SPACE4CLOUD: A Tool for System PerformAnce and Cost Evaluation of CLOUD Systems

Davide Franceschelli
Politecnico di Milano
Dipartimento di Elettronica,
Informazione e Bioingegneria
davide.franceschelli@mail.polimi.it

Danilo Ardagna
Politecnico di Milano
Dipartimento di Elettronica,
Informazione e Bioingegneria
ardagna@elet.polimi.it

Michele Ciavotta
Politecnico di Milano
Dipartimento di Elettronica,
Informazione e Bioingegneria
ciavotta@elet.polimi.it

Elisabetta Di Nitto
Politecnico di Milano
Dipartimento di Elettronica,
Informazione e Bioingegneria
dinitto@elet.polimi.it

ABSTRACT

Cloud Computing is assuming a relevant role in the world of web applications and web services. Cloud technologies allow to build dynamic systems which are able to adapt their performance to workload fluctuations delegating to the Cloud Provider the intensive tasks of management and maintenance of the cloud infrastructure. Which is the best provider for our application? The application will guarantee the required service level objectives (SLOs)? Those are relevant issues that call for a tool able to carry on cost and performance analysis of the system before its actual development. In designing a software application to be executed in a cloud environment, the most relevant issues to be addressed are determining which cloud provider to use and verifying if the target system will present the required performance levels.
The goal of this work is to provide a model-driven approach to performance and cost estimation of cloud and multi-cloud systems. We considered the IaaS (Infrastructure-as-a-Service) and PaaS (Platform-as-a-Service) levels.
The modelling of such systems has involved different abstraction levels, starting from the representation of cloud applications and ending with the modelling of the underlying insfrastructure/platform belonging to specific Cloud Providers. An initial prototype supporting our approach is also presented.

Categories and Subject Descriptors

C.4 [**Performance attributes**]: Performance of Systems; D.2.11 [**Software Engineering**]: Software Architectures; I.6.5 [**Simulation and Modeling**]: Model Development

MultiCloud'13, April 22, 2013, Prague, Czech Republic.

Keywords

Model-Driven Software Development, Performance Prediction, Cloud Computing

1. INTRODUCTION

Verifying that a software system has certain non-functional properties is now a primary concern in software engineering. Cloud Computing offers many interesting features but, at the same time, it may introduce some non-negligible issues and new challenges in application development. First, the technology offer and the pricing models are currently very heterogeneous and the selection of the cloud solution that fits the application requirements, minimizing the costs, is a non-trivial task. Nowadays, in fact, there is a considerable number of cloud providers each of them offering a proprietary cloud solution with certain configurations and cost profiles. This means that cloud costumers should consider multiple architectures and should be able to evaluate costs and performance for each of them. This task can be very challenging, even unfeasible if performed manually, as the number of possible solutions, given by the available providers and the resource configuration, may become very high.
Secondly, Cloud systems are usually multi-tenant and their performance can vary with the time of day, according to the congestion level and the competition among the applications. Therefore, analytical techniques and automatic or semi-automatic tools are needed in order to predict the Quality of Service (QoS) and to reason on software systems properties at design time. The current practice for software development tends, in fact, to relegate the non-functional features of the application to the final phases of the development process. Therefore, the entire system is designed, developed and deployed and only afterwards the QoS is measured and eventually corrective actions are undertaken. Finding out in the last phases of a project that a software system is not compliant with some required non-functional properties can be detrimental for the success of the project itself. It is, therefore, of paramount importance to accurately evaluate costs and performance of the system at design time. Several tools which permit the prediction of costs and performance on proprietary systems from design models have been developed and presented in literature,

but they do not support cloud systems and they still lack the proper level of automation envisioned by Model-Driven Software Development. In this paper, we propose a model driven approach and SPACE4CLOUD, a tool for the evaluation of the performance and costs of cloud and multi-cloud systems proposing an extension to the Palladio Component Model (PCM) and its Palladio Bench for QoS evaluation [8]. 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 and it lacks support for cloud systems. Furthermore, PCM has been developed with the goal to predict the behaviour of a static system assuming the workload, the hardware configuration and its performance are constant in time. On the contrary, on cloud based systems are dynamic by their own nature and for this reason time-dependent parameters have to be considered to predict their performance. In light of these considerations the rationale of this work is to analyse the most common cloud systems in order to derive general cloud meta-models, mapping their concepts into those defined within PCM and Palladio, considering time-dependent profiles for the most important performance parameters and developing a module in charge of the cost evaluation for the resulting model.
The remainder of the paper is organized as follows. In section 2 we summarize the state of the art for the model-based performance prediction problem. Section 3 shows the proposed meta-models while their implementation by means of the SPACE4CLOUD tool is described in Sections 4 and 5. Section 6 presents a case study based on the SPECWeb2005 benchmark and, finally, conclusions are drawn in Section 7.

