QoS parameters, they require the user to provide an accurate, subjective weighting scheme for each requested QoS parameter, which heavily influences the matching results. Hence, these methods can only perform well when the number of alternatives is small and the number of objectives is limited [32].

Zhang et al. [172] proposed a declarative decision support system called CloudRecommender for the automatic selection of infrastructure Cloud service configurations using transactional SQL semantics. For this purpose, they introduced an extensible ontology to describe the functionalities and QoS parameters of IaaS offers [171]. Their current prototype implementation allows a selection based on previous stored service information and does not support a dynamic selection based on QoS information, like latency and resource utilization.

In [164] the authors showed the effectiveness of genetic algorithms compared to linear programming methods in optimizing the composition of Cloud services. Contrary to this work, they used the Simple Additive Weighting (SAW) of four QoS parameters, which are response time, price, availability, and reputation, as fitness function for the genetic algorithm. Besides, they focused on applications, where the order of execution of each component service is important. Such kind of applications, like business workflows, are out of the focus of this work.

Dastjerdi et al. [32] proposed an approach that selects the Cloud VM images and Cloud infrastructure services for a network of virtual appliances across multiple Clouds. In their work, the selection is performed after an ontology-based discovery, using a genetic-based matching algorithm with the goal to optimize latency, reliability, and deployment cost. Their evaluation showed the cost-effectiveness of the genetic algorithm compared to their introduced Forward-Checking Based Backtracking (FCBB) algorithm, but not yet the QoS benefits from the matching.

Table 3.1 compares the previously discussed research works with the matchmaking approach proposed in this thesis in terms of their matching methods, their use of ontologies, and their applications scope. As can be seen from the table, a unique feature of the matchmaking approach proposed in this thesis is the use of a hybrid utility-based genetic algorithm (HU-GA), where a quasi-linear utility function is used as an objective function to optimize the composite services selection. The used utility function is adopted from the economic theory and has been already used in [73] for the benchmarking of single Cloud services. In contrast to the other approaches, our matching scheme supports dynamic parameters in matching, such as the Cloud-to-Cloud latency and the resource load on the Clouds. We evaluate our proposed HU-GA matching algorithm using a broker-based simulation framework with a real scientific workflow to demonstrate its cost and QoS effectiveness in addition to its time complexity.

Table 3.1.: A comparison of existing composite Cloud service selection frameworks with this work.

Criteria	CloudGenius	CloudRecommender	Dastjerdi et al.	Proposed approach
Matching methods	AHP	SQL semantics	FCBB / GA+cost	sieving / GA+utility
Ontology-based	no	yes	yes	yes
Evaluation criteria	time complexity	cost +time complexity	cost +time complexity	cost+QoS +time complexity
SLA metrics/pricing	synthetic	real	real	real
Application scope	multi-tier	multi-tier	multi-tier	scientific workflows
Framework	CumulusGenius	CloudRecommender	CloudPick	Cloud Service Broker
Dynamic matching	no	no	no	yes

## 3.3. Multi-Cloud Workflow Frameworks

While WfMS for Grids are well established in the market [165], the automation of workflows in Cloud environments is currently in the research phase. A common approach for deploying workflows on the Cloud is to extend existing Grid WfMSs [40, 30] such as the Generic Workflow Execution Service (GWES)<sup>1</sup> and Pegasus to work on Cloud environments. An example is the work in [76], which extended the Kepler WfMS to use Amazon EC2-based Cloud services. Juve et al. [95] evaluated the cost performance trade-off of executing real workflow applications on EC2 with data pre-staged from the Amazon S3 [6] Cloud storage. For this, they used the Wrangler cluster configuration tool [93] together with the Pegasus WfMS to provision and configure virtual clusters on top of the EC2 instances the same way as on High Performance Computing (HPC) and Grid clusters. They found that the choice of a storage system has a significant impact on the workflow runtime and execution costs. Buyya et al. [127] developed a workflow engine to schedule workflow tasks to Cloud resources with respect to the QoS requirements in context of the CloudBus project. The prototypical implementation of the workflow engine based on the Aneka Cloud platform [21] using EC2 resources proved its performance benefits in executing workflows on the Cloud.

In [34] Oliveira et al. introduced the SciCumulus middleware, which follows the Many Task Computing (MTC) paradigm [134] to automate the execution of workflows on the Cloud. Their simulation-based evaluation of the SciCumulus architecture showed the performance gains from using parameter sweep and data fragmentation as parallelism techniques, but not yet the monetary impact in executing workflows on commercial Clouds. Garcia et al. [72] proposed a multi-agent architecture for the concurrent and parallel execution of workflows on multi-Clouds, where consumers, brokers, Cloud providers, and Cloud resources are represented by agents. Based on a simulation testbed, they evaluated the performance benefits from the agent-based workflow execution. However, their service selection mechanism based on the Contract-Net Protocol (CNP) does not yet support the negotiation of non-functional SLA constraints. Further, Tao et al. [150] presented a framework for Intercloud service combination consisting of a workflow system, capable of managing workflow tasks running on different Clouds, and a cost-performance prediction model. The prototype implementation of the framework is restricted to the EC2-based Cloud services and does not yet support the automatic service selection.

In summary, from the above-mentioned approaches only [72, 127] addressed the deployment of workflows in multi-Cloud environments. Although many of the approaches promise the support of heterogeneous Clouds, their concrete implementations mostly support only homogeneous Cloud environments. Moreover, most of these approaches, except [95], fail to investigate the data transfer and inter-network

---

<sup>1</sup><http://www.gridworkflow.org/gwes/>

communication between the workflow tasks, which are crucial for Intercloud computing. Furthermore, the selection of the required Cloud resources for the workflow deployment is mostly based on functional SLA requirements and costs, while the non-functional SLA requirements are usually not considered. In contrast, the multi-Cloud broker framework proposed in this thesis allows a workflow deployment over heterogeneous public Clouds with respect to the user requested functional and non-functional SLAs.

## 3.4. Scheduling of data-intensive Workflows on Cloud

Within the past decade, the scheduling of data-intensive workflows has emerged as an important research topic in distributed computing. Although the support of data locality has been heavily investigated in Grid [135] and HPC, only a few approaches apply data locality for Cloud workflows. A survey of these approaches is provided in [113]. In this section we focus on works dealing with data locality-driven workflow scheduling in multi-Cloud environments.

The authors in [51] adopted a dynamic auction-based scheduling strategy called BOSS to schedule workflows in multi-Cloud environments from the game theory. Although their conducted experiments show the effectiveness of their approach in reducing the cost and makespan compared to traditional multi-objective evolutionary algorithms, the support for data locality is completely missing in their work. Szabo et al. [148] implemented a multi-objective evolutionary workflow scheduling algorithm to optimize task allocation and ordering using data transfer size and execution time as fitness functions. Their experimental results prove the performance benefits of their approach but not yet the cost effectiveness. Although the authors claim the support for multi-Cloud, in their evaluation they used only Amazon EC2-based Clouds. Yuan et al. [167] proposed a k-means clustering strategy for data placement of scientific Cloud workflows. Their strategy clusters datasets based on their dependencies and supports reallocation at runtime. Although their simulation results showed the benefits of the k-means algorithm in reducing the number of data movements, their work lacks the evaluation of the execution time and cost effectiveness. Zaho et al. [47] presented another workflow data placement strategy based on a genetic algorithm. It is able to reduce data movements among the datacenters while balancing their loads. Their algorithm outperforms the k-means algorithm in the number of data movements and load balancing. In [125] the authors use Particle Swarm Optimization (PSO) techniques to minimize the computation and traffic cost when executing workflows on the Cloud. Their approach is able to reduce the execution costs while balancing the load among the datacenters. The authors in [85] introduced an efficient data locality-driven task scheduling algorithm called BAR (balance and reduce). The algorithm adjusts data locality dynamically according to the network state and load

### 3. Related Work

on datacenters. It is able to reduce the completion time. However, it has been tested only with MapReduce [35]-based workflow applications.

Hadoop [74] is a popular open-source implementation of the MapReduce framework that allows the processing of large-scale data-analysis workflows. It comes with an integrated dynamic job scheduler, which exploits a parallel file system (HDFS) to support data locality. The scheduler ensures that each job is assigned to the closest data node that contains its required data, thereby reducing the network traffic and the makespan. In addition, the scheduler assures a load balancing between the nodes. One of the shortcomings of Hadoop is the lack of a user interface and its use of a specific programming environment. Therefore, there were some attempts to integrate it with traditional WfMS to support scientific workflows such in [157]. Moreover, the native Hadoop framework cannot execute tasks across multiple Clouds. In [163] the authors proposed a market-oriented, hierarchical scheduling strategy to execute multi-Cloud workflows. In the first step, service-level scheduling is performed by a global scheduler to allocate suitable services to each workflow task with respect to QoS constraints. In the second step, a meta-heuristic-based local scheduler performs a task-level scheduling to optimize the task-to-VM assignments inside the selected Cloud. Although this approach has some similarity to our proposed two-level multi-dimensional resource allocation scheme, it supports only cost and makespan as QoS constraints. Furthermore, it is only suitable for compute-intensive workflow applications, since the authors do not consider data locality.

The examination of the previous mentioned works shows that the support of data locality in scheduling improves the performance and minimizes the cost of workflow execution on Cloud. However, their used scheduling policies, except in [163], do not exploit the QoS of the Clouds on which the workflow is deployed. For solving this issue, a possible solution is to implement the scheduling as part of a middleware on top of the Cloud infrastructures. In this way, it would be possible to support more user-defined QoS requirements, like latency and availability in the task scheduling policies. Since a proper SLA-based resource selection can also have a significant impact on the performance and cost, it would be interesting to study the impact of the used matching policy on the scheduling performance of data-intensive workflows. To support an SLA-based workflow deployment on multi-Cloud, our proposed broker framework includes a data locality-driven task scheduler and data manager. In our evaluation with real scientific workflows, we study the impact of the matching and scheduling policies on the data movements, costs, and makespan.

## 3.5. Cloud Simulation Frameworks

Since it is difficult to evaluate complex Cloud usage scenarios and resource allocation policies on real Cloud testbeds, a common solution for Cloud researchers and developers is to use Cloud simulation environments. On one hand, the use of simulation

permits them to reproduce the modeled scenarios and, on the other hand, the refinements of the studied policies without any charged costs. In the recent years, different Cloud simulation tools have been developed. A full review of these tools is provided in [173]. A detailed comparison of their features that are relevant for this thesis is provided in Table 3.2.

CloudSim [20] is a popular event-based simulation environment, developed by the Cloud Computing and Distributed Systems (CLOUDS) laboratory at the University of Melbourne. It offers capabilities to simulate large-scale heterogeneous IaaS computing Clouds including VMs and Hosts. The main focus of CloudSim is on the modeling of VM provisioning and scheduling policies and of different Cloud applications called Cloudlets. This easy extensible simulation toolkit, written in Java, served as the basis evaluation environment for many research works on Cloud. CloudAnalyst [159] is a graphical simulation tool built on top of the CloudSim to analyze the load behavior of large social network applications, such Facebook <sup>2</sup>. The Internet traffic routing between the user bases located in different geographic locations and the Cloud datacenters is controlled in CloudAnalyst by a service broker that decides which data-center should serve the requests from each user base based on different routing policies. CloudAnalyst extended CloudSim with three different brokering policies, which are network-latency-based routing, response-time-based routing, and dynamic-load-based routing. NetworkcloudSim [64] is another extension of CloudSim that supports real HPC applications such as parallel message passing (MPI) applications and the modeling of networked datacenters. These newly added features have been included in the current CloudSim version. Since CloudSim itself lacks the support of workflows, an extension called WorkflowSim [24] has been implemented to model the deployment of workflows managed by the Pegasus WfMS on top of CloudSim modeled datacenters.

GreenCloud [102] is a packet-level Cloud Simulator, which is built on top of the network simulator NS-2 <sup>3</sup> to enable the simulation of energy-aware Cloud datacenters and model their detailed energy consumptions. This unique feature is missing in the other tools; however, it is not within the scope of this thesis. Another event-based simulator is GroundSim [124], which allows the simulation of real scientific applications on Cloud and Grid environments. A unique feature of GroudSim is the native support for scientific workflow applications through the integration of the ASKALON WfMS [48]. Therewith, it is possible to study different workflow scheduling policies and to model job failures on Cloud environments. In order to predict the cost and performance trade-offs when migrating applications to Clouds, the authors in [116] implemented the graphical simulation tool iCanCloud. This scalable tool allows for the modeling of computing datacenters with different hypervisors, networks, and storage systems. One of its features is the direct inclusion of modeled real Amazon EC2 instances and their pricing policies in the simulation.

---

<sup>2</sup><http://www.facebook.com>

<sup>3</sup><http://www.isi.edu/nsnam/ns/>

### 3. Related Work

Table 3.2.: A comparison of open source Cloud simulation tools.

Criteria	CloudSim	iCanCloud	GridSim	GreenCloud
Platform	JVM	OmNet+MPI	JVM	NS2
Language	Java	C++	Java	C++
GUI	with CloudAnalyst	yes	yes	no
Scope	provisioning policies scheduling policies	provisioning policies scheduling policies	provisioning policies scheduling policies	energy-efficiency power consumption
Applications	single/MPI	single/MapReduce/MPI	single/workflow	single
Workflow support	with WorkflowSim	no	with Askalon	no
Broker policy	Round Robin	cost+performance	cost+makespan	no
Storage support	local/SAN	local/parallel/shared	local	local
Real traces	yes	no	yes	yes
Real Clouds	no	EC2	no	no



A common issue of the Cloud simulation tools described above is the lack of support for pure storage Clouds, and also the limited support for workflow deployments on multi-Cloud. To fill these gaps and to evaluate the proposed contributions, we implement in this thesis a multi-Cloud broker simulation framework by extending the CloudSim and WorkflowSim tools. Our choice of CloudSim and WorkflowSim as simulation tools is due to their easily-extendable source code and their proven capability to model brokering and workflow scheduling policies on Cloud.

## 3.6. Summary

This chapter presented the relevant related work in the main research fields addressed in this thesis. First, existing multi-Cloud orchestration frameworks were presented. Then, the related works on the SLA-based resource matchmaking in context of composite Cloud services have been discussed. After that, the chapter presented existing multi-Cloud workflow frameworks and the conducted research work on the scheduling of data-intensive workflows on Cloud. Finally, a detailed comparison of the features included in existing Cloud simulation frameworks and their needed extensions for a simulation-based evaluation of the contributions in this thesis were provided.



## 4. Design and Implementation of a Multi-Cloud Service Broker Framework

The lack of interoperability across Clouds and the difficulty of comparing the costs and SLAs of the Clouds hindered users' ability to deploy their services on multiple Clouds. In this chapter, we address Research questions 1, 2, and 5, as specified in Chapter 1. First, we present a broker-based framework to assist users in the task of selecting and deploying their services on multi-Cloud environments with respect to their needs in terms of SLA and costs. Then, we detail the proposed ontologies for describing service requests and offers. Finally, we present a simulation-based implementation of the broker framework based on the CloudSim toolkit. Many parts of this chapter are based on [90] and [89].

This chapter is organized as follows. In Section 4.1, we present the multi-Cloud service broker framework's fundamental architecture and components. Section 4.2 describes how the SLA-based multi-Cloud service brokering is realized through our framework. In the following section, the ontologies used to describe the IaaS provider offerings and service requests are presented. Finally, in Section 4.4, we discuss challenges of implementation in the simulation framework and describe implementation details.

### 4.1. Multi-Cloud Service Broker Architectural Design

As previously mentioned in Chapter 3, current frameworks for deploying multi-Cloud services are still either in the implementation phase or do not provide all the desired functionalities needed by users. Motivated by this consideration, we propose in this section a generic architecture of a multi-Cloud service broker framework developed to facilitate the IaaS services deployment on multi-Cloud environments. The three-tier architecture of the proposed framework, depicted in Figure 4.1, is composed of the client, the Cloud service broker and the Cloud providers. The Cloud service broker, as shown in the middle of the architecture, serves as a mediator between the users and the Cloud providers. The internal components of every architec-

#### 4. Design and Implementation of a Multi-Cloud Service Broker Framework

ture part are discussed in detail in the following subsections. Their roles are briefly described in Table 4.1.

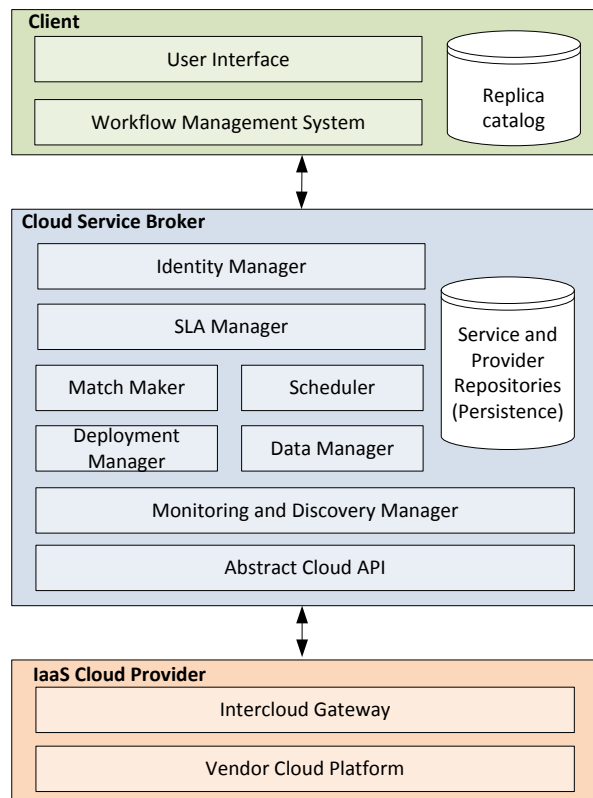


Figure 4.1.: Multi-Cloud service broker architecture.

##### 4.1.1. Cloud Service Broker

The Cloud service broker forms the core of our implemented framework by offering new, added-value services to users. Its main task is to find a suitable target infrastructure platform for running the requested user applications on multi-Cloud. Additionally, its high-level architecture design allows the deployment and monitoring of services on top of heterogeneous Cloud providers.

As can be seen from Figure 4.1, the main component of the broker is a match maker that performs a matching process where the functional and non-functional SLA requirements of users are compared to the monitored SLA metrics as well as the capacity load of the Clouds. The match maker uses different matching algorithms to make a trade-off between the cost and SLA characteristics of the selected Clouds. These are discussed in detail in Chapter 5. Another key component of the broker is the SLA manager. It controls, on behalf of the user, the entire SLA life cycle steps, including

Table 4.1.: Role of the multi-Cloud broker framework components.

<b>Component</b>		<b>Role</b>
Cloud Service Broker	Identity Manager	User authentication and Cloud single-sign-on
	SLA Manager	Management of SLA negotiation and provisioning
	Monitoring and Discovery Manager	Collecting monitoring and resource information from Clouds
	Deployment Manager	Deploying services on the selected Clouds
	Abstract Cloud API	API to interact with heterogeneous Clouds
	Data Manager	Intercloud data transfer management
	Match Maker	Selecting suitable Clouds with respect to SLA
	Scheduler	Scheduling workflow tasks
Client	Persistence	Storing service status and resources information
	WfMS	Management of workflow tasks
	User Interface	Define service requirements and check status
Clouds	Replica Catalog	map data replica to datacenter location
	Intercloud Gateway	standard service frontend for the Cloud provider
	vendor Cloud platform	The native Cloud platform hosted by the Cloud provider

negotiation and provisioning. The deployment of the matched resources as well as their high-level management (e.g. start, stop, suspend) is performed using the deployment manager component. The monitoring manager keeps the current state of the running services and continually updates the Cloud resource's information for the purpose of service discovery. All the collected monitoring data as well the service SLA requirements are stored in dedicated data repositories in the persistence layer of the broker, serving as input for the matchmaking process. Furthermore, the broker architecture contains a scheduler used to distribute the application workloads, i.e., workflow tasks, at the runtime to the selected resources according to different scheduling policies. The data manager is additionally included in the broker architecture to manage Intercloud data transfers between the deployed resources at the runtime. The entire communication of the broker with the underlying Cloud providers is realized through standard interfaces offered by provider hosted Intercloud gateways. The identity manager component, used to handle the user authentication and access control on the Cloud, is out of the focus of this thesis.

The decoupling of the broker component from the provider layer, through the use of standardized management interfaces, also allows the user to directly access his deployed service in case of a broker failure. This reduces the risk of a bottleneck within the centralized broker architecture. In order to reduce the load on one broker, and to ensure a higher availability from the user's perspective, a common solution applied to Grid brokers, is to deploy multiple redundant broker instances.

### 4.1.2. Provider Intercloud Gateway

The Intercloud gateway is the key component of our framework hosted on the provider side to interface the vendor Cloud platform. It acts as a standardized service frontend for the Cloud provider and adds the needed abstraction layer to interact with the broker and, consequently, enables the interoperability on top of the heterogeneous Cloud platforms. Its main role is to provide the broker with common management and monitoring interfaces while hiding the internal provider policies. The interaction of the Intercloud gateways with the broker is realized through an abstract Cloud API.

### 4.1.3. Client

The client provides Cloud users with an interactive user interface to submit their service requests to the broker by describing the functional and non-functional service requirements. Moreover, the user is able to manage and monitor the service after its deployment through a single management console. The client includes also a WfMS to support the deployment of workflow applications on multi-Cloud. It delivers the workflow tasks to the underlying Cloud service broker and takes care of their dependencies. Additionally, a replica catalog is used to manage data replicas.

## 4.2. SLA-based Service Brokering on Multi-Cloud

The goal of the SLA management is the management of service delivery systems to meet the QoS objectives specified in SLAs [75]. As previously mentioned in Chapter 2, the SLA management life cycle in Cloud computing has six steps, which are: discover service providers, define SLA, establish agreement, monitor SLA violation, enforce penalties for violation, and terminate SLA. In the proposed multi-Cloud framework, the broker acts on behalf of the user to handle all the SLA management tasks. In the following subsections we discuss the interaction needed between the different broker components throughout the SLA management steps. Note that the monitoring of SLA violation and the enforcement of penalties at runtime are not covered in this section, as they are out of the focus of the thesis. Nevertheless, the proposed framework architecture can support such mechanisms by integrating methods of autonomic computing like MAPE-K loops [78].

### 4.2.1. SLA Discovery and Definition

In order to discover the SLA features supported by the underlying Cloud providers, the user asks, with the assistance of the user interface, the SLA manager to retrieve the provider SLA templates published through their Intercloud gateways. An SLA

template represents, among other things, general information about the offered services and the margin values of the supported SLA parameters (e.g. supported OS, maximal number of CPU cores, and minimal guaranteed availability). The information retrieved from the SLA templates is stored in a structured data repository in the broker persistence layer.

The next step is the SLA definition. For this purpose, the user can populate a suitable SLA template with required values or even define a new SLA from scratch to describe the functional and non-functional service requirements. Since the latter greatly impact the matching process used in the next SLA management step, the user needs to provide reasonable values.

### 4.2.2. SLA Negotiation

In this step, a negotiation process, managed by the broker SLA manager, is started in order to reach an SLA agreement between the user and the appropriate Cloud providers. The ideal SLA negotiation flow of an incoming SLA user request to the service broker is illustrated by the sequence diagram in Figure 4.2. For simplicity, only one provider is shown on the right side of the figure but, in reality, multiple providers are involved in the negotiation process. The SLA negotiation is done as follows: The user submits a service request (1) with the previously defined SLA to the SLA manager. Then, the SLA Manager, after parsing the SLA definition (2), asks the match maker if it can deploy the service with the specified requirements (3). In order to respond to this request, the match maker starts a matchmaking process to find the best suitable providers (5) by matching the gathered resource properties (e.g., offered services configuration and current SLA metrics) from the monitoring manager (4) with the service requirements by applying predefined matching algorithms. At the end, the user gets a response to his request from the SLA manager with the respective matching results. Upon user acceptance, an agreement can be established (6) with the matched providers and the selected resources could be reserved for a predefined lease period. If none of the providers can be matched, the aforementioned steps may be repeated for renegotiation until an agreement is reached (7). Note that in our negotiation strategy, based on SLA-aware matchmaking, the pricing policies of the Clouds are fixed and non-discriminatory [151], thus, they are independent from the customer type or his geographic location. Additionally, we are not using bargaining-based [162] or auction-based negotiation protocols such as English-auction [22] and double continuous auction [60], which require a dynamic pricing policy. Furthermore, our negotiation objective is to benefit the equally treated users by maximizing their profit without considering the provider profit (i.e., non-cooperative negotiation).

## 4. Design and Implementation of a Multi-Cloud Service Broker Framework

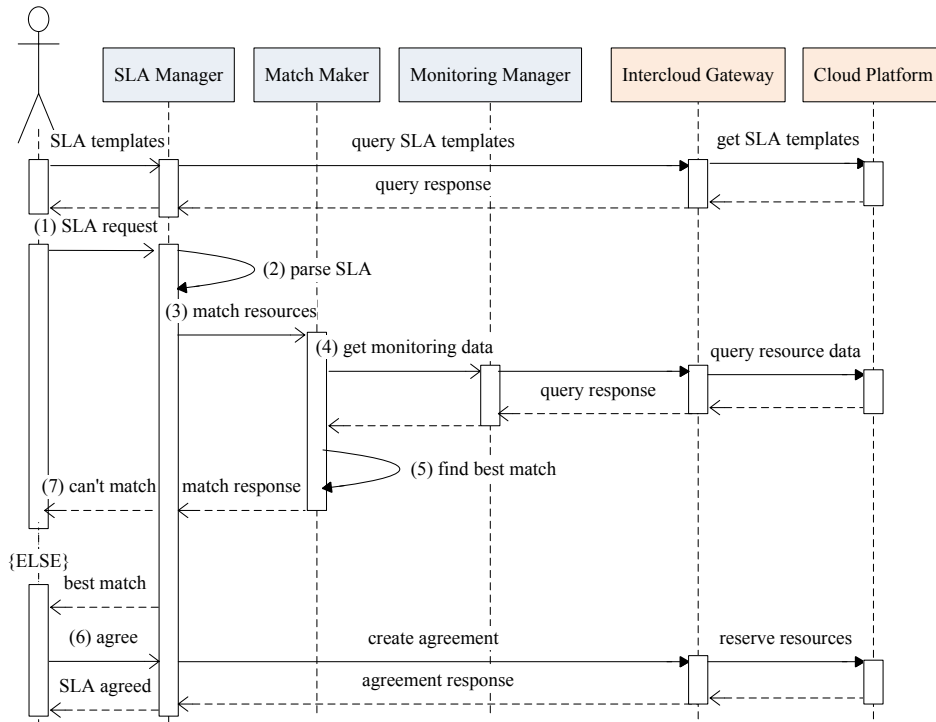


Figure 4.2.: The SLA negotiation flow.

### 4.2.3. SLA Provisioning and Monitoring

The interactions needed by the broker components to provision an agreed-upon SLA are shown by the sequence diagram in Figure 4.3. After establishing an agreement, a provision request (1) is submitted by the user to the SLA manager. After receiving the provision request, the SLA manager translates (2) the associated SLA to a service deployment request and asks the deployment manager to deploy the service (3). The deployment manager forwards the request to the appropriate Intercloud gateways to create and start the requested resources. In response, a unique service ID, used to address the deployed service, is returned. The user is then able to monitor the service (4) by requesting the metric values of the agreed QoS parameters from the monitoring manager. The user is also able to perform actions on the deployed resources (e.g., start and stop) (5). Once an agreement is terminated (6), all the created resources should be released by the Cloud platforms.

## 4.3. Ontological Model

In order to facilitate the semantic storage of SLA and monitoring data collected in the above-described SLA discovery and definition step, in particular, for complex services such as composite services, we make use of two ontologies. These provide the



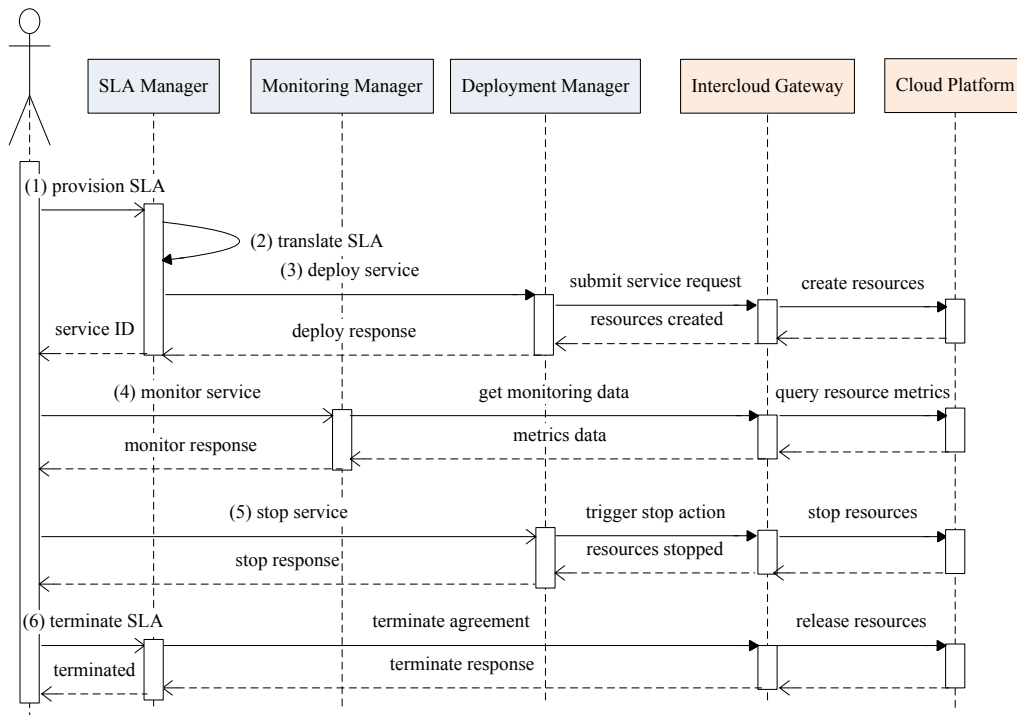


Figure 4.3.: The SLA provisioning and monitoring flow.

domain specific model and vocabulary to semantically describe IaaS service offerings as well as composite service requirements. Some of the terminologies used in these ontologies are taken from the OCCI [119] specification, which defines a standard API to access and manage heterogeneous IaaS Clouds. In the following we present in detail the two designed ontologies.

### 4.3.1. IaaS Composite Service Requester Ontology

The IaaS composite service requester ontology, as depicted in Figure 4.4, captures the user requirements, which are defined as functional properties (e.g., VM type, number of VM instances, storage size) and non-functional properties (e.g. budget, latency, and availability), which represent the QoS requirements. As can be seen from the figure, the functional requirements are specific to each single service, whereas the QoS requirements are global for the composition. This ontology also holds the current status of the composite service throughout all the SLA management steps, which can be one of: requested, matched, unmatched, deployed, failed, started, stopped, or terminated. In addition, each requested service has a unique Identifier (ID) and is associated to a specific customer type.

#### 4. Design and Implementation of a Multi-Cloud Service Broker Framework

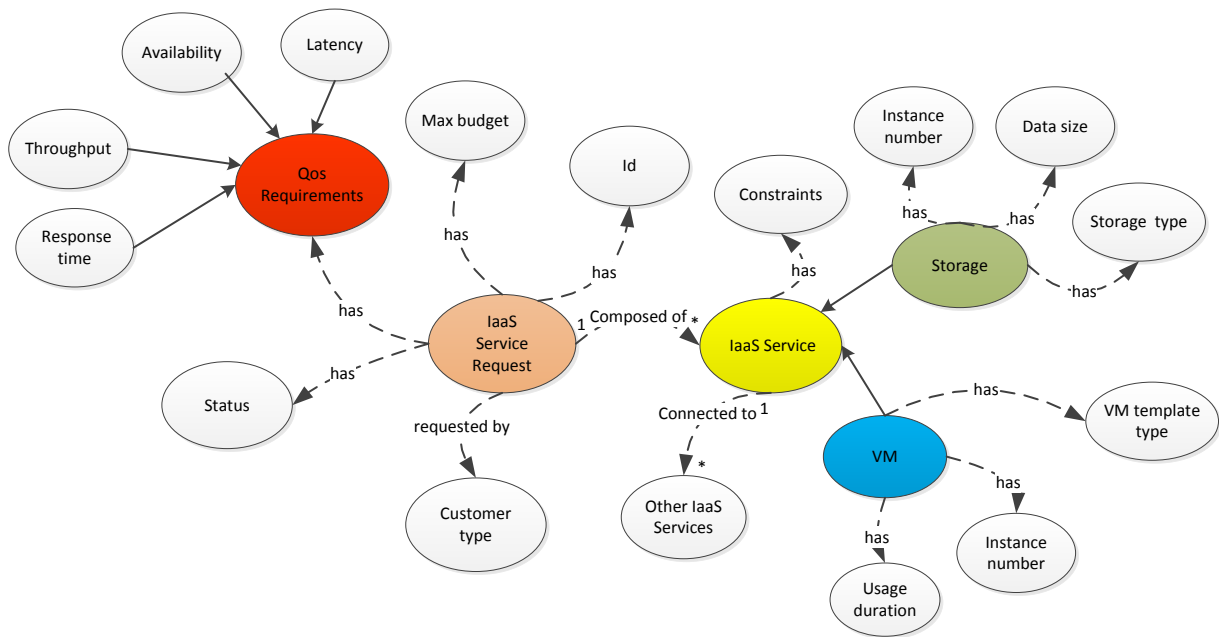


Figure 4.4.: Composite IaaS service requester ontology.

