ADAPTING CLOUD-BASED APPLICATIONS TROUGH A COORDINATED AND OPTIMIZED RESOURCE ALLOCATION APPROACH

Patrizia Scandurra[1], Claudia Raibulet[2], Pasqualina Potena[1], Raffaela Mirandola[3] and Rafael Capilla[4]
[1]*DIIMM, Dalmine (BG), Università degli Studi di Bergamo, Italy*
[2]*DISCo, Università degli Studi di Milano-Bicocca, Italy*
[3]*Politecnico di Milano, DEI, Italy*
[4]*Depto. de Ciencias de la Computacion, Universidad Rey Juan Carlos, Spain*
{*patrizia.scandurra, pasqualina.potena*}*@unibg.it, mirandola@elet.polimi.it, raibulet@disco.unimibi.it, rafael.capilla@urjc.es*

Keywords: Cloud Computing, Software Adaptation, Resource Allocation, Service-oriented Component Architecture

Abstract: Cloud computing is getting an enormous popularity for software companies as a way to save and optimize the cost of large hardware and software infrastructure organizations demand. Also, the cooperation between cloud layers constitutes a timely research challenge as allocation and optimization of (often virtualized) resources is many times done in isolation or with poor interaction. In this paper we propose a framework that adapts a cloud-based software application by providing an enhanced assembly of resources using the Pareto-optimal solution to optimize the resource allocation with tight cooperation between the cloud layers.

1 INTRODUCTION

Cloud computing is an emerging computational model where systems, both hardware and software, are seamlessly delivered and administered over the Internet as *services*, giving the illusion to have them on a local machine. This is achievable by exploiting web technologies, virtualization techniques and large web data centers. Depending on the Service Model (Mell and Grance, 2011), a *cloud* can offer Infrastructure (raw computing services such as CPU and storage) as a Service (IaaS), Platform (COTS, tools, middleware for developing and deploying applications) as a Service (PaaS), and Software (end user applications) as a Service (SaaS)[1].

The SaaS layer generally exploits the Service-Oriented Architecture (SOA) technology because, as remarked in (Tsai et al., 2010), SOA and cloud computing coexist, complement and support each other. Obviously, the adoption of SOA requires taking into account the typical features of cloud domain. In such a domain, for example, service providers publish the services in deployable packages that can be easily replicated and redeployed to different cloud hosting environments. Hence, the increasing deployment of cloud systems is leading to new research challenges based on the cooperation and integration of a variety of hardware and software resources and services that are part of the cloud.

One of the aforementioned challenges we address in this paper refers to the problem of optimizing resource allocation and adapting a cloud-based application. Cloud-based applications require dynamic adaptation (Cheng et al., 2009) for several reasons, such as: the introduction of new functionality, adaptations in the runtime environment (e.g. workflow composition), or changes in the QoS level. Since the system quality attributes are directly affected by the limitation of hardware resources, the dynamic system adaptation[2] is strongly tied to the dynamic resource allocation. For example, the system reliability might change dynamically according to various types of failures (e.g., overflow failure, timeout failure, resource missing failure, network failure, or database failure), and

[1]The lines dividing the layers are not distinctive. “Components and features of one layer can also be considered to be in another layer. For example, data storage service can be considered to be either in as IaaS or PaaS.”(Tsai et al., 2010).

[2]Throughout the paper, unless otherwise stated, the term “system” and “cloud-based application” are used interchangeable.

the cloud infrastructure must react and adapt itself to preserve the level of reliability.

In addition, the layers of a cloud architecture (i.e., IaaS, PaaS and SaaS) cooperate for the resource management through established contracts: each layer usually provides resources to the upper layers by exploiting resources of the lower layers. Cloud layers attempt to perform optimal resource allocation (e.g., the IaaS layer maximizes hardware resources utilization) in order to satisfy user demands, but existing approaches usually optimize the decisions in a single layer without considering dependencies and constraints between layers. Hence, instead of viewing layers as separate control/optimization problems, an explicit coordination between layers would allow finding the appropriate trade-off between the, usually conflicting, objectives of the layers (Litoiu et al., 2010).

In this work we address the interdependencies of cloud layers. Our approach attempts to find optimal solutions for each layer when resource allocation is needed and to support explicit cooperation between layers. Different resource allocation actions may be suitable for assuring a certain quality attribute (e.g., a different deployment of containers on VMs and/or assignment of CPU and memory to the VMs could still assure a certain reliability). Our solution allows the verification of the impact of the software adaptation actions on the qualities attributes by predicting the quality, obtained after the application of adaptation actions for each change required, as a function of the resource allocation. To this purpose, we adopt a *Pareto multi-objective optimization* (Censor, 1977) to optimize the resource allocation with tight cooperation between the cloud layers.

The paper is organized as follows. Section 2 reports some related work. Section 3 presents a small application example adopted as running case study. Section 4 describes the proposed coordination framework. The details of its realization are given and exemplified through the case study in Section 5. Section 6 outlines some technical details about our experimental cloud infrastructure. Finally, Section 7 concludes the paper and outlines some future directions.

