Enterprise applications cloud rightsizing through a joint benchmarking and optimization approach

Athanasia Evangelinou [a,*], Michele Ciavotta [b], Danilo Ardagna [b], Aliki Kopaneli [a], George Kousiouris [a], Theodora Varvarigou [a]

[a] *National Technical University of Athens, Athens, Greece*
[b] *Politecnico di Milano, Milan, Italy*

HIGHLIGHTS

- A methodology and tools that support the design and migration of applications to Cloud.
- The performance advertised by cloud providers is to be used carefully.
- The proposed benchmark procedure for migrated Cloud applications leads to reduced costs.

ABSTRACT

Migrating an application to the cloud environment requires non-functional properties consideration such as cost, performance and Quality of Service (QoS). Given the variety and the plethora of cloud offerings in addition with the consumption-based pricing models currently available in the cloud market, it is extremely complex to find the optimal deployment that fits the application requirements and provides the best QoS and cost trade-offs. In many cases the performance of these service offerings may vary depending on the congestion level, provider policies and how the application types that are intended to be executed upon them use the computing resources. A key challenge for customers before moving to Cloud is to know application behavior on cloud platforms in order to select the best-suited environment to host their application components in terms of performance and cost. In this paper, we propose a combined methodology and a set of tools that support the design and migration of enterprise applications to Cloud. Our tool chain includes: (i) the performance assessment of cloud services based on cloud benchmark results, (ii) a profiler/classifier mechanism that identifies the computing footprint of an arbitrary application and provides the best matching with a cloud service solution in terms of performance and cost, (iii) and a design space exploration tool, which is effective in identifying the deployment of minimum costs taking into account workload changes and providing QoS guarantees.

1. Introduction

Cloud computing is a disruptive phenomenon in ICT world, which has rapidly entered mainstream consciousness and gained a significant attention of various communities like researchers, business and government organization. It is obvious, in fact, that everyday services like Dropbox, Netflix or Instagram owe part of their success to the benefits of cloud such as the infinite number of resources, the ability to dynamically adapt (scale-up or scale down) accordingly to usage behaviors and the pay-as-you-go economical model.

With a rapidly increasing number of companies entering the cloud market and offering heterogeneous and constantly evolving technologies [1] the process of software design and implementation experienced a deep change. On the one hand, the cloud has meant for developers providing advanced cloud-based tools and abstractions for development, collaboration and deployment. Dynamic systems capable to react to workload fluctuations by adapting themselves in order to keep the performance unchanged can easily built and run delegating to the cloud provider the intensive tasks of infrastructure management and maintenance. On the other hand, performance unpredictability and vendor lock-in are just some of the issues that developers have to face and prove

* Corresponding author.
E-mail address: aevang@mail.ntua.gr (A. Evangelinou).

significant barriers to widespread cloud adoption. In fact, the increasing size and complexity of software systems, combined with the wide range of services and prices available in the market, puts the designer before the necessity to evaluate a combinatorially growing number of design alternatives [2] with the goal of finding a minimum-cost configuration that suits the application Quality of Service (QoS) requirements.

To carry out such a task, the system designer should consider a large number of alternatives and should be able to evaluate costs (that often depends on the application dynamics) and performance for each of them. This can be very challenging, even infeasible if performed manually, since the number of solutions may become extremely large depending on the number of possible providers and available technology stacks. What is more, in many cases the performance of these service offerings may vary depending on the application types that are intended to be executed upon them and their characteristics in terms of resource usage. Intensive RAM, CPU, disk or GPU usage may be toggled between applications that have different goals, such as scientific components, database instances or front ends.

A key challenge for customers before moving to Cloud, is to know application behavior on cloud platforms, in order to select the best-suited environment to host their application components in terms of performance and cost. In many cases the application owner might not be aware of how the migrated application component uses the available computing resources; for many application components the runtime behavior and usage of resources may not be known, mistakenly considered or altered due to structural changes during the software migration. On the other side, cloud providers might also be interested in knowing the application types hosted within their infrastructures to avoid interference effects of concurrently running VMs, which significantly degrade applications performance. Evaluating a specific and arbitrary application component over the entire range of offerings becomes a daunting task, especially when the deployment of the former may be subject to provider-specific or component-specific actions or code in each case. However, finding a more abstracted and common way for identifying both application profile and performance aspects of cloud environments may significantly reduce the effort needed in this task.

Finally, the performance characteristics of a cloud may change over time dramatically, depending on the congestion level, policies implemented by the cloud provider, and competition among running applications. Assessing the performance of an application in the cloud is a complex process that requires unbiased data and specialized models often implying skills that go beyond those commonly exhibited by software engineers. This situation calls for analytical techniques, models, application profiling and benchmarks that simplify the process of performance evaluation at design-time in order to support the user in the decision making process.

