A Multi-model Optimization Framework for the Model Driven Design of Cloud Applications

Danilo Ardagna, Giovanni Paolo Gibilisco,
Michele Ciavotta, and Alexander Lavrentev

Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, Italy
{danilo.ardagna,giovannipaolo.gibilisco,
michele.ciavotta}@polimi.it,
alexander.lavrentev@mail.polimi.it

Abstract. The rise and adoption of the Cloud computing paradigm had a strong impact on the ICT world in the last few years; this technology has now reached maturity and Cloud providers offer a variety of solutions and services to their customers. However, beside the advantages, Cloud computing introduced new issues and challenges. In particular, the heterogeneity of the Cloud services offered and their relative pricing models makes the identification of a deployment solution that minimizes costs and guarantees QoS very complex. Performance assessment of Cloud based application needs for new models and tools to take into consideration the dynamism and multi-tenancy intrinsic of the Cloud environment. The aim of this work is to provide a novel mixed integer linear program (MILP) approach to find a minimum cost feasible cloud configuration for a given cloud based application. The feasibility of the solution is considered with respect to some non-functional requirements that are analyzed through multiple performance models with different levels of accuracy. The initial solution is further improved by a local search based procedure. The quality of the initial feasible solution is compared against first principle heuristics currently adopted by practitioners and Cloud providers.

1 Introduction

The rise and consolidation of the Cloud computing paradigm had a significant impact on the ICT world in recent years. Cloud has now reached maturity; many are the technologies and services supplied by various providers, resulting in an already highly diversified market. Tools for fast prototyping, enterprise developing, testing and integration are offered, delegating to Cloud providers all the intensive tasks of management and maintenance of the underlying infrastructure. However, besides the unquestionable advantages, Cloud computing introduced new issues and important challenges in application development. In fact, current Cloud technologies and pricing models can be so different and complex that looking for the solution that minimizes costs while guaranteeing an adequate performance, might result in a tremendous task. To carry out such a labor, application designer should consider multiple architectures at once and be able to evaluate costs and performance for each of them. Moreover, while information on architectures and costs are openly available, the performance assessment aspect turns

C. Le Goues and S. Yoo (Eds.): SSBSE 2014, LNCS 8636, pp. 61–76, 2014.

out to be a far more complicated concern because Cloud environments are often multi-tenant and their performance can vary over time, according to the congestion level and the competition for resources among the different applications. Although some analytical performance models have been proposed to attain an approximate assessment of software systems performance, there is, until now, no attempt to extend those models for taking into account the specificity of Cloud solutions. Consider, for example, Palladio Component Model (PCM) and Palladio Bench [10] for Quality of Service (QoS) evaluation. PCM is a Domain Specific Language (DSL) for the description of software architecture, resource and analysis of non-functional requirements but it is limited to enterprise systems, QoS can be assessed only for the peak workload, and it lacks support for Cloud systems. On the contrary, Cloud based systems are dynamic and time-dependent parameters have to be considered to assess performance and costs. It should also be noticed that cost and performance assessments are just one side of the coin. On the other side, the problem of quickly and efficiently explore the space of possible Cloud configurations in automatic or semi-automatic way also exists.

The aim of this work is to propose and validate a novel Mixed Integer Linear Program (MILP) designed to quickly find a minimum-cost Cloud configuration for a certain application, where the feasibility of a solution is considered according to some non-functional constraints expressed in the model. To realize an accurate model, the most common Cloud systems have been analyzed deriving general meta-models and parameter values. Those meta-models have been expressed by means of a Cloud-based extension of the PCM. This extension, presented for the first time here, is able to express different kinds of QoS constraints and time-dependent profiles for most important performance parameters. The proposed MILP is finally validated against a local search based metaheuristic also designed to explore the space of alternative Cloud configurations. The MILP solution is also compared with first principle heuristics currently adopted by practitioners and Cloud providers.

The remainder of the paper is organized as follows. In Section 2 the PCM proposed extension is briefly introduced. The optimization model is introduced in Section 3, whereas Section 4 illustrates the experimental campaign the optimization model underwent and analyzes the outcomes. The State-of-the-art analysis is reported in Section 5; conclusions are finally drawn in Section 6.

2 Background: Architecture Modeling and Analyses

In order to model the application under analysis, we extended the Palladio Component Model [10]. The PCM language allows developers to represent different aspects of the application by building specific diagrams. Figure 1 shows the main components of Apache Open For Business (OfBiz), our case study application; the figure is a summary of the information that can be expressed via our PCM extension, represented in a UML-like notation.

OfBiz [1] is an enterprise open source automation software developed by the Apache software foundation and adopted by many companies. We focus here on the E-Commerce

[1] `http://ofbiz.apache.org/`

functionality of OfBiz since it is a good candidate to be implemented with Cloud technology. The left most activity diagram models the behavior of users of the system, in this example on average 70% of users will access the application to purchase some product while the remaining 30% will check the status of a scheduled order. The incoming workload is expressed in number of requests per second. Our extension allows to specify a workload profile of 24 hours. All requests generated by users are served by the *Request Handler* component. The behavior of the *checkout* functionality is described by the activity diagram associated with the request handler. To serve a checkout request, the front-end needs to perform some internal computation (e.g., calculate the shipping price), whose impact on physical resources hosting the system is shown as *Demand*, and interact with some components hosted on the back-end. In particular the request handler interacts with the *Database* component to check the availability of the desired item and with the *Payment* component to check the validity of the credit card information specified by the user. The topmost part of the diagram shows that the request handler component is deployed alone in the front-end tier while the database and the payment service are co-located in the same back-end tier.