2 STATE OF THE ART

As stated in (Litoiu et al., 2010), notwithstanding the increasing interest and diffusion of cloud computing projects (see, e.g., EU's FP7 RESERVOIR[3] and the VOLARE (Papakos et al., 2010) projects), "there is no clear vision on how different layers of the cloud, possibly in different administrative domains, can collaborate to satisfy stakeholders goals". To this purpose, in (Litoiu et al., 2010) a conceptual optimization model is proposed for satisfying the performance characteristics both locally for each cloud layer (e.g., an SaaS user requires an average application response time, and the maximum hardware utilization is wished by the IaaS providers) and globally, considering an interlayer cooperation. This paper has been inspired by the work in (Litoiu et al., 2010) and proposes a tight interlayer coordination schema for optimizing resource allocation able to deal with the adaptation of cloud-based applications.

Several research efforts have been devoted in the last years to the definition of approaches and frameworks for supporting optimization decisions for single cloud layers. Most of them typically adapt a cloud-based application by adopting different service discovery and rebinding policies (see, e.g., (Papakos et al., 2010)), varying application parametrization and properties or modifying the application structure (Baker et al., 2010), such as the flow of the activities or processes. These adaptation approaches suffer for the lack of models/techniques for analyzing the quality attributes of a cloud system (Dai et al., 2009), and are not driven by quality attribute trade-offs by considering the particular features of the cloud computing domain. Traditional web service composition methods, for example, need to be enriched (Zou et al., 2010) because service providers will likely publish their web services at different cloud platforms, such as Windows Azure Platform and Amazon S3. Furthermore, existing algorithms for workflow scheduling are not designed for multiple workflows with multiple QoS constrained scheduling and do not specifically take into account the features of the cloud computing (Xu et al., 2009).

The adaptation actions suggested by these approaches are not typically driven by the dynamic resource allocation with tight cooperation between the layers. On the opposite, as claimed in (Yau and An, 2009), the resource allocation approaches cannot support dynamically modifying runtime environment changing for example, the workflow composition, QoS requirements, or workflow priorities and resource status. Strategies have been developed, for example, for maximizing the profit of an application provider under constraints on a maximum user response time for each user class (or a minimum class throughput) (Li et al., 2009), or optimizing the power-performance tradeoff, and the adaptation benefit and cost of infrastructure providers (Jung et al., 2010).

With respect to the state of the art mentioned

[3]http://www.reservoir-fp7.eu/

above, our approach (to the best of our knowledge) is the first one that supports the adaptation of a cloud-based application (including both the static and dynamic behavior) while resources are allocated with a tight cooperation between the cloud layers. The adaptation actions may differ for adaptation cost and/or the resulting system quality while changing the resource allocation settings.

3 RUNNING CASE STUDY

This section presents a smartphone application used throughout the paper to exemplify our approach. The application, called *Multimedia Service App*, provides an end-user multimedia (text and video information) service to subscribed users. The news includes text and topical videos available in MPEG2 format. Besides, we assume that the smartphone user can require to the *Multimedia Service App* a geographical map showing its locations (Alrifai and Risse, 2009).

Figure 1 shows the software architecture of the application using the OSOA standard notation Service Component Architecture (SCA) (SCA, 2007). It is a thin client/server application: the server part is the real application hosted on the cloud, while the client (not shown in the figure) is assumed to be external and connected via a wireless network to the *Multimedia Service App*. The *Multimedia Service App* is a composite application[4] that makes use of several services for sending news to the client. The service (or service-oriented component) *ClientInterface* processes information on the client's behalf. It interacts with the core service *MultimediaService*, and also with the service *Compression* to adapt the news content to the wireless link. The service *MultimediaService* coordinates the services: *Transcoding* to adapt the video content for the smartphone format, *Translation* to adapt the text for the smartphone format and draw the geographical map, the service *Merging* to integrate the text with the video stream for the limited smartphone display, and *Locations Database* to collect information about the localization of cells and thus provide a map showing the user location.

In this work we will consider the following three functionalities: (i) require news in textual format, (ii) require news with both textual and video content (iii) provide a geographical map with user location.

[4]The SCA standard provides the composite concept as a means to assemble heterogeneous service-oriented components into logical groupings. An SCA composite usually contains a set of components, services, references, the wires (communication channels) that interconnect them.

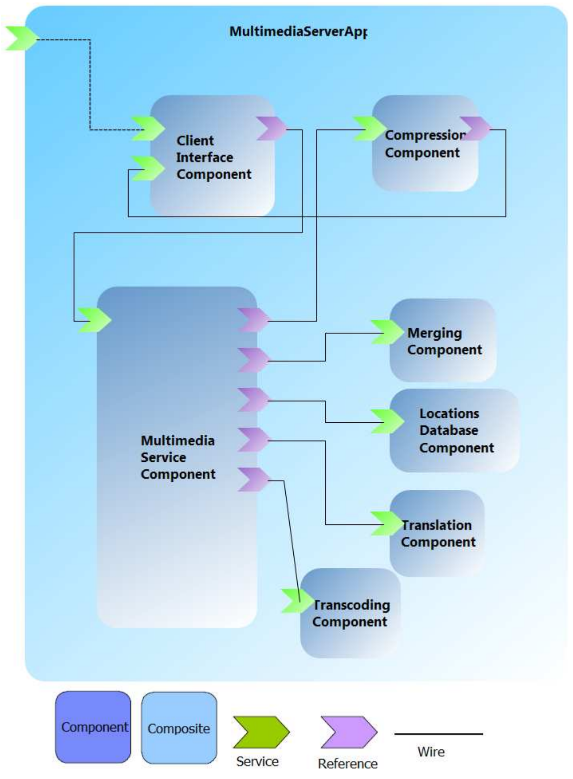

Figure 1: SCA assembly of the Multimedia Service App

4 A CLOUD COORDINATION FRAMEWORK

This section presents a three-layered coordination approach (see Figure 2) based on three frameworks (one for each cloud layer) – composing the *Cloud Coordination Framework* – that dynamically perform resource allocation. Each framework provides a set of *manager components* that handle the requests coming from the other layers. Such components of a framework interchange sensors data (related to measures of the optimization) and actuators data (to drive the optimization) with the managers of the other layers thus performing a continuous optimization activity.