2. RELATED WORK

Our research lays in the area of Model-Driven Quality Prediction (MDQP). The starting point of the MDQP process is a set of models that describe the software system using an established modelling language such as the UML. The Object Management Group (OMG) presented two versions of the UML (the UML SPT profile [22] and the UML MARTE profile [23]) in order to support resources, allocation and non-functional properties. Other approaches, however, do exist and leverages languages specifically designed for this purpose, such as KLAPER [15] and the Palladio Component Model (PCM) [8]. The second phase of the MDQP process is the automated transformation of architecture-level models into predictive performance models like Layered Queuing Network (LQN) or Markov Chains (see, e.g., [26]). Meta-models supporting model-based performance predictions are surveyed in [7] and more recently in [17] and [5]. Most of the approaches aim on the evaluation of different design options and support manual or automated optimisation of the architecture at design time. We categorise the most relevant solutions presented in literature using the following classes: rule-based, meta-heuristic, generic Design Space Exploration (DSE), quality-driven model transformations. The first class of approaches is based on feedback rules. An example of tool that exploits this class of approaches is the QVT-Rational framework proposed in [11, 12]. This tool uses the Query/View/Transformation (QVT) language proposed by the OMG for model-to-model transformations. Basically a QVT extension has been provided in order to support feedback rules defined on non-functional requirements. In this way it is possible to derive in an automatic or semi-automatic way system variants which satisfy the defined rules. Xu et al. [27], described a semi-automatic framework for PUMA [26] and proposed the JESS scripting language to specify feedback rules. Other rule-based approaches, like the ones by Smith et al. [25] make use of the so called anti-patterns. These are a collection of practices which should be avoided in the design/development of software systems. Bondarev et al. [9] presented the design space exploration DeepCompass framework developed on the ROBOCOP Component Model.
The second class is more specific and leverages particular algorithms to efficiently explore the design space in search of solutions to optimize particular quality metrics. Examples of techniques which are usually adopted are evolutionary algorithms [18] and integer linear programming. Both ArcheOpterix [6] and PerOpterix framework [20] use genetic algorithms to generate candidate solutions. Kavimandan and Gokhale [16] presented a heuristic-based model-transformation algorithm, improving an existing bin packing algorithm to optimise real-Time QoS configurations in distributed, embedded component-based systems.
Generic Design Space Exploration approaches work similarly to metaheuristic techniques, exploring the design space, but they try to encode feedback provisioning rules as a constraint satisfaction problem. The DESERT framework [21, 13], for example, can be programmed in order to support generic DSE. Finally, quality-driven model transformation approaches use model transformation to specify feedback rules or to navigate the design space. The transformations are directed by quality concerns and feedback rules can be expressed through existing languages without the need to propose new ad-hoc languages. These approaches require human intervention and they select alternatives a priori, without concretely evaluating the quality attributes. None of the above mentioned works support the design and optimisation of applications with a target cloud deployment.

3. CLOUD PLATFORM META-MODELS

In order to evaluate performance and cost of cloud applications, a general model representing a cloud environment is needed. Using different levels of abstraction it is possible to model cloud systems and to evaluate performance indices and costs with different granularity levels. Furthermore, to some extent it is also possible to perform analysis abstracting the characteristics of the solutions offered by the providers and obtain bounds for performance and costs. The goal of this work is to extend the Palladio Framework implementing a new tool referred to as System PerformAnce and Cost Evaluation for Cloud, hereinafter SPACE4CLOUD.
Cloud applications performance and costs will be derived proposing suitable cloud meta-models and generating a mapping between them and the Palladio Component Model. Palladio is then used to evaluate performance and cost of cloud applications. We followed a model-driven approach and three meta-models have been defined: Cloud Independent Model (CIM), Cloud Provider Independent Model (CPIM) and Cloud Provider Specific Model (CPSM). This three-layered architecture permits the developer to work only at the higher abstraction level, thinking of its application in terms of generic cloud elements. Afterwards, it is responsibility of the system to map the generic cloud elements into specific ones automatically deriving one or several CPMs.

3.1 CIM

A CIM represents a component based application independent from the underlaying hardware architecture. The goal of the CIM is to specifiy the components building an application, their mutual interdependencies, and the QoS constraints for the end-users. Currently we are re-using PCM *Repository*, *System* and *RDSEFF*s models [8] amazoncpsmmicroinstancedetailsextended with annotations for specifying QoS constraints.

3.2 CPIM

The CPIM is located between the CIM and the CPSM. Its goal is to represent the general structure of a cloud environment, without expressing in detail the features of cloud services offered by the available cloud providers. The CPIM proposed in this paper is derived by the generalization of the Amazon Web Services , Windows Azure , Google AppEngine , and Flexiant cloud environments.
The Palladio Resource Model is not suitable to represent applications running on cloud infrastructures, so it must be extended in order to support the modeling of cloud applications. This extension can be performed through a mapping between the CPIM and the Resource Model. Working at the CPIM intermediate level it is possible to evaluate, for example, if a cloud application has good performance or not without considering any particular cloud provider. It it possible to do this since the CPIM can contain a general representation of a cloud system with specifications and costs derived from real offers. In this way a model can be created having instances of a certain category (low, medium or high) with the minimum processing rate and the maximum cost for that category. The minimal resources configuration needed to obtain certain performance levels can be calculated using such a representation. The same analysis can be performed on costs in order to find a lower bound or to set an upper bound to the total expenditure. To obtain more detailed performance metrics and better cost estimations specific CPSMs are required. Figure 1 introduces CPIM general concepts. A *Cloud System* is owned by a *Cloud Provider* that is a specialization of the general abstract concept of *Provider*. Typically, a Cloud Provider offers to the end-users several *Cloud Services*. A Cloud Service can be classified into three main classes: IaaS, PaaS, and SaaS. A *Cloud Element* represents either a *Cloud Resource*, a *Cloud Platform* or a *Cloud Software* and can be characterized by a *Cost Profile* that defines a set of Costs. The cost profile is required to model costs that vary with the time of the day (see e.g., Amazon instances prices [1]), or costs related to the total use of resources (see, for example the storage pricing models [2, 3]). The Costs specify a unit, a value, a reference period (which is used to express the cost variability during a given time horizon which is in hour case of 24 hours), a lower bound and an upper bound. A Cloud Service can also have *Service Level Agreement Templates* which define templates for SLAs between the Cloud Provider and the Cloud costumers. A Cloud Service can also have *Scaling Policies*, which are composed by *Scaling Rules* describing the metric of interest, the threshold value and the activation rule; scale-in and scale-out scaling rules are defined. The Scaling Policy itself is defined on one or more Resource Pools, which aggregate one or more homogeneous Compute resources.
Figure 2 represents other features of Cloud Resources, Cloud Platforms and Cloud Softwares. For some Cloud Resources