The standard PCM allows application designers to build diagrams with this kind of information and derive (for every time slot) a Layered Queuing Network (LQN) model from them. LQN models can then be solved analytically or by means of a simulation in order to derive performance metrics. As opposed to [26] we suppose that the component allocation to application tiers has already been chosen by the software developer therefore will not be changed by the optimization process. Multiple QoS metrics can

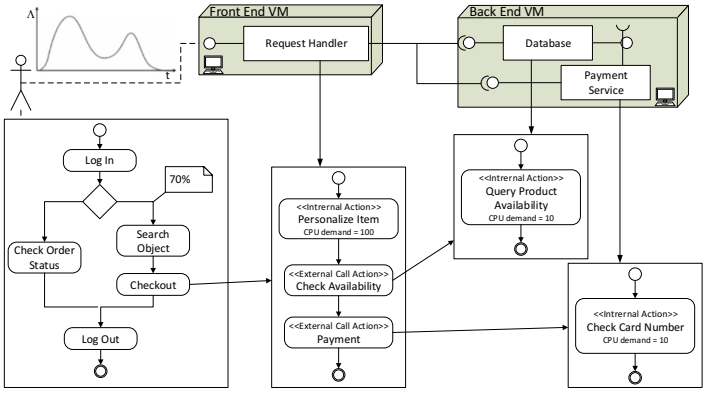


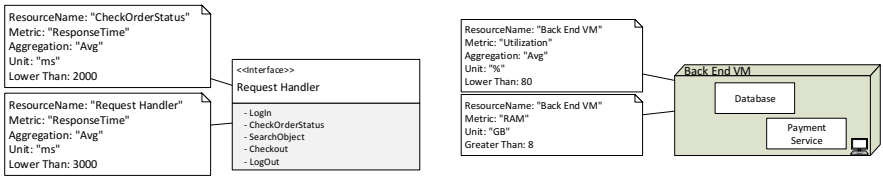
Fig. 1. OfBiz Application Example

be derived from the analysis of LQN models, in this work our focus is on response time and cost.

In a Cloud environment, infrastructural *Costs* are also difficult to compute, since the pricing policy offered by Cloud providers is very heterogeneous. In this work we refer to cost as to the sum of the prices of allocated resources, charged on a per-hour basis. This kind of pricing policy is a common denominator of all the most important Cloud provider offers, the main objective of this cost modeling is to show that cost related

aspects can be included in the optimization process, not to provide a comprehensive description of the costs related to any specific Cloud environment.

As main performance metric we consider server side request *response time*. We also suppose that all Virtual Machines (VMs) hosting the application components are located inside the same local network (e.g., the same availability zone in Amazon EC2) so that the communication between the different application layers does not cause a bottleneck.



(a) QoS constraints defined over an interface (b) Architectural constraints defined over an auto-scaling group

Fig. 2. Examples of different type of constraints that can be specified on the modeled application

In order to describe the Cloud environment the modeled application will run in, we made use of the meta models presented in [16]. We also extended those models in order to express constraints over the application QoS to drive the optimization process. Figure 2(a) shows two examples of QoS constraints that can be defined on a component. The topmost constraint expresses the fact that the functionality in charge of checking the status of an order should have an average response time lower than 2000 milliseconds. The other constraint is defined over the entire component and limits the average response time, computed over all the functionality offered by the component, to be lower than 3000 ms. Figure 2(b) shows another kind of constraints that can be expressed on the virtual hardware used to host the software components of the application. The example shows a constraint on the minimum amount of RAM that a VM needs to feature in order to run the two components and a constraint on the maximum allowed value of the CPU utilization.

3 Optimization Process

In this section we describe the hybrid optimization approach we propose, to solve the capacity allocation problem. As in [22], we implemented a two-steps approach. The first step consists in solving a Mixed Integer Linear Problem (MILP) in which the QoS associated to a deployment solution is calculated by means of an M/G/1 queuing model with processor sharing policy. Such performance model allows to calculate the average response time of a request in closed form. Our goal is to determine quickly an approximated initial solution through the MILP solution process which is then further improved by a local search based optimization algorithm (step 2). The aim of this algorithm is to iteratively improve the starting Cloud deployment exploring several application configurations. A more expressive performance model (LQN) is employed to derive more accurate estimations of the QoS by means of the LQNS tool [17]. Figure 3 shows the workflow of the optimization process. As explained in Section 2 the specification of

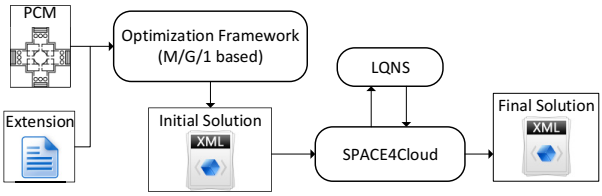


Fig. 3. Solution generation workflow

the application design is given in form of a PCM with an accompanying extension. The information contained in these models constitutes the input of the optimization problem and is passed to the optimization framework, which is the main object of this work, in order to derive an initial solution. This solution is then passed to the SPACE4Cloud tool [5], which performs an assessment of the solution using the more accurate LQN model and an heuristic optimization to derive an optimized solution.