The IaaS layer aims at maximizing the use of its Hardware Resources (HRs). Such resources are virtualized and offered as computing services (e.g., storage, CPU and memory). The IaaS layer handles service requests from the PaaS layer, such as Virtual Machines (VMs) with particular performance characteristics on memory, storage and processing capacity. The PaaS layer, which offers platform services to the SaaS layer, aims at optimizing the profit by maximizing the number of applications it hosts and minimizing the resources it uses, and penalties it pays. The PaaS layer uses resources of the IaaS layer by request-

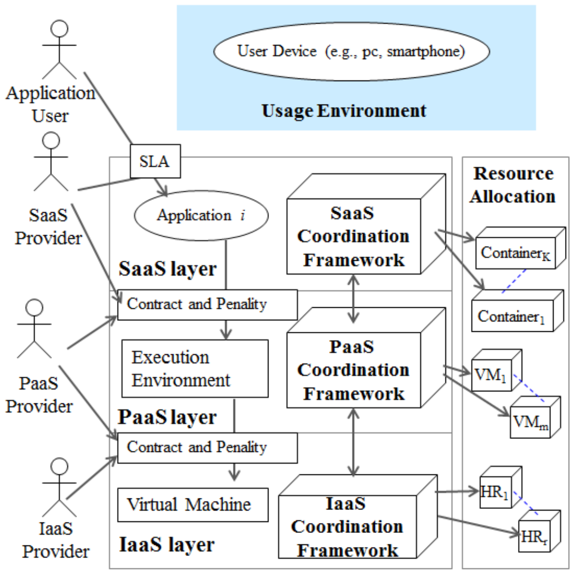

Figure 2: The *Cloud Coordination Framework* and its environment

ing VMs and storage and deploying application containers[5] on the VMs. Finally, the SaaS layer, provides services to end users and maximizes the profit, for example setting the revenue proportional to the number of users or to the throughput. It deploys the applications in PaaS containers considering topologies specific to each application.

In this section we describe how our coordination framework allows the combination of the adaptation – performed at the SaaS layer and driven by quality attributes – of a cloud-based application and of resource allocation – performed at all layers. In fact, a system quality (such as the perceived system reliability for a user application) formalized as Service Level Agreement (SLA) depends on the interaction between certain layers, i.e., to a specific contract among the stakeholders of the layers. If an adaptation request is triggered in the SaaS layer, then the layer has to combine the software adaptation actions and the resources already owned or newly requested to the PaaS layer. The PaaS layer, in turn, must find an optimal decision to satisfy the resources requested from the SaaS layer through resources it already owns (deploying additional containers) or demand more to the IaaS layer. In the last case, the IaaS, for example, must deploy and allocate additional VMs in an optimal manner. Considering quality analysis and implementation of layers' coordination, different strategies can be used depending on factors mainly related to the use of our framework for *evolution* (at re-design time) or *self-adaptation* (at run time) (Bucchiarone et al., 2010).

[5]A container is a computing infrastructure for hosting and running applications (one or more services).

4.1 A coordination scenario

Figure 3 shows a concrete example of coordination scenario as it would be performed by our coordination framework for the adaptation of a cloud-based application. Below, we describe the main steps of this process, which could be merely adopted in case of evolution. In case of self-adaptation, we should define simpler models and faster approaches must be adopted for their solution in order to allow a prompt run-time adaptation.

Step 1: SaaS Framework defines adaption actions. The SaaS framework defines software adaptation actions for addressing the required changes (claimed by a user or triggered by a monitor module) by using the multi-objective optimization (Censor, 1977). A solution of such an optimization, called *Pareto solution* or *Pareto front*, is the one that minimizes a set of objectives (e.g., adaptation cost, probability of failure and response time) under quality constraints, such as on the required minimum level of reliability.

More formally: Let AS be the *adaptation space* (i.e. the search space of our optimization process obtained considering the set of all possible candidate solutions) defined as the Cartesian product of the option sets of all application-specific adaptation actions, let s be a candidate solution, let $C \subseteq AS$ be a set of candidate solutions evaluated so far, and let q be a quality criterion with a domain D_q, and an order $\leq_q$ on D_q so that $s_1 \leq_q s_2$ means that s_1 is better than or equal to s_2 with respect to quality criterion q. Then, candidate solution s is Pareto-optimal with respect to a set of evaluated candidate solutions C, iff:

$$\forall s' \in C \quad \exists q : f_q(s) \leq_q f_q(s')$$

If a candidate solution is not *Pareto-optimal*, then it is *Pareto-dominated* by at least one other candidate solution in C that is better or equal in all quality criteria. Analogously, a candidate is *globally Pareto-optimal*, if it is Pareto-optimal with respect to the set of all possible candidates AS.

