Hybrid Multi-Attribute QoS Optimization in Component Based Software Systems

Anne Koziolek[a], Danilo Ardagna[b], Raffaela Mirandola[b]

[a] *University of Zurich, Requirements Engineering Research Group, Zurich, Switzerland*
[b] *Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, Italy*

Abstract

Design decisions for complex, component-based systems impact multiple quality of service (QoS) properties. Often, means to improve one quality property deteriorate another one. In this scenario, selecting a good solution with respect to a single quality attribute can lead to unacceptable results with respect to the other quality attributes. A promising way to deal with this problem is to exploit multi-objective optimization where the objectives represent different quality attributes. The aim of these techniques is to devise a set of solutions, each of which assures an optimal trade-off between the conflicting qualities. Our previous work proposed a combined use of analytical optimization techniques and evolutionary algorithms to efficiently identify an optimal set of design alternatives with respect to performance and costs. This paper extends this approach to more QoS properties by providing analytical algorithms for availability-cost optimization and three-dimensional availability-performance-cost optimization. We demonstrate the use of this approach on a case study, showing that the analytical step provides a better-than-random starting population for the evolutionary optimization, which lead to a speed-up of 28% in the availability-cost case.

Keywords: software architecture optimization, quality of service, availability

1. Introduction

One of the today issues in software engineering is to find new effective ways to deal intelligently with the increasing complexity of software-intensive computing system. Modern software applications have evolved not only in terms of size, but also in the criticality of the services supported. Nowadays, most human activities, including critical ones, are either software enabled or entirely managed by software. This software pervasiveness together with its use in business-critical and safety-critical applications strongly requires the realization of dependable software systems able to guarantee the achievement of quality requirements, such as performance and availability. It is thus of key importance to empower the software architects with methods and tools able to explore different design alternatives taking into account their capability to fulfil the quality requirements.

To reach this goal, the use of software architecture (SA) has emerged as an appropriate level for dealing with software qualities (Clements et al. (2001); Smith and Williams (2002)) and several efforts have been devoted to the definition of methods and tools able to evaluate quality at SA level (see, for example, Balsamo et al. (2004); Koziolek (2010); Dobrica and Niemela (2002); Smith and Williams (2002)). However, each method usually addresses a single quality attribute (e.g., performance or availability), while a major challenge in complex software development is finding the best balance between different, possibly conflicting quality requirements that a system has to meet and cost constraints (e.g., maximize performance and availability, while minimizing cost), taking into account the different possible design alternatives. However, manually searching for optimal solutions is time consuming, thus the software architect needs an automated method that efficiently explores the architectural design space with respect to the multiple quality attributes.

When considering QoS of a SA, there is usually no single global solution, and a promising way to deal with them is to exploit multi-objective optimization (Ehrgott (2005); Blum and Roli (2003)) where the objectives represent different quality attributes. The aim of these techniques is to devise a set of solutions, called Pareto optimal solutions (Ehrgott (2005)), each of which assures a trade-off between the conflicting qualities. In other words, while moving from one Pareto solution to another, there is a certain amount of sacrifice in one objective(s) to achieve a certain amount of gain in the other(s).

Thus, multi-objective SA optimization has been proposed as a method where different design alternatives are automatically generated and evaluated for different quality attributes, providing the software architect with a powerful decision making tool enabling the selection of the SA that best fits multiple quality objectives. Previous approaches in this direction use evolutionary algorithms to approximate the Pareto optimal solutions (Aleti et al. (2009); Martens et al. (2010b)), however, the derived optimization process is time-consuming.

In previous work (Martens et al. (2010a)), we have initially proposed a combined use of analytical optimization techniques and evolutionary algorithms to more efficiently identify a near-optimal set of design alternatives. However, this previous work only presented the analytical optimization algorithm for performance-cost optimization. Thus, in this paper, we extend this approach for more QoS attributes, as explained in more detail below.

The overall proposed hybrid approach takes an initial SA of the system (fulfilling its functional requirements) as input. Based on this initial solution, a search problem is formulated by defining "degrees of freedom." The identification of a significant set of design alternatives is then based on a combined use of analytical optimization techniques and evolutionary algorithms (Blum and Roli (2003)). This hybrid approach extends work on pure evolutionary optimization (Martens et al. (2010b)) by introducing a step based on analytical optimization whose goal is to derive very efficiently the Pareto optimal solutions with respect to a *simplified* optimization problem. The obtained results are used as input candidates for an evolutionary optimization of the *original* search problem. In this way, more accurate estimates for availability and performance metrics and a larger near Pareto optimal solution set can be obtained.

In this paper, we extend this hybrid approach by two means. We provide analytical optimization algorithms for (1) availability-cost optimization and (2) three-dimensional performance-availability-cost optimization. Finally, we present a more complex case study, where we conduct multiple experiments to account for the stochastic nature of evolutionary algorithms.

Given a SA model that accurately reflects the quality properties of the system, the proposed method can lead both to a reduction of development costs and to an improvement of the quality of the final system, because an automated and efficient search is able to identify more and better design alternatives. The creation of the model is cost-efficient if risks of wrong design decisions are mitigated (cf. studies by Williams and Smith (2003) and Martens et al. (2011)). Note, however, that the improvement is limited to the considered design alternatives and does not replace human designers.

The remainder of the paper is organized as follows. Section 2 introduces the adopted architectural model and quality prediction techniques. Section 3 describes the optimization process. Experimental results are presented in

Section 4. Section 5 reviews other literature proposals. Conclusions are finally drawn in Section 6.

2. Background: Architecture Modelling and Analyses

In this section, we present the architectural model and the existing quality analyses methods our approach is based on. To quickly convey our contributed concepts to the reader, we introduce an example system.

Our approach requires a component-based architecture model with performance, availability, and costs annotations as input. Balsamo et al. (2004) and Koziolek (2010) have surveyed many different methods for specifying performance models, and Smith and Williams (2002) have provided a set of guidelines on how to obtain performance models during early development stages and on how to refine such models as the implementation progresses. For availability, Gokhale (2007) provides a survey.

We adopt here the Palladio Component Model (PCM) by Becker et al. (2009), but our approach could be extended to consider other architectural performance and availability models and analysis methods. The PCM is beneficial for our purposes as it is specifically designed for component-based systems. Thus, the PCM naturally supports many architectural degrees of freedom (e.g., substituting components, changing component allocation, etc.). Additionally, the model-driven capabilities of the PCM allow an easy automated generation of alternative architecture candidates.

The example system in this paper is the so-called business reporting system (BRS), which lets users retrieve reports and statistical data about running business processes from a data base. It is loosely based on a real system (Wu and Woodside (2004)) and represents a typical enterprise system. In addition to the components reported by Wu and Woodside (2004), we have split the two main application logic components into two each, making the system more complex. Fig. 1 shows some parts of the PCM model of the BRS visualized using annotated UML diagrams. It is a 4-tier system consisting of several software components. The components are allocated on four different servers. In an open workload usage scenario, requests arrive according to a Poisson process with an arrival rate $\lambda = 0.1$ req/sec. For each request, users first use the "graphicalReport" service and then the "graphicalView" service of the system. We consider a subset of the behaviour spanning six components, which is more complex as other industrial systems we have studied before (cf. de Gooijer et al. (2012); Koziolek et al. (2011b)) and thus represents industrial problems well.

Fig. 1 also shows an excerpt of the behaviour in the lower half of the figure. The behaviour model describes internal computation of components and calls to other components. `InternalActions` represent internal resource usage; they are annotated with CPU resource demands (exponentially distributed *demand* in seconds on a 2GHz CPU) and failure probabilities (FP). `ExternalCallActions` model calls to other components.

For example, the behaviour model for the `CoreGraphicEngine.getReport` service describes the control flow that is executed when the service is called. First, some internal computation on the CPU is executed, as described by the first `InternalAction` following the start action. Then, if a detailed report is requested, the right branch in the control flow is executed, which contains a sequence of some internal computation and calls to services of the required `IDB` interface and the required `ICache` interface. To which components these calls will be forwarded is determined by the separate "system model" defining the wiring of components, here shown in the static upper part of the figure. If a simple report is requested, the left branch is executed with fewer actions. When the end action is reached, the control is returned to the caller of the `CoreGraphicEngine.getReport` service.

Components are annotated with software cost ($Cost$) in K€ (only shown for the Webserver component in Fig. 1). The initial system is deployed to four servers annotated by costs ($HCost$) in K€, availability (HA) and processing rate (PR) in GHz. The complete model can be found online (cf. Koziolek et al. (2012)).

To provide software architects with a priori performance and availability guarantees, if the application includes any loops they are annotated with a discrete probability distribution, so we assume that an upper bound for loops iterations exists.

We now briefly explain the analysis methods for the considered quality criteria performance, availability, and costs[1]:

- **Performance:** For the analytical optimization, we model the software system by introducing an M/G/1 queue for each physical server

[1]Our approach can be generalized to consider a wider set of QoS metrics under the assumption that closed formula for quality criteria are available to derive an analytical formulation of the problem

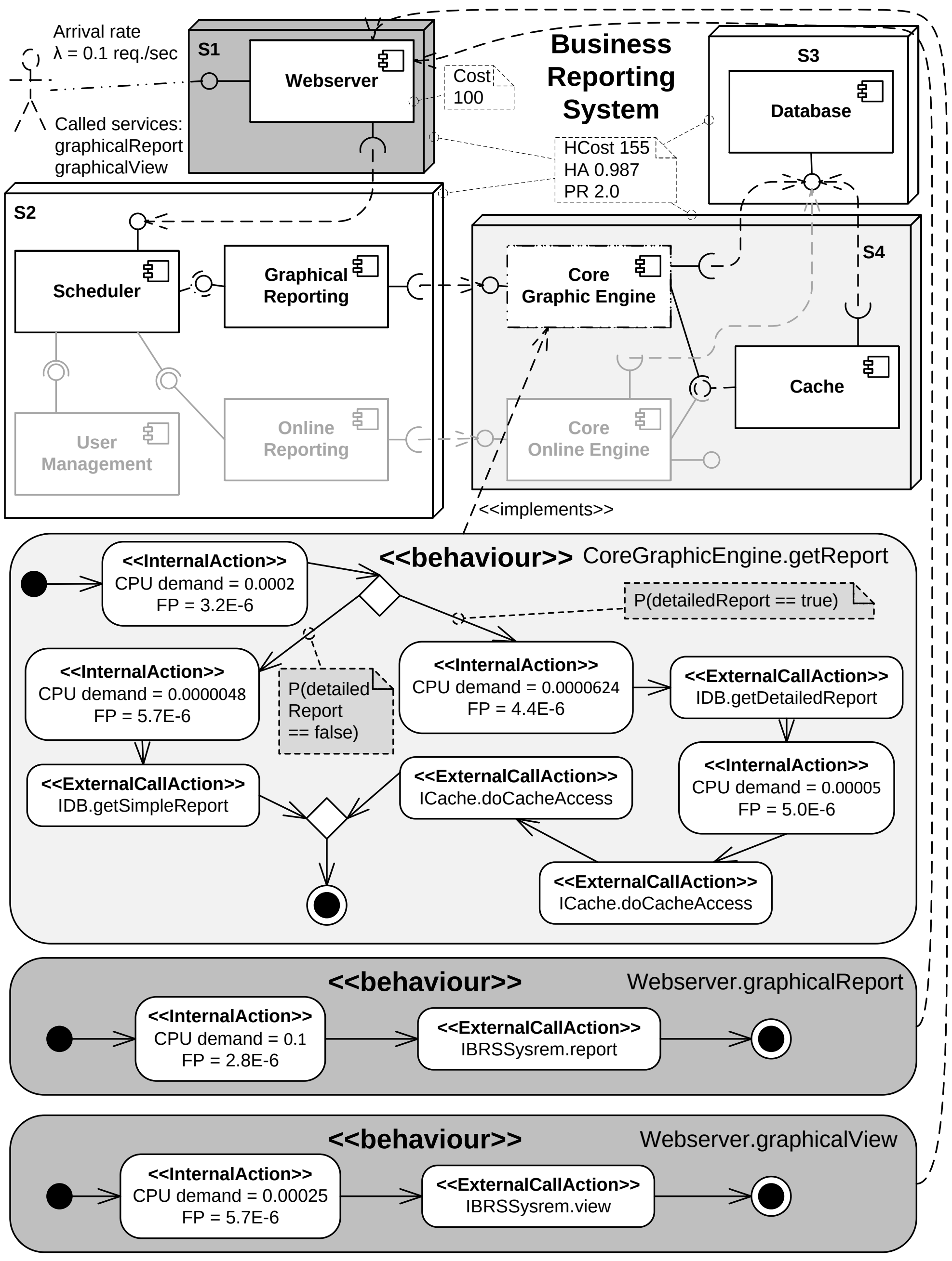
Arrival rate
λ = 0.1 req./sec
Called services:
graphicalReport
graphicalView
S1
Webserver
Cost 100
Business Reporting System
S3
Database
HCost 155
HA 0.987
PR 2.0
S2
Scheduler
Graphical Reporting
User Management
Online Reporting
S4
Core Graphic Engine
Cache
Core Online Engine
<<implements>>
<<behaviour>> CoreGraphicEngine.getReport
<<InternalAction>> CPU demand = 0.0002 FP = 3.2E-6
P(detailedReport == true)
<<InternalAction>> CPU demand = 0.0000048 FP = 5.7E-6
P(detailed Report == false)
<<InternalAction>> CPU demand = 0.0000624 FP = 4.4E-6
<<ExternalCallAction>> IDB.getDetailedReport
<<ExternalCallAction>> IDB.getSimpleReport
<<ExternalCallAction>> ICache.doCacheAccess
<<InternalAction>> CPU demand = 0.00005 FP = 5.0E-6
<<ExternalCallAction>> ICache.doCacheAccess
<<behaviour>> Webserver.graphicalReport
<<InternalAction>> CPU demand = 0.1 FP = 2.8E-6
<<ExternalCallAction>> IBRSSysrem.report
<<behaviour>> Webserver.graphicalView
<<InternalAction>> CPU demand = 0.00025 FP = 5.7E-6
<<ExternalCallAction>> IBRSSysrem.view