In the remainder of the paper, Section 3.1 describes in details the capacity allocation problem faced in this work, Section 3.2 provides its MILP formulation, while Section 3.3 outlines the heuristic local search approach.

3.1 Search Problem Formulation

The aim of the optimal selection and capacity allocation problem for Cloud applications is to minimize the usage cost of Cloud resources, while satisfying some user-defined constraints. As discussed in Section 2, an application can be described as a composition of several components $\mathcal{C}$, each of them implementing a particular set of functionalities $\mathcal{K}$ with a certain resources demand. Each component is deployed into a resource pool, or application tier, $\mathcal{I}$ composed by a set of homogeneous VMs. Such a set is not static but can scale to handle variations of the incoming workload. Since, the daily workload is periodic for many applications [11], we decided to limit our analysis to a single day horizon. Many Cloud providers charge the use of VMs per hour (e.g., Amazon EC2 *on-demand* pricing scheme[2]), hence it is reasonable to split the time horizon into 24 time slots $\mathcal{T}$ of one hour each . For the sake of simplicity, in the following we consider QoS constraints predicating on application response time. In a nutshell, the problem we deal with presents two main decision aspects, first is the selection of a certain VM type $\mathcal{V}$ for each resource pool, while the second faces the way the application has to scale in order to meet the constraints, i.e., aims at determining the optimum number of VMs to be devoted to each pool at every hour of the day. The overall workload of an application is described in terms of requests per second Λ_t.

Users interact with the application by making requests, the set of possible requests is referred to as $\mathcal{K}$. Moreover, each class of requests is characterized with a probability to be executed (α_k specified in the model, see Figure 1) and by a set of components supporting its execution (i.e., its execution path [6]). Finally, we assume that requests are served according to the processor sharing scheduling policy, a typical scheduling

[2] `http://aws.amazon.com/ec2/pricing`

policy in Web and application server containers that evenly splits the workload among all the VMs of the resource pool. As for the QoS requirements the application designer can define a threshold on the average response time $\overline{R}_k$ for a set of classes $\mathcal{K}_{Avg} \subseteq \mathcal{K}$ (as depicted in Figure 2(a)).

3.2 Analytic Optimization

In the light of the considerations made so far, the optimal capacity allocation problem can be formulated as follows:

$$\min_{\mathcal{Z},\mathcal{V}} \sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} \sum_{t\in\mathcal{T}} C_{v,t} z_{i,v,t} \tag{1}$$

Subject to:

$$\sum_{v\in\mathcal{V}} w_{i,v} = 1 \qquad \forall i \in \mathcal{I} \tag{2}$$

$$w_{i,v} \leq z_{i,v,t} \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T} \tag{3}$$

$$z_{i,v,t} \leq N w_{i,v} \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{v\in\mathcal{V}} M_v w_{i,v} \geq \overline{M_i} \qquad \forall i \in \mathcal{I} \tag{5}$$

$$\sum_{v\in\mathcal{V}} (1 - \mu_{k,c}\overline{R}_{k,c} S_v) z_{i_c,v,t} \leq \mu_{k,c} G_{k,c,t} \overline{R}_{k,c} \qquad \forall k \in \mathcal{K}, \forall c \in \mathcal{C}, \forall t \in \mathcal{T} \tag{6}$$

$$\sum_{v\in\mathcal{V}} S_v z_{i_c,v,t} > G_{k,c,t} \qquad \forall k \in \mathcal{K}, \forall c \in \mathcal{C}, \forall t \in \mathcal{T} \tag{7}$$

$$z_{i,v,t} \ \ Integer \qquad \forall i \in \mathcal{I}, \forall t \in \mathcal{T} \tag{8}$$

$$w_{i,v} \ \in \{0,1\} \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V} \tag{9}$$

Where $\mathcal{T} = \{1 \ldots 24\}$ and $G_{k,c,t} = \Lambda_t \sum_{\tilde{c}\in I_c} \sum_{\tilde{k}\in\mathcal{K}} \frac{\alpha_k p_{k,\tilde{c}}}{\mu_{\tilde{k},\tilde{c}}}$.

Table 1 summarizes the list of parameters of our optimization model and Table 2 reports the decision variables.

The value expressed by (1) represents the total usage cost of the Cloud application and it is the objective function to minimize. As $w_{i,v}$ are binary decision variables (eq. 9) equal to 1 if the VM type v is assigned to the i-th resource pool, condition (2) guarantees that exactly one type can be selected for each resource pool. Equation (8) defines a set of integer variables $z_{i,v,t}$ that represent the number of VMs of type v assigned to resource pool i at time t. Condition (3) in combination with (2) and (9) guarantees that a nonempty set of VMs is assigned to each resource pool. Moreover, condition (4) imposes the set of VMs assigned to each resource pool to be homogeneous. Indeed, if $w_{i,v} = 0$ the total number of VMs of type v assigned to resource pool i is forced to be zero as well and this happens for all $v \in \mathcal{V}$ but one (eq. (2)). Besides, if $w_{i,v} = 1$ the number of VMs assigned to the resource pool i is at least 1, for eq. (3), and at most N, which is an arbitrary large integer number. Finally, equations (5) and (6)