The solutions may differ in the adaptation actions for adapting the applications or the resource allocation (e.g., the deployment of services on the containers, which have been previously assigned by the PaaS layer). If the SaaS layer needs more resources (or does not find an admissible solution with the containers assigned by the PaaS layer), then the SaaS framework proposes the Pareto solutions to the PaaS framework. For each solution the resources necessary for its application are indicated (e.g., a new deployment topology) or ranges for container parameters (e.g., utilizations, throughput, response times of containers) are reported. Obviously, the SaaS framework tries to minimize, other than the cost for the software adaptation (e.g., the one for embedding a new service into the system), the payment for resources to the PaaS layer.

Step 2: PaaS Framework tries to satisfy the SaaS demands. Once the PaaS framework analyses the Pareto solutions obtained from the SaaS framework, it attempts to find feasible resource allocations or in case of failure, it suggests Pareto solutions to the IaaS framework. Each solution suggests, for instance, additional VMs, the allocation of VMs to containers. Similarly to the SaaS framework, the PaaS framework tries to minimize the payment for resources to IaaS layer. Section 5 presents a couple of optimization models that can be used in our framework at SaaS and PaaS layers, and shows the benefits of their interaction.

Step 3: IaaS Framework tries to satisfy the PaaS demands. The IaaS framework analyzes the Pareto set received from the PaaS framework. Similarly to the SaaS and PaaS frameworks, the IaaS framework finds a Pareto solution that minimize a set of objectives (e.g., the resources that it does not use and the penalties it pays) by trying to satisfy the PaaS requests (e.g., by allocating VMs to processor or changing VM settings, such as memory, CPU ratio, etc.).

Step 4: Contract re-negotiation between layers

If the layers' frameworks do not find a resource allocation for addressing the required systems changes, then layers' contracts may be (re)-negotiated, e.g., the PaaS layer contracts additional resources with the IaaS layer or the SLA is re-negotiated. For example, in (Papakos et al., 2010), when the context of the user device changes or the provided level of quality attributes changes then the required quality levels are re-negotiated. Note that this step will mostly depend on runtime adaptation needs.

5 THE OPTIMIZATION MODEL

In this section we present a couple of optimization models that we use in our framework at SaaS and PaaS layers together with their application to the running case study.

Specifically, the proposed SaaS model supports the adaptation of a cloud application by the service selection and the resource allocation activities. In particular, for each elementary service of an application it indicates one of its available instances. The model finds also the optimal amount of resources of the containers to be allocated to their hosted services. The model maximizes the *Profit* of the SaaS provider and the *Throughput* of the application (i.e., the average number of service responses per second) under a given throughput constraint.

At the PaaS layer, the model aims to maximize the *Profit* of the PaaS provider and the *Throughput* of the applications, which share a VM, under throughput constraints (i.e., the throughput of the applications served by the PaaS layer is assured within a certain threshold).

5.1 The SaaS problem formulation

Let us now better formalize the model. Let us assume a cloud-based application that offers a set F of functionalities to users through the composition of n elementary services. Let s_i be the i-th service ($1 \leq$ i $\leq$ n). The affiliation of the services to the functionalities is represented by the matrix $SF(n \times F)$, where an element $SF[i,f]$ is equal to 1 if the functionality f includes the service s_i, and 0 otherwise.

Let J_i be the set of instances available for the i-th service (i.e., the services provided by the SaaS provider). We assume that the instances available for the service s_i are functionally compliant with it, i.e., each instance provides at least all services provided by s_i and requires at most all services required by s_i.

Let K be the number of containers. The container deployment is expressed by the matrix $D(n,K)$, where the entry $D[i,k]$ is equal to 1 if s_i is deployed on the container k, and 0 otherwise. Each container provides resources to the services, such as CPU time, memory and bandwidth. As the average number of service-requests per second arriving to a container one resource will become a bottleneck (this is also called *critical* resource).

Model Variables. The x_{ij} ($1 \leq i \leq n$, $j \in J_i$) variables are used for selecting an instance available for the i-th service, namely x_{ij} is equal to 1 if the s_{ij} instance is chosen, and 0 otherwise.

Furthermore, the variable T_f ($1 \leq f \leq F$) represents the optimal throughput of the f-th functionality.

Function Objective. Maximizing the *Profit* of the SaaS provider and the *Throughput* of the cloud application.

The *Profit* of the SaaS provider can be expressed as:

$$PROF = \sum_{i=1}^{n} \sum_{j \in J_i} c_{ij} x_{ij} \qquad (1)$$

where c_{ij} is the price (in KiloEuros, KE) charged for using the service s_{ij}.