This work aims at proposing a methodology and a tool chain that support the migration and pricing scheme of enterprise applications to the cloud. Our tool chain exploits cloud benchmarking results and includes a profiler/classifier, which identifies the computational nature of a software component in a black box manner and a design space exploration tool, which is able to identify the cloud configuration of minimum cost fulfilling QOS constraints taking into account also daily workload patterns.

The remainder of the paper is organized as follows. In Section 2 an overview of the design methodology and the implemented tools are presented, while in Section 3 the benchmarking results acquisition is described. Section 4 introduces the profiling and classification tools, while Section 5 is devoted to the design time exploration tool. In Section 5 the experimental results achieved on a case study by combining and integrating the implemented tool chain are presented. In Section 7 related work in the respective fields is described. Conclusions are finally drawn in Section 8.

2. Overview of the design methodology and tools

The combined methodology of the tool chain appears in Fig. 1. Our approach along with the supporting tools, as an "all in one" solution allows the applications to exploit the offerings of the cloud providers in terms of performance and cost, taking into consideration their extracted computational behavior via profiling. The implemented methodology includes three phases: (i) Benchmarking (ii) Profiling and Classification and (iii) Assessment and Optimization phase.

Once the benchmark application types have been selected, the respective benchmark tests are executed automatically through the *Benchmarking Suite* on the candidate target cloud infrastructures, and their results are used to populate the *Raw Data DB*. Next, the application VM along with the benchmark application types are profiled and the computational profile of each is identified. When the profiling process is completed, the obtained computational profiles are used as an input to the *Classification Tool*. The latter determines the optimal cloud service solution for the given application in terms of performance and cost restricting the number of alternatives to be tested in the Optimization phase. Then, the stored results from the *Raw Data DB* along with the set of candidate providers and instance types obtained from the classification process, are imported within *Space4Cloud* resource DB. In the last step, the imported results are exploited during the *SPACE4Cloud* candidate solution performance assessment to evaluate how the performance metrics of the target application changes by varying the type and size of the hosting resources for subset of cloud providers that are considered as a target of the final deployment.

3. Cloud benchmarking

When considering the migration of existing applications to the cloud, it is critical to examine both the diversity of cloud providers and the varying performance issues of cloud services. Since there is an increasing number of providers offering cloud infrastructures and services a fair evaluation of such cloud systems is needed. System architects and developers have to tackle with this variety of services and trade-offs. Moreover, in some cases cloud providers offer their own metrics for evaluating and guaranteeing cloud QoS.

Hence, in order to measure performance aspects and select the cloud services that fit best to the application to be migrated, an abstracted process is implemented by using suitable tests and tools, namely benchmarking. The first step of this process is to define a set of performance stereotypes based on different application categories. The main goal of these stereotypes is to extract a number of performance characteristics of the provider that are necessary for meeting QoS requirements of the migrated cloud applications. The source of these characteristics are common application types that correspond to various popular applications and have been linked to respective benchmarks that can be used to indicate a specific offering ability to solve real-life computational problems. Thus, tests have been identified with specific workload patterns that can be mapped to concrete real world applications. Benefits of such a categorization include the ability to abstract offering performance capabilities on an application description level, thus being easily ranked according to user interests for a specific category.

Concerning the characterization of a service ability from a performance point of view, we use the *Benchmarking Suite* for benchmarking cloud platforms in order to extract performance-related data [3]. The set of the application types are reported in Table 1. The specific benchmarking tools provide a large number of application-level benchmarks which incorporate and reflect characteristics of CPU, I/O, network and data-intensive real-world applications. Therefore, these tools can stress the performance