Figure 1: Business Reporting System: PCM instance of the case study system. Components that are not used when providing the studied services graphicalReport and graphicalView are marked with grey borders. Which behaviour is executed on which server is emphasized by the same shade of grey.

(Kleinrock (1975)). For the evolutionary optimization, we use an automated transformation of PCM models into Layered Queueing Networks (Franks et al. (2009)) to derive response times.

- **Availability:** For the analytical optimization, we apply the well-known serial/parallel formulas (see, e.g. Musa et al. (1987); Avizienis et al. (2004)) to the PCM model. In particular, the availability is evaluated by considering the set of components involved, the physical servers supporting the execution, and the probability of invocations. For the evolutionary optimization, we use an automated transformation of PCM models into absorbing discrete time Markov chains (DTMC) and solve them with the PCM Markov solver (Brosch et al. (2012)).

- **Costs:** We annotate constant costs to each component and each server configuration. The software architect can choose whether costs values represent procurement costs, total costs of ownership, or other. If a server specified in the model is not used (i.e., no components are allocated to it), it is not considered in the alternative exploration process and its cost do not add to the overall costs. The goal of this simplistic model is to allow costs to be considered, not to provide a sophisticated costs estimation technique. For the latter, existing costs estimation techniques such as COCOMO II by Boehm et al. (2000) could be integrated here to obtain more accurate values.

All quality attributes combined, it is not obvious how to change the architectural model efficiently to improve the quality properties. For example, the software architect could increase the processing rate of server $\texttt{S}_1$, which would result in better performance but higher costs. The software architect could also change the component allocation or use other components with different QoS attributes.

The design space for this example is huge. Manually checking the possible design alternatives in a trial-and-error approach is laborious and error-prone. The software architect cannot easily create design alternatives that are even locally optimal for all quality criteria. Finding global optima is practically impossible because it requires modelling each alternative. In practice this situation is often mitigated by over-provisioning (i.e., incorporating fast and expensive hardware resources), leading to unnecessarily high costs.

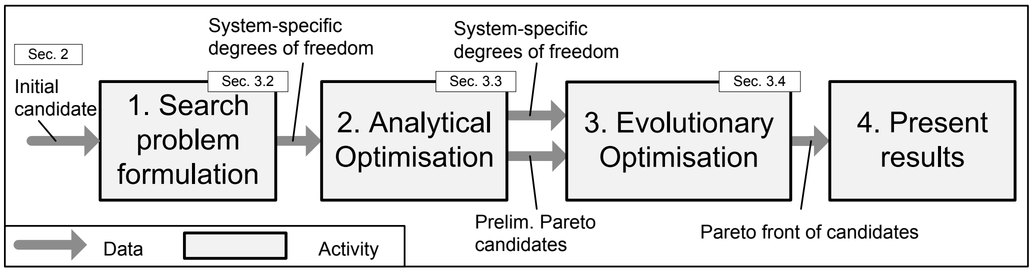
Sec. 2
Initial candidate
Sec. 3.2
1. Search problem formulation
System-specific degrees of freedom
Sec. 3.3
2. Analytical Optimisation
System-specific degrees of freedom
Sec. 3.4
3. Evolutionary Optimisation
4. Present results
Prelim. Pareto candidates
Pareto front of candidates
Data
Activity

Figure 2: Hybrid Optimization Process Overview.

3. Optimization Process

To present our hybrid optimization approach, we first provide an overview of the overall optimization framework in Section 3.1. In Section 3.2, we describe the search problem we address. Then, we describe in detail the analytical optimization (Section 3.3) and the evolutionary optimization (Section 3.4).

3.1. Overview

Our approach starts considering as input an initial architectural model of the system, named *initial candidate* in Fig. 2. In our case, this is a complete PCM model instance as shown in Fig. 1. The optimization process starts with the *search problem formulation*. In this work, we consider three degree of freedom types: (1) allocation of components, (2) server configuration, and (3) component selection. The result of this step is a set of degree of freedom instances that describe the search problem.

In the second step, the search problem is optimized using analytical techniques with simplified QoS evaluation and a simplified search space formulation. The impact of a degree of freedom (or a combination of them) is evaluated by analytical models, and the Pareto optimal candidates for the simplified search space are derived very efficiently by solving a mixed integer linear programming problem. The result of this step is a set of candidates that are Pareto optimal with respect to the simplified QoS evaluation and the simplified search space. In the third step, the results of the analytical optimization are used as input candidates for an evolutionary optimization of the original search problem. Evolutionary optimization is more time consuming, but it can consider the whole search space and obtain accurate estimates for availability and performance metrics by using more sophisticated performance models (i.e., Layered Queuing Networks, cf. Franks and Li (2012)) and availability models (i.e., DTMCs, cf. Brosch et al. (2012)).

The result of the evolutionary optimization phase is a set of near Pareto optimal candidates. This set is presented to the software architect, who can study the remaining optimal trade-offs between possibly conflicting objectives. As it will be discussed in Section 4, providing approximated Pareto solutions to the evolutionary search allows the improvement and the speed-up of the whole analysis process.

3.2. Search Problem Formulation

Candidate solutions can be evaluated for optimal trade-offs, i.e. for Pareto optimality (Ehrgott (2005)). A candidate architecture is Pareto optimal, if it is superior to all other candidate in at least one quality criterion. More formally: Let a be a candidate solution, let DS be the set of all possible candidates, and let q be a quality criterion with a value set D_q, an evaluation function $f_q : DS \rightarrow D_q$ so that $f_q(c)$ denotes the quality property of a $c \in DS$ for the quality criterion q, and an order $\leq_q$ on D_q so that $c_1 \leq_q c_2$ means that c_1 is better than or equal to c_2 with respect to quality criterion q. Then, a candidate solution a is Pareto optimal if it is better than any other candidate b in at least one quality criterion or equally good than b in all criteria, i.e. iff $\forall b \in DS\ (\exists q : f_q(a) \leq_q f_q(b)) \vee (\forall q : f_q(a) =_q f_q(b))$.

If a candidate solution is not Pareto optimal, then it is Pareto-dominated by at least one other candidate solution in DS that is better or equal in all quality criteria. The optimization problem can be formulated as follows for a set of quality criteria $Q = \{q_1, ..., q_m\}$: $\min_{c \in DS} [f_{q_1}(c), ..., f_{q_m}(c)]$. In this work, we consider three quality criteria: $q_1 = T =$ mean response time, $q_2 = A =$ availability measured as the probability of success of each request, and $q_3 = C =$ cost.

Furthermore, a candidate c is part of the so-called Pareto front with respect to a set of candidate solutions $S \subseteq DS$ if it is not Pareto-dominated by any other candidate from S, i.e. iff $\forall b \in S\ (\exists q : f_q(c) \leq_q f_q(b)) \vee (\forall q : f_q(c) =_q f_q(b))$.

In our approach, the following degrees of freedom can be considered:

Allocation of components to available servers: The mapping of components to servers can be changed. This is an integral part of most performance prediction models and availability prediction models and has large effects on the performance of a system. For example, the `Scheduler` component of the BRS system could be allocated also to $\mathtt{S}_1$, $\mathtt{S}_3$, or $\mathtt{S}_4$. When reallocating components, the number of servers can change as well. The software architect can specify the set or the maximum number of servers to be considered.

Server configuration: The available hardware resources (CPU, HDD, ...) can be changed in a certain range. In this work, we model a discrete set of servers with different CPU processing rates, availability, and costs. Thus, components can be allocated to servers with different processing rates.

Component selection: If functionally-equivalent components with different non-functional properties are available, they can be exchanged. Currently, we deem that a component B can replace a component A if B provides (i.e., implements) all interfaces provided by A and if B requires at most the interfaces required by A.

More degrees of freedom that could be considered in an automated approach are described by Koziolek (2011) and de Gooijer et al. (2012). In the search problem formulation step, the initial candidate model is automatically analysed for instantiations of these degrees of freedom. The found set of degree of freedom instances defines the search space. If desired, the software architect can also manually remove some of them.

3.3. Analytical Optimization

The goal of the analytical optimization is to quickly produce the optimal solutions for a simplified version of the optimization problem. We use a simplified search space and simplified quality analyses with closed formulas to quickly evaluate the QoS effects of single design alternatives. This allows us to formulate the optimization problem as a mixed integer linear problem, which can be comparably efficiently solved using standard solvers.

Section 3.3.1 describes the design alternatives for the analytical optimization. Then, sections 3.3.2 and 3.3.3 describe the analytical techniques developed for two dimensions Pareto analyses, considering performance-cost and availability-cost trade-off, respectively. The three dimensional analysis is described in Section 3.3.4.

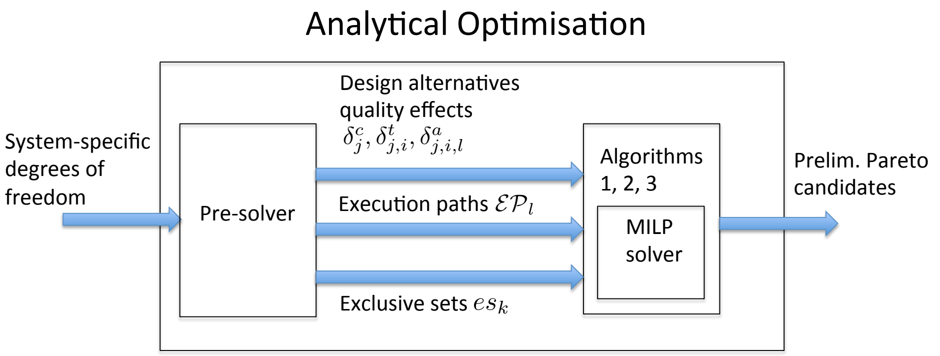
Analytical Optimisation
System-specific degrees of freedom
Pre-solver
Design alternatives quality effects $\delta^c_j, \delta^t_{j,i}, \delta^a_{j,i,l}$
Execution paths $\mathcal{EP}_l$
Exclusive sets es_k
Algorithms 1, 2, 3
MILP solver
Prelim. Pareto candidates

Figure 3: Analytical Optimization Overview.

3.3.1. Analytical Decision Space

For the simplified optimization problem, we transform the PCM model and the annotating degree of freedom model into a set of "atomic" design alternatives for which we can then determine the quality effects, as initially presented by Martens et al. (2010b).

First, the behaviour model of the PCM is transformed by a pre-solver (see Fig. 3) into a Directed Acyclic Graph (DAG) of software tasks. All different possible behaviours are represented as one execution path in this graph. As in other literature approaches (see, e.g., Canfora et al. (2005); Ardagna and Pernici (2007)), we use a probabilistic approach and label each execution path l with the probability p_l to be executed. Similarly, each task i in the path (i.e., each node in the execution graph) is labelled with the probability of execution π_i.