### 4.3.2. IaaS Cloud Provider Ontology

The IaaS provider ontology, as depicted in Figure 4.5, provides an abstract model for describing the provider service offerings, their QoS metrics, and pricing policies. In the current model, we concentrate on the modeling of IaaS offers, including storage and computation. However, the model can be easily extended to support more Cloud delivery models or QoS metrics. Each provider is presented through the ontology with a unique name, ID, and a geographical location. In addition to the pricing policies for the provided computing and storage resources, the provider ontology also captures the network traffic cost charged by each Cloud provider.

## 4.4. Simulation-based Multi-Cloud Service Broker Framework Implementation

Using the Java language, we implemented a simulation environment for the multi-Cloud service broker framework presented above. This allows us to validate the functionality of its components without the setup of a testbed with real Cloud providers, which is extremely time and cost consuming. In addition, the use of simulation allows us to fully validate the proposed broker concept with various scenarios and, hence, to study the developed resource matching and scheduling schemes (see Chapter 7).

#### 4.4. Simulation-based Multi-Cloud Service Broker Framework Implementation

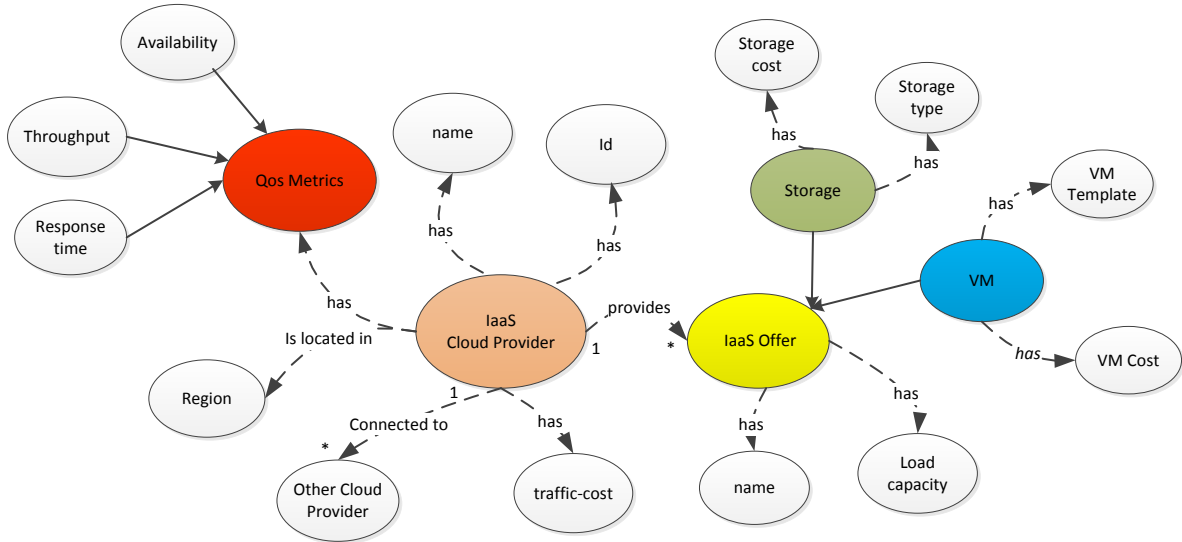


Figure 4.5.: IaaS Cloud provider ontology.

The simulation environment built on top of the CloudSim 3.0 simulation toolkit is depicted in Figure 4.6. In the following subsections we go through all the implemented components by describing the used technologies and tools.

##### 4.4.1. Implementation Details

For implementing a full-functional simulation environment for the multi-Cloud broker, multiple tools have been used and integrated with one another. Figure 4.7 shows a simplified class diagram of the simulation framework based on CloudSim. The classes shown in green are the native CloudSim classes. In the following, we detail the main implemented components and the needed extensions.

##### CloudSim Extensions

CloudSim is a scalable, open-source simulation tool offering features like support for modeling and simulation of large-scale Cloud computing infrastructures, including datacenters, brokers, hosts, and virtual machines (VMs) on a single host. In addition, the support for custom developed scheduling and allocation policies in the simulation made CloudSim an attractive tool for Cloud researchers. In our simulation environment, CloudSim is used to model large-scale and heterogeneous Cloud providers. This allows us, for the purpose of evaluation, to easily configure the amount of Cloud provider resources accessible by the broker. Nevertheless, some CloudSim extensions were needed to allow the dynamic creation, destroying and monitoring of the VMs

#### 4. Design and Implementation of a Multi-Cloud Service Broker Framework

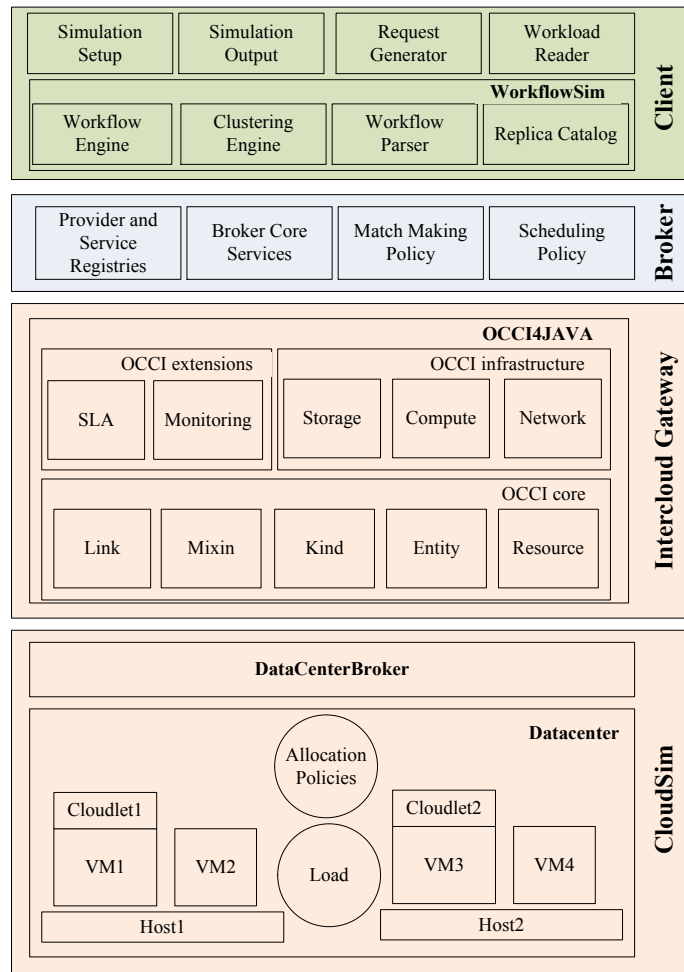


Figure 4.6.: Simulation environment.

during simulation runtime and, therefore, to enable the automatic service deployment in the broker. Furthermore, as CloudSim does not support the simulation of pure storage services, an extension has been developed in the context of a bachelor thesis, supervised within the thesis period, to model the brokering of object storage Clouds [146].

### Cloud Service Broker Implementation

We implemented the core broker services including the SLA manager, deployment manager, the match maker, and the monitoring manager as Java classes included in the Cloud service broker package. The implemented match maker functionality of the broker is extensible enough to permit the easy integration and evaluation of different

#### 4.4. Simulation-based Multi-Cloud Service Broker Framework Implementation

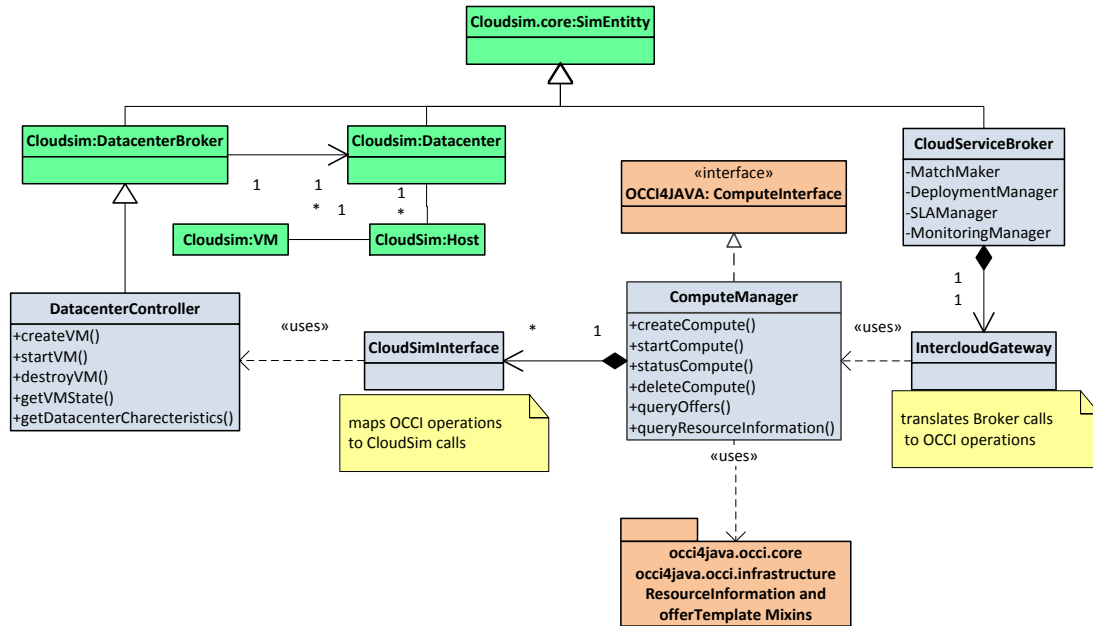


Figure 4.7.: Simulation framework class diagram.

resource matching policies. Furthermore, two persistence classes, named *ServiceRegistry* and *ProviderRegistry*, are used to store and query all the service and provider data stored using the previously presented ontologies during the simulation. The ontologies are implemented in the classes *ServiceRequest* and *Provider*, which are the abstractions of a composite service request and a Cloud provider respectively.

While looking for an abstract Cloud API to access different Cloud platforms, we found that OCCI is the most suitable for our framework. OCCI is an extensible specification for remote management of Cloud infrastructures, allowing the development of interoperable tasks over heterogeneous Clouds. The current OCCI specification, focusing on IaaS Cloud provisioning, defines three abstract resource types, which are compute, storage, and network. All the operations on resources can be requested on a REST manner over Hyper Text Transfer Protocol (HTTP) methods (GET, POST, PUT and DELETE). The use of OCCI as abstract Cloud API allows the broker to act as OCCI client against the Intercloud gateway, which runs as OCCI-server on the provider side.

#### Intercloud Gateway Implementation

In order to simulate the Intercloud gateway component serving as standard service frontend for Cloud providers, we implemented, based on the open source Java implementation for OCCI called OCCI4JAVA [120], an OCCI frontend for CloudSim. In this

#### 4. Design and Implementation of a Multi-Cloud Service Broker Framework

way, the entire communication between broker and providers is forwarded to the native CloudSim DatacenterBroker class through standard OCCI-interfaces. The use of an OCCI-based Intercloud gateway allows us to model a multi-Cloud infrastructure consisting of interoperable Clouds mediated by a Cloud service broker.

In contrast to the OCCI specification, as CloudSim simulations usually run on one host, the broker communicates with the Intercloud Gateway through simple Java object calls instead of using the defined Representational State Transfer (REST)-like methods. Furthermore, we extended OCCI4JAVA with an OCCI monitoring mixin (see Appendix C) to allow the broker to query resource properties like datacenter static information (e.g., location, supported OS, CPU architecture) and the specific datacenter monitoring metrics values from CloudSim.

##### **Request Generator**

The simulation-based evaluation of the broker requires the modeling of realistic service requests to achieve valuable evaluation results. Thus, we implemented a service *RequestGenerator* helper class that continuously generates synthetic computing service requests with different VM types at a configurable rate. The configuration of the VMs is similar to the configuration of the compute instances provided by current commercial Clouds.

##### **Workload Reader**

In order to have more realistic simulation results, we included a *WorkloadReader* class to import the service requests and resource workloads from real workload traces like the Grid workload archive [81] or the PlanetLab trace data [132]. The imported trace data is used then to dynamically generate the CloudSim Cloudlets, which model the workload on the requested VMs. The use of Grid traces is justified by the lack of public accessible real Cloud traces [152].

##### **Simulation Setup/Output**

The broker simulation based on CloudSim requires the modeling of multiple IaaS Clouds with different host and VM configurations, pricing policies, and QoS characteristics. Hence, we implemented helper classes to read the configuration of the datacenters and their pricing information from text files (in the CSV format). This allows us to conduct different experiments by easily changing the Cloud infrastructure each time.

#### 4.4. Simulation-based Multi-Cloud Service Broker Framework Implementation

In order to display the simulation results in the client during and at the end of the simulation, we implemented a helper class to collect and process the different evaluation metrics and to format the output of the results. An example of a simulation output is provided in Appendix D.

### Workflow Management System

For managing large-scale workflows in the simulation environment, we integrated WorkflowSim [24], a modeled version of the Pegasus WfMS [39] developed on top of CloudSim. WorkflowSim contains a workflow mapper to map abstract workflows to concrete workflows, which are dependent on execution sites, a workflow engine to handle the tasks and data flow dependencies, and a clustering engine to reduce the number of tasks by applying different merging techniques. In addition, WorkflowSim includes a workflow parser to import real workflow traces formatted in the Pegasus DAX Extensible Markup Language (XML) format into the simulation. To model the scheduling of workflows using our multi-Cloud broker framework, we extended WorkflowSim to use the scheduler included in the Cloud service broker. The scheduler interface allows the easy implementation of different workflow scheduling policies.

#### 4.4.2. Simple Simulation Use-Case

To show how simple simulation experiments can be conducted with the help of our implemented simulation framework, we address the deployment of synthetic, single computing service requests on Clouds selected using the broker. The simulation flow needed to process the incoming client service requests to the service broker is illustrated by the flow diagram in Figure 4.8. The simulation is done as follows: In the first step, CloudSim is initialized according to the desired simulation scenario. Then, the *RequestGenerator* starts to continuously generate VM provisioning service requests with a variable request arrival rate. All the request and provider data are maintained in the corresponding *ServiceRegistry* and *ProviderRegistry* classes during the simulation. The broker, after receiving the request, asks the match maker if the service can be deployed with the specified requirements. For this, the match maker starts a match-making process to find the best suitable provider by matching the gathered resource information from the monitoring manager with the service requirements and by applying the pre-configured matching algorithms. Upon the existence of a match, the service is automatically deployed and the requested VM is created and started on the selected CloudSim datacenter with the modeled workload traffic (Cloudlet). During the execution time, the VM status is queried periodically by the monitoring manager until the VM is destroyed. If none of the providers can be matched, the request is discarded by the broker. All the aforementioned simulation steps are repeated until reaching the preset maximum number of requests or simulation time limit. In this

#### 4. Design and Implementation of a Multi-Cloud Service Broker Framework

case, the simulation is terminated and the output results are displayed in the client.

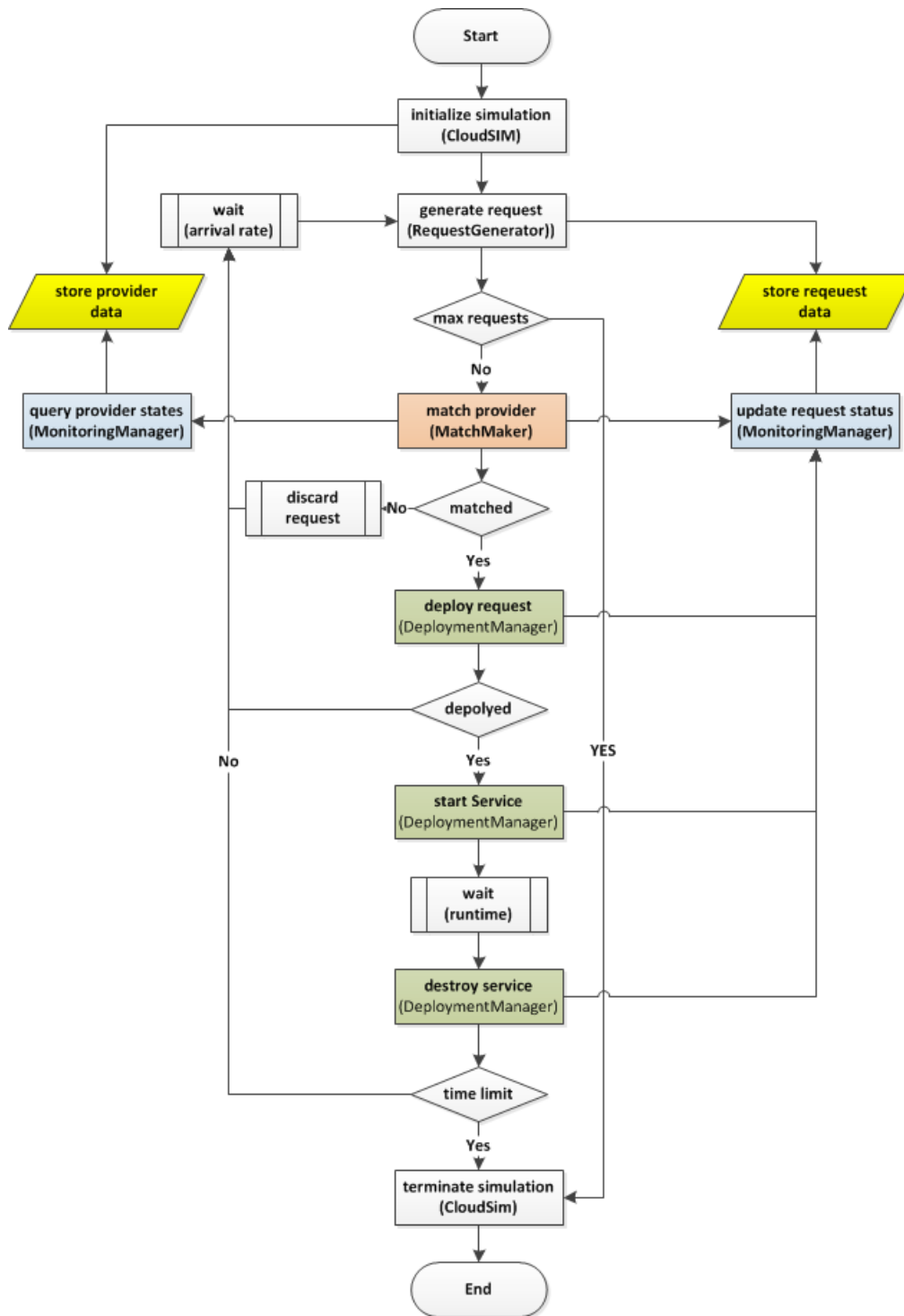


Figure 4.8.: Simulation flow.



## 4.5. Summary

In this chapter we described the architectural design of a Cloud service broker and proposed a simulation environment to test and evaluate the broker. Specifically, we focused on the SLA management and interoperability features included in the broker, allowing the customized deployment of services on multi-Cloud. In addition, this chapter presented an ontological model consisting of two ontologies, which are used to semantically describe the composite service requests and IaaS provider offerings. Finally, the chapter discussed the implementation details of the simulation environment by listing the used tools and technologies.



## 5. SLA-based Matchmaking of Composite Cloud Services

After presenting the architecture of the multi-Cloud service broker framework in the previous chapter, we focus in this chapter on solving the matchmaking problem for multi-Cloud composite services, specified as Research question 3 in Chapter 1. For this purpose, we propose two matching algorithms capable of matching user requirements in terms of functional and non-functional SLA parameters, which have been published in [88] and [87]. The main algorithm is a novel, utility-based matching algorithm adopted from the multi-attribute auction theory with the goal of maximizing the user utility. The second algorithm is a simple matching scheme called sieving, which we used as the base for evaluating the utility-based algorithm. These two policies are consumed by the broker's match maker component to automate the selection of the Clouds, which is the main broker function.

The remainder of this chapter is organized as follows. In Section 5.1, a graph-based mathematical formulation of the problem as well as our assumptions are provided. The sieving algorithm functionality is described in Section 5.2. Section 5.3 introduces the utility-based matching algorithm and analyzes its time complexity. Finally, in Section 5.4, a hybrid utility-based genetic algorithm, called HU-GA, is proposed to facilitate the selection of large-scale service compositions.

### 5.1. Matchmaking Problem Formulation

A well-known and efficient method for solving complex problems in many science fields is the use of the graph theory [41]. Motivated by this consideration, we use graphs to model the matching of multi-Cloud composite services. Each requested multi-Cloud composite service can be modeled as a fully connected, undirected graph  $G(S,E)$  called *Intercloud graph*, where each node represents a single compute or storage Cloud service and the edges represent the network connectivity between the nodes. An example of an *Intercloud graph* with three requested services (two VMs and one storage) with a possible concrete deployment is presented in Figure 5.1.

The matchmaking problem with multi-Cloud consists of finding the service composition that minimizes the deployment cost and best satisfies all user QoS requirements. This problem can be modeled as Multi-Choice Multidimensional Knapsack problem

## 5. SLA-based Matchmaking of Composite Cloud Services

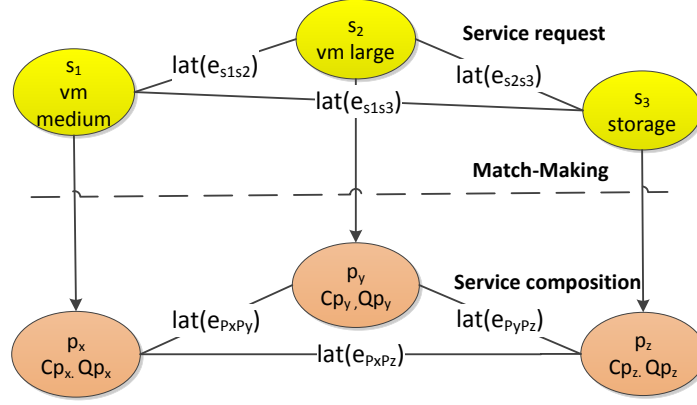


Figure 5.1.: Example of an Intercloud graph for a composite multi-Cloud service.

(MMKP), which is known to be np-hard [131, 4]. In order to formulate this multi-objective optimization problem, we introduced, based on the *Intercloud graph* representation of the composite service, a mathematical model. The latter is used to model composite service requests, Cloud providers, and candidate service compositions, which constitute the input parameters for the matching algorithms. The description of the model parameters is provided in Table 5.1.

Table 5.1.: Multi-Cloud matchmaking model.

	Parameter	Description
Request	$VM = \{vm_1, \dots, vm_a\}$	set of requested $a$ compute services
	$ST = \{st_1, \dots, st_b\}$	set of requested $b$ storage services
	$S = \{s_1, \dots, s_n\}, S = VM \cup ST$	set of $n$ services forming the composition
	$E = \{e_{s_i s_j}   s_i, s_j \in S, i \neq j\}$	set of all $f$ edges connecting the $n$ services
	$L = \{lat(e_{s_i s_j})   s_i, s_j \in S\}$	set of edge latencies
	$Q_r = \{q_1(r), \dots, q_m(r)\}$	set of $m$ required QoS values
	$C_r = \{c_{vm}(r), c_{st}(r), c_{tr}(r)\}$	max payment for VM, storage and traffic
	$D = \{d_{st}, d_{tr}\}$	required storage and data traffic size
	$T$	provisioning (lease) time
Provider	$P = \{p_1, \dots, p_l\}$	set of $l$ candidate IaaS providers
	$Q_{p_j} = \{q_1(p_j), \dots, q_m(p_j)\}$	measured QoS metrics for provider $p_j$
	$C_{p_j} = \{c_{vm}(p_j), c_{st}(p_j), c_{tr}(p_j)\}$	pricing policies for provider $p_j$
	$O_{p_j} = \{o_1(p_j), \dots, o_k(p_j)\}$	set of service types $k$ offered by provider $p_j$
Candidate	$X = \{x = \bigoplus s_i \rightarrow p_j   s_i \in S, p_j \in P\}$	set of possible service compositions $x$
	$Q_x = \{q_1(x), \dots, q_m(x)\}$	set of QoS values for service composition $x$
	$C_x = \{c_{vm}(x), c_{st}(x), c_{tr}(x)\}$	total usage cost for composition $x$

As can be seen from Table 5.1, in our problem formulation we do not consider the functional SLA parameters, since it is assumed that, for each requested service type, the functional parameters are fulfilled by the candidate Clouds if they offer a service with the requested configuration and their current load capacity allows the deployment of that service. For the QoS parameters, we consider in this thesis only four QoS parameters, which are availability, response time, throughput, and latency. The detailed description of these parameters and how they are measured are given in Table 5.2. Of course, it is possible to support more QoS parameters (e.g. reputation, security, etc.), but we argue that these four parameters impact more the performance of multi-Cloud applications and thus the user satisfaction [147].

Table 5.2.: Description of the used QoS parameters in the matching.

QoS Parameter	Measured from/to	Description
Availability	Cloud	percentage of the guaranteed service uptime
Response time	client-To-Cloud	amount of time to respond to a client request e.g. download and display of a website
Throughput	client-To-Cloud	number of processed requests per time unit e.g. size of transferred data per second
Latency	Cloud-To-Cloud	amount of time to transfer data packets between two Clouds

Since we do not know the exact amount of data to be transferred between the requested services at the runtime, we made the following assumptions on data traffic for our application needs:

1. Users are charged only for data traffic from Cloud to the Internet.
2. The data transfer inside the same provider is free of charge.
3. All the requested storage services will store the same amount of data (data is replicated).
4. The data transfer between two connected nodes is bidirectional.
5. The requested amount of data traffic is equally distributed between the connected Clouds.

Based on the mathematical model proposed above and our assumptions, the match-making problem can be formulated as follows:

$$\begin{cases} \max th(x), av(x) & \text{with } th(x) \geq th(r) \wedge av(x) \geq av(r) \\ \min rt(x), lat(x) & \text{with } rt(x) \leq rt(r) \wedge lat(x) \leq lat(r) \\ \min C_x(T) & \text{with } C_x(T) \leq C_r(T) \end{cases}$$

## 5. SLA-based Matchmaking of Composite Cloud Services

Where,  $Q_x = \{rt(x), th(x), av(x), lat(x)\}$  denote respectively the set of aggregated SLA values of response time, throughput, availability, and latency for the candidate composite service  $x$ , and  $Q_r = \{rt(r), th(r), av(r), lat(r)\}$  is the set of their minimum or maximum acceptable values. The former are calculated from their corresponding values in the component services by applying the aggregation functions used in [170], as presented in Table 5.3.  $C_x(T)$  denotes the total predicted costs (compute, storage, and data transfer cost) for using the candidate composite service  $x$  during the time period  $T$ . Based on the above assumptions,  $C_x(T)$  can be computed as follows:

$$C_x(T) = T * \left( \sum_{i=1}^a c_{vm}(vm_i) + d_{st} \sum_{i=1}^b c_{st}(st_i) + \sum_{\substack{s_i, s_j \in S, i \neq j \\ e \in E}} \frac{d_{tr} * c_{tr}(e_{s_i s_j})}{2 * f} \right) \quad (5.1)$$

with:

$$c_{tr}(e_{s_i s_j})_{s_i, s_j \rightarrow p_g, p_h} = \begin{cases} c_{tr}(p_g) + c_{tr}(p_h) & \text{if } g \neq h \\ 0 & \text{else} \end{cases} \quad (5.2)$$

Where  $p_g, p_h \in P$  denote the Cloud providers allocated respectively to the requested services  $s_i, s_j \in S$ .  $C_r(T)$  denotes the total user budget for the period of usage  $T$ . It is calculated as follows:

$$C_r(T) = T * \left( a * c_{vm}(r) + b * d_{st} * c_{st}(r) + d_{tr} * c_{tr}(r) \right) \quad (5.3)$$

Table 5.3.: Aggregated SLA attributes for a composite service  $x$ .

SLA Attribute	Aggregation function
Throughput	$th(x) = \min th(s_i) \forall s_i \in S$
Response time	$rt(x) = \max rt(s_i) \forall s_i \in S$
Latency	$lat(x) = \frac{\sum_{e \in E} lat(e_{s_i s_j})}{f} \forall s_i, s_j \in S, i \neq j, f$ is number of edges in E
Availability	$av(x) = \prod av(s_i) \forall s_i \in S$

## 5.2. Sieving Matching Algorithm

A general scenario to use our previously proposed multi-Cloud service broker framework is as following: the customer specifies several requirement parameters and the broker compares the parameters one by one with the performance metrics and pricing policies of the providers, thereby finding the best matched Clouds. This is a traditional way of solving selection problems simply in various scientific domains. In the following we present this simple matching scheme called sieving.

### 5.2.1. Sieving Selection Strategy

The sieving matching algorithm deals with two kinds of input sets, one of which are the user requirement parameters, and the other, the Cloud provider's specific performance and cost metrics. The output of the algorithm is a set of selected Clouds that match all of the requirement attributes. These inputs/outputs can be depicted using the following mathematical forms:

$$\text{Set of SLA requirement attributes: } R = \{R_1, R_2, R_3, \dots, R_m\}$$

$$\text{Set of SLA metrics for Cloud provider } i: P_i = \{M_1, M_2, M_3, \dots, M_n\}$$

$$\text{Selected providers: } C = \{C_1, C_2, C_3, \dots, C_k\}$$

Where  $R_i$  is a single requirement attribute in the form of  $A > (\text{or } <, \geq, \leq) \text{value}$  (e.g.,  $\text{price} \leq 0.02\$/h$ ),  $M_i$  is a performance attribute of a Cloud provider in the form of  $M = \text{value}$  (e.g.,  $\text{availability} = 100\%$ ) and  $C_i$  is the name of a Cloud platform. The task of the algorithm is to filter all unqualified providers out of the candidates. A Cloud is regarded as unqualified when at least one of its performance or cost attributes does not match the customer requirements.

To make this algorithm more understandable, we use an easy example with three parameters: response time, availability, and throughput, and illustrate the functionality of the algorithm in Figure 5.2 (The algorithm can handle more parameters; but we can demonstrate maximally three parameters in the graphic).

In the concrete example, the customer specified the following requirements:  $5s < R_1(\text{ResponseTime}) < 20s \wedge R_2(\text{Availability}) > 0.9 \wedge R_3(\text{Throughput}) \geq 5\text{Mbit/s}$ . In Figure 5.2, the requirements are presented with the axes, while the Cloud providers, marked with "x", are located in the 3D space based on the values they provide for the three attributes. The ovals marked "x" are Clouds that must be sieved out because they do not fulfill at least one of the user requirements. For example, the two ovals standing vertically to the x-axis are the Clouds that have a value of either smaller than 5 or greater than 20 in response time. According to the user specification  $5s < \text{ResponseTime} < 20s$ , these Clouds are regarded as unqualified. The algorithm works similarly with other parameters. Note that the axes are not equally scaled due to the large difference in user-specified parameter values.

In case more than one Cloud meets the user requirements (the four "x"s outside the ovals), we simply choose one of them randomly. Since sieving is used only for comparative study, it is not necessary to implement a complicated scheme.

The advantage of this algorithm is accuracy, i.e., the selected Clouds absolutely meet the user's requirements. However, it may be possible that the result set is empty in the case that none of the Cloud providers fulfill the criteria. The problem with the algorithm lies in flexibility, where no negotiation is allowed. Hence, it cannot handle use cases like " $R_1$  is important and  $R_3$  can be lower if no qualified provider is found".