Table 1. Optimization model parameters

System parameters

Index	
$t \in T$	time interval
$i \in \mathcal{I}$	resource pool or application tiers
$k \in \mathcal{K}$	class of request
$v \in \mathcal{V}$	type of virtual machine
Parameters	
Λ_t	number of incoming requests (workload) at time t
α_k	proportion of requests of class k in the workload
$p_{k,c}$	probability of request of class k to be served by component c
$\mu^v_{k,c}$	maximum service rate of requests of class k on component c hosted on a VM of type v
U_k	set of components serving request k
I_c	set of components that are co-located with c
$C_{v,t}$	cost of a single machine of type v at time t
M_v	memory of a virtual machine of type v
$\overline{M}_i$	memory constraint for tier i
$\overline{R}_{k,c}$	maximum average response time for the k-class of requests on component c

Table 2. Optimization model decision variables

Optimization model decision variables.

$w_{i,v}$	binary variable that is equal to 1 if the VMs type v is assigned to the i-th tier and equal to 0 otherwise
$z_{i,v,t}$	number of virtual machines of type v assigned to the i-th resource pool at time t

represent memory and QoS constraints, respectively, while (7) is the M/G/1 equilibrium condition.

As previously discussed, to evaluate the average response time of the Cloud application we model each VM container as an M/G/1 queue. However, in general, a request of class k is processed by more than a single component. Let $\Lambda_{k,t} = \alpha_k \Lambda_t$ be the incoming workload at time t for request class k and $\Lambda_{k,c,t} = p_{k,c}\Lambda_{k,t}$ the arrival rate of request class k at component c. The response time of requests in class k can then be obtained by:

$$R_{k,t} = \sum_{c \in U_k} p_{k,c} R_{k,c,t} = \sum_{c \in U_k} p_{k,c} \frac{\dfrac{1}{\mu^{\tilde{v}}_{k,c}}}{1 - \displaystyle\sum_{\tilde{c} \in I_c} \sum_{\tilde{k} \in K} \frac{\Lambda_{\tilde{k},\tilde{c},t}}{\mu^{\tilde{v}}_{k,\tilde{c}}} z_{i_c,\tilde{v},t}} \tag{10}$$

where U_k is the set of components serving class k requests and I_c represents the set of components that are co-located with c on the same VM (I_c can be obtained by standard PCM allocation diagrams, see Figure 1). In other words, the average response time is obtained by summing up the time spent by the request in each component weighted by the probability of the request to actually be processed by that component. Notice that $R_{k,c,t}$ depends on the type and number of VMs ($\tilde{v}$ and $z_{i_c,\tilde{v},t}$, respectively) the component c is allocated in at time t.

In order to simplify expression (10) we consider the lowest CPU machine a reference machine and calculate the maximum service rates for each request class and component, $\mu_{k,c}$. In this way $\mu^v_{k,c}$ can be written as:

$$\mu^v_{k,c} = \mu_{k,c}\, S_{i_c} = \mu_{k,c} \sum_v S_v\, w_{i_c,v} \tag{11}$$

where S_v is the speed ratio between the reference machine and a VM of type v, while i_c is the index of the resource pool where component c is allocated.

Let $z_{i,t} = \sum_v z_{i,v,t}$ be the number of VMs of the selected type (only one type can be selected (eq.2)) for resource pool i at time t. Therefore the following expression holds:

$$\mu^v_{k,c}\, z_{i_c,v,t} = \mu_{k,c}\, S_{i_c}\, z_{i_c,t} = \mu_{k,c} \sum_v S_v\, z_{i_c,v,t} \tag{12}$$

Under M/G/1 assumptions the response time of a request of class k processed by component c hosted on a VM of type v is given by:

$$R_{k,c,t} = \frac{\dfrac{1}{\mu_{k,c}\, S_{i_c}}}{1 - \displaystyle\sum_{\tilde{c}\in I_c} \sum_{\tilde{k}\in\mathcal{K}} \frac{\Lambda_{\tilde{k},\tilde{c},t}}{\mu_{\tilde{k},\tilde{c}}\, S_{i_c}\, z_{i_c,t}}} \tag{13}$$

By replacing (11) and (12) in (10) we can write the following constraint on the response time of requests of class k:

$$R_{k,t} = \sum_{c\in U_k} p_{k,c}\, R_{k,c,t} = \sum_{c\in U_k} p_{k,c} \frac{\dfrac{1}{\mu_{k,c}\, S_{i_c}}}{1 - \dfrac{\Lambda_t}{S_{i_c}\, z_{i_c,t}} \displaystyle\sum_{\tilde{c}\in I_c} \sum_{\tilde{k}\in\mathcal{K}} \frac{\alpha_{\tilde{k}} p_{\tilde{k},\tilde{c}}}{\mu_{\tilde{k},\tilde{c}}}} \leq \overline{R_k}, \tag{14}$$

Equation (14) is non-linear due to the presence of product $S_i\, z_{i,t}$ in the denominator; in order derive a linear model, which can be solved efficiently by MILP solvers, we explode it into a set of stricter constraints defined over the average response time of each component traversed by the request.