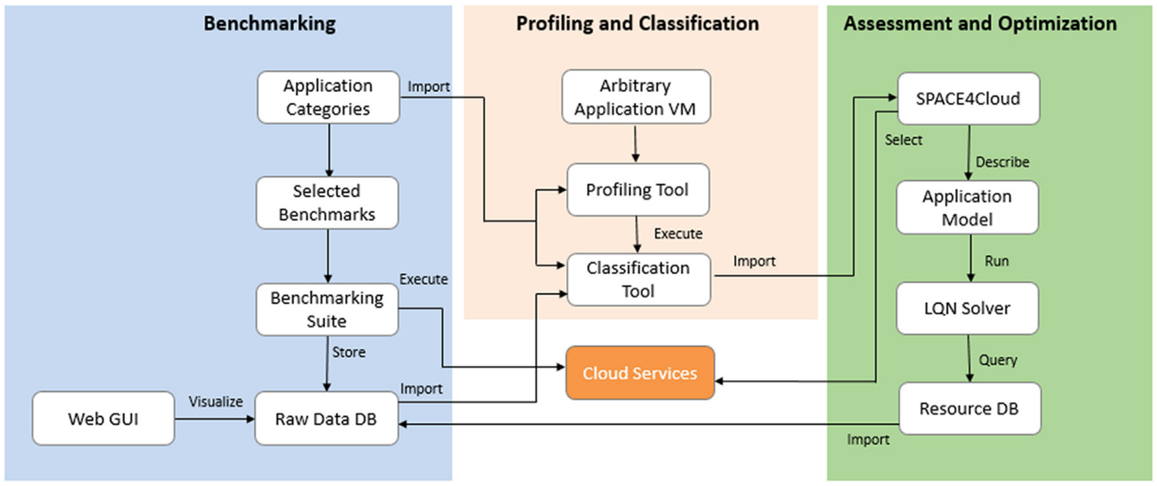

Fig. 1. Combined methodology of the joint mechanism.

Table 1
Benchmark tests used in the benchmarking process.

Benchmark test	Application type
YCSB	Databases
Filebench	File system and storage
DaCapo	JVM applications

aspects of several compute-intensive enterprise applications. What is more, they are supported by a large community of experts and have been proven reliable. Finally, users are already familiar with them, there is a lot of documentation and performance data are already available.

Such suite allows carrying out tests and getting performance metrics in a homogeneous and independent way. Metrics can then be used to collect, over the time, quantitative information about the performance offered and constitute the baseline for the selection of cloud services during the following model-based optimization phase. Results may be ranked either based on performance of the benchmark or by a combined Service Efficiency (SE) index with cost, appearing in the following equation:

$$SE = \frac{\#Clients}{w_1 \times delay + w_2 \times Cost}.$$

Service efficiency is a simplified, easily calculated and understood by the users metric that fulfills the following requirements:

- include workload aspects of a specific test
- include cost aspects of the selected offering
- include performance aspects for a given workload of a given application type
- give the ability to have varying rankings based on user interests and weights
- higher values are better.

A detailed analysis for the implementation of this formula is presented in [4].

4. Profiling and classification mechanism

Selecting a cloud provider that offers the best environment to host an arbitrary application in terms of performance and cost is a difficult challenge. To address this issue, we present a generalized mechanism which identifies the computational profile of an arbitrary application component, classifies it according to an already known and limited number of application categories and provides the fitting cloud offering by using the SE metric.

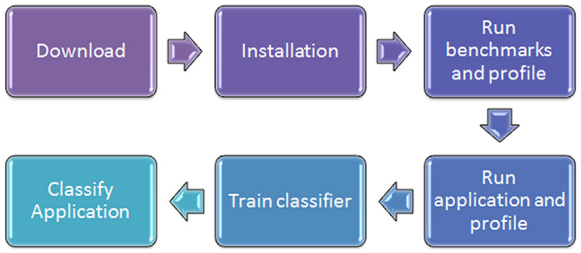

Fig. 2. Overall profiling and classification tool steps.

The benchmark application categories have been derived from the performance stereotypes defined in Section 3.

The first step to classify an arbitrary application component is to represent the performance footprint of each typical application category by a relevant benchmark application and in turn measure it by the *Profiling Tool*, one of the three components included in our innovative mechanism.

In a second step, the footprint of the application component installed in a VM in the same environment, is also measured by the same tool. Once the footprints of the benchmark applications are obtained, they are used to train the *Classification Tool*, the second component of our approach. Each of the obtained performance footprints consists of 23 features related to CPU utilization, pages faults and memory utilization, number of voluntary context switches and I/O operations network transfer. The *Classification Tool* can then determine the best fitting benchmark application type based on the arbitrary application components performance footprint by using a *kNN* (*k*-nearest-neighbor) approach. The identified benchmark footprints capture the essential runtime characteristics of the arbitrary application component and render them comparable. Finally, having the exact type of the application component and the benchmark results from different VM instance types, previously collected by the benchmarking process described in Section 3, the *Classification Tool* compares and identifies the best-suited cloud service. This comparison uses the SE metric determining the cloud provider that is most suitable for the given application in terms of performance and cost. Both tools are implemented in Java. The whole process of the generalized mechanism is described in Fig. 2.