Probabilistic approaches allow studying how a system behaves "on average." This is useful whenever the worst-case analysis may be conservative and lead to too expensive solutions.

For the sake of simplicity, in the following we assume that the application under study includes a single initial task and a single end task. Additionally, loops are peeled and transformed into a number of branches with varying number of repetitions according to the annotated probability distribution (Ardagna and Pernici (2007)). Note that, one component in the PCM can be mapped to several tasks in the DAG, as each `InternalAction` is mapped to a task.

The pre-solver then evaluates the quality metrics of each task i included in the initial candidate by means of a M/G/1 queue (Kleinrock (1975)) and the standard availability formulas (Avizienis et al. (2004)).

According to the characteristics of M/G/1 queue, the response time R_i is evaluated as:

$$R_i = \frac{D_i}{1 - U_s} \tag{1}$$

where D_i is the average service demand of a task i to the server s and U_s is the utilization of the server s, on which the task i is executed, and is computed as $\sum_{i \in T_s} D_i \cdot \lambda \cdot \pi_i$, with T_s being the set of tasks executed on server s, π_i being the probability of execution of i, and λ being the arrival rate of a Poisson process in an open workload usage scenario.

On the other hand, availability (i.e., the readiness for correct service Avizienis et al. (2004)) of each task is calculated as:

$$A_i = (1 - POFOD_i) * \sqrt[k]{A_s} \tag{2}$$

where $POFOD_i$ is the Probability Of Failure On Demand for task i (i.e., the probability that a failure happens when a request is sent to task i); A_s is the hardware availability of server s where task i is executed and k are the number of services allocated on s belonging to the same execution path.

In the following, a binary decision variable x_j is introduced for each "atomic" design alternative which can be obtained from the degrees of freedom. x_j is equal to 1 if the corresponding design alternative is implemented in the system, and 0 otherwise.

The optimization problem which can be introduced in this way is combinatorial in nature, since a Pareto optimal solution can be obtained by selecting a combination of atomic design alternatives.

Atomic design alternatives describe each possible atomic change in the SA. For example, the alternative configurations of S_1 for the reference system in Fig. 1 can be modelled introducing the binary variables x_1 (CPU downgrade to 1 GHz), x_2 (CPU upgrade to 3 GHz), x_3 (CPU upgrade to 4 GHz). Down/upgrades of servers S_2, S_3, and S_4 can be modelled analogously with variables x_4 to x_{12}. Likewise, the alternative components selection can be modelled by introducing two binary variables x_{13} and x_{14} equal to 1 iff the `WebServer` is replaced by alternative `WebServer2` and `WebServer3` implementation, respectively.

Some of the atomic design alternatives could be in conflict. For example, since only one server CPU can be changed at one instance, the following constraint has to be introduced for S_1:

$$x_1 + x_2 + x_3 \leq 1 \tag{3}$$

Indeed, since x_1, x_2 and x_3 can be 0 or 1, by introducing the constraint above, only one variable can be raised to 1. Hence, only one among the conflicting design alternatives can be selected.

Formally, we introduce an exclusive set es_k for each combination of atomic design alternatives which are in conflict with each other, because they concern the same software component and/or the same physical server where components are deployed. A parameter $es_{k,j} = 1$ is introduced indicating that the atomic design alternative j is in the exclusive set k, while $es_{k,j} = 0$ otherwise. If we consider again the constraint (3), the corresponding k-th exclusive set is characterized by $es_{k,1} = es_{k,2} = es_{k,3} = 1$, while $es_{k,j} = 0$ for all $j > 3$.

Note that, an exclusive set might include variables associated with any degree of freedom (component allocation, selection or server configuration). Variables need to be included in the same exclusive set when they correspond to design alternatives altering the selection/allocation of a software component and/or the configuration of a physical server where components are deployed. Furthermore, note that the size of exclusive sets could grow exponentially, since taking into account all of the atomic choices is also combinatorial in nature. However, since the number of possibly conflicting atomic design alternatives is usually significantly lower than the number of degrees of freedom, the analytical problem can be formulated and solved efficiently, as it will be shown in Section 4.2.

The pre-solver finally evaluates the quality effects of each design alternative j, namely:

- δ^c_j: The cost variation of the initial candidate for implementing the design alternative j.
- $\delta^t_{j,i}$: The variation of the response time (evaluated by means of M/G/1 formula) for task i if the design alternative j is implemented.
- $\delta^a_{j,i,l}$: The variation of the availability for task i along the execution path l if the design alternative j is implemented.

For example, if the arrival rate is $\lambda = 4\ req/sec$ and the S_1 CPU frequency is raised to 3 GHz (x_2 design alternative), then the service demands of the two tasks deployed in the `WebServer` and S_1 utilization initially equal to 0.1 sec, 0.00025 sec, and 0.401 are reduced by a factor $3/2 = 1.5$. Thus, the initial response times equal to 0.167 sec and 0.0004 sec become 0.091 sec and 0.0002 sec and hence the deltas $\delta^t_{j,i}$ are equal to -0.076 and -0.0002 sec. Availability variations can be computed similarly.

In the following, let $\mathcal{I}$ denote the set of indexes of the system tasks and let $\mathcal{J}$ denote the set of indexes for the atomic design alternatives arising from the degrees of freedom definition.

3.3.2. Performance-Cost Trade-off Analysis

In this section we reproduce the performance-cost optimization already presented by Martens et al. (2010b), because this case is most straightforward and thus helps to convey the idea.

Let us denote by $\tilde{C}$ the cost of the initial candidate, let $\tilde{t}_i$ be the average response time for task i invocation in the initial candidate and let $\delta^t_{j,i}$ denote the variation of the response time for task i if the design alternative j is implemented.

Then, the execution time t_i of task i can be defined depending on the selection of atomic design choices:

$$t_i = \tilde{t}_i + \sum_{j \in \mathcal{J}} \delta^t_{j,i} x_j, \forall i; \qquad \sum_{j \in \mathcal{J}} es_{k,j} x_j \le 1, \forall k$$

Since π_i is the probability of execution of task i (which can be derived as the sum of the transition probabilities of the paths in the DAG from the initial task to i), the execution time T of the whole application can then be computed as $T = \sum_{i \in \mathcal{I}} \pi_i \cdot t_i$, while the cost C corresponding to a given combination of atomic choices is given by $C = \tilde{C} + \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j$.

Algorithm 1 determines the Pareto optimal solutions by solving iteratively the problems shown in Fig. 4. It requires as input the upper $\overline{T}^{upper}$ and lower bound $\overline{T}^{lower}$ response time for the application under study, which can be computed easily by considering the maximum and minimum $\delta^t_{j,i}$ for each task i.

The Algorithm starts minimizing the system cost with the goal of providing a response time lower than $\overline{T}^{upper}$ (i.e., solving problem (P1), see step 4).

(P1) $\min C$

subject to:

$$C = \tilde{C} + \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j$$
$$t_i = \tilde{t}_i + \sum_{j \in \mathcal{J}} \delta^t_{j,i} x_j, \qquad \forall i$$
$$\sum_{j \in \mathcal{J}} es_{k,j} x_j \le 1, \qquad \forall k$$
$$\sum_{i \in \mathcal{I}} \pi_i \cdot t_i \le \overline{T}$$

(P2) $\min T$

subject to:

$$t_i = \tilde{t}_i + \sum_{j \in \mathcal{J}} \delta^t_{j,i} x_j, \qquad \forall i$$
$$T = \sum_{i \in \mathcal{I}} \pi_i \cdot t_i$$
$$\sum_{j \in \mathcal{J}} es_{k,j} x_j \le 1, \qquad \forall k$$
$$\tilde{C} + \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j \le \overline{C}$$

Figure 4: The Analytic Optimization Problems for Performance and Cost

Let x^* be the corresponding optimum solution (i.e., the set of atomic design alternatives to be implemented) and C^* be the corresponding cost. Then, the first Pareto solution is obtained by solving (P2) setting $\overline{C} = C^*$ (see step 6). Let T^* be optimum response time obtained. Indeed, no other atomic design alternative combination can lead to a better response time with a lower cost, hence x^* computed at step 6 is a Pareto solution. The process is then iterated by solving (P1) again and setting as constraint $\overline{T} = T^* - \epsilon$, where $\epsilon > 0$ is any sufficiently small constant. $IC + x^*$ at step 7 denotes the solution obtained by applying to the initial candidate IC the set of atomic design alternatives x^*.

If problems (P1) and (P2) are optimally solved, then Algorithm 1 identifies all the Pareto optimal solutions of the reduced problem. It can be shown that (P1) and (P2) are NP-hard (see Appendix), since they are as difficult as a knapsack problem. The solution algorithm complexity grows exponentially with the number of binary variables. However, as it will be shown in Section 4, current solvers are very efficient and (P1) and (P2) solutions can be computed very quickly for realistic design problems of reasonable size.

3.3.3. Availability-Cost Trade-off Analysis

For the availability analysis, the analytical problem formulation can be derived similarly. The main difference is that, in order to derive an efficient mixed integer linear formulation, the delta values have to be derived for independent application execution paths (i.e., each path from the source to the sink) and the optimization has to be iterated for each execution path. The set of initial candidates provided to the evolutionary optimization is obtained

```
   input  : $\overline{T}^{upper}$, $\overline{T}^{lower}$
   output: $Paretos$
 1 $\overline{T} \leftarrow \overline{T}^{upper}$;
 2 $Paretos \leftarrow \emptyset$;
 3 while $\overline{T}^{lower} \leq \overline{T}$ do
 4     Solve (P1). Let $x^*$ be the optimum solution found and $C^*$ its cost ;
 5     $\overline{C} \leftarrow C^*$;
 6     Solve (P2). Let $x^*$ be the optimum solution found and $T^*$ the application execution time ;
 7     $Paretos \leftarrow Paretos \bigcup \{IC + x^*\}$;
 8     $\overline{T} \leftarrow T^* - \epsilon$
 9 end
10 return $Paretos$;
```

Algorithm 1: Analytical Pareto optimality Algorithm

as the union of the analytical solutions of individual execution paths. In the following execution paths will be indexed by l and $\mathcal{EP}_l$ will indicate the set of indexes of tasks in the l-th execution path.

Let us denote with $\tilde{a}_{i,l}$, the availability for task i invocation in the execution path l of the initial candidate and let $\delta^a_{j,i,l}$ be the variation of the availability for task i along the execution path l if the design alternative j is implemented. Note that, $\delta^a_{j,i,l}$ is a real number and we have $\delta^a_{j,i,l} > 1$, if the design alternative improves the task availability, and $\delta^a_{j,i,l} \leq 1$ otherwise.

If we denote by $a_{i,l}$ the availability of task i along the execution path l according to the selection of atomic design choices, we have:

$$a_{i,l} = \tilde{a}_{i,l} \cdot \prod_{j \in \mathcal{J}} (\delta^a_{j,i,l})^{x_j}, \forall i \in \mathcal{EP}_l; \qquad \sum_{j \in \mathcal{J}} es_{k,j} x_j \leq 1, \forall k$$

The application availability along the execution path l, A_l, can then be computed as:

$$A_l = \prod_{i \in \mathcal{EP}_l} a_{i,l} = \prod_{i \in \mathcal{EP}_l} \tilde{a}_{i,l} \cdot \prod_{j \in \mathcal{J}} (\delta^a_{j,i,l})^{x_j}$$

The equation above is non-linear in the binary decision variables x_j, however it can be easily linearized by applying the logarithm function and introducing the auxiliary variable y as follows:

$$y = ln(A_l) = \sum_{i \in \mathcal{EP}_l} ln(\tilde{a}_{i,l}) + \sum_{j \in \mathcal{J}} ln(\delta^a_{j,i,l}) x_j$$

Algorithm 2 determines the Pareto optimal solutions by iteratively solving the problems shown in Fig. 5.