To do so we split the response time constraint among components of a path and take the most stringent constraint among the conditions generated by all the possible paths that the request can traverse. In other words, let:

$$r_{k,c,u} = \begin{cases} \dfrac{\dfrac{1}{\mu_{k,c}}}{\displaystyle\sum_{c\in u} \frac{1}{\mu_{k,c}}} & \text{if } c \text{ belongs to path } u \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

and let: $\overline{R}_{k,c} = \min_u\ r_{k,c,u}\ \overline{R}_k$. Instead of using constraint (13) for the response time we introduce the constraint family: $R_{k,c,t} \leq \overline{R}_{k,c}$ and after some algebra we get constraint (6).

Finally, constraint (7) represents the M/G/1 equilibrium condition obtained from (13) imposing the denominator to be greater than zero.

3.3 Local Search Optimization

The aim of this section is to provide a brief description of the optimization algorithm implemented by the SPACE4Cloud [5] tool that we used to further optimize the solution obtained by the MILP optimization problem. One of the key differences between the two optimization processes is the fact that the performance model used by SPACE4Cloud, the LQN model, is more complex and accurate than the M/G/1 models used in the analytic formulation of the optimization problem. Another differentiating factor is the way SPACE4Cloud explores the free space, in fact it uses an heuristic approach that divides the problem into two levels delegating the assignment of the type of VMs to the first (upper) level and the definition of the number of replicas to the second (lower) level. The first level implements a stochastic local search with tabu memory, at each iteration the type of the VMs used for a particular tier is changed randomly from all the available VMs, according to the architectural constraints. The tabu memory is used to store recent moves and avoid cycling of the candidate solutions around the same configurations. Once the VM size is fixed the solution may enter in a repair phase during which the number of VMs is increased until the feasibility is restored (switching to slower VMs can make the current system configuration unfeasible). The solution is then refined by gradually reducing the number of VMs until the optimal allocation is found. This whole process is repeated for a pre-defined number of iterations updating the final solution each time a feasible and cheaper one is found.

4 Experimental Results

The proposed optimization approach has been evaluated for a variety of system and workload configurations. Our solution will be compared with current approaches for capacity allocation and according to threshold based the auto-scaling policies that can be implemented at IaaS providers.

Analysis performed in this Section are intended to be representative of real Cloud applications. We have used a very large set of randomly generated instances, obtained varying the performance model parameters according to the ranges used by other literature approaches [4], [3], [36], [39] and from real system [7] (see Table 3). VMs costs and capacities have been taken from Amazon EC2, Microsoft Azure, and Flexiscale.

As, in [8,4], the request class service time threshold has been set equal to:

$$\overline{R}_k = 10 \sum_{c \in U_k} \frac{p_{k,c}}{\mu^{\overline{v}}_{k,c}}$$

where we considered as reference VM, the Amazon EC2 small with index $\overline{v}$.

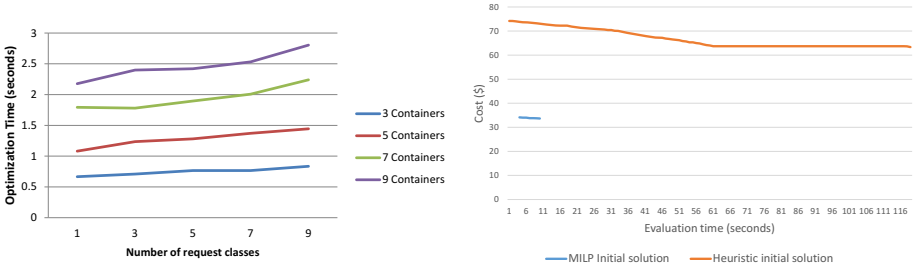
Workloads have been generated by considering the trace of a large Web system including almost 100 servers. The trace contains the number of sessions, on a per-hour basis, over a one year period. The trace follows a bimodal distribution with two peaks around 11.00 and 16.00. Multiple workloads have been obtained by adding random

Table 3. Ranges of model parameters

Parameter	Range
α_k	$[0.1; 1]$ %
$p_{k,c}$	$[0.01; 0.5]$
$\mu^v_{k,c}$	$[50; 2800]$ req/sec
$C_{v_{p,i}}$	$[0.06; 1.06]$ $ per hour
$\overleftarrow{M}_i$	[1;4] GB
N	5000 VMs
$\overline{R}_{k,c}$	$[0.005; 0.01]$ sec

Table 4. MILP sets cardinalities

Description	Variation range
Number of resource containers $\lvert\mathcal{I}\rvert$	$[1; 9]$
Time Intervals $\lvert\mathcal{T}\rvert$	$[4; 24]$
Number of Requests Classes $\lvert\mathcal{K}\rvert$	$[1; 10]$
Number of VM types $\lvert\mathcal{V}\rvert$	$[1; 12]$



(a) CPLEX optimization time varying the number of containers and request classes.

(b) Example of SPACE4Cloud execution trace.

white noise to each sample as in [4] and [24]. The MILP optimization model sets cardinality has been varied as reported in Table 4. The number of applications components has been varied between 1 and 10.

The next Section reports the results of the scalability of the MILP formulation. The quality of the MILP solution is evaluated in Section 4.2.