4.1. Profiling architecture

The *Profiling Tool* architecture appears in Fig. 3. It is designed for Linux operating systems and provides a graphical environment for

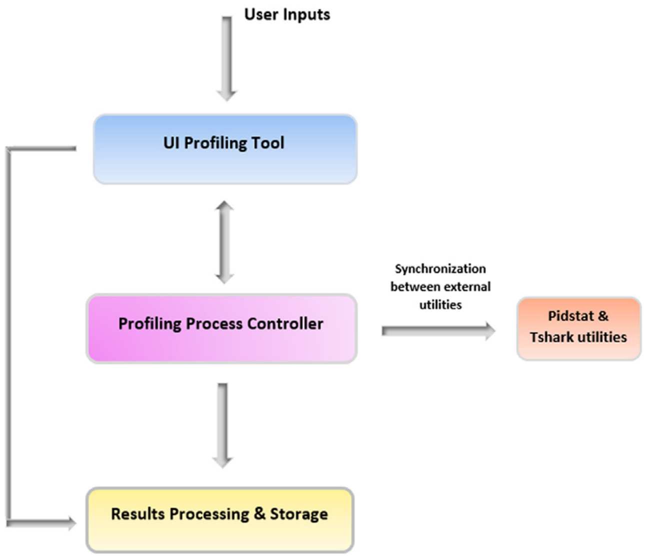
User Inputs
UI Profiling Tool
Synchronization between external utilities
Profiling Process Controller
Pidstat & Tshark utilities
Results Processing & Storage

Fig. 3. Architecture of profiling tool.

simplifying user interaction. Pidstat and Tshark external utilities, are initiated and executed by the *Profiling Tool* for performance analysis and monitoring the virtualized environments. Pidstat is used for monitoring individual tasks via PID (Process ID) and is focused on CPU utilization, I/O usage (including reading/writing to hard disk and virtual memory) and page fault information, while Tshark is responsible for monitoring network via IP. An important issue is that there are two different usage modes of the *Profiling Tool* (the application profiling and the benchmark profiling) and there is a significant difference in the level of automation between them.

The user through the UI component may set the number of the process identifier for the VM to be tested (the VM with the benchmarks or the one with the application), user credentials (username and password), paths for the start/stop scripts that control the workload traffic and the workspace (path for the produced data). The User Interface facilitates the communication between the user and the tool and makes their interaction quick and direct. In order to obtain the desired measurements, it is essential that the executing VM, the Pidstat process, the Tshark process and the *Result Collector* are executed at the same time in the best possible way. For this reason, the *Profiling Process Controller* component provides the basic mechanism for handling control and synchronization issues among all the tasks executing concurrently during the profiling process. The *Commands Executor* component serves as a library that is fully exploited by the Controller in order to perform all the essential actions related to ssh and Unix processes. Finally, the results are transformed into an appropriate format through results processing and storage component. As mentioned before, each of the profiling results can be seen as a vector which consists of 23 features representing performance and network measurements deriving from Pidstat and Tshark tools.

4.2. Classification architecture

Once both application and benchmark profiling procedures have been completed, the application developer uses the *Classification Tool* in order to map the application component to a predefined benchmark category and provide the fitting cloud offering by using the SE metric. The *Classification Tool* consists of three components: a GUI which facilitates the management of the classification process, a *Classifier*, which implements the *kNN* classification algorithm and the *Controller*, which is the general process supervisor and responsible for communicating with the DB system and

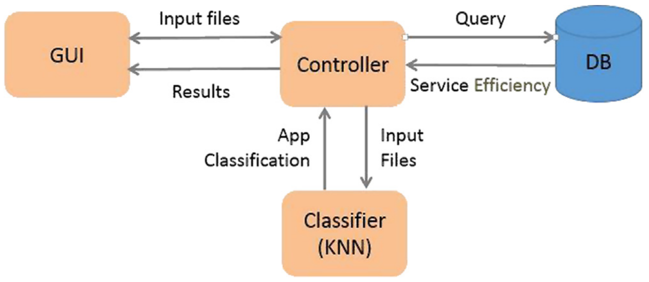

Fig. 4. Architecture of classification tool.

identifying the best instance type by calculating the SE metric. An overview of the components architecture appears in Fig. 4. Having collected the results produced during the previous steps, the user can proceed with the classification of an arbitrary application. Upon starting the classification, the *Controller* of the *Classification Tool* passes the relative information to the *Classifier*, which is responsible for matching the application to a predefined benchmark application category. Having the exact type of the application, the *Controller* interacts with the DB system, after obtaining the result from the *kNN* classifier, in order to measure the score of the SE based on normalized values for both cost and performance measurements. Finally, the *Controller* returns the ranking of the best instance types according to the highest SE, which are then used as input to the design exploration tool.