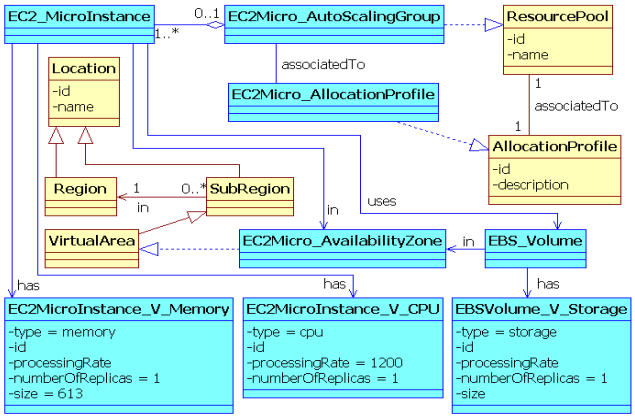

Figure 4: AWS CPSM - EC2 Micro Instance Details (yellow: CPIM classes, cyan: CPSM classes)

it is possible to define a *Location* (expressed in terms of *Region*, *SubRegion* and *Virtual Area*), which specifies where is located the hardware infrastructure providing the cloud resource. A Cloud Platform is a software framework exposing a defined set of APIs that can be used to develop custom applications and services. The platform also provides also an execution environment. *Frontend*, *Middleware*, *Backend* and *Database* platforms are three possible specializations of a Cloud Platform. Frontend platforms can host frontend services, which are directly exposed to the end users and are supposed to interact with them providing data to backend services. Backend platforms can host backend services, which are supposed to process data coming from frontend services eventually providing them some intermediate results. Middleware platforms can host services which are used to decouple Frontend instances from Backend instances. A Database platform is able to store structured or semi-structured data and can be classified into Relational DB or NoSQL DB. A Cloud Software is an application or a service that can be deployed on Cloud Platforms or can run directly on Cloud Resources. A Cloud Resource represents the minimal resource unit of a given Cloud Service and can be classified into *Compute* or *CloudStorage*. A Compute unit represents a general computational resource, like a Virtual Machine. A Storage unit is a resource able to store any kind of unstructured data and it can be classified into Filesystem Storage or Blob Storage. Figure 3 shows detailed features of a Cloud Resource that is a particular Cloud Element and a Cloud Element can be tied to one or more Cloud Elements through point-to-point *Links* in order to create a virtual network of Cloud Elements. A Resource Pool is a set of Compute Cloud Resources associated to an *Allocation Profile*, that is a set of Allocations which specify how the number of allocated instances within the Resource Pool changes in a certain reference period. A Cloud Resource is composed by Virtual Hardware Resources, which can be Virtual CPUs, Virtual Storage units or Virtual Memory units. Virtual HW Resources and Links can be characterized by an *Efficiency Profile* which specify how the efficiency (e.g., the processing rate of a Virtual CPU) changes with the time of the day due to the congestion for accessing cloud resources. In this way it is possible to represent the variability of the efficiency for both virtual hardware resources and links.