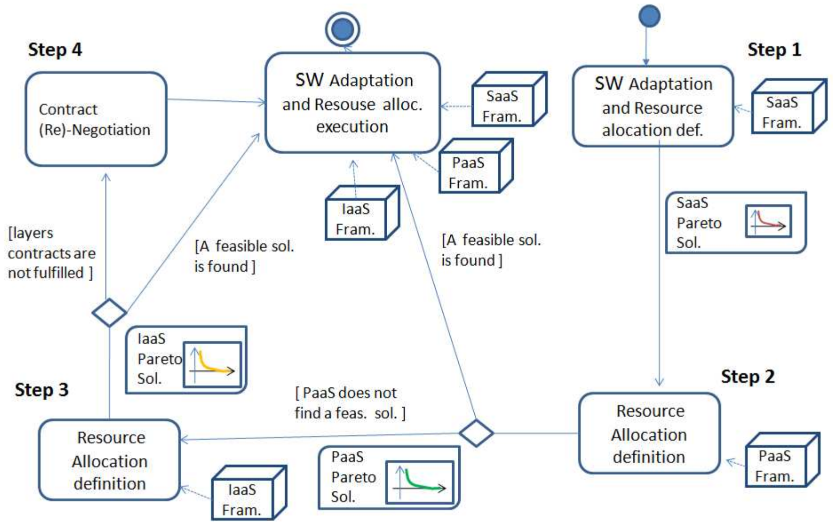
Step 4
Contract (Re)-Negotiation
SW Adaptation and Resouse alloc. execution
SaaS Fram.
PaaS Fram.
IaaS Fram.
Step 1
SW Adaptation and Resource alocation def.
SaaS Fram.
SaaS Pareto Sol.
[layers contracts are not fulfilled]
[A feasible sol. is found]
[A feasible sol. is found]
IaaS Pareto Sol.
Step 3
[PaaS does not find a feas. sol.]
Step 2
Resource Allocation definition
PaaS Fram.
Resource Allocation definition
IaaS Fram.
PaaS Pareto Sol.

Figure 3: A coordination scenario for the adaptation of a cloud application

The *Throughput* of an application can be formulated as:

$$TH = \sum_{f=1}^{F} T_f Pr_f \tag{2}$$

where Pr_f is the priority required (see (Yau and An, 2009) where the priorities for workflows are considered) for the functionality f.

Service selection constraint. For each service s_i, exactly one instance is chosen (i.e., $\sum_{j \in J_i} x_{ij} = 1, \quad \forall i = 1 \dots n$).

Service-request rates requirement constraint. For each functionality f the constraint $T_f \leq SR_f$, where SR_f is service request rate of f, is added.

Throughput requirement constraint. For the functionality f (if $TR_f \leq SR_f$, where TR_f is the throughput required by the users for f) the constraint $T_f \geq TR_f$ is added.

Available critical resource constraint. For the container k the following constraint is added: $\sum_{f=1}^{F} (\sum_{i=1}^{n} D[i,k] \cdot T_f \cdot SF[i,f] \cdot bp_{if} \cdot \sum_{j \in J_i} \lambda_{ij} \cdot x_{ij}) \leq R_k$, where R_k is the percentage of available critical resource of the container k. R_k can be either a resource assigned by the PaaS layer or the one required by the SaaS Framework for finding feasible solutions. bp_{if} is the number of invocations of s_i within the functionality f, and λ_{ij} represents the *service cost*, namely the percentage of critical resource required to serve a service-request over the total available resource critical (see (Yau and An, 2009) for more details).

Resource Allocation. Therefore, the amount of resource A_{kh} to be allocated to the service s_h deployed on the container k is:

$$A_{kh} = \sum_{j \in J_h} \lambda_{hj} \cdot x_{hj} \cdot \sum_{f=1}^{F} T_f \cdot SF[h,f] \tag{3}$$

Running Application Example: To solve the model we have implemented the *lexicographic method* (Marler and Arora, 2004) as follows. First we have solved the optimization model maximizing the SaaS profit, then we have formulated the optimization model that maximizes the throughput of the application under the SaaS profit constraint expressed as a function of the real number ε. Finally, we have found the set of Pareto optimal solutions by varying ε (i.e., we have applied the ε-constraint approach (Marler and Arora, 2004)). For the experimentation we have used the LINGO tool[6], which is a non-linear model solver, to produce the results.

We have parameterized the model with the values of the available service instances as reported in the Table 1. We have associated short names to the services as illustrated in the first column of the Table 1 (i.e., s_1, s_2,...,s_7), and deployed the services (see the last column) on three containers. The third column of the Table lists, for each service, the set of alternatives. For each alternative: the cost c_{ij} (in KiloEuros, KE) is given in the fourth column, the percentage of critical resource required to serve a service-request λ_{ij} is given in the fifth column. Finally the number of invocations bp_{fi} of s_i within the functionalities 1, 2 and 3 are given in the sixth, seven and eight column, respectively.

The critical resource of the containers is the CPU-time. Let us consider the following values: the percentage of available resource of containers 1, 2 and 3 equal to 90%, 89%, 90% respectively; the priority required for the functionalities 1, 2 and 3 are 1, 2.5 and 3, respectively. The service request rates of function-