5. QoS assessment and optimization

Design-time exploration is supported by *SPACE4Cloud* (System Performance and Cost Evaluation on Cloud), a multi-platform open-source application for the specification, assessment and optimization of QoS properties of cloud applications. In particular, this tool allows software architects to describe, analyze and optimize cloud applications following the Model-Driven Development approach. The modeling language supported by SPACE4Cloud is MODACloudsML [5], which has been devised to describe cloud architecture and express cloud-specific attributes. Among other things, MODACloudsML includes architectural and QoS constraints (e.g., VM utilization or application average response time below given thresholds), and a user-defined workload [6], necessary to assess both performance and cost of the application under different load conditions. The workload is defined for a reference day consisting of 24 time slots. This choice is consistent with the most common pricing models which allow to lease virtual resources on hourly basis.

Users can define a cloud application by specifying the related models using *Creator4Clouds*, an open-source IDE [7], whereas information about the performance of the considered cloud resources are retrieved from an external SQL database (referred to as *Cloud Resource DB*) to decouple its evolution from the one of the tool.

SPACE4Cloud can be used either to assess the running costs and performance of a full-described solution (i.e., application and cloud configuration) according to a specific cost model [6] or, providing only the application model, to find a suitable (even multi-cloud) configuration that minimizes the running cost while meeting QoS requirements, starting from the best instance types identified by the *Classification Tool*. In order to assess the performance of the application under development, *SPACE4Cloud* translates the design models, which are an extension of Palladio Component Model set (PCM) [8], into Layered Queuing Networks (LQNs) [9], a particular group of performance models that are eventually solved by appropriate tools (in particular LINE [10] or LQNS [11]).

Fig. 5 highlights the main elements of *SPACE4Cloud*, as well as its dependencies, on other third-party components. The *Initial*

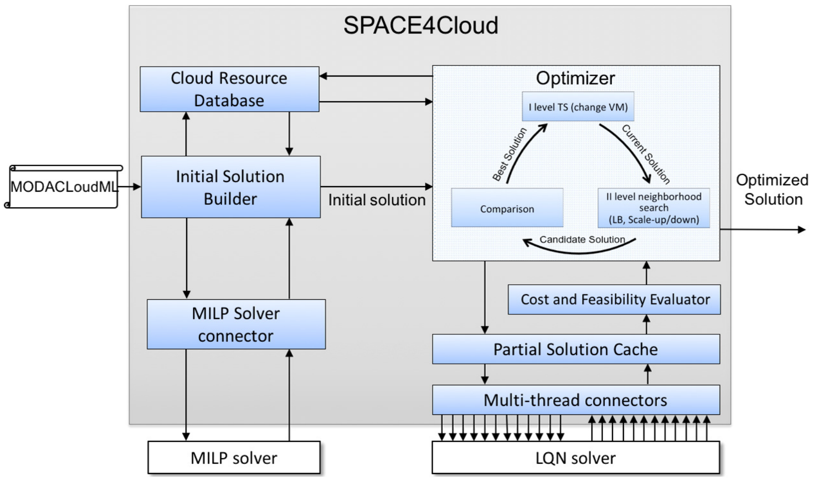

Fig. 5. Architecture of SPACE4Cloud tool.

Solution Builder is in charge of generating an initial deployment by solving a specific Mixed Integer Linear Program (MILP) built on approximate performance models and solved by a third-party solver. All experiments reported in this paper, were conducted using CPLEX [11]. The resulting solution is then used to bias the *Optimizer* components towards promising zones of the solution space [12]. A fast and effective local-search exploration procedure combining elements from both Tabu [13] and Iterated Local search [14] paradigms is the core of this component. The rationale is to improve iteratively a current solution by means of local moves, which are essentially transformation actions that, starting from a solution, lead to a new, possibly better one. More precisely, since the tool has to find the best possible cloud configuration in terms of VM type and number per application tier, the moves implemented within *SPACE4Cloud* are of two kinds. On the one hand, we devised a tabu search-based strategy that works at the level of the whole time horizon (24 h) and changes the VM type assigned to each tier among those that have been less recently considered; on the other, since changing the type of VM is a destructive move (the number of VMs for a certain tier can be far from being optimal after the move) a fast iterative method has been developed to react and re-optimize the number of VMs. Finally, since the workload can be variable over time with possible 24 different values this algorithm is applied in parallel on each time slots. The obtained new solution is then evaluated in terms of costs and performance; part of the assessment process consists in deriving a set of performance models, namely LQN models, which are then analyzed by an external solver. The *Cloud Resource Database* provides information about cloud provider offers, in terms of VM pricing models and performance metrics, which are necessary to create the LQN models. The database includes the results obtained by the *Classification Tool* to assess the performance of the different instance types (see Fig. 1).