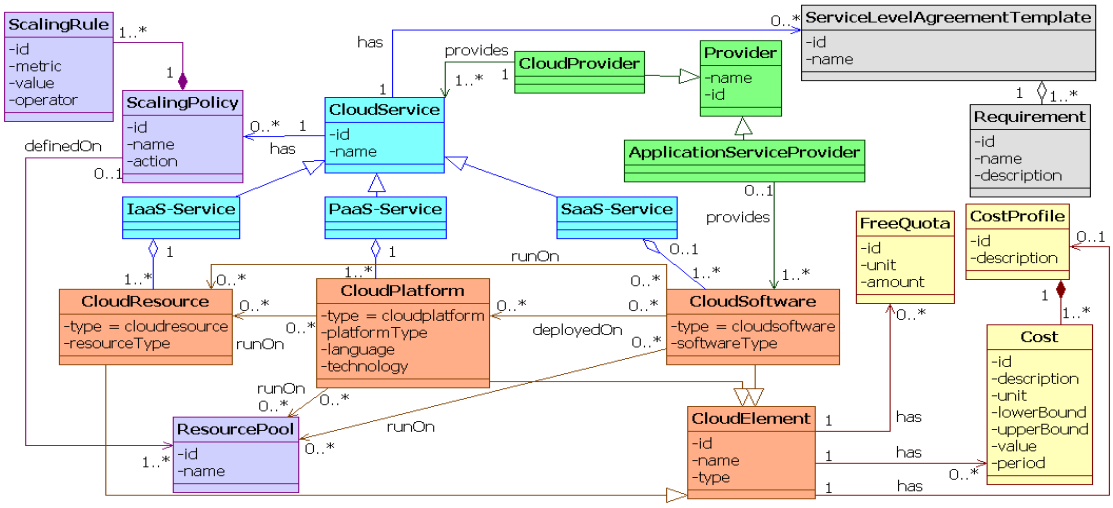

Figure 1: CPIM (Cyan: services, Yellow: free quotas and costs, Grey: SLA, Orange: elements, Purple: scaling policies, Green: provider)

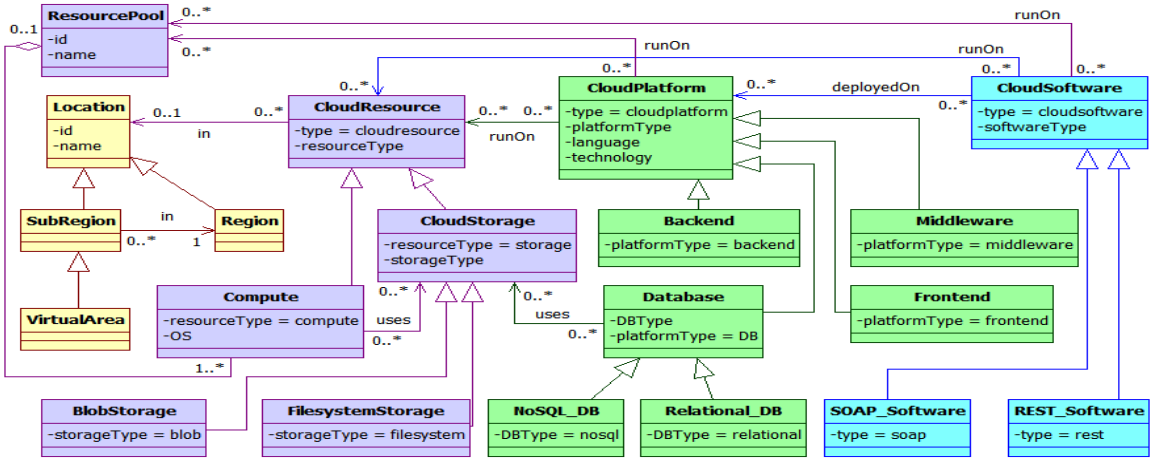

Figure 2: CPIM (Purple: resources, Green: platforms, Cyan: softwares, Yellow: locations)

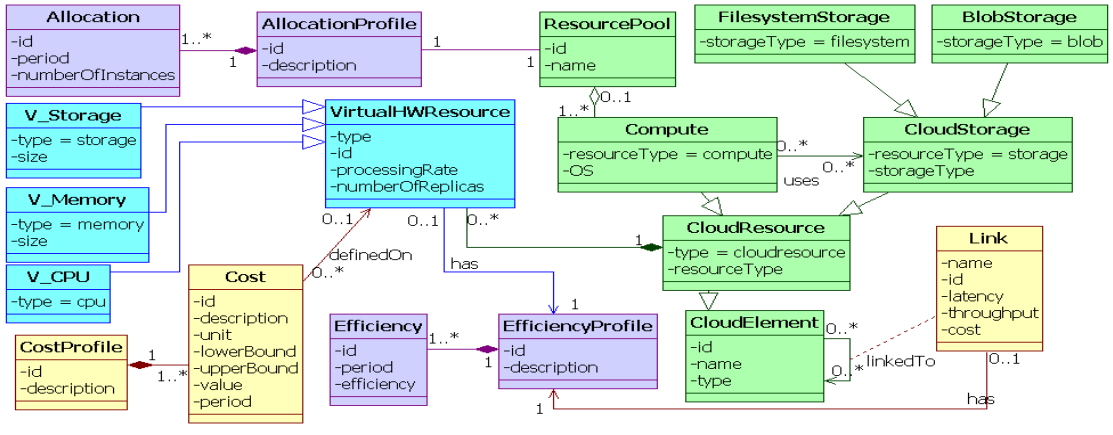

Figure 3: CPIM-IaaS (Cyan: virtual HW, Yellow: links and costs, Purple: profiles, Green: compute resource)