(P3) $\min C$
subject to:

$$C = \tilde{C} + \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j$$

$$\sum_{j \in \mathcal{J}} es_{k,j} x_j \leq 1, \quad \forall k$$

$$\sum_{i \in \mathcal{EP}_l} ln(\tilde{a}_{i,l}) + \sum_{j \in \mathcal{J}} ln(\delta^a_{j,i,l}) x_j \geq ln(\overline{A})$$

(P4) $\max y$
subject to:

$$y = \sum_{i \in \mathcal{EP}_l} ln(\tilde{a}_{i,l}) + \sum_{j \in \mathcal{J}} ln(\delta^a_{j,i,l}) x_j$$

$$\sum_{j \in \mathcal{J}} es_{k,j} x_j \leq 1, \quad \forall k$$

$$\tilde{C} + \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j \leq \overline{C}$$

Figure 5: The Analytical Optimization Problems for Availability and Cost Trade-off

Similarly to Algorithm 1, Algorithm 2 requires as input the upper $\overline{A}^{upper}$ and lower bound $\overline{A}^{lower}$ availability for the application under study, which can be easily computed by considering the maximum and minimum $\delta^a_{j,i,l}$ for each task i along every execution path. Then, for each execution path, the Algorithm starts minimizing the system cost with the goal to provide an availability value greater than $\overline{A}^{lower}$ (i.e., solving problem (P3), see step 5). Let x^* be the corresponding optimum solution and C^* be the corresponding cost. Then, the first Pareto solution is obtained by solving (P4) setting $\overline{C} = C^*$ (see step 7). Let $A^* = exp(y)$ be the optimum availability value obtained. Indeed, no other atomic design alternative combination can lead to a better availability with a lower cost, hence x^* computed at step 7 is a Pareto solution. The process is then iterated by solving (P3) again and setting as constraint $\overline{A} = A^* + \epsilon$, where $\epsilon > 0$ is any sufficiently small constant.

Note that, at the last iteration the set *Paretos* includes all of the Pareto optimal solutions obtained along the application execution paths which is used as input by the evolutionary algorithm. As in the performance-costs trade-off analysis, also the problems (P3) and (P4) are NP-hard, since they can be reduced to a knapsack problem (see Appendix).

```
   input  : $\overline{A}^{upper}$, $\overline{A}^{lower}$
   output: $Paretos$
1  $\overline{A} \leftarrow \overline{A}^{lower}$;
2  $Paretos \leftarrow \emptyset$;
3  forall the execution paths $l$ do
4      while $\overline{A} < \overline{A}^{upper}$ do
5          Solve (P3). Let $x^*$ be the optimum solution found and $C^*$
           its cost ;
6          $\overline{C} \leftarrow C^*$;
7          Solve (P4). Let $x^*$ be the optimum solution found and
           $A^* = exp(y)$ the application availability along the execution
           path $l$ ;
8          $Paretos \leftarrow Paretos \bigcup \{IC + x^*\}$;
9          $\overline{A} \leftarrow A^* + \epsilon$
10     end
11 end
12 return $Paretos$;
```

Algorithm 2: Availability vs. Cost Pareto optimality Algorithm

3.3.4. The Three-dimensional Trade-off Analysis

If we want to consider an optimization that involves execution time, availability and cost together, it is necessary to consider the optimization problems we have discussed so far at the same time. Generally, availability and performance are simultaneously in trade-off with cost. In fact hardware/software components with better performance or more resilient to failures have usually higher costs. This means that, once we have fixed a certain cost, we can decide to search for the "best" solutions with respect to availability and performance. In this way we can find solutions that, at the same cost, can be better in performance or in availability. So the set of problems for the optimization that considers all the three features (performance, availability, cost), contains the same parameter and variables of problems $(P1 - P4)$ and can be formulated as shown in Figure 6.

(PC) $\min C$

subject to:

$$C = \tilde{C} + \sum_{j\in\mathcal{J}} \delta^c_j \cdot x_j$$
$$t_i = \tilde{t}_i + \sum_{j\in\mathcal{J}} \delta^t_{j,i} x_j, \qquad \forall i$$
$$\sum_{j\in\mathcal{J}} es_{k,j} x_j \leq 1, \qquad \forall k$$
$$\sum_{i\in\mathcal{I}} \pi_i \cdot t_i \leq \overline{T}$$
$$\sum_{i\in\mathcal{EP}_l} ln(\tilde{a}_{i,l}) + \sum_{j\in\mathcal{J}} ln(\delta^a_{j,i,l}) x_j \geq ln(\overline{A}) \qquad \forall l$$

(PT) $\min T$

subject to:

$$t_i = \tilde{t}_i + \sum_{j\in\mathcal{J}} \delta^t_{j,i} x_j, \qquad \forall i$$
$$T = \sum_{i\in\mathcal{I}} \pi_i \cdot t_i$$
$$\sum_{j\in\mathcal{J}} es_{k,j} x_j \leq 1, \qquad \forall k$$
$$\tilde{C} + \sum_{j\in\mathcal{J}} \delta^c_j \cdot x_j \leq \overline{C}$$
$$\sum_{i\in\mathcal{EP}_l} ln(\tilde{a}_{i,l}) + \sum_{j\in\mathcal{J}} ln(\delta^a_{j,i,l}) x_j \geq ln(\overline{A}) \qquad \forall l$$

(PA_l) $\max y_l$

subject to:

$$y_l = \sum_{i\in\mathcal{EP}_l} ln(\tilde{a}_{i,l}) + \sum_{j\in\mathcal{J}} ln(\delta^a_{j,i,l}) x_j$$
$$\sum_{j\in\mathcal{J}} es_{k,j} x_j \leq 1, \qquad \forall k$$
$$\tilde{C} + \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} \delta^c_j \cdot x_j \leq \overline{C}$$
$$t_i = \tilde{t}_i + \sum_{j\in\mathcal{J}} \delta^t_{j,i} x_j, \qquad \forall i$$
$$\sum_{i\in\mathcal{I}} \pi_i \cdot t_i \leq \overline{T}$$

Figure 6: The Analytic Optimization Problems for all the three objectives

Problem (PC), as problems $(P1)$ and $(P3)$, minimizes the cost, but considers at the same time a constraint on the execution time and a set of constraints for the availability in each path. In the same way problem (PT), is similar to problem $(P2)$, with the addition of the constraints for the availability. Finally problem $(P4)$ is reformulated as a set of sub-problems (PA_l), one for each execution path, considering a constraint on the cost and another one on the execution time.

```
    input  : $\overline{A}^{upper}$, $\overline{A}^{lower}$, $\overline{T}^{upper}$ , $\overline{T}^{lower}$
    output: $Paretos$
 1  $\overline{T} \leftarrow \overline{T}^{upper}$;
 2  $Paretos \leftarrow \emptyset$;
 3  while $\overline{T}^{lower} \leq \overline{T}$ do
 4      Solve $(PC)$. Let $x^*$ be the optimum solution found and $C^*$
        its cost ;
 5      forall the  execution paths $l$ do
 6          $\overline{A} \leftarrow \overline{A}^{lower}$;
 7          while $\overline{A} < \overline{A}^{upper}$ do
 8              $\overline{C} \leftarrow C^*$;
 9              Solve $(PT)$. Let $x^*$ be the optimum solution found
                and $T^*$ the application execution ;
10              $\overline{T} \leftarrow T^*$;
11              Solve $(PA_l)$. Let $x^*$ be the optimum solution found
                and $A^* = \sum_{l \in \cup \mathcal{EP}} exp(y_l) * p_l$ be the estimate
12              of the application availability ;
13              $Paretos \leftarrow Paretos \bigcup \{IC + x^*\}$;
14              $\overline{A} \leftarrow A^* + \epsilon$ ;
15          end
16      end
17      $\overline{T} \leftarrow T^* - \epsilon'$
18  end
19  return $Paretos$;
```

Algorithm 3: Performance vs. Availability vs. Cost Pareto optimality Algorithm

To solve these problems, Algorithm 3 requires as inputs the upper and lower bounds for response time ($\overline{T}^{upper}$, $\overline{T}^{lower}$) and availability ($\overline{A}^{upper}$, $\overline{A}^{lower}$) of the application. The algorithm starts minimizing the system cost with the goal to provide a response time lower then $\overline{T}^{upper}$ and an availability higher than $\overline{A}^{lower}$ (i.e., solving problem (PC), see line 4). Let x^* be the corresponding optimum solution and C^* be the corresponding cost. Then, for each execution path l, fixing C^* as upper bound in cost (line 8), problem (PT) is solved, obtaining x^* as optimum solution and T^* as the corresponding execution time. Then, the first Pareto solution is obtained by setting $\overline{T} = T^*$ and solving (PA_l) (line 11). With p_l being the probability of execution of path l, A^* is evaluated as the weighted sum, according to p_l, of the availability of each path l. Hence the solution of the last computed problem is considered a Pareto optimal solution. The cycle (lines 7-15) is iterated by setting $\overline{A} = A^* + \epsilon$ as a constraint and solving (PT) again, until the upper bound $\overline{A}^{upper}$ is reached. Then constraint $\overline{T}$ is modified as $\overline{T} = T^* - \epsilon'$, and (PC) is solved again with new values for the constraints $\overline{A}$ and $\overline{T}$.

Note that the three dimensional analysis algorithm keeps the iteration on paths deriving from the availability problem in the inner cycle, while the minimization of costs is done in the outer cycle independently of each execution path. This allows the reduction of the overall number of iterations for determining the Pareto optimal solutions. It can be shown that (PC), (PT), and (PA_l) can be reduced to a multi-dimensional knapsack problem, and hence are NP-hard (see Appendix).

3.4. Evolutionary Optimization

To consider the full optimization problem and be able to use more accurate QoS analyses, the next step of our approach uses metaheuristic optimization which allows the use of any quality evaluation function. In this work, we use evolutionary optimization (see, e.g. (Blum and Roli, 2003, p. 284)), as it has been considered useful for multi-objective problems (Coello (1999)). Other metaheuristics could be used as well. More details on this choice have been described by Koziolek (2011).

Fig. 7 shows the main steps of our evolutionary search. The method is described here exemplary for our current realization in the PerOpteryx tool (Koziolek et al. (2012)) with the NSGA-II evolutionary algorithm (Deb et al. (2000)) as implemented in the Opt4J framework (Lukasiewycz et al. (2011)) with an extended reproduction step.

The process starts with an input population derived from the analytical optimization step. Individuals are then modified along degrees of freedom instances (see Section 3.1). As the software model contains all required annotations, all steps of the search can be completely automated. The population size n and the number of iterations i can be configured. If the input population size $|Paretos|$ is less than n, additional $n-|Paretos|$ random candidates

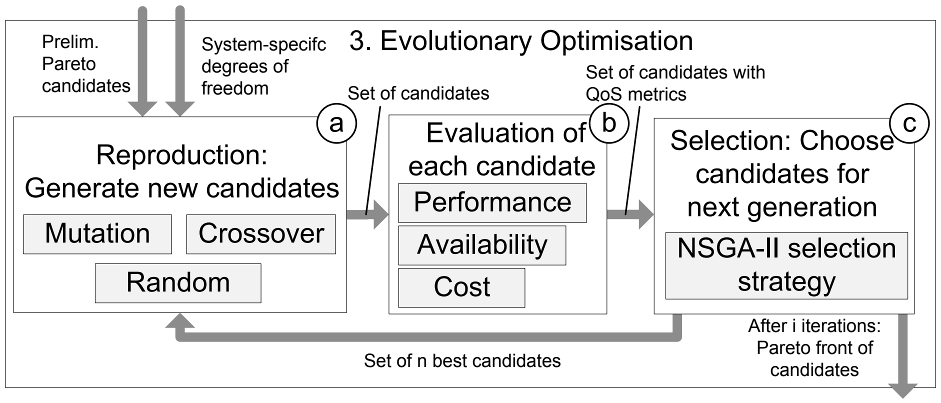
3. Evolutionary Optimisation
Prelim. Pareto candidates
System-specifc degrees of freedom
Set of candidates
Set of candidates with QoS metrics
a
Reproduction: Generate new candidates
Mutation
Crossover
Random
b
Evaluation of each candidate
Performance
Availability
Cost
c
Selection: Choose candidates for next generation
NSGA-II selection strategy
Set of n best candidates
After i iterations: Pareto front of candidates

Figure 7: Evolutionary optimization process

are generated to add diversity to the population and thus avoid convergence to local optima. The evolutionary search then iterates the following steps:

- (a) **Reproduction:** Based on the currently available candidates in the population, new candidate solutions are derived by "mutation" or "cross-over" or they are randomly created. With mutation, one or several design options are varied. With cross-over, two good candidate solutions are merged into two new ones. For each design option, it is randomly decided whether new candidate 1 receives the design decision value of parent 1 and new candidate 2 receives the value of parent 2 or vice versa (uniform cross-over). In addition to the original NSGA-II, in order to diversify the search, duplicate candidates are removed from the population and are replaced by candidates randomly generated based on the available design options.
- (b) **Evaluation:** Each yet unevaluated candidate is evaluated for each quality attribute of interest. In our case, performance, availability and/or costs metrics are predicted as described in Section 2. As a result, each candidate is annotated with the determined quality properties (i.e. mean response time, availability, and/or cost).
- (c) **Selection:** After the reproduction phase, the population has grown. In the selection phase, the population is again reduced by just keeping the n most promising candidates based on the NSGA-II selection strategy. After i iterations, the search ends here and returns the Pareto optimal candidates found so far.