## 5. SLA-based Matchmaking of Composite Cloud Services

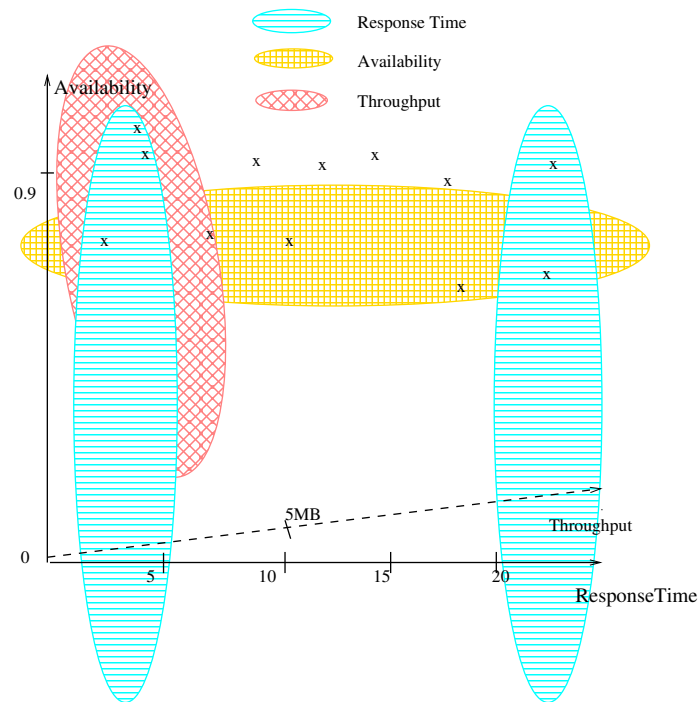


Figure 5.2.: Functionality of the sieving algorithm with three SLA parameters.

### 5.2.2. Sieving Implementation

The implementation of the sieving algorithm using the mathematical model from Table 5.1 is described using the pseudo-code in Algorithm 5.1.

```

1  Input:  $S, P, c_{vm}(r), c_{st}(r), av(r), th(r), rt(r)$ 
2  For each  $s$  in  $S$  Do
3    For each  $p$  in  $P$  Do
4      If  $s$  is Deployable in  $p.getOfferList()$  Then
5        If  $c_{vm}(p) \leq c_{vm}(r) \ \& \ c_{st}(p) \leq c_{st}(r) \ \& \$ 
6           $av(p) \geq av(r) \ \& \ rt(p) \leq rt(r) \ \& \ th(p) \geq th(r)$ 
7          Then
8            add  $p$  to CandidatesList
9          Endif
10         Endif
11       Endfor
12     If  $sizeof(CandidatesList) > 0$  Then
13       Choose random  $p$  from CandidatesList
14       CloudCompositionMap.add( $s, p$ )
15     Else return null
16     Endif
17   Endfor
18  Output: CloudCompositionMap

```

Algorithm 5.1: Sieving Matching Algorithm



As shown in Algorithm 5.1, the sieving algorithm receives as input (line 1) a service list  $S$  forming the requested multi-Cloud service composition, a list  $P$  of candidate IaaS providers, the maximal payment, and the minimum desired service quality, which is expressed with the minimally allowed throughput and availability, and the maximum allowed response time required by the customer. In the first step, for each requested service (VM or storage), the algorithm iterates through all providers (lines 2-4) to match the offers that fulfill the functional requirements. Concretely, it checks if the requested component service is offered by the provider. In addition, it checks if the current datacenter capacity load allows the deployment of the requested service type. After that, the algorithm checks (lines 5-6) if the provider's QoS metrics are within the ranges specified by the user while satisfying his maximum willingness to pay. All the matched providers, which satisfy all functional and non-functional SLA requirements, are stored in a candidates list (lines 7-11). In the second step (lines 12-17), the algorithm selects for each component service a random provider from the candidates list to serve the request and adds it to the target Cloud composition map forming the output of the algorithm. In case a requested component service cannot be matched to a Cloud, a null object is returned.

#### 5.2.3. Sieving Algorithm Time Complexity

In the following we analyze the time complexity of the sieving algorithm described above. As shown in the above pseudo-code, in the worst-case scenario of the first phase (lines 2-11), for each of the  $n$  requested component services, the algorithm has to go through the supported offer list of size  $k$  for each of the  $l$  candidate providers to match the SLA requirements. This gives a worst-case complexity of  $\mathcal{O}(n.l.k)$ . The second part of the algorithm (lines 12-17), used for selecting a random provider, has a constant time complexity of  $\mathcal{O}(1)$ . Therefore, the total sieving algorithm worst-case complexity is:

$$\mathcal{O}(n.l.k) \quad (5.4)$$

When assuming that all the candidate providers offer a constant number of service types ( $l$  and  $k$  are constants), the reduced algorithm complexity becomes:

$$\text{sieving\_complexity} = \mathcal{O}(n) \quad (5.5)$$

This result confirms that the time complexity of the sieving algorithm is proportional to the number of requested services.

### 5.3. Utility-based Matching Algorithm

A major issue of the sieving matching algorithm described above is the lack of flexibility in the matching of non-functional SLA attributes, meaning it cannot handle

use cases, such as availability being more important than throughput or selecting well-qualified providers while keeping the total costs low. In addition, the network connectivity between the Clouds and traffic costs are ignored in the matching. Therefore, we propose a new, economic, utility-based matching algorithm, which takes the payment of customers and their QoS preferences as the focus. In contrast to sieving matching, where both functional and non-functional SLA requirements must be fulfilled (hard constraints), in utility-based matching, only the former must be fulfilled, while the latter are satisfied with respect to the user preferences on a best-effort basis (soft constraints).

In this section we describe the utility-based algorithm in detail.

### 5.3.1. Utility-based Selection Strategy

In economic theory, consumer behavior is explained through preference relations facilitating pairwise comparison of goods [109]. The corresponding quality ordering is succinctly captured through ordinal utility values determined for each good [3]. These utility values can be interpreted in terms of the relative fulfillment of consumer preferences. In other words, to say that a good (service)  $A$  is preferred over a good (service)  $B$  means that  $A$  yields a higher utility than  $B$ . Thus, utility values describe consumer preference relations over goods and services. Selecting the “best” service from all functionally fitting services available from the broker side turns into the problem of finding the service with the highest utility value. That is, the assumed customer preferences have to be modeled in an appropriate utility function.

In order to model the user profit from a composite service  $x$ , which assures a service quality  $Q_x$ , we use a quasi-linear utility function [106] adopted from the multi-attribute auction theory [9]. The utility of a customer  $i$  from using the candidate service composition  $x$  during the period  $T$  is the product of his willingness to pay and the relative fulfillment of his service requirements minus the charged leasing price. It is computed as follows:

$$U_{ix}(Q_x, T) = C_{r_i}(T) * F_i(Q_x) - C_x(T), \quad (5.6)$$

Where  $C_{r_i}(T)$  represents the *maximum willingness to pay* of consumer  $i$  for an “ideal” service quality in the period  $T$  (see Equation 5.3),  $C_x(T)$  is the total service usage cost (see Equation 5.1), and  $F_i(Q_x)$  is the customer’s *scoring function* translating the aggregated service quality attribute levels into a relative fulfillment level of consumer requirements. The *scoring function* is calculated as follows:

$$F_i(Q_x) = \sum_{j=1}^m \lambda_i(q_j) * f_i(q_j) \rightarrow [0, 1], \quad (5.7)$$

Where  $\lambda_i(q_j)$  and  $f_i(q_j)$  denote respectively the relative assessed weight and the fitting function for consumer  $i$  regarding the SLA attribute  $q_j$ , where  $\sum_{j=1}^m \lambda_i(q_j) = 1$ . The fitting function maps properly to the user behavior each measured SLA attribute to a normalized real value in the interval  $[0,1]$  with 1 representing an ideal expected SLA value. Clearly, the careful choice of fitting function specification is crucial for properly describing user behavior. For the reader's convenience we refer to [103, 105] for an in-depth treatise of preference elicitation, i.e., deriving and parameterization of the fitting functions for each consumer. An example for non-linear fitting functions is provided in Chapter 7.

A broker pursuing a utility-based service selection, i.e., matching the customers with the corresponding utility-maximizing service offering, would then select the optimal service composition  $x$  if it is feasible and if it leads to the maximum utility value with:

$$U_{ix_{optimal}}(Q_x, T) = \max_{x \in X} U_{ix}(Q_x, T) \quad (5.8)$$

where  $X$  is the set of possible Cloud service compositions (solution space). Hence, the matchmaking problem can be formulated with a search for the Cloud composition with the highest utility value for the user. Note that choosing a service with negative utility is irrational as the price exceeds the willingness to pay. These service offers can therefore be discarded by the decision maker. Note that this social benchmark allocation can only be achieved under the assumption of non-strategic customers. In the presence of strategic (i.e., non truth-telling) customers, it is needed to apply screening approaches as outlined by [103]. A corresponding portfolio design problem is addressed by [104].

The functionality of the utility-based algorithm for matching a single Cloud service using the broker framework is described in the sequence diagram from Figure 5.3.

In the first step (1), the user forwards his functional requirements to the broker. In response (2), the broker checks the possible provider candidates that are able to deploy the service in order to fulfill the functional SLA requirements. In the next step, (3) the user gives his QoS requirements expressed in the weighted fitting functions and his maximal payment. After that, the broker calculates the service utility based on the current Cloud characteristics and returns the provider offer giving the maximal utility to the user (4). Upon user acceptance (5), the offer can be deployed on the selected provider (6).

#### 5.3.2. Utility-based Algorithm Implementation

The working strategy of the utility-based algorithm using the mathematical model described above is presented in Algorithm 5.2.

## 5. SLA-based Matchmaking of Composite Cloud Services

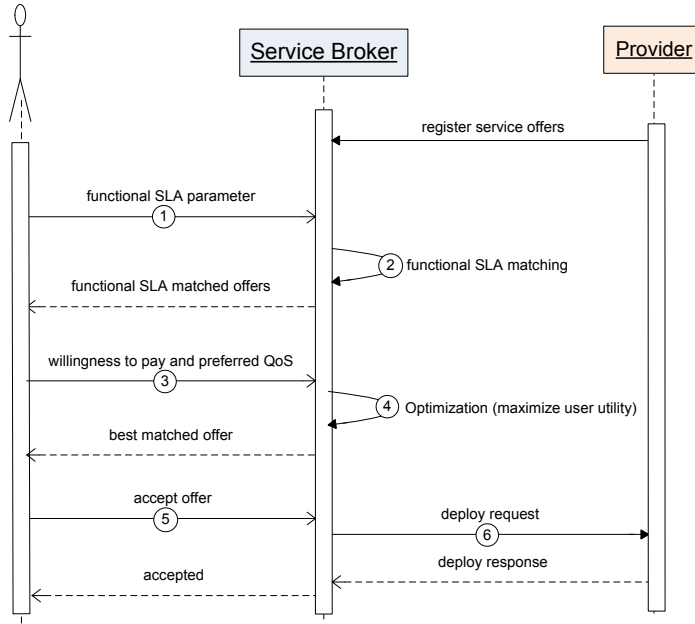


Figure 5.3.: Functionality of the utility-based matching algorithm.

```

1  Input: customer  $i, X, S, P, c_{vm}(r), c_{st}(r), f(av), f(th), f(rt), f(lat), \lambda(av), \lambda(th), \lambda(rt), \lambda(lat), T$ 
2  For each  $x$  in  $X$  Do
3      isFeasible=true
4      For each service  $s$  in  $S$  Do
5           $p = x.getAllocatedProvider(s)$  //  $p$  is allocated to  $s$  in composition  $x$ 
6          If  $s$  isnotDeployable in  $p.getOfferList()$  Then
7              isFeasible=false
8              break
9          Endif
10         Endfor
11         If feasible=true Then
12             calculate  $th(x), av(x), rt(x), lat(x), C_{r_i}(T), C_x(T)$ 
13              $F_{ix} = \lambda(av) * f(av(x)) + \lambda(th) * f(th(x)) + \lambda(rt) * f(rt(x)) + \lambda(lat) * f(lat(x))$ 
14              $U_{ix}(T) = C_{r_i}(T) * F_{ix} - C_x(T)$ 
15             If  $U_{ix}(T) > 0$  Then
16                 UtilityMap.add( $x, U_{ix}(T)$ )
17             Endif
18         Endif
19     Endfor
20     If sizeof(UtilityMap) > 0 Then
21         Sort UtilityMap by utility values in descending order
22          $x_{optimal} = UtilityMap.firstentry().getKey()$ 
23     Else return null
24     Endif
25 Output: optimal service composition  $x_{optimal}$ 
    
```

Algorithm 5.2: Utility-based Matching Algorithm

Unlike the sieving algorithm, the utility-based algorithm operates on the set  $X$  of possible composite services to find the optimal service composition to deploy the requested composite service  $S$  modeled as *Intercloud graph*. Herewith, it is also able to support the Cloud-to-Cloud traffic and latencies in the matching process. In addition, the algorithm takes as input the customer-specific fitting functions and their weight factors expressing the requested service quality (line 1). As shown in the pseudo-code presented in Algorithm 5.2, the first step (lines 2-10) followed by the utility-based algorithm, which is used to check the functional requirements, is similar to the sieving algorithm. A candidate Cloud composition is feasible if each service is allocated to a provider allowing its deployment. In the next step (lines 11-19), the non-functional requirements are checked for each feasible candidate composition by calculating its utility profit from the customer perspective using Equation 5.6. A *UtilityMap* set keeps the mapping between the gathered utility values and the corresponding service composition  $x$ . After that (lines 20-24), the algorithm sorts the positive utility values in the *UtilityMap* by a descending order. As noted before, negative values can be discarded beforehand. As a result, it returns the key of the maximum obtained utility value representing the optimal service composition for the deployment.

#### 5.3.3. Utility-based Algorithm Time Complexity

In the following, we analyze the theoretical time complexity of the utility-based algorithm described above. For a service composition request formed of  $n$  component services and with  $l$  candidate providers, there are  $x = l^n$  possible combinations to be evaluated. The time complexity of the first step (lines 2-10) is similar to the sieving algorithm with the difference that the utility-based algorithm operates on the  $x$  combinations; consequently, it has a complexity of  $\mathcal{O}(x.n.k)$ . As shown in the pseudo-code, the second step of the algorithm (lines 11-19) is composed of two sub-processes. The first sub-process is used to calculate the aggregated SLA parameters. According to Table 5.3 the aggregated availability, response time and throughput values are calculated from the  $n$  requested services with a complexity of  $\mathcal{O}(n)$ . Regarding the aggregated latency, its value is calculated based on the number of edges connecting the  $n$  services. Since the number of edges in a full connected graph is  $f = \frac{n.(n-1)}{2}$ , this gives us a complexity of  $\mathcal{O}(n^2)$  for the aggregated latency calculation. The second sub-process is used to calculate the utility for the feasible combinations. The therefore used operations have a constant complexity, that is  $\mathcal{O}(1)$ . In the worst-case scenario, if all combinations are feasible, the resulted total time complexity of the second step is then  $\mathcal{O}(x.n^2)$ . As described in the above pseudo-code, in the third step the algorithm performs a merge-sort on the *UtilityMap* values (lines 20-24). The worst-case complexity of this step is  $\mathcal{O}(x.\log x)$ .

The total worst-case complexity of the proposed utility-based algorithm is a result of the sum of its three steps complexity, which can be expressed as follows:

$$\mathcal{O}(x.(n.k + n^2 + \log x)) = \mathcal{O}(l^n.(n.k + n^2 + n.\log l)) \quad (5.9)$$

## 5. SLA-based Matchmaking of Composite Cloud Services

If we maintain the number of candidate providers  $l$  and their supported service offers  $k$  constant, the resulted total worst-case complexity becomes:

$$utility\_complexity = \mathcal{O}(n^2.l^n) \quad (5.10)$$

This result proves the exponential time complexity of the algorithm and the np-hardness of the matching problem.

### 5.4. HU-GA Matching Algorithm

The previously presented implementation of the utility-based algorithm based on exhaustive search performs the matching by calculating the utility for all possible service compositions and then selecting the composition with the maximal utility that satisfies the budget and QoS requirements. Clearly, this approach can find an exact solution of the matching problem; however, its computation cost is expensive, especially for large service compositions due to the np-hardness of the problem. Since evolutionary-based approaches are a commonly used method to solve complex optimization problems, we adopted a single objective genetic algorithm called hybrid utility-based genetic algorithm (HU-GA) to solve the matchmaking problem using the utility-based algorithm from Section 5.3. Our adoption of the genetic algorithm involves six steps which are depicted in Figure 5.4.

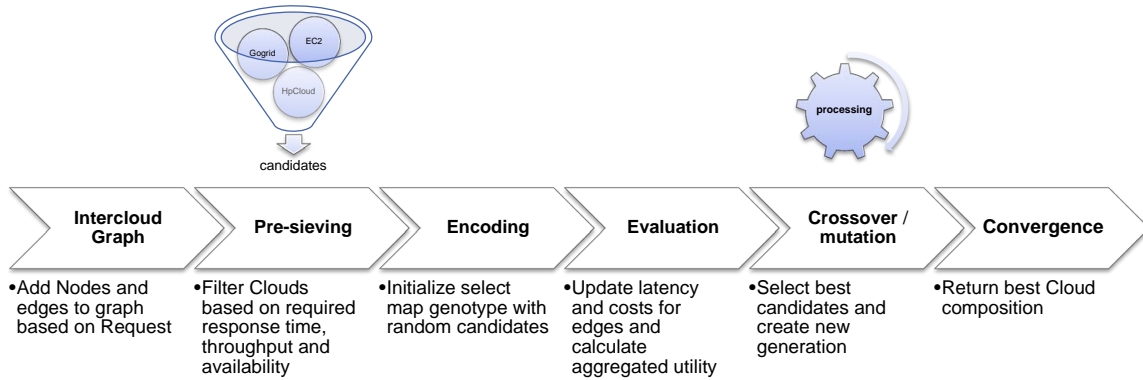


Figure 5.4.: Functionality of the hybrid utility-based genetic algorithm.

In the first step, the algorithm takes as input a composite service request presented as *Intercloud graph*. In the next step, called pre-sieving, we filter out candidate providers that do not satisfy the QoS requirements for throughput, availability, and response time similar to the sieving algorithm. Therewith, unfeasible solutions can be removed from the solution space and the convergence of the algorithm can be accelerated.

The third step is to create the population by generating random composite service candidates called individuals. As can be seen in Figure 5.5, each individual is presented with a chromosome consisting of multiple genes, which are encoded using a selection map data structure. Each entry in the map represents a gene that has a graph node presenting the requested single service as key and the list of candidate providers capable of deploying the service as value.

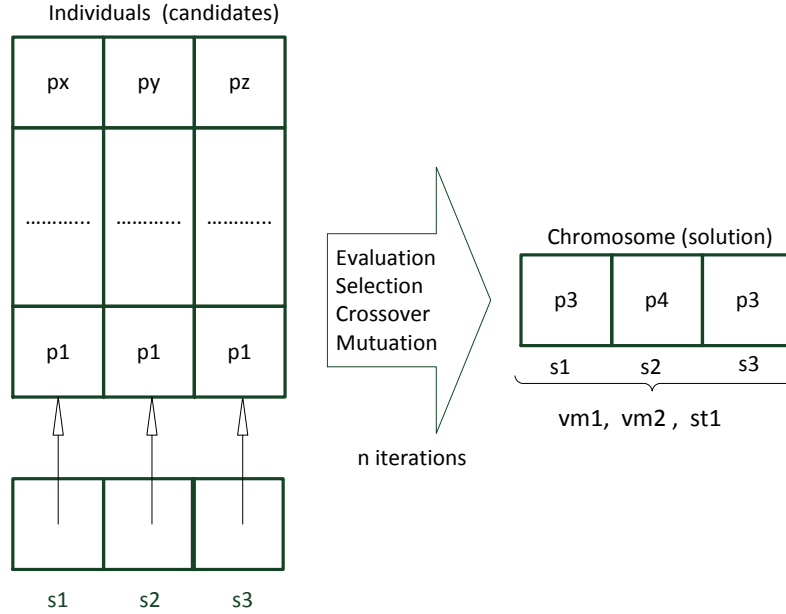


Figure 5.5.: Genetic encoding of the composite service candidates.

In the fourth step, the composite service candidates  $x$  (individuals) from each generation are evaluated against a fitness function. In our adopted HU-GA algorithm, the fitness function that needs to be maximized as shown in Equation 5.11 is equal to the utility function from Equation 5.6. Additionally, we use a "death penalty" function to penalize candidates that do not satisfy the service constraints and to discard Cloud compositions with a negative utility.

$$\max_{x \in X} \text{fitness function} = \begin{cases} 0 & \text{if constraints are violated} \\ U_{ix}(Q_x, T) & \text{otherwise} \end{cases} \quad (5.11)$$

The next step is the evolution of the population based on the crossover and mutation genetic operators. Herewith the elite candidates with the best fitness values will survive in the next generation and are used to create a new population, while the bad candidates will be discarded. This step is repeated many times until the algorithm reaches the convergence. Finally, when the genetic algorithm reaches convergence, the optimal Cloud composition is outputted as final solution.

## 5.5. Summary

This chapter addressed the problem of composite services selection on multi-Cloud with the goal to benefit users in terms of service quality and costs. At the beginning of the chapter, a mathematical formalization of the problem has been presented. In this formal model, the requested composite service is modeled with a fully connected *Intercloud graph*. Also, the methods for calculating the total cost and aggregated SLA parameters, both used as evaluation parameters in the matching, are described.

The chapter also presented two matching policies to tackle the matching problem and analyzed their time complexity. The first scheme is a simple scheme called sieving, which randomly selects the Cloud providers satisfying both functional and non-functional SLA without considering their connectivity and latencies. The second scheme is a utility-based matching scheme that leverages a quasi-linear utility function and the introduced graph-based mathematical model to find the optimal Cloud composition maximizing the user utility. In contrast to sieving, utility relies on user-specific scoring functions instead of fixed range intervals to express the user satisfaction level against the SLA. Finally, the chapter presented a hybrid utility-based genetic algorithm, called HU-GA, to facilitate the matching of large-scale service compositions. The evaluation of all the proposed algorithms will be elaborated in Chapter 7.



## 6. Optimized Multi-Cloud Workflow Deployment

A challenging task for the multi-Cloud resource allocation is how to optimize several user objectives like minimizing costs and makespan while fulfilling the user required functional and non-functional SLAs. Since data transfers in multi-Cloud are performed through Internet between datacenters distributed in different geographical locations, another challenge is how to distribute the workloads on these Clouds in order to reduce the amount and cost of Cloud-to-Cloud (Intercloud) data transfers.

In this chapter we present a multi-dimensional resource allocation scheme to optimize the deployment of data-intensive large-scale applications using our proposed multi-Cloud service framework, taking scientific workflows as our example. The scheme applies a two-level approach in which the target Clouds are selected using first the HU-GA matching algorithm presented in the previous chapter and then the application workloads are distributed to the selected Clouds using a data locality-driven scheduling policy. Furthermore, the chapter presents a replica-supported data management policy to manage the Intercloud data transfer at the runtime. This chapter addresses Research question 4 from Chapter 1. Its major parts are largely based on [91] and [86].

The chapter is organized as follows. Section 6.1 describes how the workflow deployment on multi-Cloud is performed using the proposed framework. In Section 6.2 the multi-dimensional resource allocation approach is presented. The data-aware scheduling policies are described in Section 6.3. Finally, Section 6.4 presents the implemented data management policy.

### 6.1. Broker-based Multi-Cloud Workflow Deployment

As previously mentioned in Chapter 4, our proposed Cloud service broker framework features the automatic deployment of workflows on multi-Cloud. The needed steps are shown in Figure 6.1. In the first step, the user submits a workflow description to the workflow engine together with his functional and non-functional SLA requirements. After parsing the description, the workflow engine starts a clustering process with the purpose of reducing the number of workflow tasks by applying different workflow clustering techniques (e.g., horizontal and vertical clustering). In the

6. Optimized Multi-Cloud Workflow Deployment

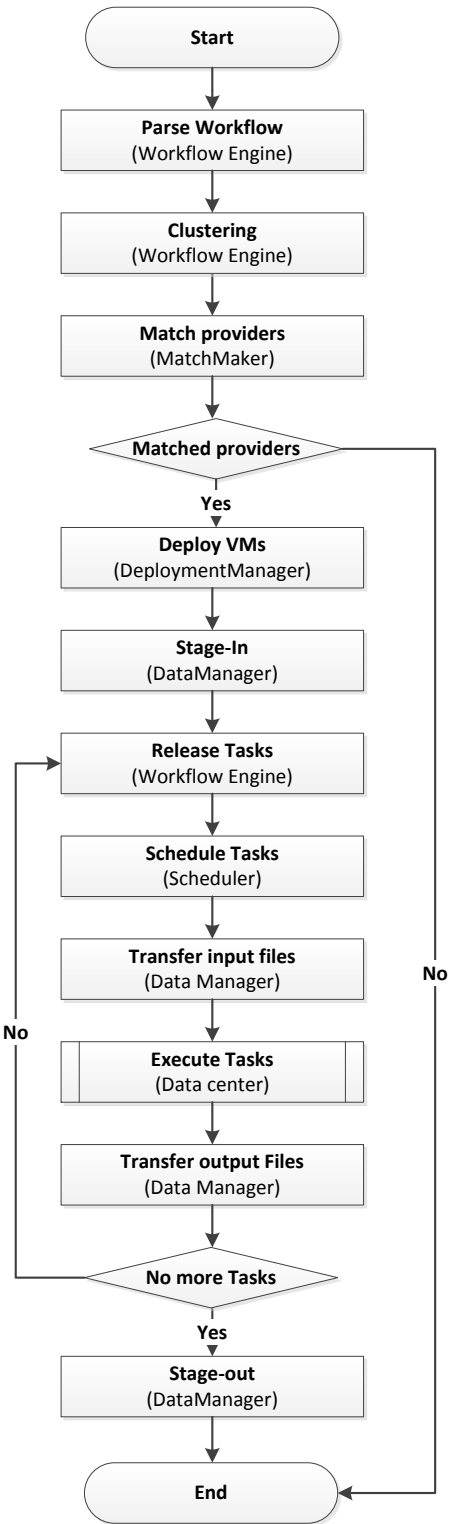


Figure 6.1.: Workflow deployment steps

following, the match maker starts a matching process to select the Clouds that can fit the user-given SLA requirements by applying the matching policy configured in the match maker. After all the requested compute (VMs) and storage resources are deployed on the selected Clouds, the data manager executes a stage-in task to transfer the input data from the client to the Cloud storage. In the next step, the workflow engine starts to continually release the reduced workflow tasks to the broker with a predefined scheduling interval with respect to their execution order. The broker scheduler assigns each received task to a target VM according to the configured scheduling policies. During the workflow execution, the data manager performs the data transfer of the tasks required input and generated output files according to a predefined data management policy. A replica catalog stores the list of data replicas by mapping workflow files to their current datacenter locations. Finally, the execution results are transferred to the Cloud storage in a stage-out task and can be retrieved via the user interface.

Note that the requested VMs and Cloud Storage are provisioned to the user until the workflow execution is finished. However, if the same workflow should be executed many times (e.g. with different input data), a long-term lease period can be specified by the user. In the next sections, the scheduling and data management policies used in the scheduler and data manager are described in detail.

## 6.2. Multi-dimensional Resource Allocation Approach

In order to optimize the deployment of large-scale multi-Cloud applications, such as data-intensive scientific workflows, using the proposed broker framework, we propose a multi-dimensional resource allocation scheme consisting of two stages. In the first phase, the target Cloud services forming the requested composite service are selected using the HU-GA matching algorithm introduced in the previous chapter. In the second phase, the workflow tasks are distributed to the allocated compute resources using a data locality-driven scheduling policy. Unlike previous works [38, 148], using our scheme we are able to optimize four deployment objectives, which are makespan, cost, data locality, and the satisfaction level against the user requested non-functional SLA including response time, availability, latency, and throughput. Furthermore, in our scheme, matching and scheduling are two independent processes, whereas in most previous works the resources selection and task scheduling are performed in a single process. The reasons for decoupling them are fourfold:

- The decoupling allows the implementation of dynamic scheduling heuristics which use dynamic SLA monitoring metrics like latency and bandwidth as decision parameters, while the static high level SLA parameters, such as cost, are considered only in the matching.

## 6. Optimized Multi-Cloud Workflow Deployment

- The non-decoupling increases the allocation problem complexity and causes more time overhead, since more objectives and constraints need to be optimized.
- The decoupling permits a generic matching without need of a detailed description of the application workloads and the internal data movements.
- With the decoupling, it is possible to change the combination of scheduling and matching policies for different application needs.

The functionality of the proposed multi-dimensional resource allocation scheme with four non-functional SLA parameters (cost, latency, throughput, and availability) and three geographical distributed Clouds is presented in Figure 6.2.

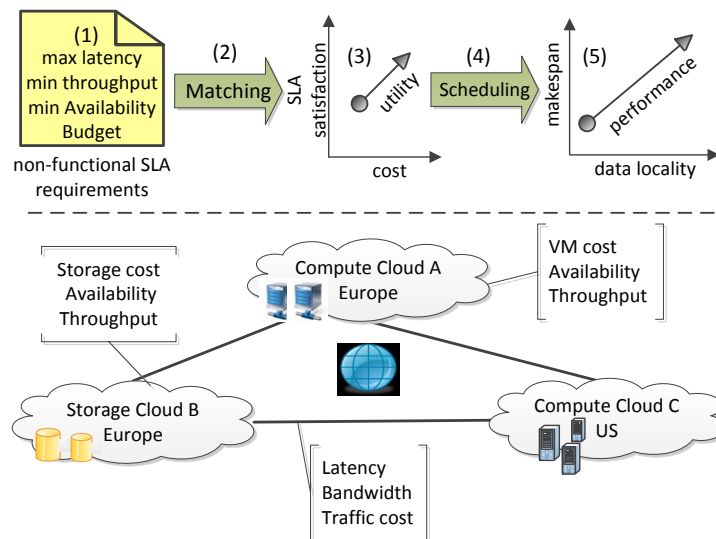


Figure 6.2.: Multi-dimensional resource allocation scheme.

As illustrated in the figure, our multi-dimensional resource allocation is performed in five steps. After that the user enters his SLA requirements and budget (step 1), a matching process is started (step 2), where the functional and non-functional SLA requirements are compared to the measured Cloud SLA metrics as well as their service usage costs for compute, storage, and data traffic. The selection of the optimal Clouds (step 3) is then performed using the HU-GA matching algorithm with the goal to maximize the user utility for the aggregated QoS and total charged cost. In the following step (step 4), the application workloads are distributed at the runtime to the selected compute resources by the broker scheduler. To perform this task, different data locality-driven scheduling policies are used to achieve a minimal data movement and improve the overall application performance (step 5). These are described in detail in the following section.

## 6.3. Data Locality-Driven Scheduling

In this section, we assume that the user requested VMs are already allocated and deployed using the HU-GA matching algorithm according to the first phase of our multi-dimensional scheme and concentrate on the second phase, scheduling. For this purpose, we propose two dynamic, greedy-based scheduling heuristics, which distribute the workflow tasks at the runtime to all the provisioned VMs. These heuristics are used to optimize the allocation of tasks to the deployed VMs at the runtime to reduce the data transfer time and size, which both affect the performance and cost. In the following subsections we describe the functionality of the two implemented scheduling policies.