3.3 CPSM

A CPSM can be thought as a particular CPIM instance representing the features of cloud services offered by a particular cloud provider. For space limitations, in this section we will present one example of Cloud Provider Specific Model referred to Amazon Web Services (AWS). Considering the Amazon cloud, we can distinguish some relevant IaaS services like: Elastic Compute Cloud (**EC2**), Simple Storage Service (**S3**), Elastic Block Store (**EBS**), Relational Database Service (**RDS**) and the NoSQL database **DynamoDB**. Amazon itself can be considered as a realization of the *Cloud Provider* concept within the CPIM. Since a Cloud Provider provides one or more *Cloud Services*, we can extend the relation *provides* to all the aforementioned cloud services. Some of them can be classified as *IaaS-Services*, like EC2, S3 and EBS; the others (RDS and DynamoDB) can be classified as *PaaS-Services*. Considering that S3 provides a storage with a flat filesystem, an S3 Instance is a realization of the *Blob Storage* of the CPIM. As far as Amazon S3 is concerned, we can define a *Cost Profile* for that type of instances representing the cost variability. This is compulsory because the S3 service is charged on the base of different price ranges, depending on the allocated capacity. EBS, instead, provides volumes with standard filesystems, so an EBS Instance is a realization of the *Filesystem Storage*. Finally, Amazon **EC2** provides a set of EC2 Instances, each of which is a realization of a *Compute Cloud Resource*. EC2 Instances are characterized by different **EC2 Prices** which are essentially *Costs* from the CPIM. Within EC2 it is possible to specify *EC2 Auto Scaling Groups* which are sets of homogeneous EC2 Instances and can be referred to the *Resource Pools* defined within the CPIM. Also, *EC2 Scaling Policies* can be defined on these EC2 Auto Scaling Groups in order to set the rules which control the scaling activities. An EC2 Scaling Policy derives from a generic *Scaling Policy* of the CPIM. For an EC2 instance it is possible to specify the *Location* in order to control reliability and delays. Also virtual hardware resources (*Virtual HW Resources*) can be specified and for each of them it is possible to specify an *Efficiency Profile*. Considering, as an example, the EC2 Micro Instance it is composed by two virtual hardware resources: **EC2 Micro Instance V_Memory** and **EC2 Micro Instance V_CPU**. This instance does not provide any local storage, so it must use an external **EBS Volume**. Figure 4 summarize what has been discussed before about the EC2 Micro Instance, showing also the relation with the *SubRegion* and *Region* of the CPIM. Within EC2, it is possible to define *Virtual Areas* (**EC2 Micro Availability Zone**) within sub-regions and to associate EC2 instances to them. The Figure shows also other features of EC2, such as the possibility to define *Resource Pools* (**EC2 Micro Auto Scaling Group**) composed by EC2 instances with the same configurations. It is also possible to associate to these pools *Allocation Profiles* (**EC2 Micro Allocation Profile**) which specify how many instances of a given type are allocated in a given time period.

4. EXTENDING PALLADIO

In this section we present our mapping of CPIM and CPSM to PCM. Palladio has been developed in order to design and simulate the performance of web applications running on physical hardware resources. Cloud-based applications, in turn, run on virtualised hardware resources, so a suitable mapping between the concept of physical resource within Palladio and the concept of virtualised resource needs to be found. In Palladio each *Software Component* must be allocated on a specific *Resource Container* and this is done specifying the *Allocation Model*. So, a Resource Container is a collection of Processing Resources which can be classified into **CPU**, **Storage** and **Delay**. Each processing resource is characterized by a *scheduling* policy defining the way the incoming requests are served and by other parameters such as the *processingRate* and the *numberOfReplicas*.
A generic resource container, therefore, can be compared to a generic *Cloud Resource* provided by a IaaS cloud provider. In fact, each *Cloud Resource* has well defined characteristics such as a virtual CPU, a virtual memory and a virtual storage. So we can define a one-to-one mapping between the CPU resource defined within Palladio and the virtual CPU which characterises a *Cloud Resource*. For what concerns memory, we do not consider a possible mapping at this level because it will be taken into account in the CPIM and in the CPSM when comparing different configurations. A Palladio storage resource (HDD) can be mapped on the virtual storage of a *Cloud Resource*, that can be either a local storage of a VM or an external storage used to store application data. Finally, a Palladio Processing Resource is characterised by the attribute *numberOfReplicas* representing the actual number of its processors. This attribute is mapped into the number of cores of a virtual CPU. Since the attribute is assigned to each resource type, we will express it for a general virtual hardware resource rather than only for a virtual CPU. Even a *Cloud Platform* can be mapped to one or more Palladio Resource Containers depending on the information available, in the worst case, in which no information about the used hardware is provided, it can always be represented as a Delay Center. The *Efficiency Profile* is used to retrieve the efficiency factor of a *Cloud Resource* in a given moment, so that the real value of the processing rate is derived from the product between the efficiency factor and the maximum processing rate of the cloud resource. This value is then used as the *processingRate* parameter of the corresponding Palladio Processing Resource. The *Allocation Profile* is used to retrieve the number of allocated instances in a pool of Cloud Resources. This information is needed to represent the variability of the pool size, so it can be used to scale the parameter *numberOfReplicas* within Palladio Processing Resources. In other words, as in [24] since VM instances in a pool are homogeneous, they are represented as a single processing resource sized with the comulated capacity of the VMs it represents. The mapping between the Palladio hardware component model and the CPIM Cloud Resources is shown in Figure 5. For simplicity, the mapping does not consider also the Allocation Profile and the Efficiency Profile. but it gives an idea of how to model a basic cloud resource within a Palladio Resource Environment. The Cost Profile does not affect the Resource Environment definition, so it is not considered in the mapping but it is used by SPACE4CLOUD to evaluate costs. SPACE4CLOUD will be presented in the next Section.

5. DESIGN AND ANALYSIS TOOL

SPACE4CLOUD supports performance and cost evaluations on a 24 hours time period for cloud systems extending the features offered by the Palladio Framework.