[6]http://www.lindo.com/

Service ID	Service name	Service alter.	Cost c_{ij}	Perc. of r. λ_{ij}	B. p. f. 1 bp_{1i}	B. p. f. 2 bp_{2i}	B. p. f. 3 bp_{3i}	Cont. Dep. $D[i,k]$
s_1	*ClientInterface*	s_{11} s_{12} s_{13}	6 9 10	0.04 0.07 0.1	3	3	3	$D[1,2]$
s_2	*Multimedia Service*	s_{21} s_{22} s_{23}	4 12 14	0.03 0.2 0.3	5	7	6	$D[2,2]$
s_3	*Locations Database*	s_{31} s_{32} s_{33}	8 10 16	0.12 0.3 0.4	0	0	1	$D[3,1]$
s_4	*Transcoding*	s_{41} s_{42} s_{43}	3 10 14	0.04 0.08 0.2	0	1	0	$D[4,3]$
s_5	*Translation*	s_{51} s_{52} s_{53}	2 5 9	0.01 0.02 0.2	1	1	1	$D[5,1]$
s_6	*Merging*	s_{61} s_{62} s_{63}	5 14 19	0.01 0.03 0.2	0	1	0	$D[6,3]$
s_7	*Compression*	s_{71} s_{72} s_{73}	3 9 15	0.09 0.1 0.2	1	1	1	$D[7,1]$

Table 1: Parameters of the available instances.

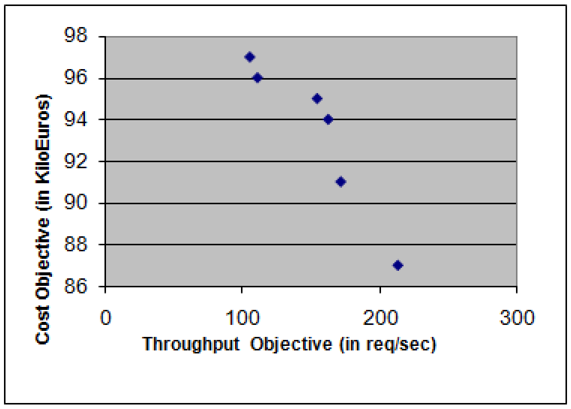

Figure 4: Pareto curve returned by the SaaS model.

alities 1, 2 and 3 are considered equal to 27, 30 and 37 req/sec, respectively. Finally, the minimum throughput of functionalities 1, 2 and 3 are set to 7, 12 and 12 req/sec, respectively.

Figure 4 reports the approximate Pareto curve obtained from solving the model. Each Pareto solution represents a configuration of services and an allocation of resources that maximize both the SaaS profit (in KiloEuros, KE) and the throughput of the application (in request/second, req/sec).

Table 2 details the results. Each row of the table represents the choice of services and the functionality throughput (i.e., a Pareto solution) that the model suggests. The choice is represented as two vectors. Each element of the first vector is the name of the service instance available, whereas each element of the second one is the functionality's throughput. Beside these vectors, the throughput objective, the SaaS profit, and the value of the ε parameter are also reported.

For example, model results claim that if throughput objective is equal to 213 req/sec (represented on the x-axis), then the maximum profit of the SaaS is 87 KE. Such solution point is: $[s_{13}, s_{21}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 27$ req/sec, $T_2 = 30$ req/sec, $T_3 = 37$ req/sec], where all the replacements services are specified, as well as the optimal throughput of each functionality. The model results also show that the optimal solution cost increases (up to 97 KE) while decreasing the throughput objective (down to 105 req/sec).

5.2 The PaaS problem formulation

Let R be the resources (assigned by the IaaS layer) provided by a VM to its m hosted applications, such as CPU time, memory and bandwidth.

Model Variables. The variable $\overline{T_i}$ $(1 \leq i \leq m)$ represents the optimal throughput of the i-th application.

Pareto Solution
$[s_{13}, s_{23}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 7$ req/sec, $T_2 = 12$ req/sec, $T_3 = 22.666$ req/sec$]$ Throu. Obj.= 105 req/sec Cost Obj.= 97 KE
$[s_{12}, s_{23}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 7$ req/sec, $T_2 = 12$ req/sec, $T_3 = 24.532$ req/sec$]$ $\varepsilon = 1$ Throu. Obj.= 110.597 req/sec Cost Obj.= 96 KE
$[s_{13}, s_{22}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 7$ req/sec, $T_2 = 14.352$ req/sec, $T_3 = 37$ req/sec$]$ $\varepsilon = 2$ Throu. Obj.= 153.882 req/sec Cost Obj.= 95 KE
$[s_{12}, s_{22}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 7$ req/sec, $T_2 = 17.614$ req/sec, $T_3 = 37$ req/sec$]$ $\varepsilon = 3$ Throu. Obj.= 162.037 req/sec Cost Obj.= 94 KE
$[s_{11}, s_{22}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 7$ req/sec, $T_2 = 21.263$ req/sec, $T_3 = 37$ req/sec$]$ $\varepsilon = 7$ Throu. Obj.= 171.157 req/sec Cost Obj.= 91 KE
$[s_{13}, s_{21}, s_{33}, s_{43}, s_{53}, s_{63}, s_{73}]$ $[T_1 = 27$ req/sec, $T_2 = 30$ req/sec, $T_3 = 37$ req/sec$]$ $\varepsilon = 10$ Throu. Obj.= 213 req/sec Cost Obj.= 87 KE

Table 2: Results of the SaaS Model.

Function Objective. Maximizing the *Profit* of the PaaS provider and the *Throughput* of the m applications.

The *Profit* can be expressed as:

$$PROF = \sum_{r=1}^{R} \sum_{i=1}^{m} \bar{c_r} \cdot \bar{A}_{ri} \tag{4}$$

where $\bar{c_r}$ represents the price (in KE) charged for using one percent of the resource r, and $\bar{A}_{ri}$ is the amount of r assigned to the application i ([7]).