### 6.3.1. DAS Scheduler

To support data locality for scheduling multi-Cloud workflows, we implemented a Data-Aware Size-based scheduler (DAS) capable of scheduling tasks to the provisioned VMs running in different Cloud datacenters with respect to the location of the required input data. The implemented data-aware scheduling policy is described using the pseudo-code in Algorithm 6.1 and Algorithm 6.2.

```

1  Input: requestedVMList, TaskList, schedulingPolicy
2  For each task T in TaskList Do
3    processTaskAffinity(T)
4    If schedulingPolicy=DAS Then
5      Taff=sizeAffnityMap
6      sort Taff by size in descending order
7    Elseif schedulingPolicy=DAT Then
8      Taff=timeAffnityMap
9      sort Taff by time in ascending order
10   Endelse
11   For each entry in Taff Do
12     site=entry.getKey()
13     For each vm in requestedVMList Do
14       If vm.getStatus()==idle
15         & vm.getDatacenter()==site Then
16           schedule T to vm
17           scheduledTaskList.add(T)
18           Break
19       Endif
20     Endfor
21   Endfor
22 Endfor
23 Output: scheduledTaskList

```

Algorithm 6.1: DAS/DAT Scheduler

## 6. Optimized Multi-Cloud Workflow Deployment

```

1  Input: matchedDatacenterList, Replica catalog R
2  For each datacenter D in matchedDatacenterList Do
3    time=0; size=0;
4    inputFileList = T.getInputFileList()
5    For each file F in inputFileList Do
6      maxBwth=0
7      siteList = R.get(F)
8      If siteList.contains(D) Then
9        size=size+F.getSize()
10     Else
11       For each site in siteList Do
12         bwth=Cloud-to-Cloud-bandwidth(site,D)
13         If bwth>maxBwth Then
14           maxBwth=bwth
15         Endif
16       Endfor
17     Endelse
18     time=time+F.getSize()/maxBwth
19   Endfor
20   sizeAffnityMap.add(D, size)
21   timeAffnityMap.add(D, time)
22 Endfor
23 Output sizeAffnityMap, timeAffnityMap

```

Algorithm 6.2: Function processTaskAffinity

As presented in Algorithm 6.1, the algorithm first iterates through the workflow tasks (line 2) and calculates for each task the total size of the required input files found in each matched datacenter (function *processTaskAffinity()* in Algorithm 6.2) and stores the result in a map data structure called task data size affinity  $T_{sizeaff}$  (line 5). If we assume that task T has m required input files  $freq_i, i \in \{1, \dots, m\}$  and there are k matched datacenters, the task affinity of the datacenter  $dc_j, j \in \{1, \dots, k\}$  is calculated using the following equation:

$$T_{sizeaff}(dc_j) = \sum_{freq_i \in dc_j} size(freq_i) \quad (6.1)$$

After sorting the task affinity map by the data size values in descending order (line 6), the policy assigns the task to the first free provisioned VM running on the datacenter  $dc_{cand}$  containing the maximum size of located input files (lines 11-22), where:

$$T_{sizeaff}(dc_{cand}) = \max_{j=1}^k T_{sizeaff}(dc_j) \quad (6.2)$$

According to the pseudo-code, we assume that a VM can execute only one task at the same time. In case all the provisioned VMs in the selected datacenter are busy, the algorithm tries the next candidate datacenters in the sorted map to find a free VM so that a load balancing between the datacenters is assured and an unnecessary waiting time for free VMs can be avoided.

### 6.3.2. DAT Scheduler

The second proposed scheduling policy, called Data-Aware Time-based (DAT) scheduler, has a strong similarity with the previously described DAS policy except in the method of calculating the task affinity. Instead of calculating the maximum size of existing input files per datacenter, the algorithm computes the time needed to transfer the missing input files to each datacenter and stores the transfer time values in a map data structure called  $T_{timeaff}$  (line 8 in Algorithm 6.1), which is calculated using the following equation:

$$T_{timeaff}(dc_j) = \sum_{freq_i \notin dc_j} transfertime(freq_i) \quad (6.3)$$

As target datacenter, the algorithm chooses the datacenter, which assures the minimum transfer time by sorting the affinity map in the ascending order (line 9).

$$T_{timeaff}(dc_{cand}) = \min_{j=1}^k T_{timeaff}(dc_j) \quad (6.4)$$

As presented in Algorithm 6.2 describing the function *processaffinity()* pseudo-code, the algorithm iterates through the matched datacenters and checks the existence of local input files for each workflow task (lines 2-9). For each missing input file, it calculates the time needed to transfer the file from a remote location to that datacenter. In order to achieve this, it fetches the replica catalog and selects a source location, which assures the maximum bandwidth and consequently the minimal transfer time to that datacenter (lines 10-19). The calculation method of the transfer time is the same used by our proposed data management policy, which is detailed in the next section.

## 6.4. Replica-based Data Management

As previously mentioned in Section 6.1, the data manager is responsible for the data stage-in/out as well as for the Intercloud data transfer before and after each task execution. In this thesis we propose a replica-based data management policy to perform the data manager tasks at the runtime. For the sake of simplicity, we assume that all the VMs deployed on the same datacenter share a local Storage Area Network (SAN) storage to store the local generated tasks output data. In addition, we assume that, at the beginning of the workflow execution, all the input files are located in the client local storage. All the data transfers between client and Clouds and between the Clouds are performed through the Internet with different throughput and bandwidth values. Algorithm 6.3 presents the pseudo-code for the implemented data transfer policy.

## 6. Optimized Multi-Cloud Workflow Deployment

```
1 Input: scheduledTaskList, replica catalog R
2 For each task T in scheduledTaskList Do
3   D=T.getVmDatacenter()
4   FileList=T.getFileList()
5   For each file F in FileList Do
6     If T.getType()=stage-in & F.getType()=input Then
7       transfer F from Client to Cloud storage
8       siteList=null
9       siteList.add(Cloud storage)
10      R.put(F, sitelist) // register file in replica catalog
11    Elseif T.getType()=stage-out & F.getType()=output Then
12      transfer F from D to Cloud storage
13    Elseif F.getType()=input Then
14      If R.get(F)=null Then
15        exit // input file missing
16      Else
17        siteList=R.get(F)
18        maxBwth=0
19        If siteList.contains(D) Then
20          transfer F from local SAN of D
21        Else
22          For each site in siteList Do
23            bwth=Cloud-to-Cloud-bandwidth(site ,D)
24            If bwth>maxBwth Then
25              maxBwth=bwth
26              sourceSite=site
27            Endif
28          Endfor
29        Else
30          transfer F from sourceSite to D // intercloud transfer with maxBwth
31          siteList.add(D)
32          R.put(F,siteList) // register input file in replica catalog
33        Endelse
34      Elseif F.getType()=output Then
35        transfer F to local SAN of D
36        siteList=R.get(F)
37        siteList.add(D)
38        R.put(F, sitelist) // register output file in replica catalog
39      Endelse
40    Endfor
41  Endfor
```

Algorithm 6.3: Data Management Policy

As can be seen from Algorithm 6.3, the algorithm iterates first through all the scheduled tasks (lines 2-5) and checks their type (stage-in/out or regular compute tasks). For stage-in tasks (lines 6-10), the data manager transfers the input data from the client to the requested Cloud storage and registers them in the replica catalog, whereas for stage-out tasks the execution results are transferred from the datacenter on which the last task ran to the Cloud storage (lines 11-12), so that the client will be able to retrieve them. Clearly, it is also possible to not use Cloud storage and therefore to transfer the input data directly from the client as needed. However, this scenario provides no advantage for data movement and, consequently, workflow performance. Moreover, for the data transfer, the client should remain connected during the workflow execution.



For the transfer of input files required to execute each compute task, the data manager iterates through the scheduled tasks and checks the existence of any local input files in the local SAN of the datacenters where their respective assigned VMs are running (line 13). In case the files are also not found in the replica catalog, the execution of the workflow will be aborted and the client will then be notified (lines 14-15). If the files are found in the local SAN, the data manager transfers them to the corresponding VM, otherwise it fetches the replica catalog and transfers the files remotely from the datacenters, which assure the maximum transfer bandwidth and consequently the minimal transfer time to the datacenters where the tasks should run (lines 16-33). At the end of each transfer, the datacenter location of the transferred files is updated in the replica catalog.

After the execution of each task, the data manager automatically initiates the transfer of its generated output files and registers them to the replica catalog. The basic strategy is to store the generated output files in the local SAN of the datacenter where the task is executed, so that all the VMs deployed on that datacenter can locally access the stored files and benefit from a free transfer (lines 34-39).

## 6.5. Summary

This chapter addressed the optimal deployment of workflows using our multi-Cloud service broker framework proposed in Chapter 4. After describing the different needed steps for the deployment process, a two-stage multi-dimensional resource allocation approach for running data-intensive workflow applications has been presented. In the first phase, the scheme applies our previously introduced HU-GA algorithms to select the suitable Clouds for users with respect to their SLA requirements and payment willingness. In the second phase, a data locality-driven scheduler brings the computation to its data during the workflow execution to reduce the Intercloud data transfers. For this purpose, two data driven greedy-based scheduler called DAT and DAS are proposed. Finally, the chapter presented the data management policy implemented in the broker's data manager to manage data replicas and the tasks input/output data transfer at runtime.



# 7. Evaluation

In this chapter, we present simulation-based evaluations of the contributions, that address the problems discussed in Chapter 1. In Section 7.1, we present the simulation testbed we used to conduct our simulations experiments using the simulation environment introduced in Chapter 4. In Section 7.2 we evaluate and validate the multi-Cloud service broker framework presented in Chapter 4. In particular, we evaluate the scalability of the match maker and workflow engine components, give comments on simulation results, and present the lessons learned that were the main motivations for the next experiments. In Section 7.3, we evaluate the utility-based matchmaking algorithm introduced in Chapter 5. In particular, we present several simulation scenarios with simple and composite services and discuss the benefits of our approach compared to the sieving matching. Then, we compare its efficiency with a prospect-based selection algorithm through a real SaaS case study. In Section 7.4, we evaluate our multi-dimensional resource allocation scheme introduced in Chapter 6 with real workflow applications and discuss the cost and performance benefits that this approach brings. Finally, in Section 7.5, we summarize the evaluations of all contributions and make final conclusions about their benefits with respect to the research questions addressed in this thesis. The results of this chapter are largely based on [88, 91, 86, 87].

## 7.1. Experimental Setup

In this section we present the configuration of the modeled IaaS Clouds and their provided services that we used to build a realistic simulation testbed as basis for our evaluation.

### 7.1.1. Modeled Public IaaS Clouds

In order to validate our brokering concept on top of heterogeneous Cloud platforms with different hardware configuration and service properties, we modeled 20 Cloud datacenters located in four world regions (Europe, USA, Asia and Australia). Each compute Cloud is made up of 50 physical hosts, which are equally divided between two different host types with 8 and 16 CPU cores, respectively. This gives a total

## 7. Evaluation

computing capacity of 600 cores and 600 GB RAM. The detailed datacenter and hosts configuration is provided in Table 7.1.

Table 7.1.: Modeled hosts and datacenter setup.

	Parameter	Value or Range
Host	CPU cores per host	8..16
	host CPU speed	1860..2660 MHZ
	host RAM size	8..16 GB
	host local storage	1 TB
Network	local datacenter: Cloud local bandwidth	100 Mbit/s
	Cloud local latency	10 ms
	Intra-continental: Cloud-to-Cloud bandwidth	30 Mbit/s
	Cloud-to-Cloud latency	25 ms
	Inter-continental: Cloud-to-Cloud bandwidth	10 Mbit/s
	Cloud-to-Cloud latency	150 ms
Datacenter	number of datacenters	20
	hosts per datacenter	50
	regions	Europe, USA, Asia and Australia

In order to make our simulation more realistic, we collected the current pay-as-you-go prices for computation, storage, and network traffic of 10 popular public IaaS providers, namely: Amazon AWS, ElasticHosts [44], GoGrid [67], Rackspace [133], CloudSigma [28], OpSource [123], CityCloud [25], VoxCloud [155], HP Cloud [77], and Flexiscale [55]. We then mapped the collected pricing policies to the modeled datacenters based on each datacenter’s regional location, operating system, and VM configurations. Please note that a Cloud provider may have more than one datacenter in different regions, but with different pricing policies. In our evaluation, we name the datacenters belonging to the same Cloud by adding the region suffix; however, we treat them as separated Clouds. Additionally, we acquired for the same datacenters the average Cloud SLA metrics values of the last three months for availability, client-to-Cloud throughput, and response time (for downloading large files from the Cloud) through CloudHarmony network tests [27] from the same client host. The collected data for each datacenter can be seen in Appendix A.1. Since this data is collected from a host located in Germany, in all our conducted experiments we assume that the Cloud users are located in Germany.

To model the network between the datacenters, we defined, based on their regional location, three constant bandwidth and latency values (local datacenter, intra-continental and inter-continental, see Table 7.1). The use of these synthetic values is justified

by the lack of free accessible Cloud-to-Cloud network metrics from CloudHarmony. Since these two metrics are of high importance for the accuracy of the matching and scheduling policies, we plan to extend the simulation framework to gather the latest SLA metrics values from third-party network monitoring services. However, the use of constant network metrics values in the simulation, similar to [159], allows us to compare different policies under the same SLA constraints.

### 7.1.2. Modeled IaaS Cloud Services

We studied the current compute services offered by the modeled 10 public Clouds and then categorized their typical offers into nine modeled VM types. Table 7.2 lists the number of equipped processing cores, the memory and local disk size of each VM type. Additionally, each VM type is assigned with a Compute Unit (CU), the unit for payment, whose value is given based on the VM's configuration, i.e., the number of cores and the size of storage. One compute unit corresponds to the price of a micro VM instance (XS). In the evaluation, we use the payment units to calculate the user budget for different VM types. For example, when the maximum acceptable price for a micro instance is given, the budget for a small VM instance is double the accepted price for a micro instance.

Since some Clouds do not support all the nine VM types (e.g. Amazon EC2 offers only the VM types XS, S.2, M.2, L.2 and XL.2), we configured the modeled datacenters to provision only the VM types supported by the respective real public Cloud. Moreover, we checked if the Clouds offer pure storage services in addition to compute services. We found that of the 20 datacenters, only 12 also provide storage services. This result is used to decide if a compute or a storage service can be deployed on a datacenter when matching the functional service requirements.

The collected pricing policies and SLA metrics data of the modeled Clouds, as well as their offered service types, are imported from separate text files at the beginning of the simulation and then stored in the broker service and provider registries, according to the provider ontology presented in Chapter 4. The information about the Clouds characteristics can be accessed during the simulation to support the broker's decision-making.

### 7.1.3. CloudSim Setup

All of our simulation experiments conducted using the implemented simulation framework are done with CloudSim version 3.0 on a notebook with CPU Intel Core i5 560M 2.67 GHZ, RAM 4 GB and using Windows 7 operating system. The default CloudSim simple VM provisioning policy is used as an internal datacenter scheduling policy. This policy allocates VMs to the host with most free cores. In order to permit the

## 7. Evaluation

Table 7.2.: Types of the modeled virtual machines (XS: micro, S: small, M: medium, L: large, XL: Xlarge); 1 CPU Core: 1 GHZ Xeon 2007 Processor of 1000 MIPS; ARCH: 64/32bits (L/XL only 64bits); OS: Linux/Windows; Bandwidth: 100 Mbit/s.

VM Type	XS	S	S.2	M	M.2	L	L.2	XL	XL.2
CPU Cores	1	1	1	2	2	4	4	8	8
RAM (GB)	0.5	1	1.7	2	3.75	4	7.5	8	15
DISK (GB)	25	50	75	100	150	200	250	300	350
Compute Units	1	2	3	4	6	8	12	16	20

dynamic sharing of CPU cores among VMs, we configured CloudSim to use a time-shared VM scheduler policy. To collect the simulation results, we repeated each of the experiments ten times from the same host and then computed the average value.

### 7.1.4. Workflow Traces

For all of our experiments conducted with multi-Cloud workflows we use two real XML formatted traces of the Montage and Epigenomics workflow-based applications. The traces are generated from real executions of the corresponding workflows using the Pegasus WfMS. The task descriptions, including runtime and input/output files information of each workflow, have been imported with the help of the WorkflowSim workflow parser from separate text files. We configured the workflow engine to release a maximum of five tasks to the broker in each scheduling interval (default value used in Pegasus). For an accurate calculation of the makespan, we extracted from each workflow trace the real delay overhead that results from clustering, post-scripting, and queuing to make the simulation as realistic as possible. In the following, we describe the workflow applications and trace characteristics.

#### Montage Workflow Application

Montage [14] is an astronomical workflow application used to construct large image mosaics of the sky obtained from the 2MASS observatory at IPAC [1]. A Montage workflow can be modeled as directed acyclic graph (DAG), where the vertices represent computing tasks, and the edges represent either data-flow or control-flow dependencies. A sample DAG graph of a 9-level Montage workflow is illustrated in Figure 7.1. All the tasks at the same horizontal level are invocations of the same binary code (see right side of the figure) operating on different input data. The imported sample trace contains 7463 tasks within 11 horizontal levels, has 3 GB of input data and generates respectively about 31 GB and 84 GB of intermediate output data and data traffic. Each task requires one CPU core to run. As Montage spends more than 80%

of the execution time in data transfer operations, it is considered as a data-intensive workflow.

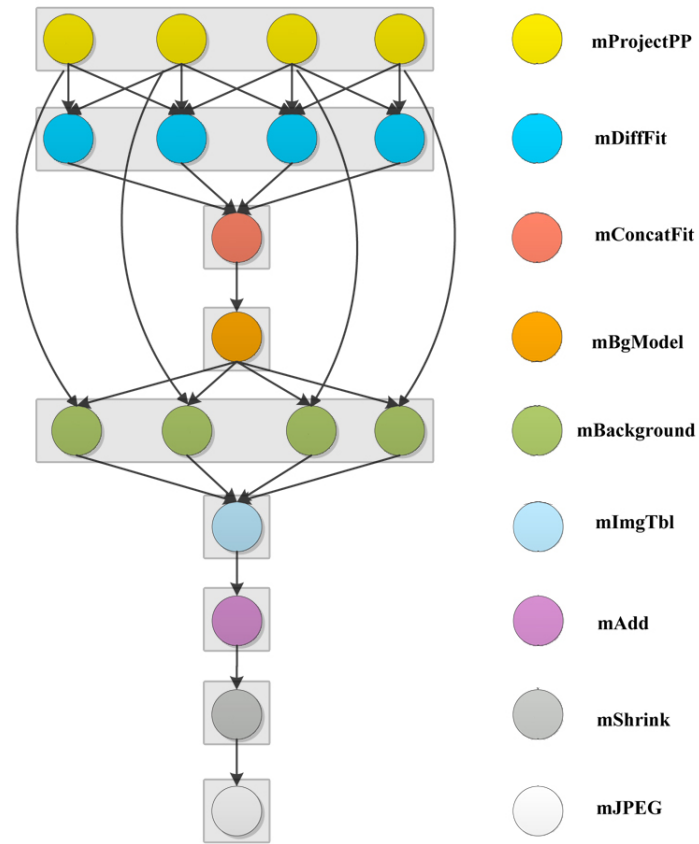


Figure 7.1.: A sample horizontally clustered Montage workflow.

As mentioned before, we use clustering to reduce the scheduling overhead of large-scale workflows on multi-Cloud. A well-suited clustering technique for the Montage workflow structure is horizontal clustering. Herewith tasks at the same horizontal level are merged into a predefined number  $k$  of clustered jobs. Table 7.3 shows the resulted total number of clustered jobs for each used  $k$ .

Table 7.3.: Total number of clustered jobs with different cluster numbers  $k$ .

Cluster number $k$ per level	20	40	60	80	100
Jobs number	92	152	212	272	332

## Epigenomics Workflow Application

Epigenomics<sup>1</sup> is a compute-intensive, workflow-based DNA sequencing application, developed by the USC Epigenome Center [46], to map the epigenetic state of human cells on a genome-wide scale. A sample directed acyclic graph (DAG) presentation of a four vertical levels Epigenomics workflow is illustrated in Figure 7.2.

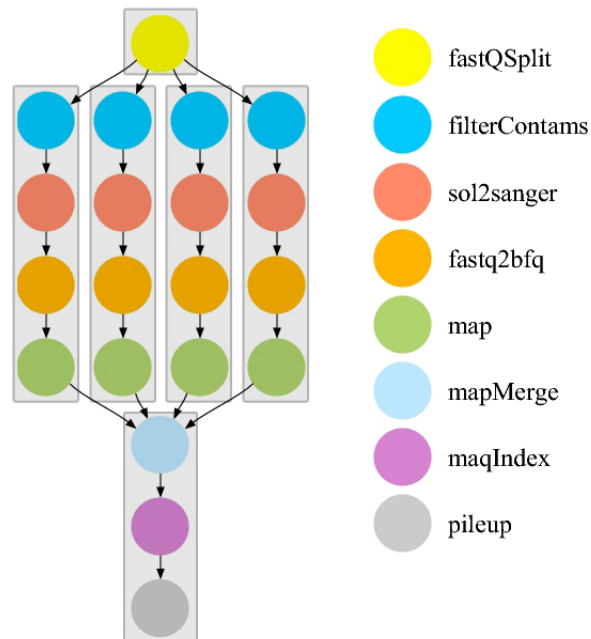


Figure 7.2.: A sample vertically clustered Epigenomics workflow [96].

As can be seen from the figure, the workflow takes the DNA sequence data generated by a genetic analyzer system as input and splits it into several chunks that can be processed parallel to one another. The sequences of each data chunk are filtered to remove noisy and contaminating sequences, and then mapped onto the correct location in a reference genome. Finally, a global map of the aligned sequences is generated and the sequence density of each position in the genome is calculated [96]. Table 7.4 shows the characteristics of the imported workflow trace consisting of 997 tasks. In order to reduce the scheduling overhead, we applied vertical clustering as merging techniques. Herewith, the sequential tasks of each vertical level are merged into a clustered job, so that the total number of tasks is reduced to 260 tasks.

<sup>1</sup>[http://pegasus.isi.edu/applications/dna\\_sequencing](http://pegasus.isi.edu/applications/dna_sequencing)



Table 7.4.: Epigenomics workflow trace characteristics.

Tasks number	Clustered jobs	Input data	Output data	Data traffic
997	260	7 GB	300 MB	505 GB

## 7.2. Evaluation of the Multi-Cloud Service Broker Framework

### 7.2.1. Match Maker and Deployment Manager Evaluation

The aim of this first simulation experiment is to test the functionality of the match maker and deployment manager components of the multi-Cloud service broker framework, with simple generated compute service requests. For this purpose, we set up a simple simulation scenario with fewer datacenters and less offered services than the ones presented in Section 7.1. In the following we describe the simulation scenario and then we present and discuss the simulation results.

#### Simulation Scenario

To conduct this experiment, we configured six heterogeneous CloudSim datacenters. Each datacenter has a unique ID and is located in a different geographical region (on four world continents). The detailed configuration for each datacenter is gathered in Table 7.5. The six datacenters have different pricing policies and can support one of two defined operating systems (Linux or Windows) and CPU architectures (x86 or x64). For this experiment, we assume that the pricing policies of each datacenter are fixed and, therefore, all the offered VM instance types are charged with the same price. Furthermore, we do not consider the cost for data traffic. Each of the modeled datacenters is made up of 50 hosts, which are equally divided between two different host types. The configuration of each host type can be seen in Table 7.6. The chosen hosts setup allows the deployment of one or more VM instances per host.

As a simple simulation scenario, we model in this experiment the broker-assisted deployment of a single VM on one Cloud. Hence, we configured the *Request Generator* to continuously generate (at a random rate varying from 0 to 60 seconds) an equal number from four synthetic user request types. Each request requires one of the four modeled VM types, which are the *micro*, *small*, *large*, and *high CPU* Amazon EC2 instance types. Based on the requested VM type, the requests have different cost limits, which are equal to the prices charged by Amazon for the corresponding type. In addition, each request prefers a different compute zone for the deployment. Table 7.7 gives the hardware configuration, requested region, and maximal acceptable cost of each request type. As can be seen from the table, the VM configurations are similar

## 7. Evaluation

Table 7.5.: Modeled datacenters configuration.

Datacenter Configuration					
Name	ID	OS	Arch	Region	Cost \$/hour
DC_A	0	Linux	x64	USA	0.3
DC_B	1000	Linux	x64	USA	0.45
DC_C	2000	Windows	x64	Europe	0.75
DC_D	3000	Linux	x64	Europe	0.55
DC_E	4000	Linux	x64	Asia	0.15
DC_F	5000	Windows	x86	Australia	0.04

Table 7.6.: Hosts setup.

Host Type	CPU (MHZ)	Cores	RAM (GB)	Bandwidth (Gbit/s)	Storage (TB)
Xeon 3040	1860	2	4	1	2
Xeon 3075	2660	2	8	1	1

to the modeled VM types *XS*, *S.2*, *L.2* and *XL.2*, presented in Table 7.2. All the configured datacenters support the provision of the four VM types when their load capacity allows.

Table 7.7.: Modeled synthetic VM request types; OS=Linux and Arch=x64.

VM Requirements						
Request ID	VM type	CPU (GHZ)	Cores	RAM (GB)	Region	Cost (\$/hour)
1	CPU high	2.5	2	1.7	USA	0.17
2	large	2	2	7.5	Europe	0.34
3	small	1	1	1.7	Asia	0.085
4	micro	0.5	1	0.63	Africa	0.02

### Deployment Rate

To test the functionality of the deployment manager, we conducted an initial experiment in which the broker randomly selects for each generated request a provider from the six datacenters, regardless of user requirement. The broker then tries to deploy the VM on the selected datacenter with respect to its current load capacity. While keeping the number of datacenters constant, we measured the deployment rate, which is defined as the percentage of successfully deployed VMs, by varying the request number from 50 to 2000. We repeated the same experiment by decreasing the number of datacenters from six to three and then to only one. The results presented in Figure

7.3, after one day of simulation time, show that the broker deployment rate scales well with the increasing number of service requests and Cloud providers.

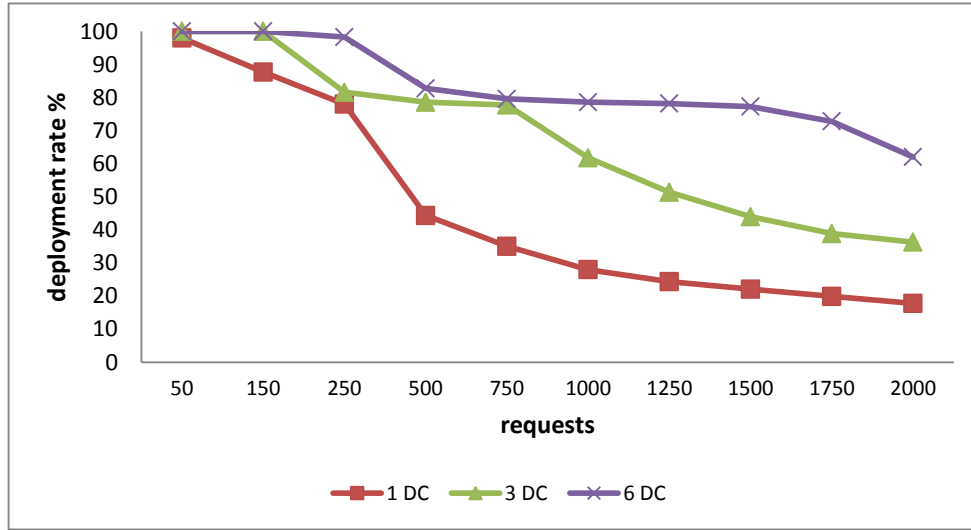


Figure 7.3.: One day broker deployment rate with different datacenter numbers and the random matching policy.

## Matching Rate

We conducted another experiment to evaluate the matching performance with different datacenter selection strategies in the broker. For this purpose, we implemented the following simple matchmaking policies in the match maker:

- **Functional SLA Cloud Matcher:** selects randomly a provider that fits all the functional SLA service requirements.
- **Location-Aware Cloud Matcher:** selects a provider located at the same region given in the service request.
- **Cost-Aware Cloud Matcher:** selects the cheapest provider below a given cost limit.
- **Hybrid Cloud Matcher:** combines both functional SLA and location-aware matching.

We repeated the previous experiment using all six datacenters and changed the matching policy used in the match maker each time. We then measured the matching rate, defined as the percentage of successfully matched requests. As depicted in Figure 7.4, when using cost or location as the matching policy, the matching rate remains constant at 75%, because the requested cost limit and location for the *micro* VM instance

## 7. Evaluation

type usually has no match. However, with the functional SLA and hybrid matching policies, the matching rate decreases continuously with the rising service demand due to the limited capacity of the provided datacenter resources. In addition, the results show that a simultaneous fulfillment of all SLA requirements always happens at the expense of a low matching rate. For example, for hybrid matching, the maximum measured matching rate was the half of the matching rate obtained with a functional SLA matching, because in this case the datacenter location also needs to be taken into account.

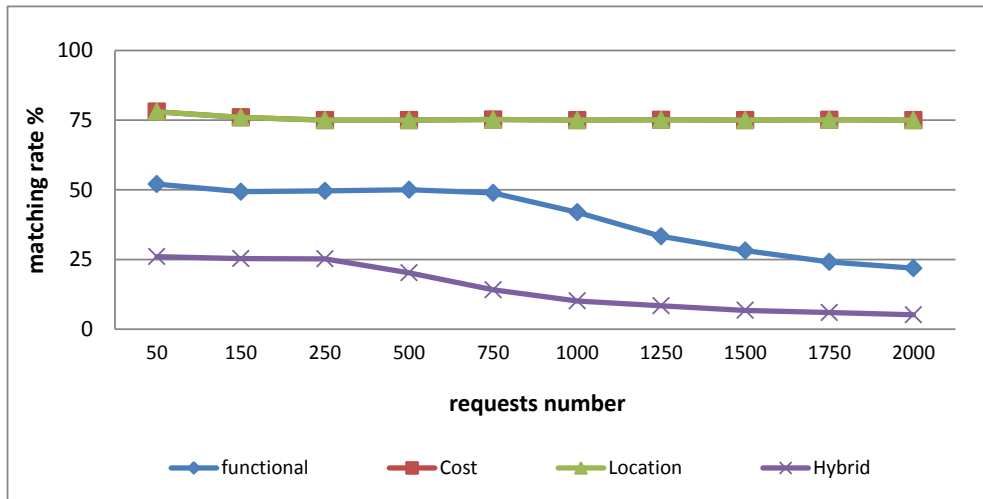


Figure 7.4.: One day matching rate for different matching policies with 6 datacenters.

### 7.2.2. Workflow Engine Evaluation

The goal of this experiment is to show the benefits of using the multi-Cloud service broker framework in executing scientific workflows compared to the use of a single Cloud. In particular, we focus on the performance and cost benefits and we identify the necessity of using clustering. We conducted this experiment using the Montage workflow trace described above in Section 7.1.4 and the modeled public Clouds in Section 7.1.1. In the following, we describe first the simulation scenario and then show and discuss the gathered results.

#### Simulation Scenario

To conduct this experiment, the following simulation scenario has been used: A user located in **Germany** requests 10 VMs of the type *small S.2* and 10 VMs of the type

*medium M.2* to run the Montage workflow using our framework. The user’s non-functional SLA requirements, given in Table 7.8, express the maximum payment willingness of the user, as well as his required values of the SLA attributes (availability, response time, and throughput), to deploy the workflow with an acceptable QoS. We configured the match maker component to use the **sieving** matching policy introduced in Chapter 5 so that only the datacenters that satisfy the entire user predefined functional and non-functional requirements are selected. The latency between the Clouds is not considered in the SLA requirements because it has no effect when using the sieving matching policy. The broker scheduler is configured to use the simple **round robin** scheduling policy. This policy allocates tasks to the first free VMs regardless of their type or datacenter location. We assume that all the VMs running in the same datacenter share a local SAN storage. The Intercloud data transfer is performed according to the data management policy presented in Chapter 6. In this experiment we do not use a dedicated Cloud storage to store the workflow input/output data. Instead, this data is transferred directly from the client when needed at runtime. In addition, we consider only the cost for provisioning the 20 VMs and do not consider the cost for data traffic.

Table 7.8.: User non-functional SLA Requirements for the Montage workflow deployment.

max Willingness (\$/hour)	min Availability (%)	max Response time (s)	min Throughput (Mbit/s)
2.7	96	10	10

For the purpose of evaluation, we modeled two simulation scenarios. In the first scenario, named “**single Cloud**”, we deploy all the workflow tasks on the EC2 EU Cloud. In the second scenario, named “**multi-Cloud**”, we use all the modeled 20 datacenters from Section 7.1.1 and let the multi-Cloud service broker automatically select the suitable datacenters to provision the requested VMs according to the used matching policy. These VMs are then provisioned to the user until the end of the workflow execution.

### Impact of Clustering on Workflow Makespan

In order to assess the scalability of the proposed brokering framework with respect to increasing cluster number  $k$ , we measured in the first experiment the total time needed to execute a single run of the sample Montage workflow in minutes for both the “single Cloud” and the “multi-Cloud” scenario. In addition, for the single Cloud scenario, we simulated the case of adding a stage-in job to transfer all required input files before executing the workflow tasks. Figure 7.5 illustrates the results achieved.