More details on the evolutionary optimization (such as the genetic encoding) is described by Koziolek (2011).

Over several iterations, the combination of reproduction and selection lets the population converge towards the real front of globally Pareto optimal solutions. The result of the optimization is a set of Pareto optimal candidates with respect to all candidates evaluated before. If the search also keeps a good diversity of candidates, the result set can be near to the global optima. However, in general, evolutionary optimization cannot guarantee globally Pareto optimal solutions. Still, previous industrial case studies we conducted have indicated that the results are helpful (de Gooijer et al. (2012); Koziolek et al. (2011b)).

3.5. Assumptions and Limitations

Our approach has mainly the assumptions and limitations of model-based quality prediction for performance, availability, and costs, discussed below. Furthermore, it inherits the assumptions of the evolutionary optimization, as discussed in detail by Martens et al. (2010b) and Koziolek (2011).

Specific assumptions to the hybrid approach concern the analytical optimization part.

- First, we assume an open request-based workload model in which requests arrive according to a Poisson process, as observed in several real systems (Paxson and Floyd (1995); Costa et al. (2004)) and assumed by most previous solutions (Liu et al. (2001); Ranjan and Knightly (2008)). This assumption greatly simplifies the performance model, thus shortening the time required to determine the performance estimates. However, one might raise the issue of whether it is adequate to certain e-commerce and enterprise applications, whose workloads are more accurately described by independent arrivals of *user sessions*, that is, sequences of inter-dependent requests (cf. Krishnamurthy et al. (2006)). To address this issue, we rely on the results from Zhang et al. (2008), which show that the performance (i.e., response time) and resource requirements of *session-based systems* can be accurately captured by a simplified model based on the assumption of *independent request arrivals*, such as the one proposed here, provided that the distribution of different request types is the same as in the original session-based system.

- Furthermore, the hybrid approach will only work well if the quality properties of the system can be meaningfully approximated by the simplified quality evaluations. This might not be the case for some system, e.g. systems where the use of passive resources heavily influences the observed performance. Still, even in the worst case the analytical part will only provide a useless starting population to the evolutionary algorithm, but the evolutionary algorithm might overcome this over time. To overcome this problem, one might add a disagreement detection step before starting the evolutionary part. In this step, random solutions by the analytical part can be re-evaluated by the detailed quality evaluations, and if the disagreement in the results is too large, the analytical starting population could be discarded in favour of a purely random starting population.

The main underlying assumption of model-based quality prediction approaches in general is that a SA model annotated with quality information is available. The models require quality attribute annotations that reflect the quality properties of the system under study well. Furthermore, information like the usage profile (operational profile) needs to be known for performance and availability evaluations. For the quality prediction approaches used in this paper, the accuracy of the prediction has been discussed in papers presenting these prediction approaches (Franks et al. (2009); Brosch et al. (2012)). A more extensive discussion is provided by Koziolek (2011).

4. Experimental Results

This section reports the results of the quality optimization performed for the BRS system (cf. Figure 1) to demonstrate the applicability and usefulness of our approach and is organized as follows. Section 4.1 describes the degrees of freedom adopted. Section 4.2 summarizes the performance of the hybrid optimization. Finally, Section 4.3 compares the outcome of the hybrid approach to purely evolutionary optimization.

Notice that we do not compare our prediction results from the models with actual measurements from the system implementation. For our validation, we assume that the underlying modelling and prediction methods are sound and deliver accurate prediction results as discussed in Section 3.5.

Configuration	Processor Speed PR	Availability HA	Cost $HCost$
C_1	1 GHz	0.99986	145
C_2	2 GHz	0.99993	155
C_3	3 GHz	0.99995	267
C_4	4 GHz	0.99997	884

Table 1: Available Server Configurations for BRS

4.1. Search Problem Formulation

In the BRS under study we have considered the following degrees of freedom:

Component allocation: For the evolutionary optimization, all used components except the `Webserver` and the `Database` can be freely allocated to the four servers. The `Webserver` cannot be reallocated (e.g. for security reasons), the `Database` can only be allocated to servers S_1 or S_3. For the analytical optimization, we consider three allocation degrees of freedom: (1) The `GraphicalReporting` component can be allocated to servers S_1 to S_4, (2) the `Cache` component can be allocated to servers S_1 to S_4, and (3) the `Database` component can be allocated to servers S_1 or S_3.

Component selection: The `Webserver` can be realized using third party components. The software architect can choose among three functional equivalent implementations: `Webserver2` with cost 150 and `Webserver3` with cost 80. Both have less resource demand than the initial `Webserver`. `Webserver2` has better availability for the requests of type "view", while `Webserver3` has better availability for the requests of type "report".

Server configuration: For the analytical optimization, we consider four different server configurations C_1 to C_4 with varying processor speed (PR in GHz), hardware availability (probability HA), and cost $HCost$ available as shown in Table 1. For the evolutionary optimization, we allow for a continuous change of processing rate from 1 to 4 GHz. For the costs model, we analysed Intel's CPU price list (Intel Corporation (2010)). We fitted a power function to this data, so that the resulting costs of one server s with processing rate pr_s is $cost_s = 0.7665\ pr_s^{6.2539} + 145$ with coefficient of determination $R^2 = 0.965$.

The degrees of freedom are mapped into an optimization problem including 132 binary variables. For example, x_1-x_3, x_4-x_6, x_7-x_9, and x_{10}-x_{12} specify the four servers' up/downgrades. x_{13} and x_{17} are introduced to model the two `WebServer` component alternative implementations.

x_{14} to x_{16} combine the use of the first alternative `WebServer2` (x_{13}) with the upgrades of server S_1. For example, x_{14} denotes "CPU downgrade to 1 GHz for S_1 and use of `WebServer2`". Thus, x_{14} to x_{16} cannot be set to one together with any of x_1-x_3 and x_{13}, because the latter four decision variables also affect S_1. Consequently, we introduce an exclusive set which forbids that more that one variable out of x_1-x_3 and x_{13}-x_{16} can be selected:

$$es_1 : x_1 + x_2 + x_3 + x_{13} + x_{14} + x_{15} + x_{16} \leq 1$$

For brevity, we do not present all final exclusive sets in this paper. Details can be found online (Koziolek et al. (2012)).

Analogous to the above, x_{18} to x_{20} combine the use of the second alternative `WebServer3` (x_{17}) with the upgrades of server S_1.

x_{21} is associated with the allocation of the `GraphicalReporting` component to S_1 and x_{22}-x_{36} combine this decision with upgrades of the source server S_2 and the target server S_1. Similarly, x_{37}-x_{68} model the allocation of `GraphicalReporting` component to servers S_3 or S_4.

Analogously, x_{69}-x_{116} model the reallocation of the `Cache` component to any of the servers, and x_{117}-x_{132} model the allocation of the `Database` component to server S_1.

The exact values considered in the case study, including the architecture model, are provided online (Koziolek et al. (2012)).

4.2. Hybrid Approach Performance

Table 2 shows the statistics of the optimization runs (cand. = candidate(s), it. = iteration) which have been performed single threaded on a single core of an Intel Core 2 T7200 CPU @ 2GHz. The analytical optimization step is performed by running *CPLEX* IBM ILOG (2010), a state-of-the-art integer linear programming solver based on the branch-and-cut technique (Wolsey (1998)).

Our pre-solver, which determines optimization problem parameters and the exclusive sets used as input by CPLEX, is implemented in Java. Table 2 reports the generation time of our pre-solver and CPLEX optimization time in the first two columns.

The analytical optimization found 12 optimal candidates for the availability-cost case and 16 for the three dimensional case, reported in the third column of Table 2.

Analysis	Gen. Time	CPLEX Time	Input cand.	Avg. cand. in front	Avg. duration evol.	Mean d per cand.	Avg. duration total
Avail.-Cost	< 1	41 sec	12	24.5	12.4 min	0.25 sec	13.1 min
3D	sec	98 sec	16	63.3	67.5 min	1.34 sec	69.1 min

Table 2: Performance of the Hybrid Approach

For the evolutionary optimization, our PerOpteryx tool follows the process described in Section 3.4. The number of candidates per iteration was set to 30, a value sufficiently larger than the number of optimal candidates found by the previous, analytical step to add diversity and leave room for more solutions. The number of iterations was set to 200 and half of the population was replaced by offspring in each iteration, so that each runs analysed $3015 = 201 * 30 / 2$ candidates in total. The stop criterion of the search was a manually chosen maximum number of iterations. The results had to be inspected to determine whether the search had converged up to then.

For performance prediction, we use the Layered Queueing Network Solver (LQNS) by Franks et al. (2009). The solver was configured with a convergence value of 0.01 and an underrelaxation coefficient of 0.5. For availability and cost prediction, we use the PCM Markov solver by Brosch et al. (2012) and the PCM costs solver, respectively.

The evolutionary optimization found a Pareto front with on average 24.5 candidates for the availability-cost case and 63.3 for the three dimensional case (fourth column of Table 2). The average duration of the evolutionary optimization step was 12.4 min and 67.5 min, respectively (fifth column). We can derive the average duration for evaluating one of the 3,015 evaluated candidates (sixth column). Finally, the total duration of both the analytical step and the evolutionary step is 13.1 min and 69.1 min, respectively (seventh column).

We expect the scalability of the approach to mainly depend on two aspects independently, the number of degrees of freedom / decision variables and the complexity of quality evaluation. We observe that the analytical part requires only little time compared to the evolutionary part, so it is not the main performance factor. As current solvers scale well up to 10,000 variables on a single core and more recent solver versions even make use of multiple cores (see IBM ILOG (2010)), we expect that the analytic part also scales up to larger problem instances, and even more if further relaxations of the problems are used (i.e., the CPLEX solver can be stopped without reaching the optimality of each individual MILP problem as we did here, by selecting as stopping criteria a given bound, e.g., 5% (IBM ILOG (2010))).

The evolutionary part has been recognized to be useful for hard problems (Deb (2001)). Here, the problem also becomes more complex as the number of decision variables grow, but also other factors determine the performance for evolutionary algorithms (Deb (2001)). However, note that the BRS system is already a comparably complex system. For example, previous industry case studies we considered (de Gooijer et al. (2012); Koziolek et al. (2011b)) had fewer degrees of freedoms (decision variables).

With respect to the complexity of quality evaluation, the analytical one is efficient as it uses closed formulas for quality predictions. The more detailed quality evaluation approaches used in the evolutionary part also have been shown to be applicable to complex industrial systems.

4.3. Comparison of Hybrid and Pure Evolutionary

In this section, we first describe the metrics used for comparing the optimization outcome in Section 4.3.1 and then present the results in Section 4.3.2.

4.3.1. Metrics for Comparison

The performance of an optimization approach is typically measured by assessing the quality of the solutions and the time needed to generate the solutions (cf. Zitzler et al. (2008)).

First, to account for the stochastic nature of evolutionary algorithms, all experiments are replicated several times. For each experiment setting X (i.e. running the optimization purely evolutionary $X = E$ or hybrid $X = H$), a set of runs $\{X_r \,|r = 0, \dots, n\}$ is performed. At each iteration i, a run X_r has produced a Pareto front, i.e. a sample, which we denote with $P(X_r^i)$. To compare optimization approaches, we do not require a complete characterization of the random variable $P(X^i)$, but we are only interested in the distribution of the quality metrics (see below). Statistical tests are performed for a chosen iteration to assess the results.