Similarly to the SaaS model, the *Throughput* objective can be formulated as:

$$TH = \sum_{i=1}^{m} \bar{T_i}\bar{Pr_i} \tag{5}$$

where $\bar{Pr_i}$ is the priority required for the application i.

Service-request rates requirement constraint. For the application i the constraint $\bar{T_i} \le SR_i$, where SR_i is service request rate of i, is added.

Throughput requirement constraint. For the application i (if $TR_i \le SR_i$, where TR_i is the throughput required by the users for i) the constraint $\bar{T_i} \ge TR_i$ is added.

Available critical resource constraint. For the resource r the following constraint is added: $\sum_{i=1}^{m} \bar{T_i} \cdot \lambda_{ri} \le av_r$ where av_r is the percentage of available resource r of the VM, and λ_{ri}, similarly to the *service cost*, is the percentage of resource r required to serve a service-request of the application i over the total available resource critical.

Resource Allocation. Therefore, the amount of resource r to be allocated to the application i is:

$$\bar{A}_{ri} = \bar{\lambda}_{ri} \cdot \bar{T_i} \tag{6}$$

[7] $\bar{T_i}$ can be expressed as $\sum_{f=1}^{F} T_f \cdot pexec_f$, where $pexec_f$ is the probability that the f-th application functionality will be invoked. It must hold: $pexec_f >= 0$ and $\sum_{f=1}^{F} pexec_f = 1$.

5.3 Cooperation between SaaS and PaaS optimization models

In this section we outline two examples where the SaaS and the PaaS model cooperate together. We show how this latter model allows adapting the smartphone application, as proposed by the Pareto solutions in Figure 4, by using the PaaS resource allocation activity (see *Step 2* in Section 4).

Running Application Example: The container 1 of the smartphone application (here also called *app* 1) belongs to a VM, which is also shared by other two cloud applications (here also called *app* 2 and *app* 3, respectively).

Let us consider the following values: the price charged for using one percent of VM's CPU equal to 2 KE; the VM has the 100% of CPU; the priority required for the *app*1, *app* 2 and *app* 3 are set to 2, 1.5 and 1, respectively. The probabilities that the functionalities 1, 2 and 3 of *app*1 are considered equal to 0.3, 0.3 and 0.4, respectively. Thus the average service request rate SR_1 and throughput requirement TR_1 of *app* 1 are equal to 31.9 and 10.5 req/sec, respectively.

Example 1: *PaaS Framework assures the requests of the smarthphone provider and the maximum throughput for app 2 and app 3.*

Let us consider the following values for *app* 2 and *app* 3, respectively: the percentages of CPU usage at a very low average load are 0.6% and 0.8%; the service request rates are 20 and 24 req/sec; the throughput requirements are 8 and 9 req/sec.

The PaaS model for the SaaS Pareto point (153.882 req/sec, 95 KE) – for which the average optimal throughput $\overline{T_1}$ of *app* 1 is equal to 21.205 req/sec – finds the following feasible solution: $[\overline{T_1} = 21.205$ req/sec, $\overline{T_2} = 20$ req/sec, $\overline{T_3} = 24$ req/sec]. The PaaS profit is equal to 138.68 KE, while the throughput objective is equal to 96.411 req/sec.

Example 2: *PaaS Framework satisfies the smarthphone provider, but not the maximum throughput for app 2 and app 3.*

Starting from the initial values of the model parameters of *Example 1*, let us increase to: 2% the percentage of CPU required by *app* 2 and *app* 3; 24 and 34 req/sec the service requests rates of *app* 2 and *app* 3, respectively.

The PaaS model for the SaaS Pareto point (213 req/sec, 87 KE) – for which $\overline{T_1}$ is equal to 31.9 req/sec – finds the following feasible solution: $[\overline{T_1} = 31.9$ req/sec, $\overline{T_2} = 14.8$ req/sec, $\overline{T_3} = 9$ req/sec]. The PaaS profit is equal to 200 KE, while the throughput objective is equal to 95 req/sec.

These examples highlight the benefits of the coordination between the SaaS and the PaaS layers. They show how the SaaS provider is able to adapt the smartphone application by using the resource allocation activity of the PaaS layer.

6 EXPERIMENTAL ENVIRONMENT

Today, many cloud applications in the SaaS layer are implemented using service-oriented assemblies such as the SCA (SCA, 2007) approach. SCA is an XML-based metadata component model that copes with heterogeneity by allowing various implementation technology (as Java, C++, Spring, PHP, BPEL, Web services, etc.), middleware programming APIs, and communication protocols.

We have been conducting our experiments on a private cloud environment based on the software framework Eucalyptus (Nurmi et al., 2009). Eucalyptus is an open source cloud computing software framework that implements the IaaS model by allowing users to run and control entire virtual machines on cloud infrastructure. On the top of such virtual machines, as application containers we install: the Java platform enterprise edition (Java EE), Apache Tomcat, various web services containers, and (more importantly) the SCA runtime platform Apache Tuscany (Tuscany, 2010).