## 7. Evaluation

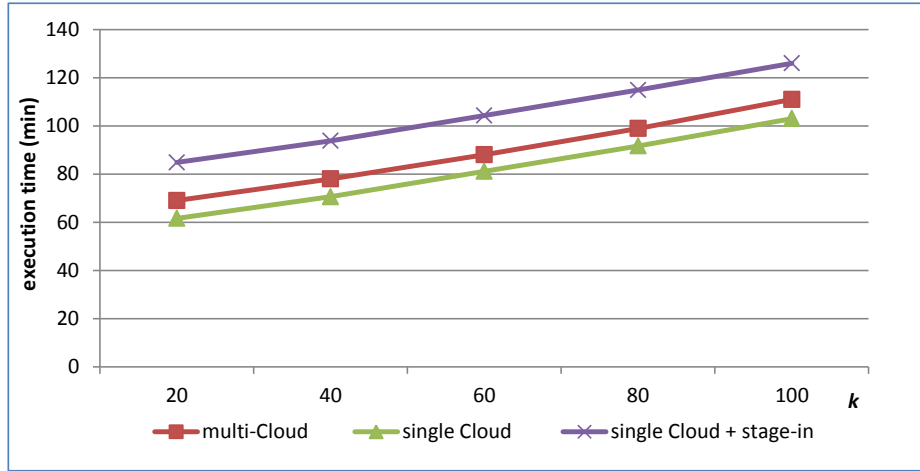


Figure 7.5.: Workflow execution time with different cluster numbers  $k$ .

As depicted in the figure, a 20 stepwise increase of  $k$  results for all the simulated scenarios in an increase of about 10 minutes in the workflow makespan. This demonstrates that our framework scales well with the increasing number of workflow jobs. Clearly, the single Cloud case benefits from a shorter execution time because, for the “multi-Cloud” case, the Intercloud data transfer is more time-consuming than a transfer from the local SAN storage. However, when adding a stage-in job to the single Cloud case, the execution of the workflow needs more time, as all the other tasks should wait for the stage-in job to finish before their execution starts.

### Impact of Clustering on Data Transfer

To evaluate the resulting data transfer overhead in the multi-Cloud scenario, we repeated the previous experiment to measure the proportion of time needed to transfer all the required files compared to the total pure computing time (includes clustering delay) for all workflow tasks. The results with different cluster numbers are shown in Figure 7.6. Note that for the Intercloud transfer time calculation, we used the predefined local and Cloud-to-Cloud bandwidth constants, as we do not know the real bandwidth values. All the data transfers are performed by the data manager according to the transfer policy introduced in Chapter 6. It can be seen from the figure that the total pure computing time decreases with larger  $k$ . This decrease is caused by the logical decrease of the clustering delay, since, with a bigger cluster number, the number of merged tasks in a clustered job is reduced. However, the data transfer time remains between 180 and 190 minutes for a  $k$  between 20 and 100. To show the benefit of using clustering, we repeated the same experiment by disabling the clustering. We found that about double the time is spent for the data transfer compared to a deployment with clustering enabled.

## 7.2. Evaluation of the Multi-Cloud Service Broker Framework

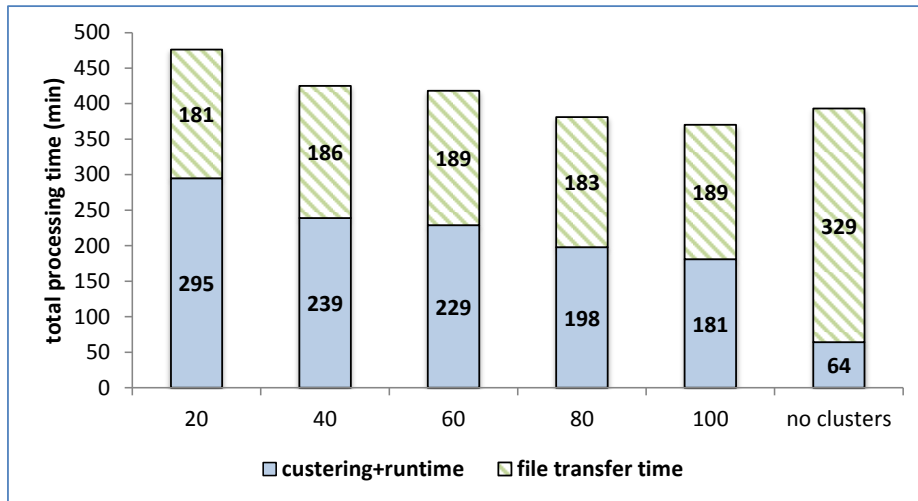


Figure 7.6.: File transfer overhead for the multi-Cloud case with different cluster numbers  $k$ .

In order to classify the types and origins of the transferred files, we counted the amount of VM-to-SAN-Storage (local), Cloud-to-Cloud (Intercloud) and client-to-Cloud transfers during the workflow execution. The total number and size of the transferred files during a deployment with  $k=20$  are depicted in Figure 7.7. The results prove that the transfer time overhead is heavily affected by the Intercloud transfers. Therefore, a reduction of the number of transfers between the Clouds will probably improve the workflow execution performance. This can be achieved by using more efficient brokering and scheduling policies in the broker than the ones used in this experiment (sieving as matching policy and round robin as scheduler).

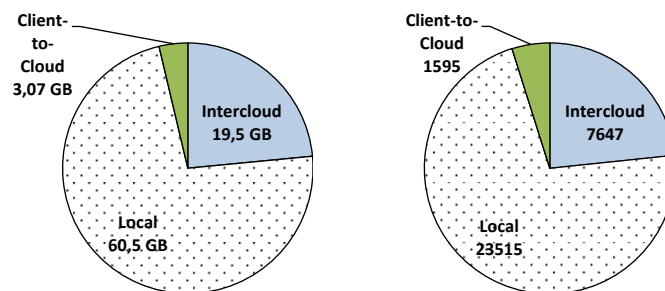


Figure 7.7.: Total size (left) and number (right) of transferred files for the multi-Cloud case with  $k=20$ .

## 7. Evaluation

### QoS and Cost Evaluation

We compared the average non-functional SLA values (cost, availability, response time, and throughput) of the matched datacenters for the multi-Cloud case with the corresponding values in the case of deploying the workflow on the single Amazon EC2 EU Cloud. The results are depicted in Table 7.9.

Table 7.9.: Average non-functional SLA values for the matched datacenters.

Scenario	Datacenters	Cost (\$/h)	Availability (%)	Response time (s)	Throughput (Mbit/s)
single	EC2 EU	2.55	99.97	3.63	19.11
Multi	EC2 EU FlexiScale EU CityCloud EU	2.27	99.86	3.91	21.66

It can be seen that, besides Amazon EC2 EU, FlexiScale EU and CityCloud EU also fulfill all the SLA user requirements listed in Table 7.8. The other Clouds fail either because their expected execution costs are above the maximum user's payment willingness or because they cannot deliver the minimum required client-to-Cloud throughput due to their non-closeness to the client (meaning they are not located in the same geographical region as the client). The table also shows that the multi-Cloud case works better than the single Cloud case in terms of cost saving. This lies in the fact that FlexiScale and CityCloud deliver the requested VM types with lower costs and with almost the same or better service quality than Amazon EC2.

### 7.2.3. Results Discussion

The first experiment conducted in Section 7.2.1 showed that an increase of the number of concurrent providers results in more resource heterogeneity and therefore improves the broker's matching rate. The results also prove the good scalability of the broker and, in particular, of the match maker. Furthermore, the results prove that the accuracy of the queried monitoring information by the monitoring manager (e.g. datacenter load capacity) heavily impacts the performance of the matching policy, especially for the functional SLA matching. In fact, the support of more than one SLA parameter in the matching increases the customer satisfaction, but at the cost of a low match rate. Thus, the matching algorithm should optimize this trade-off by modeling the dependency between the customer utility function and his requested functional and non-functional SLA parameters, while considering the current provider's monitoring information. The final experiment results justify the need for efficient matching policies, like our proposed utility-based matching algorithm in Chapter 5, to increase the match rate while satisfying the user SLA requirements.



The second experiment presented in Section 7.2.2 showed that one Cloud may execute a workflow application faster than using multiple Clouds because the inter-task data transfer is performed within a single platform. However, when the input data of the workflow task must be transferred from other locations, e.g. other Clouds, to the target platform, the multi-Cloud workflow framework may work better in terms of execution time. Hence, the use of Cloud storage for storing input and output data has more advantages in the multi-Cloud case. We also found that clustering is a necessary technique for running workflows on multiple Clouds because it reduces the number of tasks and hence reduces the amount of data to be transferred between the tasks. Finally, the results prove the benefit from using the broker framework to deploy workflows on multi-Cloud in reducing the user payment by selecting Clouds with lower cost and better service quality. In the next sections we will evaluate the use of more efficient matching and scheduling policies for the workflow execution to improve the performance and costs and consequently increase the user satisfaction.

## 7.3. Evaluation of the Matching Algorithms

In this section we evaluate the efficiency of the SLA-based matching algorithms introduced in Chapter 5 with simple and complex composite service requests. Therefore, we implemented the sieving and utility-based matching algorithms as a new match-making policy in the broker and conducted experiments with different scenarios. In the first subsection we apply the simple version of the utility-based algorithm, implemented based on an exhaustive search to match single Cloud services. In the next subsection we use its genetic-algorithm based version (HU-GA) to match a composite workflow service. Finally, we compare the HU-GA algorithm with a prospect-based selection algorithm in a real SaaS case study.

### 7.3.1. Matching of Single Cloud Services

#### Simulation Scenario

In this experiment, we investigate the efficiency of the proposed sieving and utility-based algorithms in matching single Cloud services using a *Desktop Cloud Service (DCS)* use case. This is an application of the virtualization technology (e.g., IBM PowerVM [80], VMware<sup>2</sup>, Xen<sup>3</sup>) to desktop computing. In the DCS computing model, users connect to single virtual machines running desktop operating systems on servers

---

<sup>2</sup><http://www.vmware.com>

<sup>3</sup><http://www.xensource.com>

## 7. Evaluation

provisioned in a remote datacenter. Users interact with their desktops through remote access protocols (e.g., RDP<sup>4</sup>, ICA<sup>5</sup>) using thin-client devices that provide Graphic User Interface (GUI) interaction, but do not necessarily perform any end-user computing. These desktop services can be provided with varying levels of responsiveness, availability, throughput, and cost. Depending on the customer type requirements, different flavors of desktop service may be relevant. To better illustrate customer heterogeneity, we consider the following use cases with three different types of DCS customers: “enterprise user”, “Internet cafe” and “online game provider”. These use cases exemplify varying levels of expectations with respect to availability level, response time and throughput:

- **Enterprise users** ( $i = 1$ ) have high requirements regarding availability, response time and throughput when running DCS solutions. As expenses for IT services account for only a small fraction of the total costs, these users are typically willing to accept higher prices for such high quality IT services.
- **Internet cafe** ( $i = 2$ ) relies primarily on revenue from users accessing Internet. Therefore, its concern is having the desktops highly available and responsive. Unresponsive or faulty desktops will discourage customers from using the Internet cafe and hence reduce its revenues. At the same time Internet cafes are concerned with the cost of the service as desktop provisioning constitutes a substantial fraction of the overall cost base.
- **Online game provider** ( $i = 3$ ) has high requirements regarding response time to ensure real-time interaction between distributed game participants.

Table 7.10 depicts the customer types described above, including the mathematical representation of their preferences using scoring functions, serving as input for the utility-based matching. The numerical values of the  $\lambda$  weights and the fitting function shapes are chosen to represent the properties of the customer types described above. The fitting functions range between 0 and 1 with 0 denoting the lowest and 1 the highest correspondence of preferences. Such customer-specific functions are usually gained from user experience or from usage profiles collected on the provider side.

Based on the real use cases described above, we configured the request generator to continuously produce up to 20000 DCS requests (at a random rate varying from 0 to 30 seconds) for single VM deployments on Cloud. The requests generated are equally distributed between all the VM types presented in Table 7.2 and the three defined customer types. Among other things, each request specifies the VM type, the customer’s willingness to pay for one CU, the customer type, and the desired non-functional parameters, including availability, response time, and throughput. For the utility-based algorithm, the non-functional parameters are implicitly specified in the fitting function of the corresponding customer type. For the sieving algorithm, we use sample values depicted in Table 7.11, which correspond to the scoring value of 0.8 in

---

<sup>4</sup><http://www.microsoft.com>

<sup>5</sup><http://www.citrix.com>

Table 7.10.: Preferences of DCS customers  $i$  expressed using fitting functions  $f$  and relative weights  $\lambda$ ;  $\gamma = 0.0005$ ;  $\beta = 1 - \gamma$ .

$i$	Availability		Response Time		Throughput	
	$\lambda_{av}$	$f(av)$	$\lambda_{rt}$	$f(rt)$	$\lambda_{th}$	$f(th)$
1	$\frac{1}{3}$	$\frac{\gamma}{\gamma + \beta e^{(-av+87)}}$	$\frac{1}{3}$	$\frac{\gamma}{\gamma + \beta e^{(rt-12.5)}}$	$\frac{1}{3}$	$1 - \beta e^{-0.1th}$
2	$\frac{2}{5}$	$\frac{\gamma}{\gamma + \beta e^{(-0.9(av-84)}}$	$\frac{1}{5}$	$\frac{\gamma}{\gamma + \beta e^{(0.3(rt-55)}}$	$\frac{2}{5}$	$1 - \beta e^{-0.2th}$
3	$\frac{1}{5}$	$\frac{\gamma}{\gamma + \beta e^{(-0.6(av-76)}}$	$\frac{3}{5}$	$\frac{\gamma}{\gamma + \beta e^{(0.9(rt-20)}}$	$\frac{1}{5}$	$1 - \beta e^{-0.9th}$

the fitting functions of the utility-based algorithm, which is a high value forming a big challenge for this algorithm to compete with the sieving algorithm.

 Table 7.11.: Non-functional service requirements of customers  $i$  for the sieving algorithm.

$i$	max Willingness (\$/hour per CU)	min Availability (%)	max Response time (s)	min Throughput (Mbit/s)
1	0.06	96	3.5	16.3
2	0.03	94	25	8.2
3	0.025	91	10	1.8

### Matching Rate and Provider Coverage

The customers' DCS requests are delivered to the broker as input, where suitable providers are selected to deploy and then start their VMs, based on one of the two matching policies. Using the matching results, we calculated the matching rate and the provider coverage. To study the impact of the payment willingness on the matching, for each algorithm we first applied the numbers in Table 7.11 for customers *willingness to pay* (named "**single-case**") and then doubled these numbers (named "**double-case**") to observe the reaction of both algorithms. Figure 7.8 shows the matching rate of the sieving and utility-based algorithms. As depicted in the figure, the matching rate, which defines the percentage of matched requests, goes down with the increasing number of requests for all four cases. This is clearly due to the restriction of the computing capacity on the Cloud, i.e., when a Cloud is fully loaded with VM instances, there is no free host for more VM requests. For the "single" case of both

## 7. Evaluation



Figure 7.8.: Matching rate of the sieving and utility-based algorithms.

algorithms (the lower two curves) it can be seen that the sieving algorithm has a better matching rate. However, the curve for the utility-based algorithm is much closer to the one for sieving and the matching rate of the utility-based algorithm improves after 5000 requests. These results prove that the matching rate of utility matching works better than sieving matching when the number of service requests increases. In contrast, sieving works better with a small request number (under 5000). This is because with fewer than 5000 requests with sieving, more datacenters are matched and therefore more resource capacity is available. Additionally, it can be observed that the doubling of the payment willingness causes a rapid improvement in the matching rate for utility matching, which then outperforms the sieving matching rate even starting from 2000 requests. Overall, for both single and double cases the matching rate with the sieving scheme decreases drastically after 5000 requests, while the reduction with the utility scheme is more stable. The slow decrease of the matching rate for utility matching with increasing number of requests is due to the steady increase of the matched datacenters and resource capacity. This experiment indicates that the utility-based algorithm scales well with the customer requests and also performs better with high-paying customers.

Figure 7.9 shows the number of different providers selected by each matching algorithm in a single matching process with a certain number of service requests. It can be seen that the sieving algorithm (the two flat lines in the figure) selects for both scenarios between a constant value of Clouds for serving the customer request. Thus, the algorithm always matches the same seven Clouds in the “single” case, and then selects the more expensive providers as well, in case their price remains within the limitation of the customer’s budget, and their obtained utility remains positive. Therefore, the utility-based algorithm provides more benefits to the providers because every Cloud may have the opportunity to serve the customer requests. It can also be seen from

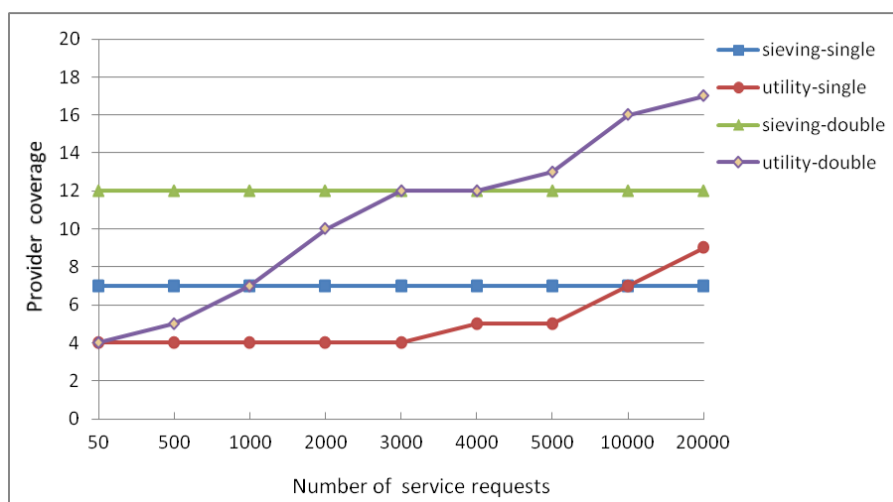


Figure 7.9.: Number of covered providers by the matching algorithms.

the figure that the doubling of the payment willingness causes utility to match more providers by selecting more expensive providers.

Figure 7.10 depicts the top selected providers after the first 5000 requests for the utility-based and sieving algorithms in their “single” and “double” cases. We can observe that the datacenters located in Europe are mostly matched by both algorithms, as they better fulfill the throughput and response time QoS requirements due to their proximity to the customer. However, the utility-based algorithm allows, through its economic-based matching more competitiveness and fairness between the selected providers than the sieving algorithm, which is based on random selection of providers.

### Cost Evaluation

We repeated the above simulation experiment to evaluate the cost-effectiveness of the used matching policies in the broker. We generated 5000 requests of the VM type *small S* with Linux OS for each of the three customer types by changing the matching policy each time. Based on the corresponding matching results and the current “pay-as-you-go” pricing policy of the selected providers, we then calculated the average expected VM cost per hour for each customer. Please note that for this experiment, the data traffic costs are not considered in the calculation of the cost. The results for each matching algorithm are illustrated in Figure 7.11.

It can be seen that the utility-based algorithm in both “single” and “double” cases is, from a user perspective, more cost-saving than the sieving algorithm, as it usually prioritizes the cheapest providers while considering the service quality expectations

## 7. Evaluation

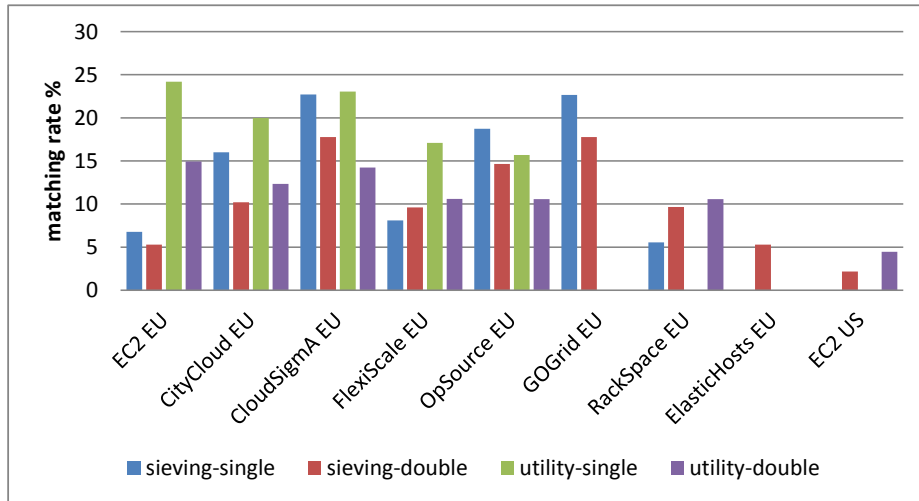


Figure 7.10.: Match percentage per provider after 5000 requests.

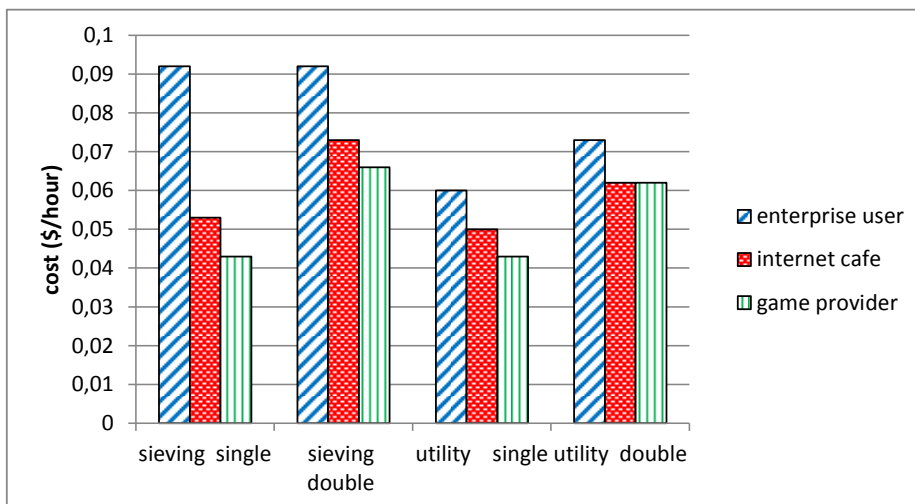


Figure 7.11.: Average cost for 5000 requests of VM type *small* with linux OS.

of the customers at the same time. Since the matching strategy for both algorithms is dependent upon payment willingness, the impact of doubling the payment willingness on cost is stronger. Note that for the enterprise users, with sieving matching the same three providers could always be matched, therefore the payment has no effect on the costs. For the same reason, the average VM costs with the utility-based matching for game provider and Internet cafe users in the “double-case” are equal.

### 7.3.2. Matching of Composite Cloud Services

For the experimental evaluation of our proposed HU-GA matchmaking scheme with a real case study, we investigate in this subsection the deployment of the Epigenomics bioinformatic application presented in Section 7.1.4 using our multi-Cloud service broker framework. One major reason for running the DNA sequencing workflows on the Cloud is to use the Cloud feasibility in scaling-in and scaling-out [141]. Depending on the number of sequences, the analysis work can take several days. For a fast diagnosis, it is necessary to involve more computing capacities for larger data or to reduce the resource number when the data set is small. In this case, Cloud is an ideal choice for this kind of applications. In addition to the computing facilities offered by Clouds, the scalable storage systems and analysis tools offered by Cloud also permit us to realize the vision of “data-driven medicine” [31]. An example of commercial companies providing genomic analysis in the Cloud to hospitals and researchers is DNAnexus<sup>6</sup>, which offers its services based on the Amazon AWS Cloud services.

It may be possible to run the DNA sequencing workflows on a single Cloud [95], however, it is better or even necessary to use multiple Clouds because the resources on one Cloud can be insufficient, or there could be a resource limit to the customers. In the following subsections, after presenting the simulation scenario, we explore the performance and cost benefits of running DNA sequencing workflows using our broker framework together with the HU-GA matching scheme.

#### Simulation Scenario

For the purpose of evaluation, we implemented the HU-GA algorithm as a new broker matchmaking policy using the Java-based Opt4J [108] genetic framework. For all experiments, we configured Opt4J to use a population size of 100 with a maximal generation number of 1000 and a crossover rate of 0.95. For a comparative study with HU-GA, we use the simple sieving matching algorithm. Additionally, as a scheduling policy, we use the simple **round robin** scheduler, which schedules workflow tasks to the first free available VMs in the composite service regardless of the datacenter location. Since Epigenomics workflows are not data-intensive, it is not necessary to use a more complex data locality-driven scheduling policy.

The modeled use case scenario of a Epigenomics workflow deployment using our broker framework is depicted in Figure 7.12. As the figure shows, the deployment consists of the following steps: First, the user gives his functional requirements for the workflow deployment by requesting from 10 to 50 VMs and one storage Cloud to store the workflow data. The half number of VMs is of the type *small S.2* and the other half is of type *medium S.2*. After acquiring the user QoS and budget requirements, the

---

<sup>6</sup><https://www.dnanexus.com>

## 7. Evaluation

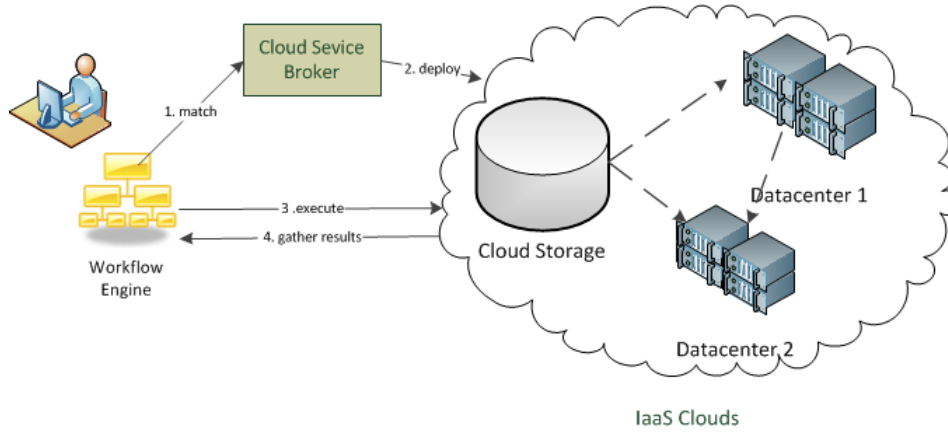


Figure 7.12.: Multi-Cloud DNA sequencing workflow deployment use case.

workflow engine forwards the user composite service request to the Cloud service broker (1) to select the suitable Clouds for the deployment based on the configured matching policy. After that, the requested resources are deployed (2), the workflow engine transfers the input data from the client to the Cloud storage, and then starts the execution of the workflow (3). Finally, the output data is stored in the Cloud storage when the execution is finished and can be fetched from the user client (4).

For the purpose of evaluation, we modeled two simulation scenarios. In the first scenario, named “**unconstrained**”, the number of VMs that can be deployed per datacenter is unlimited. In the second scenario, named “**constrained**”, we added a constraint in the user request to increase the chance of a multiple Cloud deployment by limiting the maximal number of VMs per datacenter to half of the total requested VMs.

The user requested minimal and maximal values for the QoS parameters (response time, availability, Cloud-To-Cloud latency, and throughput) to deploy the Epigenomics workflow on the Cloud are given in Table 7.12. These values are consumed by the sieving matching algorithm to select the random candidate Clouds for the deployment. The fitting functions and the relative weight for each QoS parameter, which are

Table 7.12.: User QoS requirements for the multi-Cloud DNA sequencing workflow deployment.

max Response time (s)	min Availability (%)	max Latency (ms)	min Throughput (Mbit/s)
25	95	50	12

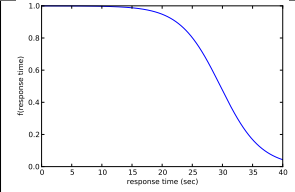
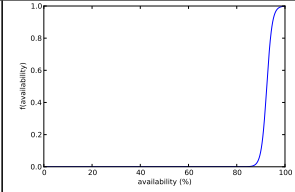
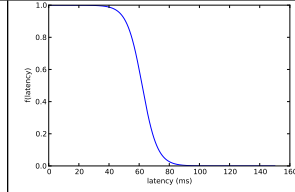
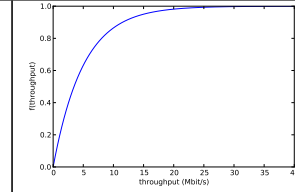
both part of the input for our proposed HU-GA matching algorithm, are given in Table 7.13. The fitting functions are set so that a score value higher than 0.8 corresponds to the minimal QoS requirements given in Table 7.12. It can be seen from the table that in the modeled user request, the Cloud-to-Cloud latency has more importance for the



### 7.3. Evaluation of the Matching Algorithms

user because of the high amount of inter-task data that needs to be transferred at run-time between Clouds. Hence, low latency values between the Clouds are required. The second place of user concern is the availability of the Clouds. Whereas both the client-to-Cloud throughput and response time have no significance because of the relatively small amount of input/output data that need to be transferred from/to the client. For all our conducted experiments, we fixed the lease time period  $T$  in which the requested Cloud resources are provisioned for the user to two days.

Table 7.13.: User preferences for the multi-Cloud DNA sequencing workflow deployment expressed using fitting functions  $f$  and relative weights  $\lambda$ ;  $\gamma = 0.0005$ ;  $\beta = 1 - \gamma$ .

Response time		Availability		Latency		Throughput	
$\lambda_{rt}$	$f(rt)$	$\lambda_{av}$	$f(av)$	$\lambda_{lat}$	$f(lat)$	$\lambda_{th}$	$f(th)$
$\frac{1}{10}$	$\frac{\gamma}{\gamma + \beta e^{(0.3(rt-55))}}$	$\frac{3}{10}$	$\frac{\gamma}{\gamma + \beta e^{(-0.9(av-84))}}$	$\frac{5}{10}$	$\frac{\gamma}{\gamma + \beta e^{(0.2(lat-100))}}$	$\frac{1}{10}$	$1 - \beta e^{-0.2th}$
							

The maximum acceptable hourly price for each requested *small S.2* and *medium M.2* VM type as well as the maximum gigabyte price for data storage and network traffic are given in Table 7.14. Clearly, these budget limits express the medium payment willingness of the users of such types of scientific applications.

Table 7.14.: Maximal payment for VMs, storage, and traffic for the DNA sequencing workflow deployment.

VM small S.2 (\$/hour)	VM medium M.2 (\$/hour)	Cost storage (\$/GB)	Cost traffic (\$/GB)
0.09	0.18	0.1	0.1

### Time complexity and Convergence

In order to evaluate the time complexity of the HU-GA algorithm with the increasing number of requested VMs in the “unconstrained” scenario, we measured in an initial experiment the time consumed by the genetic algorithm to find the optimal composite service and then compared the results with the sieving algorithm. Figure 7.13 illustrates the results. As can be seen from the figure, the time complexity of HU-GA increases exponentially as the number of VMs increases from 10 to 50, reaching 40 seconds with 50 VMs. Although the complexity values brought by our approach

## 7. Evaluation

are up to three orders of magnitude compared to the sieving algorithm, it is negligible compared to the workflow makespan, which can take many hours.

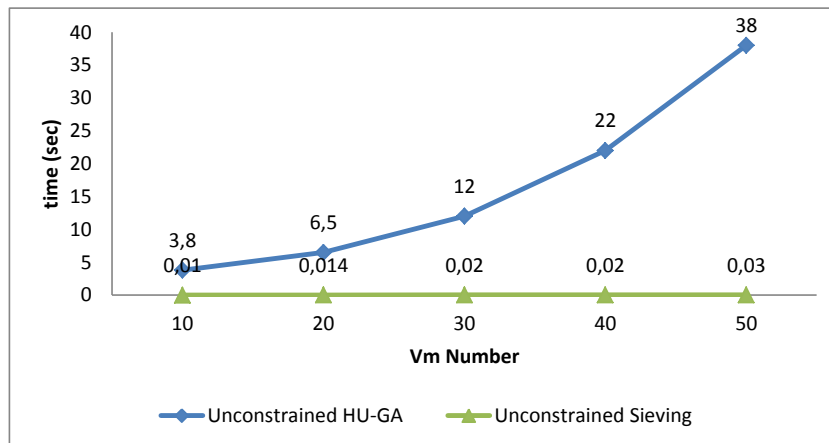


Figure 7.13.: HU-GA and sieving time complexity for different VM numbers.

We repeated the above experiment by measuring the number of iterations needed for the convergence of the genetic algorithm for both the “constrained” and “unconstrained” use cases. Additionally, in order to assess the impact of the pre-sieving process used in the HU-GA algorithm on the algorithm convergence, we conducted the experiments first with enabling and then with disabling pre-sieving. The results for all scenarios are presented in Figure 7.14.