Quality Indicators. For assessing the quality of solutions, we use a modified, symmetric coverage indicator (based on the asymmetric indicator defined by Zitzler and Thiele (1999)) defined as follows. Let P_1 and P_2 be the Pareto fronts to compare. Furthermore, let $P_1^* \subseteq P_1$ be the Pareto front subset of

P_1 that is not dominated by any candidates in P_2, i.e. the candidates of P_1 that are also the Pareto front of the union of both P_1 and P_2.

Then, our coverage indicator $\mathcal{C}^*(P_1, P_2)$ is defined as the share of candidates in $P_1^* \cup P_2^*$ that come from P_1:

$$\mathcal{C}^*(P_1, P_2) := \frac{|P_1^*|}{|P_1^*| + |P_2^*|} \in [0, 1]$$

If $\mathcal{C}^*(P_1, P_2) > 0.5$ then P_1 is considered better than P_2 because P_1 has more candidates not dominated by P_2. Note that $\mathcal{C}^*$ is symmetric as $\mathcal{C}^*(P_1, P_2) = 1 - \mathcal{C}^*(P_2, P_1)$.

The standard coverage indicator may be misleading if the Pareto fronts overlap each other with varying distances to the true optimal Pareto front. Thus, we additionally use the additive binary ϵ indicator presented by Zitzler et al. (2002), which gives a factor by which a Pareto front P_1 is worse than another P_2 with respect to all objectives. It determines the minimum factor ϵ by which the Pareto front P_1 needs to be moved so that it dominates P_2.

Using this indicator, the distance of the Pareto fronts is taken into account as well. Note, however, that the distance measure is subject to the scales of the objectives and thus can be misleading in a different way. For example, availability and costs are measured on quite different scales: Candidates have values close to 1 (e.g., 1 - 3E-4) for reliability and large values (e.g., 1000 K€) for costs. In such a setting, any small change in cost will be valued much more by the ϵ indicator than a comparably large change in reliability.

To mitigate this problem to a certain extent, we normalize the Pareto fronts before determining the ϵ indicator. For the required reference point, we use, for each objective, the largest value found for all candidates. Let z be this reference point and $N(P_1, z)$ and $N(P_2, z)$ be the so normalized Pareto fronts based on P_1 and P_2.

Then, by $\mathcal{E}(P_1, P_2)$ we denote the additive ϵ indicator $I_{\epsilon+}$ as defined by Zitzler et al. (2002) and as implemented in the JMetal framework (cf. Durillo and Nebro (2011)) using the normalized Pareto fronts $N(P_1, z)$ and $N(P_2, z)$ as an input:

$$\mathcal{E}(P_1, P_2) := I_{\epsilon+}(N(P_1, z), N(P_2, z))$$

Front P_1 is superior to P_2 if $\mathcal{E}(P_1, P_2) < 1$ and $\mathcal{E}(P_2, P_1) > 1$ (cf. Zitzler et al. (2002)). As the indicator is not symmetric, situations may arise where $\mathcal{E}(P_1, P_2) < 1$ and $\mathcal{E}(P_2, P_1) < 1$, in which case the two fronts overlap and none is objectively better. In this case, $\mathcal{E}(P_1, P_2) < \mathcal{E}(P_2, P_1)$ indicates that P_1 is better with respect to the reference point z used for normalization.

However, a reference point induces a preference model. This preference model might not reflect the actual preferences of the decision maker. For example, the range of response time values found by the optimization might be considered wide by a human decision maker because values range from 1 second to 300 seconds while only values from 1 second to 3 seconds are acceptable solutions. In such situations, a response time change within the interesting interval $[1, 3]$ will be valued only little by the ϵ indicator, as it is relatively small with respect to the maximum value of 300 seconds. Thus, the resulting ϵ indicator values need to be interpreted carefully. Integrating preference models into the ϵ indicator could encounter this problem, but is out of scope of this work.

Time Savings. To assess the time savings achieved by the hybrid approach, we define a novel speed-up metric based on the coverage indicator to compare the time efficiency of two optimization techniques. We do not use the ϵ indicator $\mathcal{E}(P_1, P_2)$ here because it is asymmetric and thus cannot be readily used to compare the fronts over time.

The time savings metric $\mathcal{T}$ determines how many iteration steps earlier one optimization run has found a solution with equivalent quality. Because each iteration has a similar duration, this measures the computational effort of a run while is independent of execution time measurement errors such as additional load on the executing machine.

To compare two runs A and B, we first compare which run had the better results at the final iteration i_{max}. Let $S \in \{A, B\}$ be the superior run, defined as follows:

$$S = \begin{cases} A & \text{if A is superior or equivalent, i.e. if } \mathcal{C}^*(P(A^{i_{max}}), P(B^{i_{max}})) \geq 0.5 \\ B & \text{else} \end{cases}$$

The worse run is denoted as W with $W \in \{A, B\}, W \neq S$.

Then, we determine the smallest iteration s in which the better run S has a Pareto front $P(S^s)$ that is superior or equivalent to the results of the slower run W at the final iteration i_{max} (front $P(W^{i_{max}})$). This means that we determine the smallest iteration s so that $\mathcal{C}^*(P(S^s), P(W^{i_{max}})) > 0.5$.

For a fair comparison, we also determine the smallest iteration w in which run W has already found a front $P(W^w)$ that is equivalent to the front $P(W^{i_{max}})$: $\mathcal{C}^*(P(W^w), P(W^{i_{max}})) \geq 0.5$. Then, run S has found an equivalent solution $w - s$ iterations earlier.

Let $x \in \{s, w\}$ denote the iteration of A used for the comparison, and $y \in \{s, w\}$ denote the iteration of B used for the comparison. That is, if A was superior, $x := s$ and $y := w$ and vice versa if B was superior.

To compare the speed of the hybrid approach and the pure evolutionary, we also take into account the time for the CPLEX solver t by considering how many iterations the pure evolutionary optimization can complete while the CPLEX solver is running. Let d be the duration of the pure evolutionary optimization. Then, the number of iterations c that can be executed in time t is $c = \frac{i_{max}}{d} * t$, where $\frac{i_{max}}{d}$ is the number of iterations the pure evolutionary optimization can complete in one unit of time.

Then, $\mathcal{T}$ is defined as the absolute runtime improvement of run A over run B with respect to $\mathcal{C}^*$ measured as number of iterations:

$$\mathcal{T}(A, B) = y - x - c$$

$\mathcal{T}(A, B)$ is positive if run A is superior, and negative if run B is superior.

4.3.2. Results

To visualize the results of the approach, Fig. 8 shows example runs for availability and costs optimization. The found Pareto fronts of the hybrid approach and the pure evolutionary are similar for high failure probabilities, but the hybrid approach finds significantly better solutions with low probability of failure. While the pure evolutionary approach finds a higher number of solutions, the right part of its Pareto front is fully dominated by the most reliable solution found by the hybrid approach. We observe that the analytical input had already similar quality than the results found by the pure evolutionary run and that the hybrid run was able to further improve the analytical input.

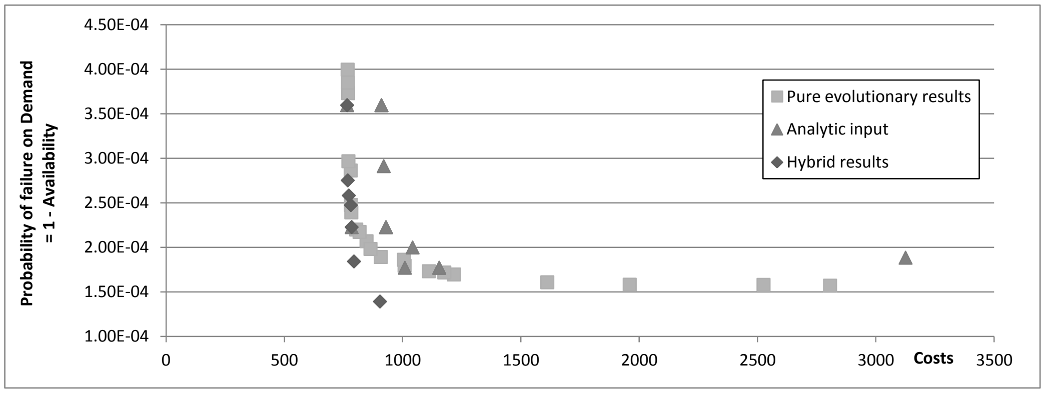

Figure 8: Example results of an availability and cost optimization run.

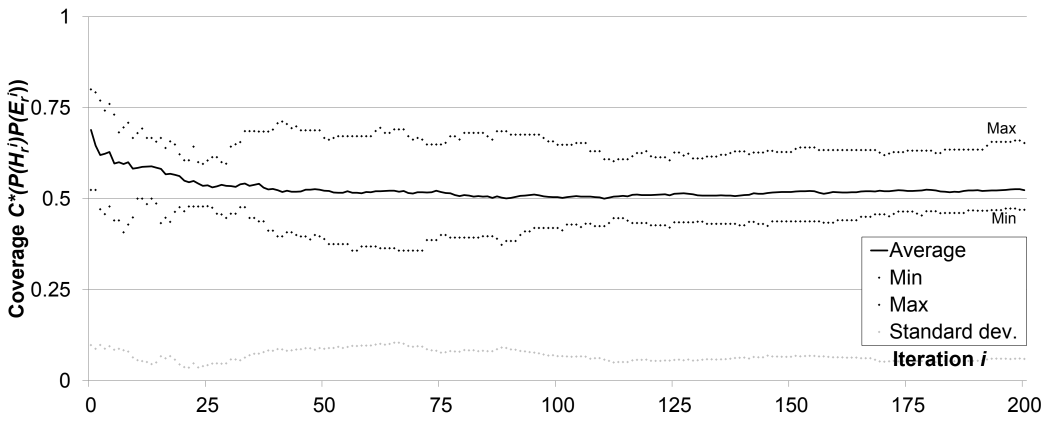

Figure 9: Pareto Front Coverage $C^*(P(H_r^i), P(E_r^i))$ of Hybrid Runs H over Pure Evolutionary Runs E for $r \in 0, ..., 9$ for 3D Optimization

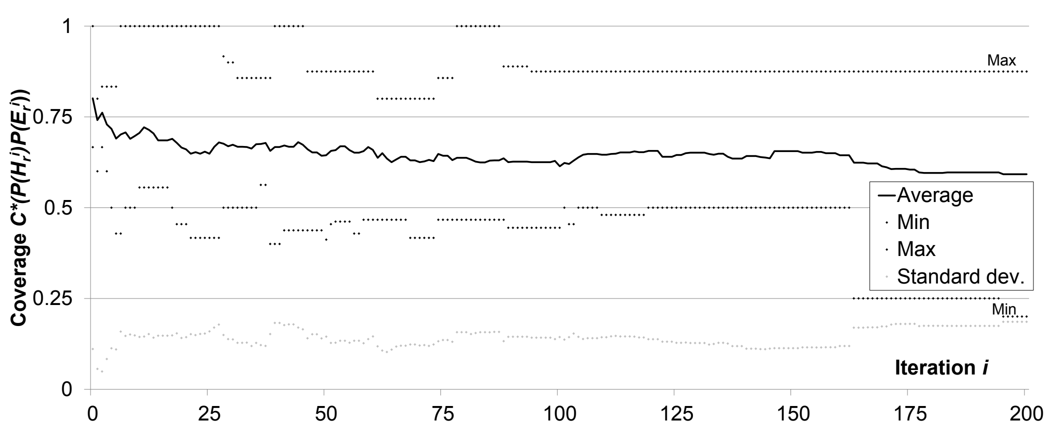

Figure 10: Pareto Front Coverage $C^*(P(H_r^i), P(E_r^i))$ of Hybrid Runs H over Pure Evolutionary Runs E for $r \in 0, ..., 9$ for Availability and Costs Optimization.

Figures 9 and 10 show the evolution of the coverage indicator over the course of the optimization runs for the availability-cost case and the three dimensional case, respectively. We observe that in both cases, the hybrid runs start with a clear advantage due to the use of the analytical starting population: The mean coverage at iteration 0 is 0.8 (availability costs case) and 0.69 (three dimensional case). The pure evolutionary optimization is able to recover part of this benefit during the course of the optimization, but, especially in the availability-costs case, does not reach the same quality of the results.

The results for the ϵ indicator are shown in Fig. 11. All results for all runs for $\mathcal{E}(H, E)$ and $\mathcal{E}(E, H)$ are smaller than 1, so none of the fronts is objectively superior to the others. Additionally, the values determined by the ϵ indicator are similar (mostly between 0.01 and 0.1). For future work, it might be interesting to incorporate preference models to allow for a more detailed interpretation of these results.