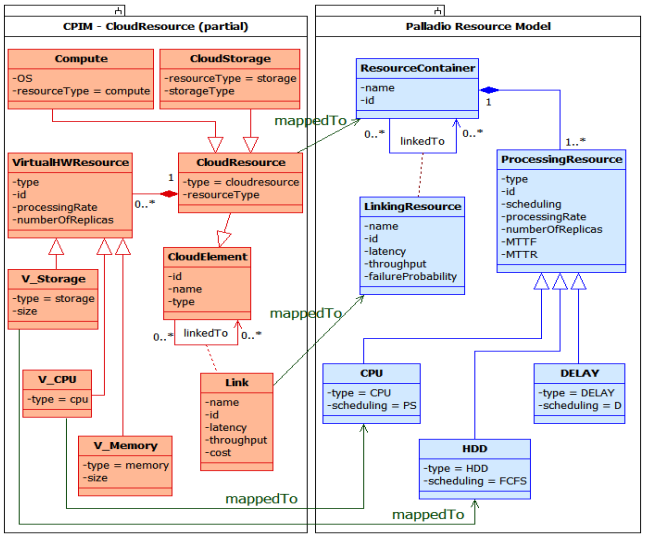

Figure 5: Mapping between Palladio and Cloud IaaS

We used the model-driven approach introduced in section 3 to represent cloud systems and we derive Palladio models suitable for performance analysis through model to model transformation. The tool has been developed in Java technologies as an Eclipse plug-in. We have implemented a step-by-step wizard to create the mapping bateween CPIM and CPSM to PCM models and launch automatically the perfromance analysis. Since multiple providers can be specified at the CPSM layer, the tool allows to support what-if analyses and perform design time exploration considering multiple clouds as candidate for the final deployment.
Since the Palladio cost model is limited and cannot include the cost characteristics of cloud systems, the cost derivation is complex and will be discussed in details in the following. The tool SPACE4CLOUD provides support for cost derivation, considering each association between Cloud Elements and Palladio Resource Containers. For each couple Cloud Element - Palladio Resource Container, we have the following parameters:

A	=	Allocation Profile
C	=	Set of Costs related to the Cloud Element
H	=	Set of "per hour" costs within C
G	=	Set of "per GB/month" costs within C
$VHR[c]$	=	Virtual Hardware Resource associated to cost c
CP	=	Cost Profile related to the Cloud Element
C^{CP}	=	Set of costs within CP
H^{CP}	=	Set of "per hour" costs within C^{CP}
G^{CP}	=	Set of "per GB/month" costs within C^{CP}
$VHR^{CP}[c]$	=	Virtual Hardware Resource associated to cost c

We can distinguish two kinds of costs: costs which are fixed with respect to the time (H, G) and time-variant costs (H^{CP}, G^{CP}). The firsts are those costs not associated to any cost profile, whereas the others are associated to a cost profile and each of them is defined for a certain *period* value. Furthermore, the H, H^{CP} sets represent the sets of costs which are charged on a "per hour" basis, while the G, G^{CP} sets represent the costs which are charged on a "per GB/-month" basis. Since the costs belonging to the G, G^{CP} sets are related to storage units, they are not affected by the allocation specification within the Allocation Profile specified for the container, because these resources do not scale. Costs belonging to the H, H^{CP} sets, instead, are related to scalable compute resources, so they are affected by the allocation specifications defined within the Allocation Profile. Taking into account the previous considerations, we can derive the total cost given by the H set in the following way:

$$COST_H = \sum_{h \in H} \left(value(h) * \sum_{a \in A} size(a) \right)$$

So, for each element of H we compute the cost related to the "total equivalent hours" by multiplying the cost value for the sum of allocated resources. This is equivalent to sum up the products between the cost value and the allocation specification, because costs belonging to H are not time-variant. For what concerns costs belonging to G, they do not depend neither on allocation specifications, neither on time. These costs depend only on the size of the allocated storage unit, so we need to retrieve the size of the Virtual Hardware Resource (VHR) which is expressed in terms of GB. Costs related to storage units can be associated to capacity ranges, so we need a function F to derive the partial cost related to the allocated size. Furthermore, the derived cost is expressed in terms of $/month, while we want it in terms of $/day. Considering the most simple case in a month we have 730 hours so in order to derive the daily costs, we have to divide the monthly cost by 730, obtaining the hourly cost, and then multiply for 24. So, the total cost related to G is:

$$COST_G = \frac{24}{730} * \sum_{g \in G} F\left(g, size(VHR[g])\right)$$

The F function take into account the cost specification g, the associated *lowerBound* and *upperBound* and the size of the virtual resource in order to calculate the cost. The time-variant costs belonging to the H^{CP} set are derived using the following expression:

$$COST_{H^{CP}} = \sum_{h \in H^{CP}} \left(value(h) * size(A[period(h)])\right)$$

The H^{CP} set is supposed to contain 24 cost specifications, one for each hour of the day. The sub-expression $A[period(h)]$ corresponds to the allocation specification related to the same period on which is defined the actual cost h. So, the total cost is given by the sum of the products between the cost values and the sizes of the allocation specifications having the same *period* value.
Costs belonging to G^{CP}, as usual, do not depend on allocation specifications, so we can use an expression similar to the one used for $COST_G$:

$$COST_{G^{CP}} = \sum_{g \in G^{CP}} \frac{F\left(g, size(VHR^{CP}[g])\right)}{730} \qquad \text{with}$$

Finally, the total daily cost for a given container is given by:

$$COST = COST_H + COST_G + COST_{H^{CP}} + COST_{G^{CP}}$$

6. CASE STUDY: SPECWEB2005

In this section we discuss some preliminary analyses we performed with the SPACE4CLOUD tool. We studied the SPECWeb2005 benchmark under varying workload and deployment conditions. SPECweb2005 is an industry benchmark for evaluating the performance of web systems [4].