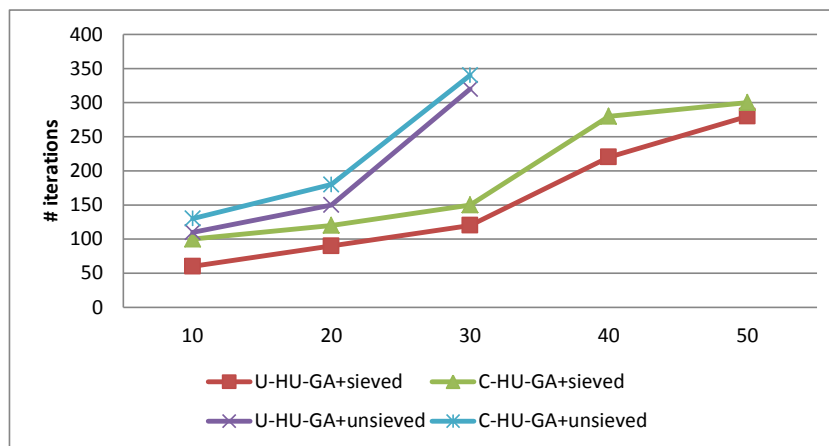


Figure 7.14.: Unconstrained (U)/constrained (C) HU-GA convergence for different VM numbers.

As depicted in the figure, the continual increase of the VM number results in a steady increase of the iteration number for all scenarios. Thus, we conclude that more time

is needed to perform the matching. It can be seen that the fastest convergence is reached with the “pre-sieved” HU-GA in the “unconstrained” scenario, followed by the “pre-sieved” HU-GA in the “constrained” scenario. This result proves the benefit of pre-sieving in improving the genetic algorithm performance. We also observed that, starting from 40 VMs, the “unsieved” HU-GA algorithm is terminated (after 1000 iterations) without reaching the convergence. Because of this result, all of the next conducted experiments are performed with a “pre-sieved” HU-GA algorithm. The maximal achieved utility in the convergence values with increasing number of VMs as well as sample convergence graphs can be seen in Appendix B.1.

### Workflow Makespan

We repeated the previous experiment to measure the workflow makespan in minutes after a single run of the Epigenomics workflow on the Cloud resources allocated either using the sieving or HU-GA matching algorithm. In addition, we compared the result with the predicted theoretical makespan value (baseline) calculated based on the number of requested VMs in the composite service. The results for the “unconstrained” scenario with different numbers of requested VMs are shown in Figure 7.15.

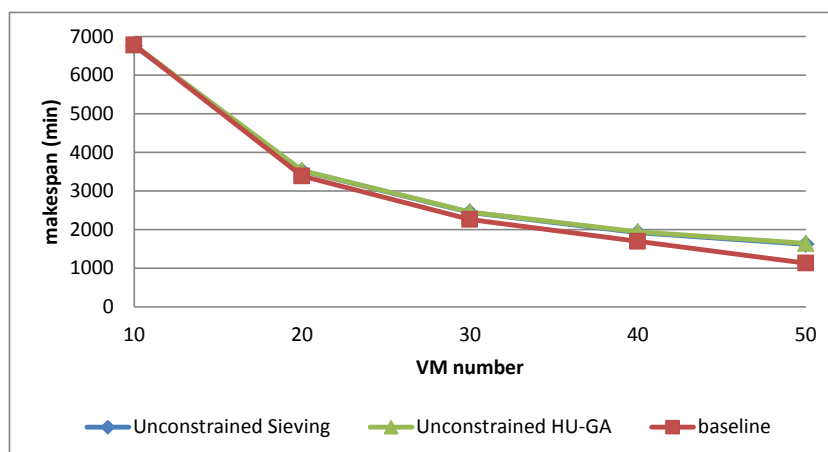


Figure 7.15.: Workflow makespan in minutes with HU-GA-and sieving.

It can be seen from the figure that for both algorithms the makespans are very close. This is explained by our assumption that the VMs on all the matched Clouds have the same hardware configuration and performance, and by the fact that both algorithms have the tendency to select Clouds in the same user region to take advantage of low data transfer time and latency. We observed also that the execution of the workflow within the requested two days lease time is only possible with more than 30 VMs, otherwise the deadline will be violated and the user will be charged for the additional

## 7. Evaluation

needed time. The difference from the theoretical makespan is due to the overhead resulting from the file transfer, particularly with a large number of requested VMs in the composite service.

### QoS Evaluation

One of the important criteria to prove the effectiveness of the matching algorithms is the QoS resulted from the deployment of the requested composite services on the matched Clouds. For this purpose, we calculated for the “unconstrained” and “constrained” simulation scenarios with 50 requested VMs the aggregated values for the availability, response time, throughput, and latency according to Table 5.3. The results for each SLA attribute with the sieving and HU-GA algorithms are depicted in Figure 7.16.

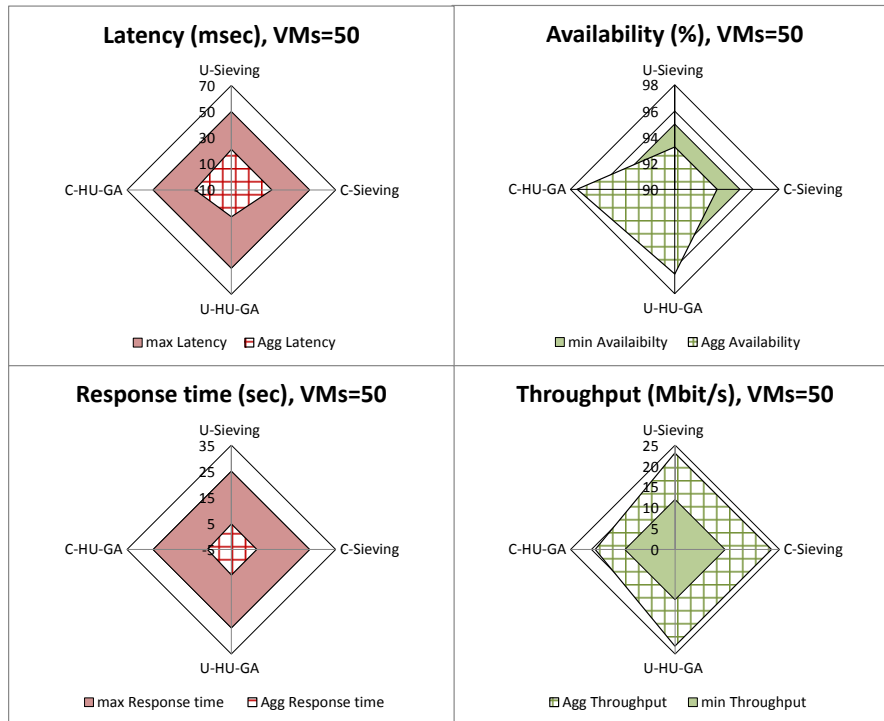


Figure 7.16.: Average aggregated SLA values for unconstrained (U)/constrained (C) HU-GA and sieving for a VM number of 50.

It can be seen from the figure that, except for the availability attribute, both algorithms are able to keep the QoS values inside the requested user range (lightly colored). The low aggregated availability value obtained with sieving matching is due to its random strategy in choosing Clouds, which leads to a multiplicative decrease of the availability, in particular, with a high number of component services in the

composition. Moreover, we observed that the use of HU-GA matching assures better aggregated values for latency and availability, as they are both of high importance for the user, which is expressed with high weights in their respective scoring functions. The good values of the client-to-Cloud throughput and response time are explained by the tendency of both algorithms to choose the Clouds located in Europe to reduce the time for transferring input and output data from/to the user client. Furthermore, the unconstrained use case gives the best results in terms of latency as, in this case, the chance to deploy all the VMs in one Cloud is higher. Please refer to Appendix B.2 to see the full list of the matched datacenters for each scenario.

## Cost Evaluation

In this subsection, we evaluate the impact of the used matching policies on the execution costs, including costs for compute, storage, and traffic. Thus, we calculated the total charged cost after a single run of the Epigenomics workflow for a different number of VMs. As the VMs are charged on an hourly basis, we rounded the makespan up to the nearest hour and then calculated the cost based on the rounded values. Our cost calculation does not consider additional costs like license and VM images costs, which are charged by some Cloud providers. For the traffic cost calculation, we measured the amount of data traffic transferred between two Clouds during the execution and then applied the current pricing policies. The resulted total costs for the different use cases are depicted in Figure 7.17.

We can observe from the figure that the HU-GA algorithm allows up to 25% cost-saving compared to sieving for both “unconstrained” and “constrained” scenarios. The best cost-saving is achieved with “unconstrained” HU-GA algorithm due to its tendency to deploy all the requested VMs in the cheapest provider if the required QoS is fulfilled. The small increase of the costs with a higher number of used VMs results from the low makespan, which compensates the additional costs of the leased VMs.

In order to evaluate the amount of cost-saving in the requested lease period  $T$  of two days, we use a metric called Cost-Budget-Ratio (CBR) defined as follows:

$$CBR = \frac{C_x(T)}{C_r(T)} * 100 \quad (7.1)$$

where  $C_x(T)$  and  $C_r(T)$  respectively denote the total lease cost and the total user budget for the period  $T$ . These are calculated according to Equations 5.1 and 5.3. Based on the above metric definition a low CBR value is expressed with a rise in cost-saving. We repeated the previous experiment to compare the obtained CBR values with different matching policies in the two deployment scenarios. The results are provided in Table 7.15.

## 7. Evaluation

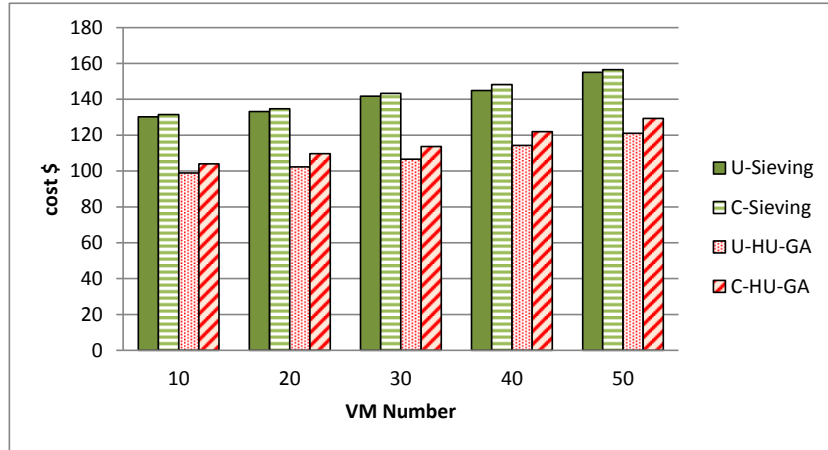


Figure 7.17.: Total workflow execution cost with unconstrained (U)/constrained (C) HU-GA and sieving for different VM numbers.

Table 7.15.: CBR values with unconstrained (U) and constrained (C) HU-GA and sieving for different VM numbers.

VMs	10	30	30	40	50
U-Sieving	49.22	61.03	66.12	69.85	72.92
C-Sieving	50.27	59.84	66.57	70.95	73.4
U-HU-GA	36.53	46.13	50.67	53.31	55.04
C-HU-GA	42.15	49.48	54.07	56.88	58.77

It can be seen from the table that, for all scenarios, the budget constraints could be fulfilled. Here we also confirm that HU-GA allows more cost-saving than sieving by keeping the costs under 60% of the user requested budget. In contrast, a workflow deployment with sieving is more economic only with a small number of VMs. In addition, we can also prove that the “unconstrained” HU-GA deployment gives the minimal CBR values and consequently ensures the best cost-saving.

### 7.3.3. Case Study: CAD-aaS

The aim of this case study is to evaluate the HU-GA utility-based algorithm with a prospect-based composite service selection approach [52] developed in parallel with this work. The evaluation is done with a real world SaaS scenario with the same simulation setup from Section 7.1. In the following, we compare both approaches and present the simulation scenario and the achieved results.

### Comparison of Utility and Prospect-based Selection

The prospect theory [97] is an alternative model to the classic expected-utility theory [153] for describing decision-making under uncertainty. Its basic principle is to model human behavior in assessing potential gains and losses rather than focusing on a final outcome, such as in the case of MCDM methods. The evaluation of gains and losses is expressed relative to a reference point through an “S-shaped” satisfaction function, which is analogical to the scoring function used in the utility-based selection. An example of a satisfaction function for the availability of QoS parameter based on different importance weights is shown in Figure 7.18. The function shape is influenced by the importance weight assigned to availability by the user. In this concrete example, the user accepted boundaries are 99.5 and 100 and the reference point has a score of 0.5 (meaning that a score over 0.5 expresses a gain). The score achieved for a normalized value of 0.4 (availability 99.7%) is usually under 0.5, that is, an unsatisfactory value.

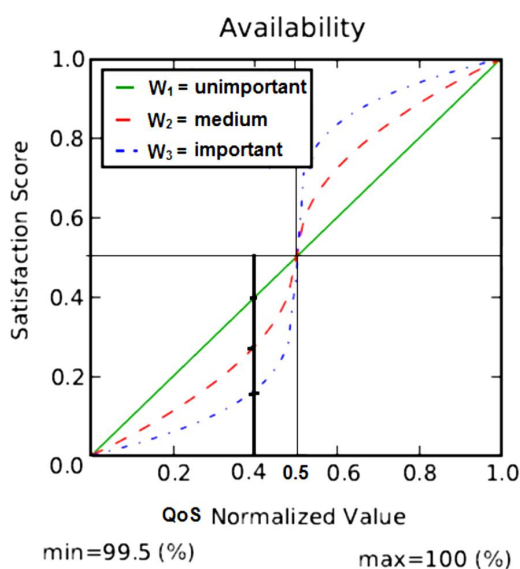


Figure 7.18.: Sample satisfaction function for the availability [52].

The multi-Cloud selection approach based on the prospect-theory, which is used for the purpose of comparison, is composed of three phases. In the first phase, the candidate providers are ranked with respect to the specific SLA requirements of each single service, called sub-SLA. In the second phase, the set of provider candidates are ranked with respect to the SLA requirements of the whole service composition, called meta-SLA. A final score is then calculated based on an additive weighting of the sub-SLA and meta-SLA scores. The candidate composition with the highest final score is then chosen by the algorithm for the deployment. Since the ranking is performed using SLA attribute-specific satisfaction functions, the user has to determine

## 7. Evaluation

when an attribute yields a gain or a loss by assigning value boundaries and importance weights to each of them. The key differences between this approach and the proposed utility-based algorithm are summarized in Table 7.16:

Table 7.16.: Comparison between utility and prospect-based selection.

Factor	Utility-based Selection	Prospect-based Selection
cost significance	multiplicative, not scored	scored like QoS
fitting function	preferred values	gains and losses
weights meaning	relative importance	user sensitivity
selection strategy	maximize user utility	increase user satisfaction
SLA consideration	meta-SLA	meta-SLA and sub-SLA
decision input	scoring functions, weights budget	QoS and budget boundaries weights, satisfaction functions

### Simulation Scenario

The following simulation scenario has been modeled: a Computer Aided Design (CAD) software provider located in Europe aims to deploy its CAD software on multiple public IaaS Clouds to save on cost and to increase the service quality. The software is offered to end-users as SaaS application, called CAD-as-a-Service (CAD-aaS), in three software editions: enterprise, professional, and standard, which have different infrastructure requirements. The users can use the corresponding edition of the CAD software on-demand through the internet by remotely accessing the user interface from their thin client desktop or mobile device. Since the simulation results for different software editions are similar, we address only the experiment with the standard edition.

The various components of the CAD-aaS application in its standard edition, forming a composite service, are depicted in Figure 7.19. As can be seen from the figure, this deployment requires one *small S.2* VM for the application user interface (UI), one *large L.2* VM with GPU features for the computation component, and 100GB of Cloud storage to store the CAD models. Due to privacy and performance concerns, the SaaS provider requires that the storage Cloud be located in Europe. Each requested service has its specific SLA requirement, termed as sub-SLA, while the global composite service SLA requirements are termed meta-SLA. The numbered edges in the figure represent the interaction steps needed between the components to process a CAD-aaS customer request. In addition, a score is assigned to each edge expressing how strong is the data traffic.

The minimal and maximal non-functional meta-SLA values requested by the SaaS provided for availability, throughput, and latency as well as his maximum budget for



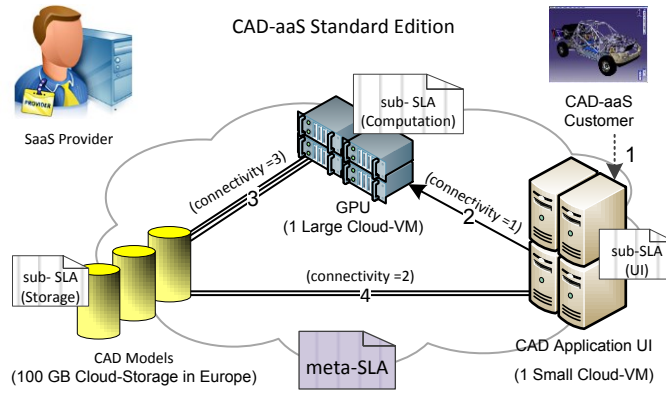


Figure 7.19.: Composite infrastructure service for a CAD-aaS standard edition [52].

VMs, storage, and traffic, are presented in Table 7.17. The leasing time has been set to one month.

Table 7.17.: Non-functional meta-SLA requirements of the CAD-aaS standard edition.

Cost VM CU (\$/hour)	Cost storage (\$/GB)	Cost traffic (\$/GB)	Availability (%)	Throughput (Mbit/s)	Latency (ms)
0.07	0.1	0.1	96	5	100

### Simulation results

In order to compare the matching efficiency of both selection algorithms, we conducted a simulation experiment using the scenario described above for the standard CAD-aaS edition. For this experiment, we used the same scoring functions from Section 7.3.2 for the utility-based algorithm, but with equally weighted QoS attributes. Since the cost factor is of high importance for the low paying, standard-edition customers compared to other QoS parameters, we assigned an importance weight value of 0.8 for cost and an importance weight value of 0.25 for all other QoS parameters for the prospect-based algorithm. Then we applied these weights and their respective boundaries to build the specific satisfaction functions, which have the same shape as in Figure 7.18. Furthermore, to dominate the influence of meta-SLA on sub-SLAs in service selection, the meta-SLA and sub-SLA have been assigned as weights 0.9 and 0.1, respectively.

Figure 7.20 depicts the results for the aggregated QoS values of the matched service composition (left side) and the total charged cost (right side) for both algorithms. As can be seen from the figure, both algorithms are able to satisfy the SaaS provider requested meta-SLA requirements. Moreover, the SLA metrics measured for both algorithms look similar, with a slight advantage for the utility-based algorithm regarding

## 7. Evaluation

the total cost. This is due to the fact that cost is a multiplicative factor in the utility function; thus, it has more influence on the selection. The results also demonstrate the tendency of both algorithms to select services from the same Cloud datacenter to minimize the latency and traffic cost. This explains the aggregated latency value of 10 ms obtained for both approaches and, therefore, no traffic costs are charged. Furthermore, both algorithms prioritize datacenters located in Europe to assure the minimal required throughput between customer and the user interface. As datacenter, Amazon EC2 EU and VoxCloud EU, respectively, have been selected by the prospect-based and utility-based algorithms.

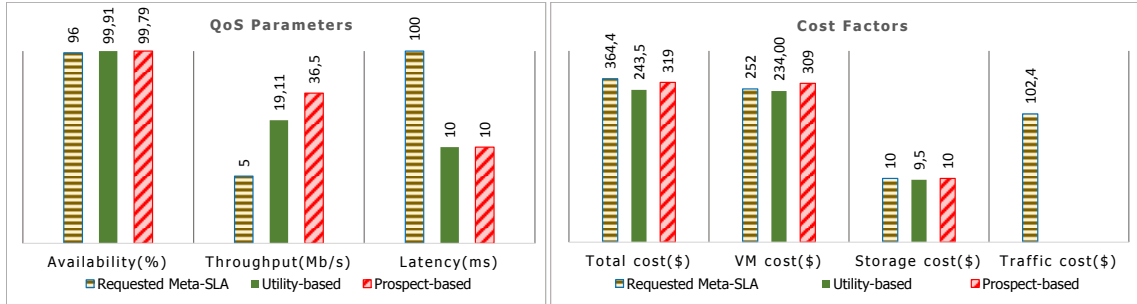


Figure 7.20.: Aggregated QoS values and cost with prospect-based and utility-based selection.

### 7.3.4. Results Discussion

The DCS experiment in Section 7.3.1 showed that the utility-based matching algorithm outperforms sieving in terms of matching rate and provider coverage in particular when the Clouds are heavily loaded with requests. Moreover, its cost-efficiency compared to Sieving has been proved for different customer types.

The simulation experiments with the DNA-sequencing workflow in Section 7.3.2 showed the benefit of using HU-GA compared to sieving matching in reducing the total execution costs, as well improving the QoS of the workflow deployment. This profit is of more importance, particularly when running the workflow on a large service composition. Although the HU-GA algorithm needs more time to perform the matching, this time can be neglected compared to the real workflow makespan and therefore it pays off.

The case study with the CAD-aaS simulation scenario on multi-Cloud presented in Section 7.3.3 showed that HU-GA competes perfectly with an efficient prospect-based selection algorithm, in particular by reducing the total cost of the deployment.

## 7.4. Evaluation of the Multi-dimensional Resource Allocation

In order to validate our multi-dimensional resource allocation scheme proposed in Chapter 6, we investigate in this simulation experiment the deployment of data-intensive multi-Cloud workflow applications, using the example of the Montage application (see Section 7.1.4). Our idea of deploying data-intensive workflow applications on multi-Cloud comes from the observation of the following scenario: A customer works on several Clouds and stores data on them. There is a demand for jointly processing all of the data to form a final result. This scenario is similar to the collaborative work on the Grid. For example, the Worldwide LHC Computing Grid (WLCG) [160] involves more than 170 computing centers, where the community members often work on a combined project and store their data locally on their own sites. A workflow application within such a project must cover several Grid resource centers to process the locally stored or replicated data. In the following subsections, we present the simulation scenario of deploying Montage using the multi-Cloud service broker framework and show the benefits of using our proposed resource allocation scheme in terms of cost and performance. In particular, we demonstrate the effects of the HU-GA based Clouds selection and data-aware scheduling policies on makespan, Intercloud data transfer, and costs.

### 7.4.1. Simulation Scenario

We conducted the same Montage experiment from Section 7.2.2 by adding one storage Cloud to store the workflow input/output data. The input data are transferred from the client to Cloud storage by the workflow engine before the start of workflow execution, whereas the output data are stored in the Cloud storage when the execution is finished. If the user executes the same workflow multiple times on the same provisioned Cloud resources, our implemented data locality scheduling scheme reuses the existing replicated input data in order to save on data transfers and costs. For simplicity, we consider in our evaluation only the first run of the workflow.

To evaluate our resource allocation scheme with a large-scale application such as Montage, we modeled the following two simulation scenarios. In the first scenario, named “**EU-deployment**”, all the requested VMs and storage Cloud are deployed in the same user region (Europe). In the second scenario, named “**EU-US deployment**”, all the 10 VMs of type *small S.2* are deployed on Clouds located in the US region, while the 10 VMs of type *medium M.2* and the storage Cloud are located in Europe. The non-functional SLA requirements for both scenarios, given in Table 7.18, express the user’s desired ranges for the aggregated availability, Cloud-to-Cloud latency and

## 7. Evaluation

client-to-Cloud throughput in order to deploy the workflow with an acceptable quality. These values are consumed by the sieving matching algorithm described in Chapter 5 to select the target Clouds for the workflow deployment. Please note that for this experiment, the response time is not considered in the SLA requirements.

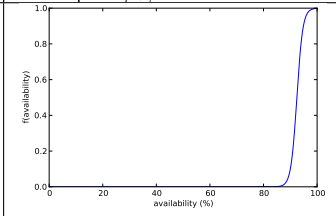
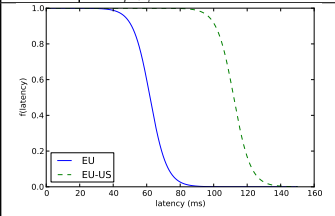
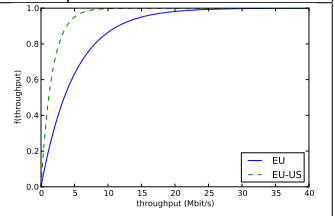
Table 7.18.: User non-functional SLA requirements for EU and EU-US scenarios.

Deployment scenario	min Availability (%)	max Latency (ms)	min Throughput (Mbit/s)
EU	95	50	12
EU-US	95	100	2

The fitting functions and the relative weight for each SLA parameter, both required for the HU-GA matching algorithm, are given in Table 7.19. As we can see from the table, for the first scenario, the user prefers Clouds with low Cloud-to-Cloud latency values to favor the data transfer time, while for the second scenario, availability is equally weighted with the latency and the client-to-Cloud throughput has more significance.

Table 7.19.: User preferences for EU and EU-US scenarios expressed using fitting functions  $f$  and relative weights  $\lambda$ ;  $\gamma = 0.0005$ ;  $\beta = 1 - \gamma$ .

Scenario	Availability		Latency		Throughput	
	$\lambda_{av}$	$f(av)$	$\lambda_{lat}$	$f(lat)$	$\lambda_{th}$	$f(th)$
EU	$\frac{3}{10}$	$\frac{\gamma}{\gamma + \beta e^{(-0.9(av-84))}}$	$\frac{6}{10}$	$\frac{\gamma}{\gamma + \beta e^{(0.2(lat-100))}}$	$\frac{1}{10}$	$1 - \beta e^{-0.2th}$
EU-US	$\frac{2}{5}$	$\frac{\gamma}{\gamma + \beta e^{(-0.9(av-84))}}$	$\frac{2}{5}$	$\frac{\gamma}{\gamma + \beta e^{(0.2(lat-150))}}$	$\frac{1}{5}$	$1 - \beta e^{-0.6th}$

The maximum user payment willingness for each VM type as well as for storage and network traffic are given in Table 7.20.

Table 7.20.: Maximal payment for VMs, storage, and data traffic for EU and EU-US scenarios.

VM small S.2 (\$/hour)	VM medium M.2 (\$/hour)	Cost storage (\$/GB)	Cost traffic (\$/GB)
0.09	0.18	0.1	0.12

### 7.4.2. Impact of Clustering

To evaluate the impact of clustering with respect to increasing cluster number  $k$ , we measured in an initial experiment the total time needed to execute a single run of the sample Montage workflow and the total consumed time for data transfer in minutes for the “EU-deployment” scenario. For this experiment, we configured the broker to use **HU-GA** matching policy together with **DAS** as a scheduling policy. For an accurate calculation of the execution time, we extracted from the workflow trace the real delay overhead resulting from clustering, post-scripting, and queuing. Figure 7.21 illustrates the results achieved.

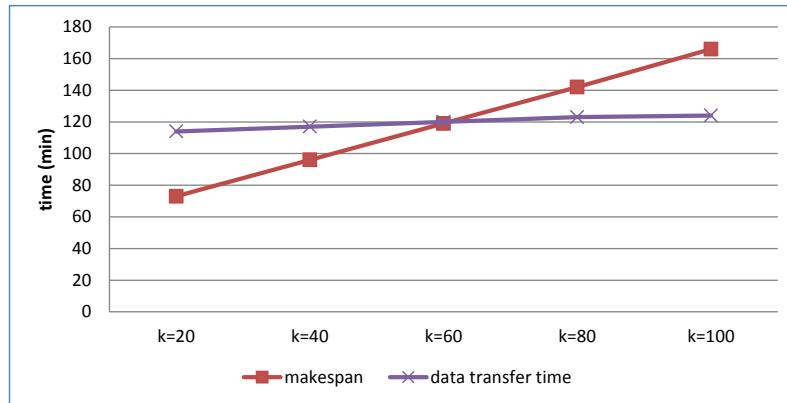


Figure 7.21.: Workflow makespan and total data transfer time with HU-GA+DAS EU for different cluster numbers  $k$ .

As depicted in the figure, the continual increase of  $k$  results in a steady increase of workflow execution time, since more jobs are queued. This demonstrates that our allocation scheme scales well with the increasing number of clustered workflow jobs. Although the transfer time is nearly constant, as file transfers are performed with a high throughput and low latency values because the matched Clouds are close to the user, we observed a slow decrease of the transfer time for small numbers of  $k$ . In the latter case, more files are merged in a clustered job, so that the amount of Intercloud transfers is heavily reduced. For all of the next experiments, we fixed the cluster number to  $k=20$ , as it gives us the best results in terms of makespan and consequently execution cost.

### 7.4.3. Impact of Resource Allocation on Workflow Makespan

We repeated the previous experiment with the “EU” and “EU-US” scenarios with the different matching and scheduling policies presented in Chapters 5 and 6. For a comparative study, we executed the workflow using a simple round robin scheduler

## 7. Evaluation

and the scheduler from the literature known as Min-Min [59], which prioritizes tasks with minimum runtime and schedule them on the *medium* M.2 VM types. The results with  $k=20$  for all the possible combinations are shown in Figure 7.22.

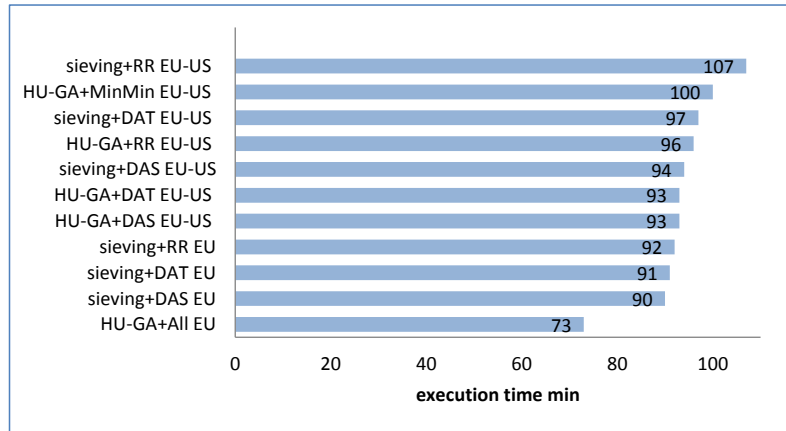


Figure 7.22.: Workflow makespan with different scheduling and matching policies for  $k=20$ .

It can be seen from the figure that for both scenarios, the use of utility-based HU-GA matching algorithms combined with the DAS or DAT scheduler gives the lowest execution time compared to sieving. This result proves how the efficiency of the HU-GA matching affects the scheduling performance. For the EU scenario, the HU-GA algorithm has a tendency to deploy all the requested VMs on the cheapest datacenter to save cost and minimize latency so that the user is charged only for the costs to transfer input/output data from/to storage Cloud. This explains the same makespan and Intercloud transfer obtained with different scheduling policies. We observed also that our implemented DAS and DAT scheduler outperform round robin and Min-Min, especially for the EU-US scenario (up to 25% faster).

### 7.4.4. Impact of Resource Allocation on Intercloud Data Transfer

In order to assess the impact of our multi-dimensional scheme on reducing the Intercloud data transfers, we measured the size of Intercloud transfers and the ratio of transfer time over the total consumed processing time for the previous simulation experiment. The results for different combinations of matching and scheduling policies with  $k=20$  are depicted respectively in Figure 7.23 and Figure 7.24.

It can be seen from Figure 7.23 that the DAS scheduler keeps the Intercloud transfer size under 10 GB for both scenarios with HU-GA and sieving matching policies. Next to DAS on saving Cloud-to-Cloud data movements is DAT, whereas MinMin and Round robin occupy the last places. The evaluation results for the transfer time

#### 7.4. Evaluation of the Multi-dimensional Resource Allocation

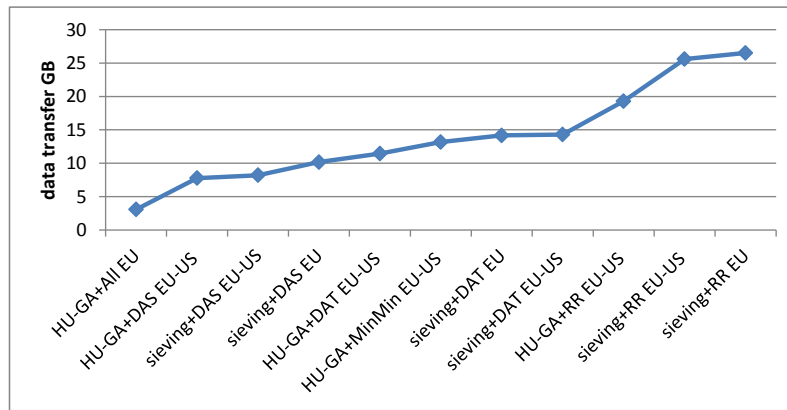


Figure 7.23.: Total amount of Intercloud transfers with different scheduling and matching policies for  $k=20$ .