4.1 Scalability Analysis

Tests have been performed on a VirtualBox virtual machine based on Ubuntu 12.10 server running on an Intel Xeon Nehalem dual socket quad-core system with 32 GB of RAM. CPLEX 12.2.0.0 [3] has been used as MILP solver.

In order to guarantee statistical independence of our scalability results, for each test we considered ten different instances with the same size. The results reported here have been obtained by considering 10,000 total runs.

Figure 4(a) reports a representative example and shows how CPLEX optimization time (i.e., the time required to optimally solve the model) for optimizing a system with 10 components varies by changing the number of containers and request classes. On average CPLEX is able to find a solution in 0.5-3 seconds. In the very worst case, considering a system including 5 containers, 9 requests classes and 9 components the optimization time was 8.72 seconds.

[3] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

Table 5. Results comparison

First Feas. Time Savings	Final Opt. Time Savings	First Feas. Cost Savings	Final Opt. Cost Savings	MILP Feas. Initial Sol.	HEU Feas. Initial Sol.
-57.2%	414.5%	130.7%	98.5%	78.5%	82.0%

4.2 Initial Solution Quality Evaluation

In general the performance of a heuristic based optimization approach is measured by assessing final local solutions values and the time needed to generate such solutions. We evaluate the benefits of the MILP formulation by comparing the final solution that can be obtained by SPACE4Cloud considering as initial solution the MILP configuration and the one obtained by the following heuristic:

- for all containers, the cheapest VM type available at the Cloud provider satisfying also the memory constraint (5) is adopted;
- as in other literature approaches [36,39] and coherently to the auto-scaling policies that can be implemented by current IaaS providers[4], the number of VMs of each resource pool is determined such that the average CPU utilization is lower than a given threshold $\overline{\rho}$.

In our experiments we set $\overline{\rho} = 0.6$ as in [39]. More in details, the performance metrics we considered in our comparison are:

- the time required by SPACE4Cloud to find the first feasible solution;
- the time required by SPACE4Cloud to converge to the final local optimum;
- the cost of the first feasible solution;
- the cost of the final local optimum;
- the percentage of initial feasible solutions obtained by the MILP and heuristic approach.

Table 5 summarizes the results achieved. The results reported here have been obtained by considering 100 total runs. Figures report the average percentage improvements that can be obtained by adopting the MILP formulation. The precentages have been evaluated by considering the ratio $(Y - X)/X$, where X and Y are the MILP and heuristic performance, respectively (negative values means that the heuristic solution performs better). The first two columns report the percentage time saving obtained to identify the first feasible solution and the final local optimum. The third and fourth columns report the percentage of cost reduction of the first feasible and final local optimum solution, while the last two columns report the average percentage of QoS constraints that are satisfied by the two initial solutions. Even if the MILP approach introduces an overhead to find the first feasible solution (the first feasible solution is obtained by the heuristic with around 57% lower time), the hybrid approach outperforms the heuristic, reducing the time to converge to the final local optimum and improving significantly the cost of the final optimum solution.

[4] `http://aws.amazon.com/elasticbeanstalk/`

As an example, Figure 4(b) reports the execution trace of SPACE4Cloud for optimising a system including 5 containers, 9 request classes, and 9 components. On the x axis the overall optimization time is reported (including the time required to determine the initial solution by the MILP or the heuristic and the local search execution time), while the y axis reports the Cloud daily resource cost. The blue and red lines are the costs of the best current solution obtained by considering as initial solution the MILP and the heuristic solution, respectively. The initial gap shows the time required by identifying the first initial solution (which is more evident for the MILP trace where around 3 seconds are needed). Even if the evaluation of the MILP solution introduces an initial delay, the local search performance are significantly improved. In this specific case the final solution is around 88% cheaper, while the time required by the local search to identify the final local optimum is reduced by almost an order of magnitude.

5 Related Work

In the last two decades prediction and assessment of QoS characteristics of complex software systems has emerged as an important requirement. An effective way of dealing with this requirement is to integrate non functional properties prediction techniques into the software development process by means of appropriate tools and techniques. In these kind of approaches a model of the software architecture is annotated with non functional elements that are used to predict the performance of the application. The output of this analysis is used by application designers as feedback to modify the architecture in order to meet the requirements.

The Model-Driven Quality Prediction (MDQP) approach is to model the application with UML models, in order to support the specification of non functional properties the Object Management Group (OMG) introduced two profiles called *Schedulability, Performance and Time* (SPT) [31] and *Modeling and Analysis of Real-Time and Embedded Systems* MARTE [32]. This approach of extending UML is not the only one that deals with the analysis of non functional properties of software systems, Becker et al. developed the PCM [10], a language that can be used to model an application and its non functional properties and, with the support of the PCM-Bench tool, derive a LQN model to estimate the performance of the running system. The automated transformation of architecture-level models to predictive models is the second phase of the MDQP process (see, e.g., [37]). Many meta-models have been built to support performance prediction, some surveys of these models, their capability and their applicability to different scenarios can be found in [9,23,1]. The output of the performance analysis performed using these models is used to optimize the architecture of the application at design time in (semi-)automatic way.

We divide the most relevant approaches extending the classification presented by Martens et al. in [27]. The classes used to categorize the different solutions are: rule-base, meta-heuristic, generic Design Space exploration (DSE), quality-driven model transformations.