The results for the speed-up metric $\mathcal{T}(H, E)$ for final iteration $i_{max} = 200$ are shown in Fig. 12. We observe that in most cases, the hybrid approach H was faster than the pure evolutionary approach E. The average speed-up is $\mathcal{T}(H, E) = 57.8$ for the availability-cost case and $\mathcal{T}(H, E) = 3.7$ for the three dimensional case. The best speed-up is thus achieved in the availability-cost case, where the average speed-up in relation to iteration y (calculated as the average of $\frac{y-x-c}{y}$) is 28%.

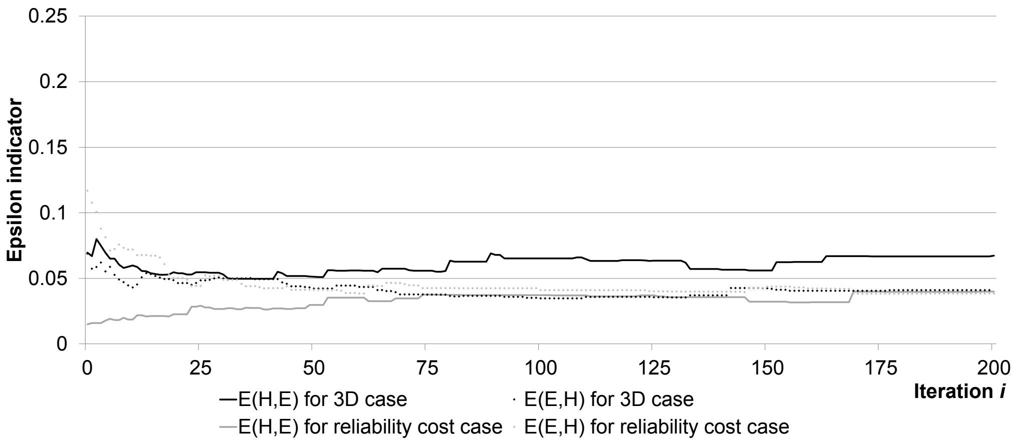

Figure 11: Epsilon indicator $\mathcal{E}$ Comparing Hybrid Runs H and Pure Evolutionary Runs E (Averaged over Runs $r \in 0, ..., 9$).

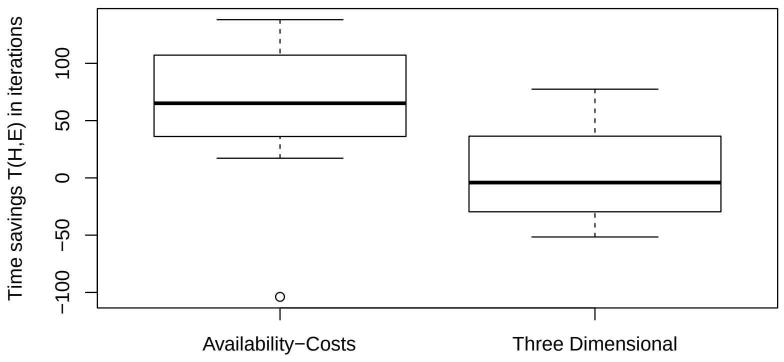

Figure 12: Time savings $\mathcal{T}(H, E)$ of hybrid runs. The thick lines denote the median, the boxes denote the 75% and 25% quantile, the whiskers extend to the most extreme data point which is no more than 1.5 times the interquartile range from the box, and the circle denotes an outlier outside the whiskers.

To test the significance of the results for the availability cost case, we used a paired, one-sided T-test as performed by the `t.test` method of the R tool on the iterations x and y found for each of the 10 pairs of runs, because we can assume that the duration is approximately normally distributed. The null hypothesis is that the mean difference of the iteration $y - x - c$ is zero or smaller. This null hypothesis is rejected in favour of the alternative hypothesis that the difference is larger than zero, i.e. the hybrid runs are faster (p-value $= 0.012$).

The results for the three dimensional case are indifferent and not significant in any direction (p-value is 0.389).

To conclude, our results show that the hybrid step provides the evolutionary optimization with a better-than-random starting population (first iteration in Figures 10 and 9), which is a clear advantage for the optimization in the availability-cost case (average speed-up of 28%).

In the three-dimensional case, although the starting population is advantageous, the speed-up at iteration 200 is smaller and not significant. Here, the parameter configuration of the evolutionary algorithms (such as mutation rate and crossover rate) might have favoured fast evolution instead of intensification of the results. This would mean that the pure evolutionary runs can use the time to catch up with the hybrid runs, while the hybrid runs cannot effectively improve upon the found solutions due to e.g. too much disruption by too aggressive crossover operators. However, this possible explanation needs further investigation, which is subject to our future work.

5. Related Work

Our approach is based on software performance prediction (Smith and Williams (2002); Balsamo et al. (2004); Koziolek (2010)) and architecture-based software availability analysis (Gokhale (2007)). We categorize closely related approaches into: (i) scenario based SA analysis, (ii) rule-based approaches, and (iii) metaheuristic-based approaches.

Scenario-based SA analysis approaches: The definition of a SA model can embody not only the software qualities of the resulting system, but also the trade-offs decisions taken by designers (Bass et al. (2003); Clements and Northrop (2001); Yang et al. (2009)). The efforts to explore such trade-offs have produced the so-called scenario-based architecture analysis methods, such as SAAM and ATAM (Kazman et al. (1994, 1998)) and others reviewed by Dobrica and Niemela (2002). These methods analyse the SA with respect to multiple quality attributes exploring also trade-offs concerning software qualities in the design. The outputs of such analysis include potential risks of the architecture and the verification result of the satisfaction of quality requirements. These methods provide qualitative results and are mainly based on the experience and the skill of designers and on the collaboration with different stakeholders. With respect to these works, our goal is to provide the software architect with a tool able to analyse the multiple objective problem in a quantitative way by allowing the automatic generation of several design architectures.

Rule-based approaches: Xu (2008) presents a semi-automated approach to find configuration and design improvement on the model level. Based on a LQN model, performance problems (e.g., bottlenecks, long paths) are identified in a first step. Then, rules containing performance knowledge are applied to the detected problems.

McGregor et al. (2007) have developed the ArchE framework. ArchE assists the software architect during the design to create architectures that meet quality requirements. It helps to create architectural models, collects requirements (in form of scenarios), collects the information needed to analyse the quality criteria for the requirements, provides the evaluation tools for modifiability or performance analysis, and suggests improvements.

Cortellessa et al. (2012) proposed an approach for automated feedback generation based on software performance antipatterns. They encode antipatterns and their solution as logical predicates referring to a software architecture model and performance evaluation results. Based on these predicates, refactored software architecture models can be suggested.

Kavimandan and Gokhale (2009) present an approach to optimize component allocation in the context of distributed real-time embedded component-based systems. They use heuristic rules to deploy components together that have a compatible configuration. In total, only allocation is considered as a degree of freedom, but the authors also mention that their approach could be combined with other approaches.

All rule-based approaches share a common limitation. The model can only be changed as defined by the improvement rules. However, especially performance is a complex and cross-cutting quality criterion. Thus, optimal solutions could lie on search paths not accessible by rules. At the same time, the encoded knowledge is complementary for an optimization approach like presented in this paper and could be integrated into the evolutionary optimization step as tactic operators (Koziolek et al. (2011a)).

Metaheuristic-based approaches: Aleti et al. (2009) present a generic framework to optimize architectural models with evolutionary algorithms for multiple arbitrary quality attributes. As a single degree of freedom, they vary the deployment of components to hardware nodes.

Canfora et al. (2005) optimize service composition costs using genetic algorithms while satisfying SLA constraints. Services are assumed to have fixed performance metrics that do not change for changing composition. Only service selection is considered as a degree of freedom, and trade-offs with other quality criteria are not considered.

Menascé et al. (2010) have developed the SASSY framework for generating service-oriented architectures based on quality requirements. Based on an initial model of the required service types and their communication, SASSY generates an optimal architecture by selecting the best services and potentially adding patterns such as replication or load balancing. As the allocation of components is irrelevant in SASSY's service architecture, the quality evaluations are simpler and allocation degrees of freedom cannot be considered. Thus, the approach is not suitable for component-based architectures in general.

To summarize, these metaheuristics are limited to a given set of degrees of freedom, whereas our evolutionary step is extendible to many degrees of freedom that can be even specified by the user (Koziolek and Reussner (2011)). None of the approaches considers a combination with analytical optimization or other means to create a superior starting population.

6. Conclusions

In this paper, we extended our hybrid approach for multi-attribute QoS optimization of component based software systems by providing availability-cost optimization and three-dimensional performance-availability-cost optimization. The core idea of our approach is the combination of analytical optimization of a simplified search problem with evolutionary optimization for refining the results with more accurate quality evaluation and the full search space. In our case study, we show that the hybrid approach can speed-up the software architecture optimization compared to pure evolutionary optimization: In the availability-costs optimization, we observed a significant speed-up of 28%. Hence, the integration of the analytical and evolutionary approaches is effective.

The proposed approach can lead both to a reduction of development costs and to an improvement of the quality of the final system, because an automated and efficient search is able to identify more and better design alternatives, and allows the software architect to make optimal trade-off decisions.

Future work will validate the overall approach by considering real industrial case studies. We will also extend the analytical problem formulation in order to consider applications with parallel components execution and/or which can be modelled by means of closed queueing networks. Furthermore, the evolutionary search will be implemented as a parallel algorithm and an automated stop criterion will be developed. Additional quality metrics will be also considered and the optimization of cloud-based systems will be also investigated.

Appendix

In this section we show how the analytical optimization problems (P1)-(P4), (PC), (PT), and (PA_l) are as difficult as the binary knapsack (KP) and multiple dimension knapsack (MKP) problems, which are NP-hard (Wolsey (1998)). Hence, the analytical optimization step is also NP-hard.

The classical 0-1 Knapsack Problem (KP) is to pick up items from a knapsack for maximum total value, so that the total resource does not exceed the resource constraint $W > 0$ of the knapsack. Let there be n items with values $v_1, v_2, \ldots, v_n$ and the corresponding resources required $w_1, w_2, \ldots, w_n$ $(\forall j \in [1, n], w_j > 0)$.

Mathematically the KP can be formalized as:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} v_j x_j \\ \sum_{j=1}^{n} w_j x_j \quad & \leq \quad W \\ x_j \quad & \in \quad \{0,1\} \end{aligned}$$

By relaxing exclusive sets constraints, (P1) can be rewritten as:

$$\min \left(\tilde{C} + \sum_{j\in\mathcal{J}} \delta_j^c \cdot x_j \right) = \min \sum_{j\in\mathcal{J}} \delta_j^c \cdot x_j$$

subject to:

$$\sum_{i\in\mathcal{I}} \pi_i \cdot (\tilde{t}_i + \sum_{j\in\mathcal{J}} \delta_{j,i}^t x_j) \leq \overline{T} \Leftrightarrow \sum_{j\in\mathcal{J}} \left(\sum_{i\in\mathcal{I}} \pi_i \cdot \delta_{j,i}^t \right) x_j \leq \overline{T} - \sum_{i\in\mathcal{I}} \pi_i \cdot \tilde{t}_i$$

Indeed, $\tilde{C}$ is a constant independent of the decision variables and can be dropped by the objective function. By setting $v_j = -\delta_j^c$, $w_j = \sum_{i\in\mathcal{I}} \pi_i \cdot \delta_{j,i}^t$ and recalling that for any optimization problem $\max_{x\in X} f(x) = -\min_{x\in X} f(x)$, (P1) can be reduced to a KP and hence is NP-hard.

With the same arguments, by relaxing the exclusive set constraints, (P2) can be reduced to a KP setting $v_j = -\sum_{i \in \mathcal{I}} \pi_i \cdot \delta^t_{j,i}$, $w_j = \delta^c_j$, and $W = \overline{C} - \tilde{C}$.

Similarly, (P3) becomes a KP by setting $v_j = -\delta^c_j$, $w_j = -\sum_{i \in \mathcal{EP}_l} ln(\delta^a_{j,i,l})$, and $W = -ln(\overline{A}) + \sum_{i \in \mathcal{EP}_l} ln(\tilde{a}_{i,l})$, while for (P4) is sufficient to set $v_j = ln(\delta^a_{j,i,l})$, $w_j = \delta^c_j$, and $W = \overline{C} - \tilde{C}$.