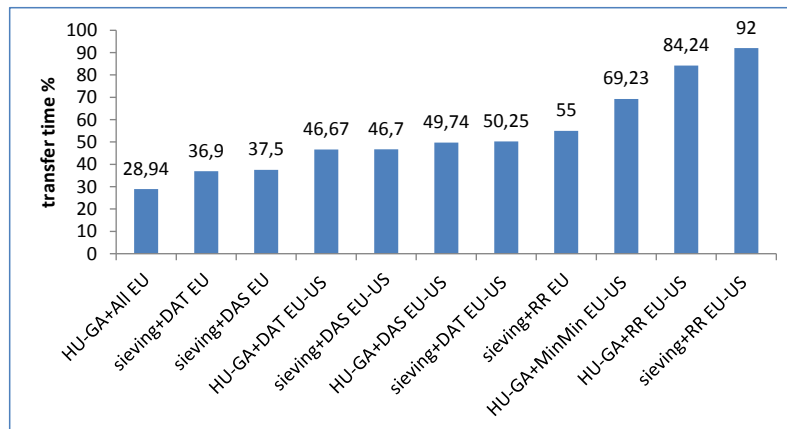


Figure 7.24.: Percentage of data transfers with different scheduling and matching policies for  $k=20$ .

ratio also prove that DAS and DAT are able to reduce the transfer time ratio up to 37% and 50% respectively for the EU and EU-US scenario. In contrast, the use of round robin and Min-Min scheduling, regardless of the matching policy, results in very high transfer ratios in particular for the EU-US scenario, which consequently creates a disadvantage for the workflow makespan. Therefore, data locality has more impact when Clouds are not close to the user, as, in this case, latency and throughput have a greater effect on the transfer time.

### 7.4.5. Cost Evaluation

In this subsection, we evaluate the impact of matching and scheduling policies on the traffic costs for both simulation scenarios. The previous experimental results show that the use of HU-GA matching and data-aware scheduling heavily reduces the amount of Intercloud transfers and consequently the data traffic costs. This result is demonstrated in Figure 7.25, in which the amount of compute, storage, and traffic costs for different combinations after a single workflow run is illustrated. From the figure, we found that the utility-based matching combined with the DAS scheduler have more of a benefit for the compute and traffic cost compared to the other use cases. For example, comparing utility and sieving with DAS, respectively, up to 25% and 15% cost saving can be achieved with the EU and EU-US deployment. Since the storage is used only to store the input and output data, its charged costs are constant for all scenarios.

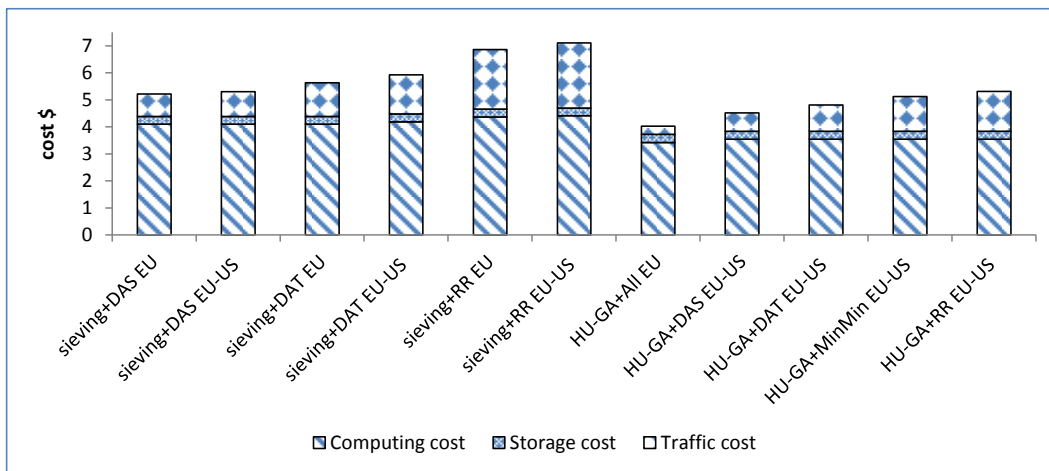


Figure 7.25.: Total execution cost with sieving (left) and HU-GA (right) matching for  $k=20$ .

We conducted another experiment by adding another storage Cloud located in the US region for the EU-US scenario. The gathered cost results are depicted in Figure 7.26. For the purpose of comparison, we do not consider the time needed to transfer the input files from the Client to the US located storage Cloud. It can be seen from the figure that the total costs for the two storage Clouds use-case are very close to the one storage use-case, even with the doubled storage cost. This is due to the tendency of the utility algorithm to deploy VMs located in the US and the added storage Cloud to the same provider to save on traffic costs. This cost saving is more meaningful when running the same workflow multiple times.



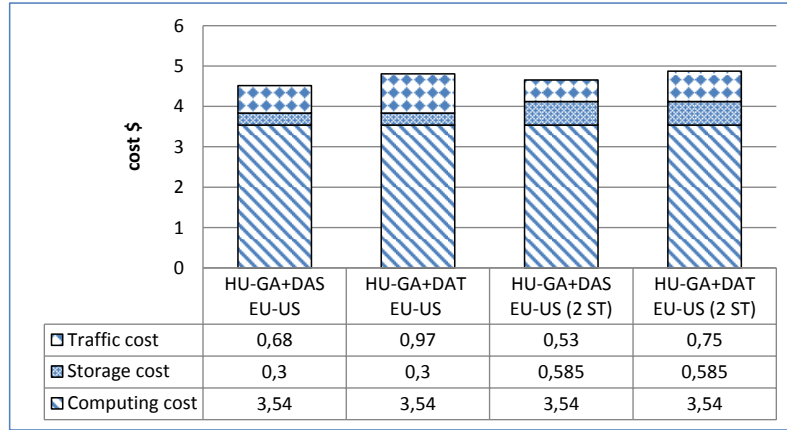


Figure 7.26.: Total execution cost with one and two storage Clouds (ST) for  $k=20$ .

### 7.4.6. Results Discussion

The experimental results gathered above showed the benefits of combining efficient matching policies and data locality-driven scheduling in reducing the amount of Intercloud transfers and the total execution costs as well improving the workflow makespan. In particular, the best results for our multi-dimensional resource allocation scheme are achieved by combining the HU-GA matching algorithm and the DAS scheduling policy. Moreover, we demonstrated that the impact of the optimal selection of the Cloud services in the matching phase is of great importance for the scheduling phase at runtime. Thus, we found that the best fulfillment of the aggregated latency and throughput SLA metrics for the requested composite service, when using HU-GA-based matching, reduces the makespan and the amount of Intercloud data transfer and consequently the costs. Finally, the experiments showed that the replication of the Cloud storage benefits the data traffic for large-scale, data-intensive applications without affecting the total costs.

Based on these observations, we conclude that by including the multi-dimensional resource allocation scheme in our multi-Cloud service broker framework, we can best satisfy the SLA requirements of users as well their budget expectations when running workflow applications on multi-Cloud. The scientific Montage workflow application used in this case study simply serves as an example for the evaluation. However, the proposed framework is capable of supporting the deployment of different applications after minor adjustments.

## 7.5. Summary

In this chapter, we have presented the simulation-based evaluations of the contributions achieved throughout this thesis. We have detailed the experimental setup in Section 7.1 and the different simulation scenarios within which multiple experiments were conducted and demonstrated the performance and cost benefits from our proposed approaches. These benefits can be summarized as follows:

- As demonstrated in Section 7.2, our multi-Cloud service broker framework scales well with a rising number of single and composite service requests. We have also justified the need of efficient matching and scheduling policies for the optimal deployment of services on multi-Cloud. Also, we demonstrated the benefits of clustering techniques for running workflows.
- As demonstrated in Section 7.3, our proposed utility-based matching algorithm is able to match cost-efficient Clouds within the expected service quality even with large service compositions, when compared to the sieving algorithm. Also, its matching efficiency competes perfectly with a prospect-based selection algorithm.
- In Section 7.4, we demonstrated that the use of our multi-dimensional resource allocation approach with data-intensive scientific workflows significantly reduces the amount of Intercloud transfers (up to 50%) and the total execution costs (up to 25%) as well as improves the workflow makespan (up to 20%).

The benefits listed above confirm the expected added values from using a multi-Cloud service broker, such as the cost, interoperability and performance gains, which were the motivation of the work conducted in this thesis. An extensive list of possible extensions and future improvements to the proposed contributions is provided in Chapter 8.

## 8. Conclusion and Outlook

In this chapter, we conclude the thesis by summarizing its contributions and their implications for the advancement of multi-Cloud service brokering research. We present in Section 8.1 a summary of the thesis contributions, and in Section 8.2 we discuss the constraints of the achieved contributions, and the possible extensions to mitigate them. Finally, Section 8.3 presents the potential future research directions that go beyond this research work.

### 8.1. Summary

The main objective of this research work is to find an answer to the following fundamental question:

*How can an adequate broker framework for automating and optimizing the deployment of multi-Cloud applications be realized?*

In the following, we summarize the contributions achieved throughout this thesis, which answer the above question by addressing the five research questions presented in Section 1.2.

The intention of Research question 1 is to design an adequate broker framework to assist users in selecting and managing their single and composite services on top of heterogeneous IaaS Clouds. To address this question, we implemented a generic architecture of a multi-Cloud service broker framework. The framework architecture, presented in Chapter 4, includes components for SLA management, matchmaking and monitoring and discovery, as well as components to schedule workloads and manage data transfers between Clouds. Thus, it supports different multi-Cloud applications, including workflows. In addition, the framework includes service and provider data repositories to store the service and provider SLA information. For the data representation in the repositories, we implemented an ontological model consisting of two ontologies, which are used to semantically describe the composite service requests and IaaS Cloud provider offerings. This ontological model addresses mainly Research question 2.

For addressing Research question 5, we implemented a simulation environment based on CloudSim and WorkflowSim to evaluate the broker framework and validate its

## 8. Conclusion and Outlook

functionality. Herewith, we modeled the multi-Cloud infrastructure as well as different usage scenarios extracted from real workload traces. Furthermore, to make our simulations more realistic, we used the real pricing policies and QoS metrics of current public Clouds to model the characteristics of the Clouds. Using the simulation testbed, we demonstrated the good scalability of the framework with single and multi-Cloud workflow services. From our experiments with simple matching policies, we identified the need for efficient matching policies in the broker in order to make the trade-off between cost, performance, and quality. The evaluation of scientific workflows proved the benefit from deploying workflows on a multi-Cloud compared to a single Cloud in reducing the user payment and improving the service quality, and identified the need for clustering and enhanced data management and scheduling policies.

Research question 3 focuses on the problem of composite services selection on multi-Cloud with the goal of maximizing the benefit for the user. This question is initially addressed in Chapter 5 by formalizing the selection problem using a graph-based, mathematical model. Then, the problem is tackled using a utility-based matching algorithm adopted from the multi-attribute auction theory. This matching policy selects the Clouds that maximize a quasi-linear utility-function calculated based on the user SLA preferences and his payment willingness. The latter is calculated based on the connectivity, size, and cost of data traffic between the single services forming the composition. The algorithm is able to find the optimal computing and storage services with respect to the user functional and non-functional SLAs, including availability, response time, throughput, latency and budget. To improve the time performance of the algorithm, particularly, with large service compositions, we introduced a hybrid utility-based genetic algorithm, called HU-GA, which uses the utility as fitness function. For a comparative study with the utility-based algorithm, we implemented a simple matching policy called sieving, which randomly selects the Cloud providers satisfying all SLA requirements of the single services without considering the latency and data traffic. The evaluation of the utility-based algorithm for matching single services with a DCS use-case with different customer types demonstrated its cost efficiency and good matching performance when compared to sieving. We evaluated the matching of composite services using the HU-GA matching policy with a real DNA sequencing workflow application. The experimental results showed the benefits of HU-GA matching compared to sieving in reducing the total execution costs as well improving the QoS, in particular when running the workflow on top of a large service composition. Also, the comparison with a prospect-based selection algorithm with a CAD-aaS case study approved the matching efficiency of our proposed HU-GA matching algorithm.

Focusing on the optimal deployment of data-intensive, large-scale applications on multi-Cloud with the example of scientific workflows, Research question 4 is addressed in Chapter 6 by proposing a two-stage multi-dimensional resource allocation scheme. In the first phase, the scheme applies our proposed HU-GA matching algorithm to select the target Clouds based on the user SLA requirements and bud-

get. In the second phase, a data locality-driven scheduler distributes the application workloads to the previously selected Clouds at runtime. For this purpose, two data locality-driven, greedy-based schedulers called DAT and DAS have been proposed. Furthermore, we proposed a replica-based data management policy to reduce the Cloud-to-Cloud data transfers during the execution. Using the simulation testbed, we evaluated the proposed scheme with a real data-intensive astronomical workflow application in different deployment scenarios. From our experiments, we demonstrated the benefits from combining utility-based matching and data locality-driven scheduling in reducing the amount of Intercloud data traffic (up to 50% ) and the total execution costs (up to 25% ) as well improving the workflow makespan (up to 20%).

Overall, this work distinguishes itself from existing research achievements with the following unique contributions:

1. A scalable generic multi-Cloud service broker framework
2. An efficient and cost-saving utility-based matching algorithm for selecting composite Cloud services
3. Effective data locality-driven scheduling and data management policies for deploying multi-Cloud workflows
4. Realistic scenarios and modeled Cloud environment for the simulation-based evaluation

## 8.2. Constraints and Future Work

In this section, we discuss the constraints of the research contributions achieved throughout this thesis, as stated in Section 1.3. In addition, future work directly linked to the recognized open issues is presented.

As previously mentioned in Section 5.1, in our matching problem formulation we assumed that the data traffic is equally distributed between the single components forming a service composition. Although this assumption relieves users of the burden of detailing traffic behavior, it is not applicable for different kinds of applications. Since the data traffic size affects the accuracy of the cost calculation and consequently the matching results, it is necessary to look for alternative methods to more accurately predict the distribution of data traffic within a composite service. Therefore, the analysis of data traffic distribution for different multi-Cloud applications is subject for future research.

The evaluation of our proposed HU-GA matching algorithm presented in Chapter 7 has been done on a commodity computer hardware with a composite service consisting of maximally 50 VMs. Although we were able to evaluate the performance

## 8. Conclusion and Outlook

of the algorithm compared to sieving with this simulation setup, it would also be worthwhile to evaluate its performance with larger service compositions. A possible extension is to implement a parallel version of the algorithm running on faster hardware. Moreover, the genetic algorithm in HU-GA is implemented with a single objective, that is, maximizing the user utility. To study the effect of using more than one objective on the matching results, multi-objective evolutionary algorithms such as NSGA-II [36] and SPEA-2 [175] would be exploited.

In Chapter 7, the multi-dimensional resource allocation scheme has been evaluated using a real data-intensive scientific workflow application. Herewith, we were able to prove the performance benefits from using the proposed scheduling and data management policies. Since the workflow structure impacts the distribution of tasks in the scheduler and the number of data transfers, it would be interesting to observe its effect on the performance and deployment cost. Therefore, it would be of great importance to evaluate this scheme with popular data-processing workflow applications like MapReduce.

The proposed ontologies describing the service requests and provider offerings presented in Chapter 4 served in this thesis to represent the data stored in the broker service and provider repositories, which are consumed for performing the matching. A potential future work would be to exploit the ontologies for selecting the candidate Clouds that fulfill the functional requirements in the SLA discovery phase. In a second phase, the utility-based matching can be performed to match the non-functional requirements. To perform the ontology-based discovery through semantic match-making, such as the work in [32], we could use existing ontology languages, like WSMO, and semantic rule languages, like SWRL. An interesting feature of these tools is the possibility to dynamically update and extend the ontologies at the runtime.

As previously mentioned in Chapter 4, the SLA-based matching algorithms presented in this thesis assume that the QoS parameters and pricing policies are under control of the Clouds and are fixed during the selection phase. In order to allow the mutual negotiation of the SLA parameters between the broker and the Clouds, we plan to explore agent-based automated negotiation strategies such as the time- and trade-off-based negotiation [49]. Herewith, it would be possible for the broker to adjust the negotiated SLA parameters based on the customer type or the requested service.

Our prototype implementation of the broker based on the CloudSim toolkit presented in Chapter 4 allows comprehensive evaluations, which are not possible on current real Clouds. Although the conducted simulation experiments are made as realistic as possible, several reasonable extensions and shortcomings are recognized. Firstly, in our simulation setup, we consider the Cloud-to-Cloud bandwidth and latency as constant metrics. Since these two QoS parameters are considered in the matching as well as in the data transfer policy, more accurate network models need to be integrated in the simulation to achieve more realistic results. Secondly, our assumption that the QoS metrics (i.e., availability, throughput, and response time) of the modeled Clouds remain constant during the simulation differs from real Cloud deployment.

Since the QoS metrics of real Clouds change frequently, the collection of the QoS metrics from third-party monitoring tools needs to be automated in order to support dynamic QoS parameters in the simulation. Furthermore, in our simulation setup we modeled datacenters made up of 50 physical hosts characterized by a constant computing capacity. However, the computing capacity of the real Clouds is dynamic and is managed by provider-specific host allocation policies. Thus, it is unlimited from a user perspective and the Clouds rarely become overloaded. A possible future work could be to investigate the resource allocation also from a provider perspective and its effect on the datacenter load.

Due to the integration of OCCI as standard interface for the interaction between the broker and the Clouds, a future deployment of the broker framework on real production Clouds requires only minimal changes. Nevertheless, it is worth mentioning that the simulation environment differs from the deployment on real Cloud infrastructure in the following ways:

- Real Cloud SLAs are mostly limited to guarantee and monitor only the availability as QoS metric. Consequently, the support for other SLA metrics, like response time, is still missing. Therefore, a real deployment of the broker should rely on third-party monitoring services to gather the missing QoS metrics.
- We use a common Intercloud gateway implemented based on OCCI to access and manage the modeled Clouds. Unfortunately, most current public Clouds do not provide OCCI-based standardized interfaces to their customers. Therefore, the use of OCCI-based adapters to access commercial Clouds is a subject of further developments.
- Unlike our assumption in the simulation setup that each modeled VM type has the same hardware configuration, real Clouds provide VMs based on different hardware and virtualization technologies. Thus, the impact of these factors on the VM performance needs to be taken into account.
- We use "pay-as-you-go" pricing policies in the costs calculation. Since some commercial Clouds offer discounted prices for long-term or short-term use periods, like the Amazon spot-instance pricing model, the broker framework will be adapted to support different pricing policies.

## 8.3. Complementary Research

This section presents research challenges in the area of multi-Cloud service brokering that are beyond the scope of this thesis. In the following, we discuss these challenges and identify several possible research directions:

## 8. Conclusion and Outlook

- **Fault-tolerant brokering:** Due to the dynamic change of Cloud offerings, pricing policies, and SLA requirements at the runtime, the broker should detect any SLA violations within a reasonable amount of time. Also, hardware or network failures on the Cloud directly impact the availability and performance of the deployed user service and consequently cause SLA violations. The support for fault tolerance allows a broker to react to these violations by adapting the matching partially or completely (e.g., trigger a redeployment, scale up/down resources) [121] and enforcing the SLA. A resulting challenge for the broker is how to make the right decision at the right time. For solving this issue, one could use methods of autonomic computing like MAPE-K loops [78]. Therefore, suitable adaptation rules should be defined (e.g., price change, performance degradation) and maintained in a knowledge database. A strategic decision for the brokerage is to coordinate the matching adaptation with or without user involvement.
- **Federated brokering:** A real deployment of a multi-Cloud broker may span only the Clouds located at a specific geographic area or domain due to technical or regulatory restrictions. Moreover, centralized, single-broker architectures raise availability concerns caused by a single point of failures. In order to permit the brokering of multiple Clouds across different geographical locations or domains, peer-to-peer or federated broker approaches become feasible in the future. With this federation concept, brokers are able to share knowledge about their own domains and coordinate the monitoring and management tasks. For enabling the communication between the brokers themselves, one could use reliable and scalable messaging protocols, such as XMPP, which is currently used to manage and monitor large Grid infrastructures like the European Grid Initiative [43].
- **Enhanced matchmaking:** In addition to price and QoS considerations, the matchmaking policy could be enhanced to consider the details about the network topology connecting the participating Clouds, since it heavily influences the data transfer performance. In addition, to use the Clouds reputation as a decision criteria, the broker should maintain a history database for rating the Clouds. Moreover, the Cloud provider specific constraints (e.g. admission control, resource limitations, or energy-management policies) could play a role in decision-making. Finally, one could implement methods for predicting the application needs in terms of computing resources (e.g., type and number of VM instances) to meet a predefined deadline. This method would be consumed by the matching policy for performing the selection.
- **Social, economic, and legal aspects:** In order to achieve a better market success, the legal entity offering multi-Cloud brokering solutions should take into consideration the social, economic, and legal aspects. For this purpose, concepts for ensuring the traceability of brokerage decisions, competition among providers, and social trust need to be investigated. Furthermore, the social acceptance of



the broker and its non-discriminative decision-making would be subjects of future studies.

- **Identity management:** Security aspects are still a big concern hindering the adoption of Cloud computing. An open challenge in the context of broker-based multi-Cloud access is how to implement the single-sign-on and identity management across heterogeneous Clouds to unburden users from maintaining different identities while accessing Clouds. We believe that the federated identity concepts would allow the decoupling of authorization and authentication and their centralization through a broker. For supporting federated identity in the broker, one could use the current open standards like OpenID, SAML, and OAuth, which are often successfully applied in the context of social media applications today.



# Appendix



# A. Appendix 1

## A.1. SLA Metrics of the Modeled Public Clouds

Table A.1 shows the SLA metrics of the public Clouds collected from CloudHarmony, which we used in most of our simulation experiments. These data are dependent on the location of the measurement as well as the network connection and time. The metrics shown are the average availability and response time of an entire month and the average throughput (for 5 Megabyte file download) of a single week. They were acquired in Germany from the same client host. In all our conducted experiments we suppose that all the customers are located in Germany.

Table A.1.: Modeled datacenters SLA metrics measured from Germany.

Datacenter	Location	Availability (%)	Response Time (s)	Throughput (Mbit/s)
EC2 EU	Ireland	99.97	3.63	19.11
EC2 US	Virginia	99.98	8.59	2.39
EC2 JP	Tokio	99.91	21.19	2.73
ElasticHosts US	Texas	99.96	12.12	1.41
ElasticHosts EU	England	99.99	2.87	15.42
GoGrid US	Virginia	100	8.35	3.13
GoGrid EU	Netherland	99.96	1.45	48.52
Rackspace EU	England	99.96	2.73	11.28
RackSpace US	Texas	99.96	11.32	1.76
CloudSigma EU	Switzerland	99.95	2.69	29.2
CloudSigma US	Nevada	99.93	12.58	1.58
VoxCLOUD US	New York	99.93	8.27	1.96
VoxCLOUD SG	Singapore	99.93	24.65	0.63
VoxCLOUD EU	Netherland	99.93	4.76	36.05
OpSource EU	Netherland	99.98	2.91	27.5
OpSource AU	Australia	99.95	28.89	0.85
OpSource US	Virginia	99.96	8.64	2.47
CityCloud EU	Sweden	99.93	4.18	23.13
HP Cloud US	Nevada	100	9.08	1.34
Flexiscale EU	England	99.5	4.04	24.51



## B. Appendix 2

### B.1. HU-GA Matching Algorithm Convergence

In the Epigenomics workflow deployment simulation experiment described in Section 7.3.2, we observed a steady linear increase of the maximal achieved utility by increasing the number of requested VMs in the composite service at the convergence. This can be seen in Figure B.1 for the “constrained” and “unconstrained” deployment scenarios.

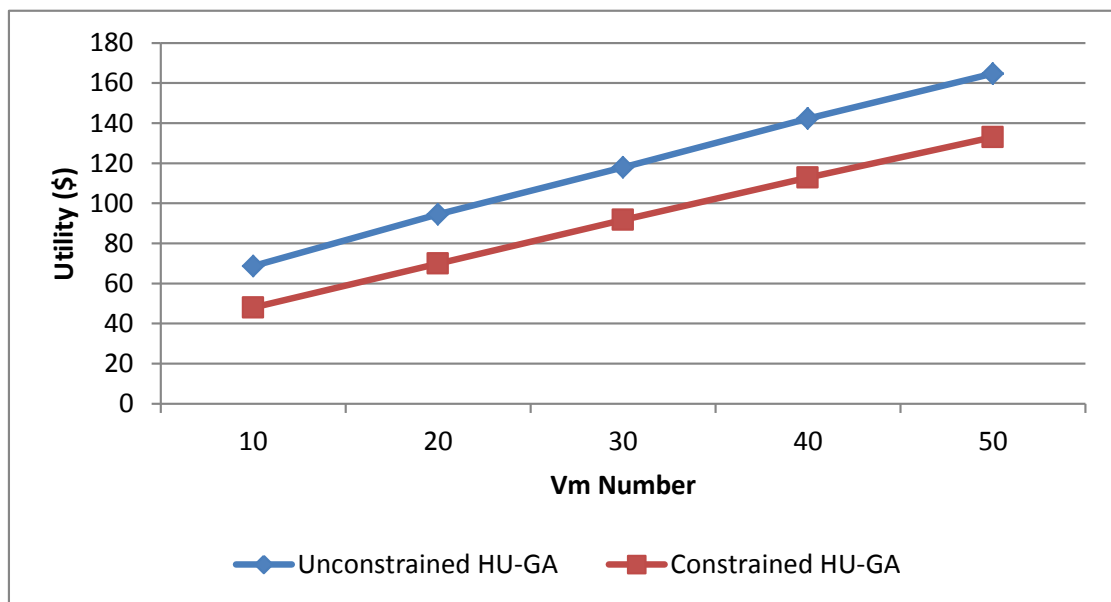


Figure B.1.: Maximal utility values at the HU-GA convergence for different VM numbers.

Figure B.2 shows a screen shot from the Opt4J convergence plot for the “constrained” deployment scenario with 30 VMs with (left) and without sieving (right). In the later case the candidate composite services with negative utilities (blue line) are also evaluated; consequently more, iterations are needed for the convergence.

## B. Appendix 2

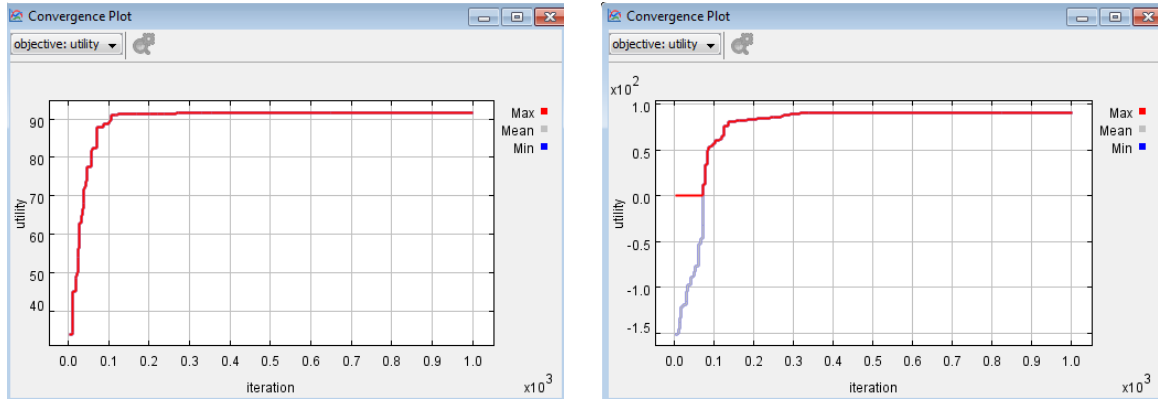


Figure B.2.: Opt4J convergence plot with 30 VMs for constrained HU-GA sieved (left) and unsieved (right).

## B.2. Matched Datacenters

Table B.1 shows the matched datacenters with HU-GA algorithm for the “constrained” and “unconstrained” scenarios to deploy the requested VMs and Cloud storage according to the simulation scenario detailed in Section 7.3.2. The table shows that all the selected datacenters are located in Europe due to their closeness to the user. The most matched datacenter is CityCloud EU, as it offers the cheapest prices combined with good SLA metric values. As storage Cloud, VoxCloud EU is the most matched followed by CloudSigma EU. It can be also seen that for the “constrained” deployment, the VMs are equally matched between Amazon EC2 EU and CityCloud EU. The candidate datacenters matched by the Sieving algorithm are CityCloud EU, Amazon EC2 EU, Flexiscale EU, CloudSigma EU, and VoXCloud EU.

Table B.1.: Matched datacenters with unconstrained (U) and constrained (C) HU-GA for different VM numbers; ST=storage.

VMs number	10		20		30		40		50	
Scenario	U	C	U	C	U	C	U	C	U	C
CityCloud EU	10	5	20	10	30	15	40	20	50	25
Amazon EC2 EU	0	0	0	10	0	15	0	20	0	25
CloudSigma EU	0	5	0	1ST	0	0	0	0	0	0
VoxCloud EU	1ST	1ST	1ST	0	1ST	1ST	1ST	1ST	1ST	1ST



## C. Appendix 3

### C.1. OCCI Monitoring Extensions

A status action is not yet included in the OCCI core specification, but it is required for querying and updating the status of each compute instance running on the Cloud platform. Therefore, we extended the OCCI compute resource by adding a status action (`/infrastructure/compute/action#status`). The compute instance state, as specified in OCCI, can be active, inactive, suspended, or failed. Furthermore, since the current OCCI specification does not support the monitoring of the Cloud SLA metrics, we implemented two new Mixin classes called *offertemplate* and *resourceinformation* by extending the OCCI Mixin interface. The *resourceinformation* Mixin is used to query the current providers resource information and QoS metrics, while the *offertemplate* Mixin is used to query the providers supported service Offers. For each provider exactly one *resourceinformation* Mixin and one or more *offertemplate* Mixin needs to be initialized at the simulation start. The attribute values of the Mixins are updated during the simulation by the monitoring manager and can be queried from OCCI Client. The list of queryable attributes of each Mixin can be seen below:

```
1 Scheme: "http://schemas.ogf.org/occi/monitoring#"
2 Term: resourceinformation
3 Attributes:
4 occi.monitoring.providername, //CloudSim datacenter name
5 occi.monitoring.cost, // cost per CPU core
6 occi.monitoring.os, // Linux or windows
7 occi.monitoring.arch, // 32 or 64 bit
8 occi.monitoring.region, // provider location
9 occi.monitoring.freecores, //overall free CPU capacity
10 occi.monitoring.usedcores, //overall free CPU capacity
11 occi.monitoring.maxcpuspeed, // in MHZ
12 occi.monitoring.availability, // in %
13 occi.monitoring.responsetime, // in s
14 occi.monitoring.throughput // in Mbit/s
```

Source code C.1: Resourceinformation OCCI Mixin

### C. Appendix 3

```
1 Scheme: "http://schemas.ogf.org/occi/sla#"
2 Term: offertemplate
3 Attributes:
4   occi.sla.providertype, // CloudSim datacenter name
5   occi.sla.offertype, // VM small, medium, large, etc. or
   storage
6   occi.sla.cost, // usage cost \$/hour
7   occi.sla.status // available or not available
```

Source code C.2: Offertemplate OCCI Mixin

## D. Appendix 4

### D.1. Sample Simulation Output

This screen shot shows the output of the simulation in which the user requests a micro VM deployment. This request has been assigned the ID 49 and the datacenter DC.B have been selected by the broker for provisioning the VM. When the VM is deployed, a unique service Uniform Resource Identifier (URI) is outputted to the user by the corresponding Intercloud gateway to manage the service at runtime. This URI is mapped internally inside the datacenter to a local VM ID.

```
1 1681.0 ServiceRequestGenerator: New Request #49
2     Os: Linux, Arch: x64, Cores: 1, CPU: 0.5 GHZ, Memory:
3     0.633 GB,
4     Hostname: Host_49, Storage: 2.5GB, Region: 4, Cost: 0.02
5 1681.0 MatchMaker: Matching best Provider for Service Request
6     #49
7 1681.0 MatchMaker: matched Provider for Service Request #49 is
8     DC_B
9 1690.0 MonitoringManager: Updating state of Request #45
10 1690.0 IBM-Gateway: Request State of VM #2006
11 1691.0 DeploymentManager: Deploying Service Request #49 in DC.B
12 1691.0 DC.B-Gateway: VM creation Request with ID #1011
13 1691.0 DC.B-Gateway: Trying to Create new VM #1011 in DC.B
14 1691.0 DeploymentManager: Service Request #49 received
15 1691.0     Service URI: 51d7e4ad-29ab-4926-8d8a-8c6ba6df11f1
16 1691.0 DC.B-Gateway: VM #1011 has been created in DC.B, Host
17     #11
18 1701.0 MonitoringManager: Updating state of Request #49
19 1701.0 DC.B-Gateway: Request State of VM #1011
20 1737.0 DeploymentManager: Starting Service #49
21 Simulation completed.
```

Source code D.1: Sample Simulation Output



# Acronyms

<b>AHP</b>	Analytic Hierarchy Process
<b>API</b>	Application Programming Interface
<b>AWS</b>	Amazon Webs Services
<b>CAD</b>	Computer Aided Design
<b>CBR</b>	Cost-Budget-Ratio
<b>CU</b>	Compute Unit
<b>DAG</b>	Directed Acyclic Graph
<b>DAS</b>	Data Aware Size-based Scheduler
<b>DAT</b>	Data Aware Time-based Scheduler
<b>DC</b>	Datacenter
<b>DCS</b>	Desktop Cloud Service
<b>EC2</b>	Elastic Compute Services
<b>EU</b>	Europa
<b>GA</b>	Genetic Algorithm
<b>GUI</b>	Graphic User Interface
<b>HPC</b>	High Performance Computing
<b>HTTP</b>	Hyper Text Transfer Protocol
<b>HU-GA</b>	Hybrid Utility-based Genetic Algorithm
<b>IaaS</b>	Infrastructure as a Service
<b>ID</b>	Identifier
<b>I/O</b>	Input/Output
<b>MAUT</b>	Multi-Attribute Utility Theory
<b>MCDM</b>	Multi-Criteria Decision-Making
<b>OCCI</b>	Open Cloud Computing Interface

## *Acronyms*

- OS** Operating System
- PaaS** Platform as a Service
- QoS** Quality of Service
- REST** Representational State Transfer
- S3** Simple Storage System
- SaaS** Software as a Service
- SAN** Storage Area Network
- SLA** Service Level Agreement
- SME** Small and Medium Enterprises
- SOA** Service Oriented Architecture
- SOAP** Simple Object Access Protocol
- UI** User Interface
- URI** Uniform Resource Identifier
- US** United States
- VM** Virtual Machine
- WfMS** Workflow Management System
- WWW** World Wide Web
- XML** Extensible Markup Language

# Bibliography

- [1] 2MASS. 2MASS at IPAC. [Online], 2014. <http://www.ipac.caltech.edu/2mass/> (accessed:2014-03-20).
- [2] Lifeng Ai, Maolin Tang, and Colin J. Fidge. Partitioning composite web services for decentralized execution using a genetic algorithm. *Future Generation Comp. Syst.*, 27(2):157–172, 2011.
- [3] G. A. Akerlof. The market for lemons: Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, 1970.
- [4] Mohammad Alrifai, Thomas Risse, Peter Dolog, and Wolfgang Nejdl. A scalable approach for qos-based web service selection. In George Feuerlicht and Winfried Lamersdorf, editors, *Service-Oriented Computing ICSOC 2008 Workshops*, volume 5472 of *Lecture Notes in Computer Science*, pages 190–199. Springer Berlin Heidelberg, 2009.
- [5] Amazon Elastic Compute Cloud. [Online], 2014. <http://aws.amazon.com/ec2/> (accessed: 2014-03-20).
- [6] Amazon Simple Storage Service. [Online], 2014. <http://aws.amazon.com/s3/> (accessed: 2014-03-20).
- [7] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D’Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan. ModacLOUDS: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 50–56, June 2012.
- [8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, University of California at Berkeley, February 2009.
- [9] John Asker and Estelle Cantillon. Properties of scoring auctions. *The RAND Journal of Economics*, 39(1):69–85, 2008.
- [10] Microsoft Azure Platform. [Online], 2014. <http://www.microsoft.com/windowsazure/> (accessed: 2014-03-20).