LQNs have been preferred to other performance models as they can be used to represent complex systems (e.g., multi-tier applications) and competition among application requests at software layer. In this work we adopted LINE [10] for all the experiments, as to the best of our knowledge, it is the only solver able to take into account cloud performance variability through random environments [15]. What is worth to be pointed out at this point, is that the evaluation of a single candidate solution generated by the Optimizer is a time-consuming task resulting in a bottleneck for the entire optimization process. This happens because a solution comprises 24-hour deployment configurations, each of them leading to a different LQN model to be evaluated also in terms of costs and feasibility. For this reason, in order to speed up the evaluation process, we realized a multi-thread connector component managing the parallel evaluation of the 24 LQN models of a single solution and a cache-based proxy (Partial Solution Cache), to store and retrieve the evaluation of previous solutions for each hour in the time horizon. These additions greatly boost the overall evaluation process since the optimizer tends to generate similar solutions; thus by caching partial evaluations the tool is able to avoid unnecessary evaluations of performance models. An extensive analyses of the scalability of the *SPACE4Cloud* tool is available in [16] and demonstrated that a (local) optimal solution for instances of realistic size (e.g., including up to three cloud providers, tens of components and functionalities) can be identified in less than 30 min.

6. Experimental analysis

In the following sections, we provide an in-depth description of a case study based on the *HTTPAgent* application as well as on the achieved results for its optimal design. Section 6.2 reports the results of our benchmarking activity, while Section 6.3 presents the profiling process that has been followed in order to obtain *HTTPAgent* and benchmark profiles. In Section 6.4 the classification process of *HTTPAgent* is described, providing the optimal cloud service solutions for different workload executions. In Section 6.5 we demonstrate and validate our overall approach. Finally, a discussion of the achieved results is drawn in Section 6.6.

6.1. A case study: the HTTPAgent application

This section is devoted to introduce the case study used as testbed for the cloud migration approach presented in this work.

The experimental analysis presented throughout Section 6 has been performed by considering a software component called *HTTPAgent*, which belongs to *Modelio Constellation* modeling platform developed by Softeam [17]. Modelio Constellation is a web-based software-as-a-service application modeling environment. The Extended PCM models (part of MODACloudsML) allow to represent the considered application under different point of view. For instance, Fig. 6(c) showcases a deployment model for the application. The simplified version of Constellation considered here is characterized by four components. In particular there is an *Administration Server* that exposes a WebService interface to provide the users with a GUI to retrieve, modify and update the available projects and read their configuration. This component uses the *Administration Database* to store the access permission policies. The

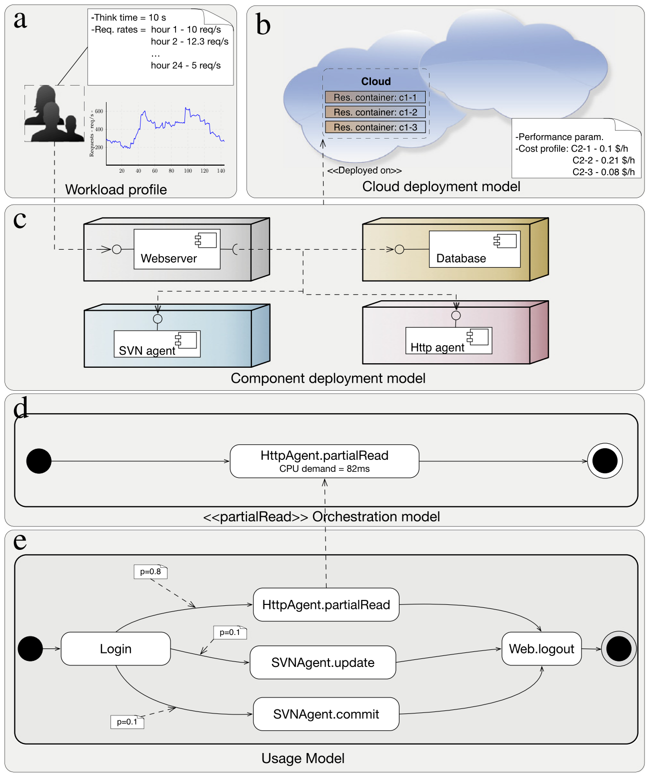

Fig. 6. Constellation platform: Extended PCM instance of the case study.