To show that (PC), (PT), and (PA_l) are NP-hard we need to consider the multi-dimensional formulation of the knapsack problem.

A multiple-dimension knapsack problem (MKP) is one kind of knapsack where the resources are multi-dimensional, i.e. there are multiple resource constraints for the knapsack, for example the weight but also the size. The multi-dimensional variant was shown to be NP-complete around 1980 (see, e.g., Wolsey (1998)).

Formally, let there be n items, let v_j be the value of the j-th item, $w_{j,k} > 0$ the amount of resource k required by the j-th item, and $W_k > 0$ the amount of the k resource. Then the MKP is:

$$\begin{array}{rcl} \max & & \sum_{j=1}^{n} v_j x_j \\ \sum_{j=1}^{n} w_{j,k} x_j & \leq & W_k; \quad \forall k \\ x_j & \in & \{0,1\} \end{array}$$

Proceeding as before, by relaxing exclusive set constraints (PC) becomes a bi-dimensional knapsack by setting $v_j = -\delta^c_j$, $w_{j,1} = \sum_{i \in \mathcal{I}} \pi_i \cdot \delta^t_{j,i}$, $w_{j,2} = -ln(\delta^a_{j,i,l})$, $W_1 = \overline{T} - \sum_{i \in \mathcal{I}} \pi_i \cdot \tilde{t}_i$, and $W_2 = -ln(\overline{A}) + \sum_{i \in \mathcal{EP}_l} ln(\tilde{a}_{i,l})$. The reduction of (PT) and (PA_l) to MKP is also straightforward.

Acknowledgements.

The work reported in this paper has been partially supported by the EU FP7 Q-ImPrESS project and by the Swiss Research Foundation (SNF) as a Forschungskredit of the University of Zurich, grant no. 53510401.

References

Aleti, A., Björnander, S., Grunske, L., Meedeniya, I., 2009. Archeopterix: An extendable tool for architecture optimization of AADL models. In: Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES 2009). IEEE Computer Society, pp. 61–71.

Ardagna, D., Pernici, B., June 2007. Adaptive service composition in flexible processes. IEEE Trans. on Soft. Eng. 33 (6), 369–384.

Avizienis, A., Laprie, J. C., Randell, B., Landwehr, C., Jan.-March 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing 1 (1), 11 – 33.

Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M., May 2004. Model-Based Performance Prediction in Software Development: A Survey. IEEE Trans. on Software Engineering 30 (5), 295–310.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, Second Edition. Addison-Wesley, Reading, MA, USA.

Becker, S., Koziolek, H., Reussner, R., 2009. The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82, 3–22.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys 35 (3), 268–308.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., Steece, B., 2000. Software Cost Estimation with Cocomo II. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Brosch, F., Koziolek, H., Buhnova, B., Reussner, R., 2012. Architecture-based reliability prediction with the palladio component model. Transactions on Software Engineering 99 (PrePrints), doi: 10.1109/TSE.2011.94.

Canfora, G., Penta, M. D., Esposito, R., Villani, M. L., 2005. An approach for QoS-aware service composition based on genetic algorithms. In: Beyer, H.-G., O'Reilly, U.-M. (Eds.), Proc. of Genetic and Evolutionary Computation Conference (GECCO). ACM, pp. 1069–1075.

Clements, P. C., Kazman, R., Klein, M., 2001. Evaluating Software Architectures. SEI Series in Software Engineering. Addison-Wesley.

Clements, P. C., Northrop, L., Aug. 2001. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley.

Coello, C. A. C., 1999. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems 1, 269–308.

Cortellessa, V., Di Marco, A., Trubiani, C., 2012. An approach for modeling and detecting software performance antipatterns based on first-order logics. Software and Systems Modeling, 1–42.

Costa, C., Cunha, I., Borges, A., Ramos, C., Rocha, M., Almeida, J., Ribeiro-Neto, B., 2004. Analyzing Client Interactivty in Streaming Media. In: Proc. 13th ACM International World Wide Web Conference (WWW).

de Gooijer, T., Jansen, A., Koziolek, H., Koziolek, A., 2012. An industrial case study of performance and cost design space exploration. In: Kurian John, L., Krishnamurthy, D. (Eds.), Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering (ICPE 2012). Boston, USA, iCPE Best Industry-Related Paper Award.

Deb, K., 2001. Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester, UK.

Deb, K., Agrawal, S., Pratap, A., Meyarivan, T., 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Parallel Problem Solving from Nature PPSN VI. Vol. 1917/2000. Springer, pp. 849–858.

Dobrica, L., Niemela, E., Jul 2002. A survey on software architecture analysis methods. IEEE Trans. on Software Engineering 28 (7), 638–653.

Durillo, J. J., Nebro, A. J., 2011. jmetal: A java framework for multi-objective optimization. Advances in Engineering Software 42 (10), 760 – 771.

Ehrgott, M., 2005. Multicriteria Optimization. Springer.

Franks, G., Li, L., 2012. Efficiency improvements for solving layered queueing networks. In: ICPE. pp. 279–282.

Franks, G., Omari, T., Woodside, C. M., Das, O., Derisavi, S., 2009. Enhanced modeling and solution of layered queueing networks. IEEE Trans. on Software Engineering 35 (2), 148–161.

Gokhale, S. S., January-March 2007. Architecture-based software reliability analysis: Overview and limitations. IEEE Trans. on Dependable and Secure Computing 4 (1), 32–40.

IBM ILOG, 2010. IBM ILOG CPLEX.
http://www-01.ibm.com/software/integration/optimization/cplex/about/.

Intel Corporation, 2010. Intel®processor price list, effective feb 8th, 2010. http://www.intc.com/priceList.cfm, last visit March 10th, 2010.

Kavimandan, A., Gokhale, A. S., 2009. Applying model transformations to optimizing real-time QoS configurations in DRE systems. In: Quality of Softw. Architectures. Springer, pp. 18–35.

Kazman, R., Bass, L., Abowd, G., Webb, M., May 1994. SAAM: A method for analyzing the properties of software architectures. In: Intl. Conf. on Softw. Engineering. IEEE, pp. 81–90.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carrière, S., 1998. The architecture tradeoff analysis method. In: Intl. Conf. on Engineering of Complex Computer Systems. IEEE, pp. 68–78.

Kleinrock, L., 1975. Queueing Systems. Vol. I: Theory. Wiley Interscience, (Published in Russian, 1979. Published in Japanese, 1979. Published in Hungarian, 1979. Published in Italian 1992.).

Koziolek, A., Jul. 2011. Automated improvement of software architecture models for performance and other quality attributes. Ph.D. thesis, Institut für Programmstrukturen und Datenorganisation (IPD), Karlsruher Institut für Technologie, Karlsruhe, Germany. URL `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000024955`

Koziolek, A., Ardagna, D., Koziolek, H., Mirandola, R., Reussner, R., 2012. Details on case study for the extended hybrid optimization approach. sdqweb.ipd.kit.edu/wiki/PerOpteryx/Hybrid_Optimization_Case_Study_2012.

Koziolek, A., Koziolek, H., Reussner, R., 2011a. Peropteryx: automated application of tactics in multi-objective software architecture optimization. In: Crnkovic, I., Stafford, J. A., Petriu, D. C., Happe, J., Inverardi, P. (Eds.), Joint proceedings of the Seventh International ACM SIGSOFT Conference on the Quality of Software Architectures and the 2nd ACM SIGSOFT International Symposium on Architecting Critical Systems (QoSA-ISARCS 2011). ACM, New York, NY, USA, pp. 33–42.

Koziolek, A., Reussner, R., Jun. 2011. Towards a generic quality optimisation framework for component-based system models. In: Crnkovic, I., Stafford, J. A., Bertolino, A., Cooper, K. M. L. (Eds.), Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering. CBSE '11. ACM, New York, NY, USA, New York, NY, USA, pp. 103–108.

Koziolek, H., 2010. Performance evaluation of component-based software systems: A survey. Performance Evaluation 67 (8), 634–658, special Issue on Software and Performance.

Koziolek, H., Schlich, B., Bilich, C., Weiss, R., Becker, S., Krogmann, K., Trifu, M., Mirandola, R., Koziolek, A., 2011b. An industrial case study on quality impact prediction for evolving service-oriented software. In: Taylor, R. N., Gall, H., Medvidovic, N. (Eds.), Proceeding of the 33rd international conference on Software engineering (ICSE 2011), Software Engineering in Practice Track. pp. 776–785.

Krishnamurthy, D., Rolia, J., Majumdar, S., 2006. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems 32(11).

Liu, Z., Squillante, M., Wolf, J. L., October 2001. On Maximizing Service-Level-Agreement Profits. In: Proc. of ACM Eletronic Commerce Conference.

Lukasiewycz, M., Glaß, M., Reimann, F., Teich, J., 2011. Opt4J: a modular framework for meta-heuristic optimization. In: GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, Dublin, Ireland, pp. 1723–1730.

Martens, A., Ardagna, D., Koziolek, H., Mirandola, R., Reussner, R., 2010a. A Hybrid Approach for Multi-Attribute QoS Optimisation in Component Based Software Systems. In: Heineman, G., Kofron, J., Plasil, F. (Eds.), Research into Practice - Reality and Gaps (Proceedings of the 6th International Conference on the Quality of Software Architectures, QoSA 2010). Vol. 6093 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, pp. 84–101.

Martens, A., Koziolek, H., Becker, S., Reussner, R. H., 2010b. Automatically improve software models for performance, reliability and cost using genetic algorithms. In: WOSP/SIPEW International Conference on Performance Engineering. ACM.

Martens, A., Koziolek, H., Prechelt, L., Reussner, R., 2011. From monolithic to component-based performance evaluation of software architectures. Empirical Software Engineering 16 (5), 587–622.

McGregor, J. D., Bachmann, F., Bass, L., Bianco, P., Klein, M., 2007. Using arche in the classroom: One experience. Tech. Rep. CMU/SEI-2007-TN-001, Software Engineering Institute, Carnegie Mellon University.

Menascé, D. A., Ewing, J. M., Gomaa, H., Malex, S., Sousa, J. a. P., 2010. A framework for utility-based service oriented design in SASSY. In: Proc. of Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW). ACM, pp. 27–36.

Musa, J. D., Iannino, A., Okumoto, K., 1987. Software reliability: measurement, prediction, application. McGraw-Hill, Inc., New York, NY, USA.

Paxson, V., Floyd, S., 1995. Wide Area Traffic: the Failure of Poisson Modeling. IEEE Trans. on Networking 3 (2), 226–244.

Ranjan, S., Knightly, E., 2008. High-Performance Resource Allocation and Request Redirection Algorithms for Web Clusters. IEEE Trans. on Parallel and Distr. Systems 19 (9), 1186–1200.

Smith, C. U., Williams, L. G., 2002. Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley.

Williams, L. G., Smith, C. U., 2003. Making the Business Case for Software Performance Engineering. In: Proceedings of the 29th International Computer Measurement Group Conference, December 7-12, 2003, Dallas, Texas, USA. Computer Measurement Group, pp. 349–358.

Wolsey, L., 1998. Integer Programming. John Wiley and Sons.

Wu, X., Woodside, M., 2004. Performance Modeling from Software Components. SIGSOFT Softw. Eng. Notes 29 (1), 290–301.

Xu, J., 2008. Rule-based automatic software performance diagnosis and improvement. In: International Workshop on Software and Performance. ACM, pp. 1–12.

Yang, J., Huang, G., Zhu, W., Cui, X., Mei, H., 2009. Quality attribute tradeoff through adaptive architectures at runtime. Journal of Systems and Software 82 (2), 319–332.

Zhang, Q., Cherkasova, L., Mi, N., Smirni, E., 2008. A Regression-Based Analytic Model for Capacity Planning of Multi-Tier Applications.

Zitzler, E., Knowles, J., Thiele, L., 2008. Quality Assessment of Pareto Set Approximations. Vol. 5252 of LNCS. Springer-Verlag, Berlin, pp. 373–404.

Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans. Evolutionary Computation 3 (4), 257–271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., da Fonseca, V. G., 2002. Performance assessment of multiobjective optimizers: An analysis and review. IEEE Trans. on Evolutionary Computation 7, 117–132.