Rule-Based Approaches. Rule-based approaches use feedback rules to modify the architecture of the application. The QVT-Rational framework proposed in [13,14] extends

the Query, View, Transformation language defined by OMG by adding the support for feedback rules for semi-automatic generation of system architectures. The PUMA [37] framework aims at filling the gap between the design model and the performance models and has been extended with support for feedback rules specification in JESS [38]. Other language-specific tools, like the one proposed by Parsons et al. [34] for Java EE, are able to identify performance anti-patterns in existing systems specified by a set of rules; the tool reconstruct the model of the application using different monitoring techniques and analyze the generated model against the stored rules. Rule-based model to model transformation approaches has been proposed by Kavimandan and Gokhale [20] to optimize real-time QoS configuration in distributed and embedded systems.

Meta-Heuristics. Meta-heuristics use particular algorithms that are specifically designed to efficiently explore the space of design alternatives and find solutions that are optimized with respect to some quality criteria. In [25], Li et al. propose the Automated Quality-driven Optimization of Software Architecture (AQOSA) toolkit that implements some advanced evolutionary optimization algorithms for multi-objective problems, the toolkit integrates modelling technologies with performance evaluation tools in order to evaluate the goodness of generated solution according to a cost function. Aleti et al. proposed a similar approach in [2]; they presented the ArcheOpterix framework that exploits evolutionary algorithms to derive Pareto optimal component deployment decisions with respect to multiple quality criteria. Meedeniva et al. developed a multi-objective optimization strategy on top of ArcheOpterix in [28] to find trade-off between reliability and energy consumption in embedded systems. PerOpteryx [21] use a similar approach to optimize software architectures modeled with the Palladio framework [10] according to performance, reliability and cost. Other approaches combine analytical optimization techniques with evolutionary algorithms to find Pareto optimal solutions, an example of this is presented by Koziolek et al. in [22] with a particular focus on availability, performance and cost. A tabu search (TS) heuristic has been used by Ouzineb et al. [33] to derive component allocation under availability constraints in the context of embedded systems. The SASSY [29] framework developed by Menascé et al. starts from a model of a service-oriented architecture, performs service selection and applies patterns like replication and load balancing in order to fulfill quality requirements. Finally, Frey et al. [18] proposed a combined metaheuristic-simulation approach based on a genetic algorithm to derive deployment architecture and runtime reconfiguration rules while moving a legacy application to the Cloud environment.

Generic Design Space Exploration (GDSE). Generic Design Space Exploration approaches encode feedback rules into a Constraint Satisfaction Problem (CSP) in order to explore the design space. The DeepCompass [12] framework proposed by Bondarev et al. perform design space exploration according to a performance analysis of component-based software on multiprocessors systems. The DESERT framework [30,15] performs a general exploration of design alternatives by modeling system variations in a tree structure and using Boolean constraints to cut branches without feasible solutions. The latest version of this framework, DESERT-FD [15] automates the constraint generation process and the design space exploration. The GDSE [35] framework proposed by Saxena et al.is a meta-programmable system domain-specific DSE problems,

it provides a language to express constraints and support different solvers for candidate solution generation. A similar approach is proposed by Jackson et al. in the Formula [19] framework; Formula allows the specification of non-functional requirements, models and meta-models by first-order logic with arithmetic relations, the problem is solved by the Z3 Satisfiability Modulo Theory (SMT) solver to generate several design alternatives that comply with the specified requirements.

Most of the presented works are tailored to solve very particular problem and lack of generalization on the quality attributes supported for the design space exploration. Moreover, only one approach [18] tackles directly the problem of building architectures for the Cloud environment but it focuses on the migration of legacy applications.

6 Conclusions

In this paper, a hybrid approach for the cost minimization of Cloud based applications has been proposed. The MILP formulation that implements the first step of the hybrid approach is able to identify a promising initial solution for a local search optimization procedure which outperforms, both in terms of overall optimization time and final solution costs, first principles heuristics based on utilization thresholds. The proposed approach can lead to a reduction of Cloud application 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.

Ongoing work focuses on the extension of the MILP formulation and the local search to multiple Cloud deployments and on QoS analyses of real case studies.

References

1. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: A systematic literature review. IEEE Trans. Soft. Eng. 39(5), 658–683 (2013)
2. Aleti, A., Stefan Björnander, S., Grunske, L., Meedeniya, I.: Archeopterix: An extendable tool for architecture optimization of aadl models. In: MOMPES 2009 (2009)
3. Almeida, J., Almeida, V., Ardagna, D., Cunha, I., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing 70(4), 344 (2010)
4. Ardagna, D., Casolari, S., Colajanni, M., Panicucci, B.: Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. Journal of Parallel and Distributed Computing 72(6), 796 (2012)
5. Ardagna, D., Ciavotta, M., Gibilisco, G.P., Casale, G., Pérez, J.: Prediction and cost assessment tool - proof of concept. Project deliverable (2013)
6. Ardagna, D., Mirandola, R.: Per-flow optimal service selection for web services based processes. Journal of Systems and Software 83(8), 1512–1523 (2010)
7. Ardagna, D., Panicucci, B., Trubian, M., Zhang, L.: Energy-aware autonomic resource allocation in multitier virtualized environments. IEEE Trans. Serv. Comp. 5(1), 2–19 (2012)
8. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Soft. Eng. 33(6), 369–384 (2007)
9. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Trans. Soft. Eng. 30(5), 295–310 (2004)

10. Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1), 3–22 (2009)
11. Birke, R., Chen, L.Y., Smirni, E.: Data centers in the cloud: A large scale performance study. In: CLOUD 2012 (2012)
12. Bondarev, E., Chaudron, M.R.V., de Kock, E.A.: Exploring performance trade-offs of a jpeg decoder using the deepcompass framework. In: WOSP 2007 (2007)
13. Drago, M.L.: Quality Driven Model Transformations for Feedback Provisioning. PhD thesis, Italy (2012)
14. Drago, M.L., Ghezzi, C., Mirandola, R.: A quality driven extension to the qvt-relations transformation language. Computer Science - R&D 27(2) (2012)
15. Eames, B., Neema, S., Saraswat, R.: Desertfd: a finite-domain constraint based tool for design space exploration. Design Automation for Embedded Systems 14(1), 43–74 (2010)
16. Franceschelli, D., Ardagna, D., Ciavotta, M., Di Nitto, E.: Space4cloud: A tool for system performance and costevaluation of cloud systems. In: Multi-cloud 2013 (2013)
17. Franks, G., Hubbard, A., Majumdar, S., Neilson, J., Petriu, D., Rolia, J., Woodside, M.: A toolset for performance engineering and software design of client-server systems. Performance Evaluation 24, 1–2 (1996)
18. Frey, S., Fittkau, F., Hasselbring, W.: Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In: ICSE 2013 (2013)
19. Jackson, E., Kang, E., Dahlweid, M., Seifert, D., Santen, T.: Components, platforms and possibilities: towards generic automation for mda. In: EMSOFT 2010 (2010)
20. Kavimandan, A., Gokhale, A.: Applying model transformations to optimizing real-time qos configurations in dre systems. Architectures for Adaptive Software Systems, 18–35 (2009)
21. Koziolek, A.: Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes. PhD thesis, Germany (2011)
22. Koziolek, A., Ardagna, D., Mirandola, R.: Hybrid multi-attribute QoS optimization in component based software systems. Journal of Systems and Software 86(10), 2542–2558 (2013)
23. Koziolek, H.: Performance evaluation of component-based software systems: A survey. Performance evaluation 67(8), 634–658 (2010)
24. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., Jiang, G.: Power and performance management of virtualized computing environments via lookahead control. Cluster Computing 12(1), 1–15 (2009)
25. Li, R., Etemaadi, R., Emmerich, M.T.M., Chaudron, M.R.V.: An evolutionary multiobjective optimization approach to component-based software architecture design. In: CEC 2011 (2011)
26. Martens, A., Ardagna, D., Koziolek, H., Mirandola, R., Reussner, R.: A hybrid approach for multi-attribute qoS optimisation in component based software systems. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) QoSA 2010. LNCS, vol. 6093, pp. 84–101. Springer, Heidelberg (2010)
27. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: WOSP/SIPEW 2010 (2010)
28. Meedeniya, I., Buhnova, B., Aleti, A., Grunske, L.: Architecture-driven reliability and energy optimization for complex embedded systems. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) QoSA 2010. LNCS, vol. 6093, pp. 52–67. Springer, Heidelberg (2010)
29. Menascé, D.A., Ewing, J.M., Gomaa, H., Malek, S., Sousa, J.P.: A framework for utility-based service oriented design in SASSY. In: WOSP/SIPEW 2010 (2010)
30. Neema, S., Sztipanovits, J., Karsai, G., Butts, K.: Constraint-based design-space exploration and model synthesis. In: Embedded Software, pp. 290–305 (2003)
31. OMG. UML Profile for Schedulability, Performance, and Time Specification (2005)

32. OMG. A uml profile for marte: Modeling and analysis of real-time embedded systems (2008)
33. Ouzineb, M., Nourelfath, M., Gendreau, M.: Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems. Reliability Engineering & System Safety 93(8), 1257–1272 (2008)
34. Parsons, T., Murphy, J.: Detecting performance antipatterns in component based enterprise systems. Journal of Object Technology 7(3), 55–91 (2008)
35. Saxena, T., Karsai, G.: Mde-based approach for generalizing design space exploration. Model Driven Engineering Languages and Systems, 46–60 (2010)
36. Wolke, A., Meixner, G.: TwoSpot: A cloud platform for scaling out web applications dynamically. In: Di Nitto, E., Yahyapour, R. (eds.) ServiceWave 2010. LNCS, vol. 6481, pp. 13–24. Springer, Heidelberg (2010)
37. Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (puma). In: WOSP 2005 (2005)
38. Xu, J.: Rule-based automatic software performance diagnosis and improvement. In: WOSP 2008 (2008)
39. Zhu, X., Young, D., Watson, B., Wang, Z., Rolia, J., Singhal, S., McKee, B., Hyser, C., Gmach, D., Gardner, R., Christian, T., Cherkasova, L.: 1000 islands: An integrated approach to resource management for virtualized data centers. Journal of Cluster Computing 12(1), 45–57 (2009)