SPECweb2005 includes a Web Server, a Back-End Simulator, and several Clients. The Web Server is the main element and represents the System Under Test. The Back-End Simulator (BESim) is the component used to simulate a database serving requests coming from the web server for dynamic page generation. Finally, clients act as load injectors according to a closed workload model.
For what concerns the available workloads (SpecWeb2005 provides multiple and standardized workloads) we choose the *Banking* suite and we considered four main pages which are the most resource intensive: Account information access and maintenance, Bill pay, Money transfer, and Loan. Five clients have been used during the experiments, the Web Server has been deployed on an Amazon EC2 Medium Instance, while the clients and the Back-End Simulator have been hosted by Amazon EC2 Extra-Large Instances to guarantee they are not the system bottleneck.
Using the SPACE4CLOUD tool we have been able to create a Palladio representation of the system and we run several experiments on it. Initially we performed the calibration of the performance model, indentifying PCM resource demand parameters. We ran several tests varying the number of users on Amazon EC2 considering 1, 1500, 3900, and 5100 users that correspond to a CPU utilization of 1%, 10 %, 50 %, and 80 %, respectively, in the real system. The response time measured during the tests have been used to estimate the CPU time of individual pages execution.

Class	CPU Time
login	0.0019
account_summary	0.0007
check_details	0.0023
logout	0.003

Table 1: PCM SPECweb Model - CPU Time

Results are reported in Table 1 and have obtained by considering the regression technique proposed in [19]. More details are reported in [14].
For model validation, we ran several additional tests considering 1, 10, 50, 100, 300, 900, 1500, 2700, 3900, and 5100 users. The response time measures were compared against the performance measures obtained by Palladio feed by the CPU estimates determined as result of the model calibration. We performed the tests twice with population values of 1500 and 3900 and three times the test with a population value of 5100. Palladio performance measures were obtained by simulations through Palladio Simucom by using 95 % confidence interval as stopping criteria. Figure 6 shows the response times related to the *login* page obtained in the real system and by the Palladio performance model. We can notice that the response times obtained with Palladio are similar to the ones obtained in the real system for light load. When the number of users increases, the difference between the model estimates and the real measures becomes larger but the model still remains conservative. This is more evident when considering the tests with 5100 users. For model validation, we performed three different measures and we obtained response time varying by an order of magnitude. We argue that this variability mainly depends on the system congestion or possibly to a VM live migration performed by infrastructure management software used by the provider. A similar behaviour has been obtained also for the other pages. The obtained results, unsatisfactory

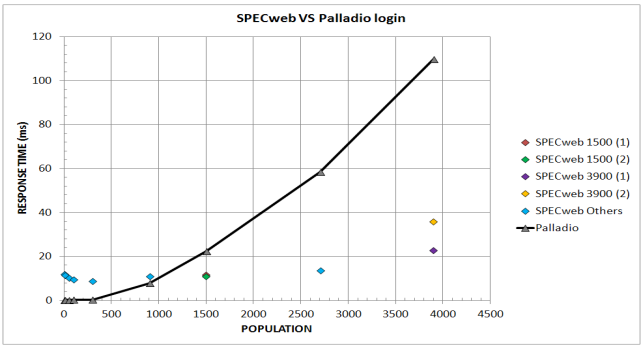

Figure 6: SPECweb2005 VS Palladio - *login* response times

for heavy workloads, demonstrate that the Layered Queueing Network (LQN) models used by Palladio needs to be extended in order to cope with cloud system performance variability. We are planning to extend LQN by considering phase type distributions [10]
In a second class of analysis, we compared the services of two well-known cloud provider, Amazon and Flexiscale, when running the SPECweb2005 *Banking* suite. For what concerns the workload profile, we considered a closed workload with a population that varies during the day. In particular, we considered realistic workloads created from a trace of requests relative to the web-site of a large university in Italy. We choose Amazon EC2 Medium Instances (webserver) and EC2 Amazon Extra-Large Instances (BESim). As far as Flexiscale is concerned we adopted servers with configurations with performance similar to those of Amazon machines, we used Flexiscale Servers with one virtual core (webserver) and two virtual cores (BESim); For simplicity, we report here only the results obtained during the peak hours, that is between 10:00 and 17:00. A suitable Allocation Profile (where the number of instances varies between four and five) has been generated and used for both Amazon and Flexiscale. The average response time on the two infrastructures was in the range 40-170 ms. We noticed that Amazon always performs better than Flexiscale and this is due to the fact that Amazon machines are more powerful than the Flexiscale ones. However, the difference between the response times is not constant for all the considered time intervals. On average, Amazon provides response times which are from 12%, to 20% lower than the ones of Flexiscale.
The SPACE4CLOUD tool also calculated the costs for both solutions that are reported in Figure 7. In this case, Amazon instances are more expensive than the Flexiscale, around 52.94% higher.