SVN Agent uses SVN to provide versioning, sharing and conflict detection with the aim of enabling multiple users to work simultaneously on the same project. To offload the previous component from some of the burden, the *HTTPAgent* has been added to provide read-only access to the models. Constellation can be deployed both on public and private cloud and it is supposed to undergo a variable workload overtime (see Fig. 6(a) and (b)).

To better understand how the system is used we employ the Usage model shown in Fig. 6(c). This model showcases a typical interaction between a user and the system in the scenario in which she/he has already retrieved a copy of the model to work on. In the picture three simple possible interactions are depicted, which are represented as branches. In the upper branch, after the login the typical user interacts with the system requiring a partial read of the project. This activity is by far the most common, happening the 80% of the time. Less frequently the user updates the model from SVN (10%) or commit her/his changes (10%). Note that, in the optimization step performance metrics are evaluated and, possibly guaranteed, along all individual execution paths and not only for the critical path [18]. The interaction scenarios and the associated probabilities shown in Fig. 6 have been gathered by mining the logs of the current version of Modelio. Finally, the Orchestration model reports for each exposed functionality the CPU demand and (where required) a graph of calls to other functionalities. In the case considered here, `HttpAgent.partialRead` shows a demand of 82 ms. This demand is calculated on Amazon m1.large instance.

In the frame of this work, we base our study only considering the *HTTPAgent* component as, according to the usage model, it has to handle most of the workload; therefore selecting the most suitable VM type and related number of replicas so as to minimize the execution costs is of paramount importance. Furthermore, it is also the only component of Constellation that can scale horizontally, for technical reasons.

6.2. Benchmarking process

In order to examine the performance aspects of cloud environments we proceeded to the execution of performance measurements on various cloud providers. The results from the benchmarking process were stored in an internal Raw data DB in order to be used during the analysis of the migration of the Constellation service and for selecting the best cloud environment in terms of performance.

During the experimental process for investigating differences in VM performance, we utilized workloads from DaCapo [19], YCSB [20] and Filebench [21] benchmarking tools. DaCapo is designed to facilitate performance analysis of Java VMs, while YCSB and Filebench measures databases performance and file system and storage respectively. The selected workloads from each test were running on instances in three different cloud environments: Amazon EC2, Microsoft Azure, and Flexiant. For all three cases different types of VM instances were considered. Regarding the selected application benchmarks from each of the benchmark tools, briefly are listed below, while a detailed analysis can be found in [22].

- **DaCapo**: avrora, eclipse, fop, h2, jython, pmd, tomcat, xalan.
- **YCSB**[1]: the considered benchmarks included workloads from *workload a* to *workload f*.
- **Filebench**: fileserver, varmail, videoserver, webproxy, webserver.

The execution of the tests took place at specific hours (at different time intervals) during a period of eight months (from July 2014 to February 2015) and the average values were extracted for each case. Moreover, the different time zones of the three different provider locations were taken into consideration so that the peak hours were the same in each zone. After completing the benchmarking process the results were retrieved from the local database, processed and the plots reported in Fig. 5 were obtained. For space limitation, the Figure reports the results of the Dacapo benchmark only but similar consideration can be obtained for Filebench and YCSB.

From the graphs it is evident that the performance for a specific workload varies and depends on both the type of workload and the VM instance size. For instance, for the DaCapo benchmark the workload performance across Azure (A2 Standard), Amazon (m1.large) and Amazon (m1.medium) is almost similar except for some cases where Amazon provides better results for the workload h2 while Azure was better for the avrora workload. What appears also from Fig. 7, and is more evident in the rating produced through the application of the Service Efficiency formula in Fig. 8, is the fact that in many cases lower capability VMs prove to be significantly more efficient. This may be due to the fact that the workload of the test does not drive the VM resources to the limit, thus when costs are included it is evident that higher capability VMs are not necessary for the specific case.

Concerning the cost for the execution of benchmarks on cloud environments, the time configuration for each workload execution from Filebench Suite, was set to 300 s. However, for DaCapo and YCSB workloads the total execution time depended on the VM instance types. In order to calculate the total time of execution, one should consider that the experimental benchmarking scenario lasted eight months, as previous mentioned, and performed three times a week and two times per day on all Cloud Services listed on Table 5. An example of the total time execution of all the workloads which run once on m1.small, m1.medium and m1.large instance types of Amazon EC2 and the relative cost are represented

[1] https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads.