The platform Tuscany supports different technology/protocols (e.g., WSDL binding to consume/expose web services, JMS binding to receive/send Java Message Service, etc.) and recently, with the incubation project Nuvem, addresses some cloud-specific issues. Using this platform, applications (possibly containing components implemented in different languages) can be integrated by representing them in terms of SCA assemblies deployed and spanned across multiple *SCA domains*. An SCA domain typically represents a set of services providing an area of business functionality that is controlled by a single organization. As an example, for the accounts department in a business, the SCA domain might cover all financial related function, and it might contain a series of composites dealing with specific areas of accounting, with one for customer accounts, another dealing with accounts payable (SCAspec, 2007). We use Tuscany as the primary infrastructure solution for creating and hosting the domains, components, etc. These domains are then spread across the cloud and enterprise infrastructure, thus creating a composite service-oriented application.

Finally, our optimization framework uses (as we already mentioned previously) Lingo to solve the optimization models at all three layers. Up to now, we have been focused on the coordination of the SaaS and PaaS layers, upon each we build a coordination manager component to deal with handling requests. Similarly to the Eucalyptus framework that implements high-level system components in terms of Web services, we have chosen to implement each manager component as a stand-alone Web service exposing well-defined language-agnostic API in the form of a WSDL document. In this paper, we focused on the description of the overall workflow of the compound managers behaviors. Providing details about the fine-grained architecture of such managers is out of the scope of this paper.

7 CONCLUSIONS AND FUTURE WORK

We have argued the need that cooperation between the cloud layers should be improved in order to satisfy the resource allocation in an optimal way thus avoiding collapsing the cloud. Such layered coordination schema, as shown in this paper on a smartphone application, benefits the adaptation of a cloud-based application. To this extent, a couple of optimization models – designed for the SaaS and PaaS layers – are presented, and their application/coordination is also illustrated through the case study.

Currently, we are implementing a prototype of our coordination framework to handle multiple distributed applications and large scale infrastructures by following both a centralized and a decentralized paradigm. We intend to apply our approach on realistic examples to validate it, study its scalability, and compare it with existing approaches. We believe that our approach may provide a valid solution to overcome some of their drawbacks. For example, using our approach it would be possible to consider the particular features of the cloud domain (e.g., services can be deployed in different clouds) by expressing the quality attributes of an application as a function of the dynamic resource allocation.

REFERENCES

Alrifai, M. and Risse, T. (2009). Combining global optimization with local selection for efficient QoS-aware service composition. In *WWW*, pages 881–890.

Baker, T., Taleb-Bendiab, A., Randles, M., and Karam, Y. (2010). Support for adaptive cloud-based applications via intention modelling.

Bucchiarone, A., Cappiello, C., Nitto, E. D., Kazhamiakin, R., Mazza, V., and Pistore, M. (2010). Design for adaptation of service-based applications: Main issues and requirements. In *ICSOC/ServiceWave 2009 Workshops*, LNCS, pages 467–476.

Censor, Y. (1977). Pareto Optimality in Multiobjective Problems. *Appl. Math. Optimiz.*, 4:41–59.

Cheng, B. H. C. et al. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26.

Dai, Y.-S., Yang, B., Dongarra, J., and Zhang, G. (2009). Cloud service reliability: Modeling and analysis. *Proc. of 15th Pacific Rim Inter. Symp. on Depend. Comp.*

Jung, G., Hiltunen, M. A., Joshi, K. R., Schlichting, R. D., and Pu, C. (2010). Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. *Distributed Computing Systems, International Conference on*, 0:62–73.

Li, J., Chinneck, J., Woodside, M., Litoiu, M., and Iszlai, G. (2009). Performance model driven qos guarantees and optimization in clouds. In *Proc. of the ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 15–22.

Litoiu, M., Woodside, M., Wong, J., Ng, J., and Iszlai, G. (2010). A business driven cloud optimization architecture. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 380–385. ACM.

Marler, R. and Arora, J. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395.

Mell, P. and Grance, T. (September 2011). The NIST definition of cloud computing. National Institute of Standards and Technology.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In Cappello, F., Wang, C.-L., and Buyya, R., editors, *CCGRID*, pages 124–131. IEEE Computer Society.

Papakos, P., Capra, L., and Rosenblum, D. S. (2010). Volare: context-aware adaptive cloud service discovery for mobile systems. In *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*, ARM '10, pages 32–38.

SCA (2007). OSOA. Service Component Architecture (SCA) www.osoa.org.

SCAspec (2007). SCA Assembly Model Specification, Version 1.00, March 15 2007.

Tsai, W.-T., Sun, X., and Balasooriya, J. (2010). Service-oriented cloud computing architecture. *Information Technology: New Generations, Third International Conference on*, pages 684–689.

Tuscany (2010). Apache Tuscany. `http://tuscany.apache.org/`.

Xu, M., Cui, L., Wang, H., and Bi, Y. (2009). A multiple qos constrained scheduling strategy of multiple workflows for cloud computing. *Parallel and Distributed Processing with Applications, International Symposium on*, 0:629–634.

Yau, S. and An, H. (2009). Adaptive resource allocation for service-based systems. In *Internetware '09: Proceedings of the First Asia-Pacific Symposium on Internetware*.

Zou, G., Chen, Y., Yang, Y., Huang, R., and Xu, Y. (2010). Ai planning and combinatorial optimization for web service composition in cloud computing. In *Proc. International Conference on Cloud Computing and Virtualization (CCV-10)*.