7. CONCLUSIONS

Cloud technologies are promising but provider selection, performance and cost evaluation are challenging for companies that decide to deploy their applications in the cloud. This work provides a model-driven approach to address this problem. We have proposed a Cloud Provider Independent Model to represent general cloud systems focusing on those aspects which affect performance and costs. Furthermore, we have implemented a Java tool that exploits the features offered by Palladio to run performance analysis for cloud

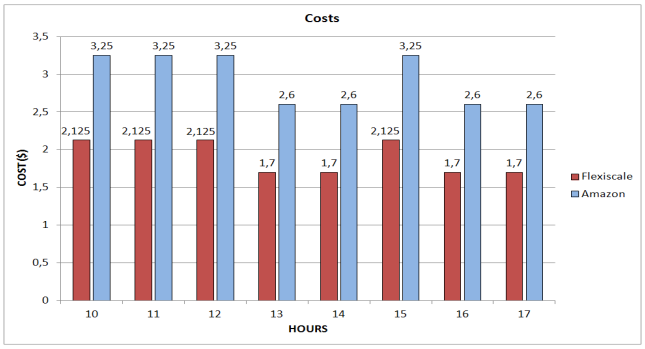

Figure 7: Amazon VS Flexiscale - Cost Comparison

systems and that support cost evaluation. The preliminary results we achieved to validate our approach have shown that the performance estimates are accurate for light workloads, while are very conservative for heavy loads. As part of our future work, we plan to extend LQN models to achieve higher accuracy in the performance estimates and take into account cloud performance variability. Finally, we plan to create an optimization engine able to explore the space of all the possible configurations returning to the end-user the best (in terms of costs, for instance) solution that satisfies the QoS constraints and supporting the cloud provider selection.

8. ACKNOWLEDGEMENTS

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds).

9. REFERENCES

[1] http://aws.amazon.com/ec2/#pricing.
[2] http://aws.amazon.com/s3/#pricing.
[3] http://www.windowsazure.com/en-us/pricing/details/.
[4] Specweb2005 - http://www.spec.org/web2005/.
[5] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *Software Engineering, IEEE Transactions on*, PP(99):1–1, 2013.
[6] A. Aleti, S. Stefan Björnander, L. Grunske, and I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of aadl models. In *Proc. of Workshop MOMPES 2009*.
[7] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *Software Engineering, IEEE Transactions on*, 30(5):295–310, 2004.
[8] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.
[9] E. Bondarev, M. R. V. Chaudron, and E. A. de Kock. Exploring performance trade-offs of a jpeg decoder using the deepcompass framework. In *Proc. of the Workshop WOSP 2007*.
[10] G. Casale, P. G. Harrison, and M. G. Vigliotti. Product-form approximation of queueing networks with phase-type service. *SIGMETRICS Performance Evaluation Review*, 39(4):36, 2012.
[11] M. Drago. *Quality Driven Model Transformations for Feedback Provisioning*. PhD thesis, Italy, 2012.
[12] M. L. Drago, C. Ghezzi, and R. Mirandola. A quality driven extension to the qvt-relations transformation language. *Computer Science - R&D*, 27(2), 2012.
[13] B. Eames, S. Neema, and R. Saraswat. Desertfd: a finite-domain constraint based tool for design space exploration. *Design Automation for Embedded Systems*, 14(1):43–74, 2010.
[14] D. Franceschelli. Space4cloud an approach to system performance and cost evaluation for cloud. Master's thesis, Politecnico di Milano, Italia, 2012.
[15] V. Grassi, R. Mirandola, and A. Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *Proc. of the Workshop WOSP 2005*.
[16] A. Kavimandan and A. Gokhale. Applying model transformations to optimizing real-time qos configurations in dre systems. *Architectures for Adaptive Software Systems*, pages 18–35, 2009.
[17] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010.
[18] R. Li, R. Etemaadi, M. T. M. Emmerich, and M. R. V. Chaudron. An evolutionary multiobjective optimization approach to component-based software architecture design. In *Proc. of Congress, CEC 2011*.
[19] Z. Liu, L. Wynter, C. Xia, and F. Zhang. Parameter inference of queueing models for it systems using end-to-end measurements. *Performance Evaluation*, 63(1):36–60, 2006.
[20] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proc. of Conference WOSP/SIPEW 2010*.
[21] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts. Constraint-based design-space exploration and model synthesis. In *Embedded Software*, pages 290–305, 2003.
[22] OMG. *UML Profile for Schedulability, Performance, and Time Specification*, 2005.
[23] OMG. A uml profile for marte: Modeling and analysis of real-time embedded systems, 2008.
[24] C. Rathfelder, S. Becker, K. Krogmann, and R. Reussner. Workload-aware system monitoring using performance predictions applied to a large-scale e-mail system. In *WICSA/ECSA*, 2012.
[25] C. Smith and L. Williams. More new software performance antipatterns: Even more ways to shoot yourself in the foot. In *Proc. of Conference CMG 2003*.
[26] M. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (puma). In *Proc. of Workshop WOSP 2005*.
[27] J. Xu. Rule-based automatic software performance diagnosis and improvement. In *Proc. of Workshop WOSP 2008*.