## Bibliography

- [11] D. Berardi, D. Calvanese, G. De Giacomo and M. Lenzerini, and M. Mecella. Automatic Services Composition based on Behavioral Descriptions. *International Journal of Cooperative Information Systems (IJCIS)*, 14(4):333-376, 2005.
- [12] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. Heuristics for qos-aware web service composition. In *Web Services, 2006. ICWS '06. International Conference on*, pages 72–82, Sept 2006.
- [13] David Bernstein, Erik Ludvigson, Krishna Sankar, Steve Diamond, and Monique Morrow. Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In Mark Perry, Hideyasu Sasaki, Matthias Ehmann, Guadalupe Ortiz Bellot, and Oana Dini, editors, *ICIW*, pages 328–336. IEEE Computer Society, 2009.
- [14] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers Conference*, volume 5493 of *SPIE 04*, pages 221–232, September 2004.
- [15] Martin Bichler. An experimental analysis of multi-attribute auctions. *Decis. Support Syst.*, 29(3):249–268, October 2000.
- [16] Benjamin Blau. *Coordination in Service Value Networks—A Mechanism Design Approach*. phdthesis, University of Karlsruhe, 2009.
- [17] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. *Market-Oriented Cloud Computing and The Cloudbus Toolkit*, pages 319–358. John Wiley, Inc, 2013.
- [18] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I, ICA3PP'10*, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.
- [19] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009.
- [20] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.
- [21] Rodrigo N. Calheiros, Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya. The Aneka platform and QoS-driven resource provisioning



- for elastic applications on hybrid Clouds. *Future Generation Computer Systems*, 28(6):861–870, June 2012.
- [22] R.J. Cassady. *Auctions and Auctioneering*. University of California Press, 1967.
- [23] CCUC-DG. Cloud Computing Use Cases White Paper Version 4.0. Technical report, Cloud Computing Use Case Discussion Group, 2010.
- [24] Weiwei Chen and Ewa Deelman. WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments. In *Proceedings of the 8th IEEE International Conference on eScience*, Chicago, USA, October 2012.
- [25] CityCloud. [Online], 2014. <http://www.citycloud.eu> (accessed: 2014-03-20).
- [26] K.P. Clark, M. Warnier, and F. M.T. Brazier. An intelligent Cloud Resource Allocation Service. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER2012)*, Porto, Portugal, 2012.
- [27] CloudHarmony. Network Tests. [Online], 2014. <http://www.cloudharmony.com> (accessed: 2014-03-20).
- [28] CloudSigma. [Online], 2014. <http://www.cloudsigma.com/> (accessed: 2014-03-20).
- [29] Cloudswitch. [Online], 2014. <http://www.cloudswitch.com/> (accessed: 2014-03-20).
- [30] Condor Middleware. [Online], 2014. <http://research.cs.wisc.edu/condor/> (accessed: 2014-03-20).
- [31] Fabricio F. Costa. Big data in biomedicine. *Drug Discovery Today*, -(0):-, 2013.
- [32] Amir Vahid Dastjerdi, Saurabh Kumar Garg, and Rajkumar Buyya. QoS-aware Deployment of Network of Virtual Appliances Across Multiple Clouds. *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 415–423, November 2011.
- [33] AmirVahid Dastjerdi and Rajkumar Buyya. *A Taxonomy of QoS Management and Service Selection Methodologies for Cloud Computing*, pages 109–131. CRC Press, October 2011.
- [34] Daniel de Oliveira, Eduardo Ogasawara, Fernanda Baião, and Marta Mattoso. SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing*, CLOUD '10, pages 378–385, Washington, DC, USA, 2010.

## Bibliography

- [35] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
- [36] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.
- [37] Ewa Deelman, Gideon Juve, and G. Bruce Berriman. Using Clouds For Science, is it Just Kicking The Can Down the Road? In *Proceedings of the International Conference on Cloud Computing and Services Science, CLOSER 2012*, pages 127–133, Porto, Portugal, April 2012.
- [38] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08*, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.
- [39] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
- [40] Bastian Demuth, Schuller Bernd, Holl Sonja, Milad Daivandy Jason, Giesler André, Huber Valentina, and Sild Sulev. The UNICORE Rich Client: Facilitating the Automated Execution of Scientific Workflows. In *Proceedings of 6th IEEE International Conference on eScience*, pages 238–245, 2010.
- [41] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- [42] I. Egambaram, G. Vadivelou, and P. Sivasubramanian. Qos based Web Service Selection. *Journal of Ubiquitous Computing and Communication*, 2012. available at [http://www.ubicc.org/files/pdf/PaperID20460\\_460.pdf](http://www.ubicc.org/files/pdf/PaperID20460_460.pdf).
- [43] EGI. European Grid Initiative. [Online], 2014. <http://www.egi.eu> (accessed: 2014-03-20).
- [44] ElasticHosts. [Online]. <http://www.elastichosts.com> (accessed: 2014-03-20).
- [45] Vincent Chimaobi Emeakaroha. *Managing Cloud Service Provisioning and SLA Enforcement via Holistic Monitoring Techniques*. phdthesis, TU Vienna, Austria, April 2012.
- [46] USC Epigenome Center. [Online], 2014. <http://epigenome.usc.edu> (accessed: 2014-03-20).

- [47] Zhao Er-Dun, Qi Yong-Qiang, Xiang Xing-Xing, and Chen Yi. A data placement strategy based on genetic algorithm for scientific workflows. In *Computational Intelligence and Security (CIS), 2012 Eighth International Conference on*, pages 146–149, 2012.
- [48] T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, and M. Wiczorek. Askalon: a grid application development and computing environment. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 10 pp.–, Nov 2005.
- [49] P. Faratin, C. Sierra, and N.R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence*, 142(2):205 – 237, 2002. International Conference on MultiAgent Systems 2000.
- [50] Hamid Mohammadi Fard, Radu Prodan, Juan Jose Durillo Barrionuevo, and Thomas Fahringer. A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments. *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 300–309, May 2012.
- [51] Hamid Mohammadi Fard, Radu Prodan, and Thomas Fahringer. A Truthful Dynamic Workflow Scheduling Mechanism for Commercial Multicloud Environments. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1203–1212, June 2013.
- [52] Soodeh Farokhi, Foued Jrad, Ivona Brandic, and Achim Streit. Hs4mc: Hierarchical sla-based service selection for multi-cloud environments. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2014)*, in press, Barcelona, Spain, 2014.
- [53] Ana Juan Ferrer, Francisco Hernandez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Ral Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Sriji K. Nair, George Kousiouris, Kleopatra Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forg, Tabassum Sharif, and Craig Sheridan. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.
- [54] Jos Figueira, Vincent Mousseau, and Bernard Roy. Electre methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, volume 78 of *International Series in Operations Research and Management Science*, pages 133–153. Springer New York, 2005.
- [55] FlexiScale: Utility Computing On Demand. [Online], 2014. <http://www.flexiscale.com/> (accessed: 2014-03-20).

## Bibliography

- [56] Global Inter-Cloud Technology Forum. Use Cases and Functional Requirements for Inter-Cloud Computing. [Online], 2010. [http://www.gictf.jp/doc/GICTF\\_Whitepaper\\_20100809.pdf](http://www.gictf.jp/doc/GICTF_Whitepaper_20100809.pdf) (accessed: 2013-12-20).
- [57] Tm Forum. Tm Forum. [Online], 2014. <https://www.tmforum.org/> (accessed: 2014-03-20).
- [58] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.
- [59] R.F. Freund, Michael Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, F. Mirabile, L. Moore, B. Rust, and H.J. Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, pages 184–199, 1998.
- [60] D. Friedman and J. Rust. *The Double Auction Market: Institutions, Theories, and Evidence*. Proceedings volume in the Santa Fe Institute studies in the sciences of complexity. Addison-Wesley Publishing Company, 1993.
- [61] B. Furht and A. Escalante, editors. *Handbook of Cloud Computing*. Springer, 2010.
- [62] Google App Engine. [Online], 2014. <http://code.google.com/appengine/> (accessed: 2014-03-20).
- [63] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. A framework for ranking of cloud computing services. *Future Gener. Comput. Syst.*, 29(4):1012–1023, June 2013.
- [64] S.K. Garg and R. Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 105–113, Dec 2011.
- [65] Inc Gartner. Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services. [Online], 2009. <http://www.gartner.com/newsroom/id/1064712> (accessed: 2013-12-20).
- [66] gLite WMS. [Online], 2014. <http://web.infn.it/gLiteWMS/> (accessed: 2014-03-20).
- [67] GoGrid. [Online], 2014. <http://www.gogrid.com> (accessed: 2014-03-20).
- [68] Google Docs. [Online], 2014. <http://docs.google.com/> (accessed: 2014-03-20).
- [69] Gridbus Project. Gridbus service broker. [Online], 2014. <http://www.cloudbus.org/broker/> (accessed: 2014-03-20).

- [70] Nikolay Grozev and Rajkumar Buyya. *Inter-Cloud architectures and application brokering: taxonomy and survey*, pages 1–22. John Wiley, Inc, 2012.
- [71] Nikolay Grozev and Rajkumar Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.
- [72] J. Octavio Gutierrez-Garcia and Kwang Mong Sim. Agent-based Cloud Workflow Execution. *Integrated Computer-Aided Engineering*, 19:39–56, 2012.
- [73] Steffen Haak and Michael Menzel. Autonomic benchmarking for cloud infrastructures: an economic optimization model. In *Proceedings of the 1st ACM/IEEE workshop on Autonomic computing in economics*, pages 27–32. ACM, 2011.
- [74] Hadoop. Hadoop. [Online], 2014. <http://hadoop.apache.org/> (accessed: 2014-03-20).
- [75] Jens Happe, Wolfgang Theilmann, Andrew Edmonds, and Keven T. Kearney. A reference architecture for multi-level sla management. In Philipp Wieder, Joe M. Butler, Wolfgang Theilmann, and Ramin Yahyapour, editors, *Service Level Agreements for Cloud Computing*, pages 13–26. Springer New York, 2011.
- [76] Marcus Hardt, Thomas Jejkal, Isabel Campos, Enol Fernandez, Adrian Jackson, Daniel Nielsson, Bartek Palak, and Marcin Plociennik. Transparent Access to Scientific and Commercial Clouds from the Kepler Workflow Engine. *Computing and Informatics*, 31:1001–1015, 2012.
- [77] HP Cloud. [Online], 2014. <http://www.hpcloud.com/> (accessed: 2014-03-20).
- [78] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28, August 2008.
- [79] IBM Services Architecture team. Web Services architecture overview. [Online], 2014. <http://www.ibm.com/developerworks/webservices/library/w-ovr/> (accessed: 2014-03-20).
- [80] IBM PowerVM. [Online], 2014. <http://www.redbooks.ibm.com/abstracts/sg247940.html> (accessed: 2014-03-20).
- [81] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H.J. Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672 – 686, 2008.
- [82] G.N. Iyer, R. Chandrasekaran, and B. Veeravalli. Auction-based vs. incentive-based multiple-cloud orchestration mechanisms. In *Communication, Networks and Satellite (ComNetSat), 2012 IEEE International Conference on*, pages 1–5, July 2012.

## Bibliography

- [83] Jamcracker. [Online], 2014. <http://www.jamcracker.com/> (accessed: 2014-03-20).
- [84] Jclouds library. [Online], 2014. <http://jclouds.apache.org/> (accessed: 2014-03-20).
- [85] Jiahui Jin, Junzhou Luo, Aibo Song, Fang Dong, and Runqun Xiong. Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 295–304, 2011.
- [86] Foued Jrad, Jie Tao, Ivona Brandic, and Achim Streit. Multi-dimensional resource allocation for data-intensive large-scale cloud applications. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2014), in press*, Barcelona, Spain, 2014.
- [87] Foued Jrad, Jie Tao, Ivona Brandic, and Achim Streit. SLA Enactment for Large-scale Healthcare Workflows on Multi-Cloud. *Future Generation Computer Systems*, 2014.
- [88] Foued Jrad, Jie Tao, Rico Knapper, Christoph M. Flath, and Achim Streit. A utility-based approach for customised cloud service selection. *Int. J. Computational Science and Engineering*, 2013.
- [89] Foued Jrad, Jie Tao, and Achim Streit. Simulation-based evaluation of an intercloud service broker. In *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 140–145, 2012.
- [90] Foued Jrad, Jie Tao, and Achim Streit. Sla based service brokering in intercloud environments. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, pages 76–81. SciTePress, 2012.
- [91] Foued Jrad, Jie Tao, and Achim Streit. A broker-based framework for multi-cloud workflows. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 61–68. ACM, 2013.
- [92] Adrian Juan-Verdejo and Henning Baars. Decision support for partially moving applications to the cloud: The example of business intelligence. In *Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services, HotTopiCS '13*, pages 35–42, New York, NY, USA, 2013. ACM.
- [93] Gideon Juve and Ewa Deelman. Automating Application Deployment in Infrastructure Clouds. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11*, pages 658–665, Washington, DC, USA, 2011. IEEE Computer Society.
- [94] Gideon Juve and Ewa Deelman. *Scientific Workflows in the Cloud*, pages 71–91. Computer Communications and Networks. Springer London, 2011.

- [95] Gideon Juve, Ewa Deelman, G. Bruce Berriman, Benjamin P. Berman, and Philip Maechling. An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2. *Journal of Grid Computing*, 10:5–21, 2012.
- [96] Gideon M. Juve. *Resource management for scientific workflows*. phdthesis, University of Southern California, May 2012.
- [97] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, pages 263–291, 1979.
- [98] K. Keahey and T. Freeman. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In *Proceedings of the First Workshop on Cloud Computing and its Applications*, October 2008.
- [99] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Wiley series in probability and mathematical statistics. Applied probability and statistics. Cambridge University Press, 1993.
- [100] John Kennedy, Andrew Edmonds, Victor Bayon, Pat Cheevers, Kuan Lu, Miha Stopar, and Damjan Murn. Sla-enabled infrastructure management. In Philipp Wieder, Joe M. Butler, Wolfgang Theilmann, and Ramin Yahyapour, editors, *Service Level Agreements for Cloud Computing*, pages 271–287. Springer New York, 2011.
- [101] A. Kertesz, G. Kecskemeti, and I. Brandic. Autonomic sla-aware service virtualization for distributed systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 503–510, Feb 2011.
- [102] D. Kliazovich, P. Bouvry, Y. Audzevich, and S.U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5, Dec 2010.
- [103] R. Knapper, B. Blau, T. Conte, A. Sailer, A. Kochut, and A. Mohindra. Efficient contracting in cloud service markets with asymmetric information—a screening approach. In *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*, pages 236–243. IEEE, 2011.
- [104] R. Knapper, C.M. Flath, B. Blau, A. Sailer, and C. Weinhardt. A multi-attribute service portfolio design problem. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–7, dec. 2011.
- [105] P. Köhler, A. Anandasivam, M. Dan, and C. Weinhardt. Customer Heterogeneity and Tariff Biases in Cloud Computing. In *Proceedings of the International Conference on Information Systems (ICIS), Saint Louis, USA, 2010*.

## Bibliography

- [106] S. Lamparter, S. Ankolekar, S. Grimm, and R. Studer. Preference-based Selection of Highly Configurable Web Services. In *Proc. of the 16th Int. World Wide Web Conference (WWW'07)*, pages 1013–1022, Banff, Canada, May 2007.
- [107] F. Lécué and A. Léger. A Formal Model for Semantic Web Service Composition. In *ISWC the 5th International Semantic Web Conference*, pages 385–398, November 2005.
- [108] Martin Lukasiewicz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, pages 1723–1730, Dublin, Ireland, 2011.
- [109] J.G. March. Bounded rationality, ambiguity, and the engineering of choice. *The Bell Journal of Economics*, pages 587–608, 1978.
- [110] Medigrid D-Grid Project. [Online], 2014. <http://www.medigrid.de> (accessed: 2014-03-20).
- [111] P. Mell and T. Grance. The NIST Definition of Cloud Computing. [Online], 2011. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf> (accessed: 2014-03-20).
- [112] Michael Menzel and Rajiv Ranjan. Cloudgenius: Decision support for web server cloud migration. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, New York, NY, USA, 2012.
- [113] R. Mian, P. Martin, A. Brown, and M. Zhang. Managing data-intensive workloads in a cloud. In Sandro Fiore and Giovanni Aloisio, editors, *Grid and Cloud Database Management*, pages 235–258. Springer Berlin Heidelberg, 2011.
- [114] G. Modica and O. Diand Tomarchio. A Semantic Discovery Framework to support supply-demand Matchmaking in Cloud Service Markets. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER2012)*, Porto, Portugal, 2012.
- [115] MOSAIC EU Project. [Online], 2014. <http://www.mosaic-fp7.eu/> (accessed: 2014-03-20).
- [116] Alberto Nez, JoseL. Vzquez-Poletti, AgustinC. Caminero, GabrielG. Casta, Jesus Carretero, and IgnacioM. Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.
- [117] Shahzad Nizamani, Peter Dew, and Karim Djemame. A qualityaware cloud management service for computational modellers. *International Journal of Cloud Computing*, 2(4):340–363, 2013.



- [118] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseffand, and D. Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *Proceedings of Cloud Computing and Its Applications*, October 2008. Available: <http://eucalyptus.cs.ucsb.edu/wiki/Presentations> (accessed: 2014-03-20).
- [119] Open Cloud Computing Interface specification OCCI. [Online], 2014. <http://www.occi-wg.org/> (accessed: 2014-03-20).
- [120] OCCI4JAVA JAVA-based OCCI Implementation. [Online], 2014. <https://github.com/occi4java/> (accessed: 2014-03-20).
- [121] Daniel Oliveira, Kary a. C. S. Ocaña, Fernanda Baião, and Marta Mattoso. A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds. *Journal of Grid Computing*, 10(3):521–552, August 2012.
- [122] OpenStack Cloud Software. [Online], 2014. <http://openstack.org/> (accessed: 2014-03-20).
- [123] OpSource. [Online], 2014. <http://www.opsources.net/> (accessed: 2014-03-20).
- [124] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. Groudsim: An event-based simulation framework for computational grids and clouds. In MarioR. Guarracino, Frdric Vivien, JesperLarsson Trff, Mario Cannatoro, Marco Danelutto, Anders Hast, Francesca Perla, Andreas Knpfer, Beniamino Martino, and Michael Alexander, editors, *Euro-Par 2010 Parallel Processing Workshops*, volume 6586 of *Lecture Notes in Computer Science*, pages 305–313. Springer Berlin Heidelberg, 2011.
- [125] S. Pandey, Linlin Wu, S.M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407, 2010.
- [126] Suraj Pandey. *Scheduling and management of data intensive application workflows in grid and cloud computing environments*. University of Melbourne, Department of Computer Science and Software Engineering, 2011.
- [127] Suraj Pandey, Dileban Karunamoorthy, and Rajkumar Buyya. *Workflow Engine for Clouds*, pages 321–344. John Wiley, Inc, 2011.
- [128] M. Papazoglou. *Web services: principles and technology*. Addison-Wesley, 2008.
- [129] D.F. Parkhill. *The Challenge of the Computer Utility*. Number S. 246 in *The Challenge of the Computer Utility*. Addison-Wesley Publishing Company, 1966.

## Bibliography

- [130] Dana Petcu. Consuming resources and services from multiple clouds. *Journal of Grid Computing*, pages 1–25, 2014.
- [131] David Pisinger. *Algorithms for Knapsack Problems*. phdthesis, University of Copenhagen, February 1995.
- [132] PlanetLab Project. [Online], 2014. <http://www.planet-lab.org> (accessed: 2014-03-20).
- [133] The Rackspace Open Cloud. [Online], 2014. <http://www.rackspace.com/cloud/> (accessed: 2014-03-20).
- [134] Ioan Raicu, Ian T. Foster, and Yong Zhao. Many-Task Computing for Grids and Supercomputers. In *Proceedings of the IEEE Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS 08, pages 1–11, Austin, TX, USA, November 2008.
- [135] Ioan Raicu, Ian T. Foster, Yong Zhao, Philip Little, Christopher M. Moretti, Amitabh Chaudhary, and Douglas Thain. The quest for scalable support of data-intensive workloads in distributed systems. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 207–216, New York, NY, USA, 2009. ACM.
- [136] Christoph Redl, Ivan Breskovic, Ivona Brandic, and Schahram Dustdar. Automatic sla matching and provider selection in grid and cloud computing markets. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, GRID '12, pages 85–94, Washington, DC, USA, 2012. IEEE Computer Society.
- [137] Rightscale. [Online], 2014. <http://www.rightscale.com> (accessed: 2014-03-20).
- [138] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, 53(4):535–545, July 2009.
- [139] T.L. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Advanced book program. McGraw-Hill, 1980.
- [140] M. Sathya, M. Swarnamugi, P. Dhavachelvan, and G. Sureshkumar. Evaluation of QoS Based Web- Service Selection Techniques for Service Composition. *International Journal of Software Engineering*, 1(5), 2010.
- [141] Michael C Schatz, Ben Langmead, and Steven L Salzberg. Cloud computing and the dna data race. *Nature biotechnology*, 28(7):691, 2010.

- [142] SensibleCloud. [Online], 2014. <http://www.sensiblecloud.com/> (accessed: 2014-03-20).
- [143] E. Sirin, B. Parsia, D. Wu, J.A. Hendler, and D.S. Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–96, 2004.
- [144] SLASOI FP7 EU Project. [Online], 2014. <http://sla-at-soi.eu> (accessed: 2014-03-20).
- [145] B. Sotomayor, R. Montero, I. Llorente, and I. Foster. Capacity Leasing in Cloud Systems using the OpenNebula Engine. In *The First Workshop on Cloud Computing and its Applications*, October 2008.
- [146] Tobias Sturm, Foued Jrad, and Achim Streit. Storage cloudsim: A simulation environment for cloud object storage infrastructures. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2014)*, in press, Barcelona, Spain, 2014.
- [147] Dan Sullivan. *The Definitive Guide to Cloud Acceleration*. realtime publishers, 2013.
- [148] Claudia Szabo, Quan Z. Sheng, Trent Kroeger, Yihong Zhang, and Jian Yu. Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows. *Journal of Grid Computing*, October 2013.
- [149] Metsch T., Edmonds A., and Bayon V. Using cloud standards for interoperability of cloud frameworks. Technical report, SLASOI project, April 2010.
- [150] J. Tao, D. Franz, H. Marten, and A. Streit. An Implementation Approach for Inter-Cloud Service Combination. *International Journal on Advances in Software*, 5:65–75, 2012.
- [151] H. R. Varian. Price discrimination. In R. Schmalensee and R. Willig, editors, *Handbook of Industrial Organization*, volume 1, pages 597 – 654. Elsevier, 1989.
- [152] David Villegas, Ivan Rodero, Liana Fong, Norman Bobroff, Yanbin Liu, Manish Parashar, and S.Masoud Sadjadi. The role of grid computing technologies in cloud computing. In Borko Furht and Armando Escalante, editors, *Handbook of Cloud Computing*, pages 183–218. Springer US, 2010.
- [153] J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1944.
- [154] William Voorsluys, James Broberg, and Rajkumar Buyya. *Introduction to Cloud Computing*, pages 1–41. John Wiley & Sons, Inc., 2011.
- [155] VoxCloud. [Online]. <http://www.internap.com> (accessed: 2014-03-20).

## Bibliography

- [156] H. Wang, P. Tong, and P. Thompson. QoS-Based Web Services Selection. In *Proceedings of the IEEE International Conference on e-Business Engineering*, 2007.
- [157] Jianwu Wang, Daniel Crawl, and Ilkay Altintas. Kepler + hadoop: A general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 12:1–12:8, New York, NY, USA, 2009. ACM.
- [158] L. Wang, R. Ranjan, J. Chen, and B. Benatallah. *Cloud Computing: Methodology, Systems, and Applications*. Taylor & Francis, 2011.
- [159] B. Wickremasinghe, R.N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 446–452, April 2010.
- [160] WLCG. Worldwide LHC Computing Grid. [Online], 2014. <http://glite.web.cern.ch/glite/> (accessed: 2014-03-20).
- [161] Linlin Wu and Rajkumar Buyya. Service Level Agreement (SLA) in utility computing systems. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, pages 1–20, 2011.
- [162] Linlin Wu, S.K. Garg, R. Buyya, Chao Chen, and S. Versteeg. Automated sla negotiation framework for cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 235–244, May 2013.
- [163] Zhangjun Wu, Xiao Liu, Zhiwei Ni, Dong Yuan, and Yun Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293, 2013.
- [164] Zhen Ye, Xiaofang Zhou, and Athman Bouguettaya. Genetic algorithm based qos-aware service compositions in cloud computing. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications: Part II, DASFAA'11*, pages 321–334, Berlin, Heidelberg, 2011. Springer-Verlag.
- [165] Jia Yu and Rajkumar Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *SIGMOD Rec.*, 34(3):44–49, September 2005.
- [166] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow scheduling algorithms for grid computing. In Fatos Xhafa and Ajith Abraham, editors, *Metaheuristics for Scheduling in Distributed Computing Environments*, volume 146 of *Studies in Computational Intelligence*, pages 173–214. Springer Berlin Heidelberg, 2008.

- [167] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A data placement strategy in scientific cloud workflows. *Future Gener. Comput. Syst.*, 26(8):1200–1214, October 2010.
- [168] M. Zeleny. *Multiple Criteria Decision Making*. McGraw-Hill Series in Quantitative Methods for Management. McGraw-Hill, 1982.
- [169] L. Zeng, B. Benatallah, H.H. A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.
- [170] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web, WWW '03*, pages 411–421, New York, NY, USA, 2003. ACM.
- [171] Miranda Zhang, Rajiv Ranjan, Armin Haller, Dimitrios Georgakopoulos, Michael Menzel, and Surya Nepal. An ontology-based system for cloud infrastructure services' discovery. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, pages 524–530. IEEE, 2012.
- [172] Miranda Zhang, Rajiv Ranjan, Surya Nepal, Michael Menzel, and Armin Haller. A declarative recommender system for cloud infrastructure services selection. In *Proceedings of the 9th International Conference on Economics of Grids, Clouds, Systems, and Services, GECON'12*, pages 102–113, Berlin, Heidelberg, 2012. Springer-Verlag.
- [173] Wei Zhao, Yong Peng, Feng Xie, and Zhonghua Dai. Modeling and simulation of cloud computing: A review. In *Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific*, pages 20–24, Nov 2012.
- [174] Zimory going beyond. [Online], 2014. <http://www.zimory.de/> (accessed: 2014-03-20).
- [175] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100. CIMNE, Barcelona, Spain, 2002.



# Curriculum Vitae

## Foued Jrad CV

### Personal Information

Date of birth: 05. September 1979  
Place of birth: Monastir, Tunisia  
Citizenship:: German

### Education

Since 06/2011 PhD study at the Steinbuch Centre for Computing  
Karlsruhe Institute of Technology, Germany

10/1999 to 12/2005 Study of electrical engineering  
Leibniz University of Hanover, Germany  
Diploma engineer degree in electrical engineering  
Final grade: 1.83, among the best 10%

Electives Telecommunication engineering and communication systems

Study thesis Implementation and evaluation of an indoor-positioning method  
for W-LAN using neuronal networks  
Institute of communications technology  
Leibniz University of Hanover, Germany

Diploma thesis Visualization concept and wireless interaction for an infokiosk  
Institute for Systems Engineering, System and Computer  
Architecture  
Leibniz University of Hanover, Germany

09/1998 to 07/1999 Preparatory courses for German universities  
Studienkolleg of the Leibniz University of Hanover, Germany

06/1998 Secondary School of Jemmal, Tunisia  
Ranked among third best baccalaureate graduate on technology in Tunisia

## **Work Experience**

- 04/2008 to 05/2011      Research assistant at Steinbuch Centre for Computing  
Karlsruhe Institute of Technology, Germany  
Worked on D-Grid DGI-2 and EGI EU FP7 projects  
Tasks: Grid computing, On-Call service, service monitoring,  
user helpdesk support, technical advisory board member.
- 02/2006 to 03/2008      Research assistant at Center for Computing and Communication  
RWTH Aachen, Germany  
Worked on D-Grid DGI-1 project  
Tasks: Grid computing, web portals, Cluster administration
- 07/2004 to 12/2004      Internship at Siemens AG, Information and Communication Networks,  
Munich, Germany  
Extension of a web-based CTI (Computer Telephony Integration) application
- 09/2000 to 12/2004      Worked as student assistant on several university institutes  
Leibniz University of Hanover, Germany  
Tasks: Visual C++ programming, matlab, microelectronic labor

## **Research Abroad**

- 11/2013 to 01/2014      DAAD KHYS PhD scholarship for a research stay abroad  
Institute of Information Systems  
TU Vienna, Austria

## **Supervised Student Thesis**

- 02/2010      Amine Miled: "Aspect-oriented monitoring in Grid" (Diploma thesis)
- 04/2013      Yousri Mhedheb: "Energy-aware virtual machines scheduling on the Cloud" (Diploma thesis)
- 08/2013      Tobias Sturm: "Implementation of a simulation environment for Cloud object storage infrastructures" (Bachelor thesis)

August 5, 2014