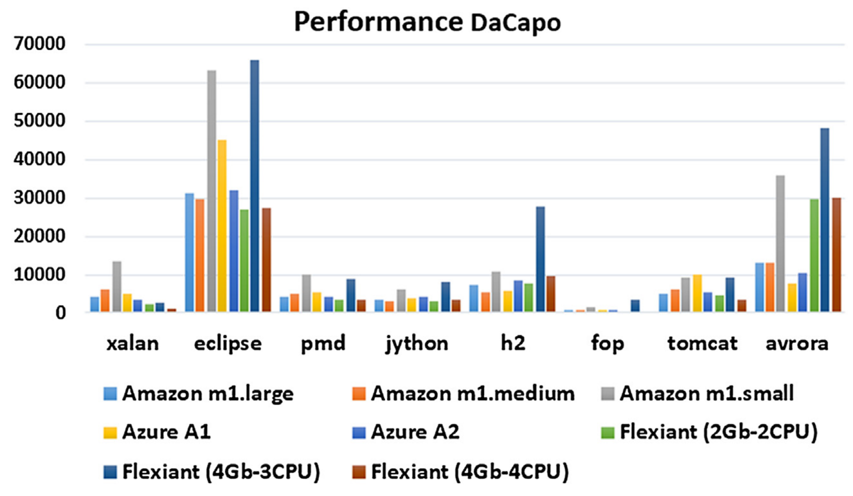

Fig. 7. Performance time in ms for DaCapo benchmark application.

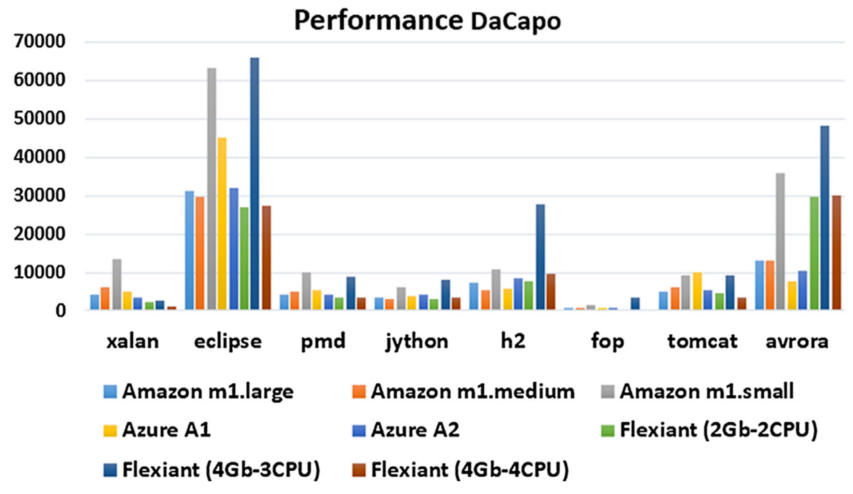

Fig. 8. Service efficiency metric for DaCapo benchmark application.

on Table 2. On the same table the relevant information with regard to the time consumed, the role and the cost for all the phases of the proposed approach are presented. The benchmarking of Cloud Providers is a process that should be repeated periodically at regular intervals over time. Thus, we recommend that only a Benchmark Provider should undertake this demanding role of providing a performance estimation of Cloud Providers to the customers with the relative cost.

6.3. *HTTPAgent and benchmark applications profiling*

Prior to the execution of the profiling process, both the *Profiling Tool* and the external utilities were installed on the physical host. The application profiling experiment was conducted in three steps. The first step included the installation of the *HTTPAgent* component in a VM (the image was created on VMware virtualization environment) and hosted locally in a Linux physical environment (Ubuntu 14.04), as the *Profiling Tool* has been designed for Linux operating systems.

In the second step, Apache Jmeter tool was used to generate workload towards the application. The Apache Jmeter was installed on a VM using the same configuration parameters as the aforementioned ones. However, Jmeter was installed on a separate VM hosted on a different physical environment in order to avoid misleading interference with Pidstat and Tshark results.

In the third step, Jmeter along with Pidstat and Tshark were synchronized through the *Profiling Tool* in order to generate traffic to the application and at the same time monitor the application execution performance and resources utilization. Finally, the results produced during the third step were processed and then stored in order to be compared with the exported results from the benchmarking phase.

Regarding the type of tested workloads, we do profiling and classification based on the typical case scenario and not on worst-case execution time (WCET) which would be too costly. However, if there are case specific workloads, we recommend the application developer to create artificial workloads (e.g. using Apache Jmeter) based on user sequence following a normal set of actions. For instance, in both HHTPAgent and NewsAsset multimedia cross-channel applications [23] the profiles of three distinct workloads have been created, indicating actual usage by the clients during normal operation. In turn, each of the previous profiles is used as an input to the classification phase. Concerning the HTTPAgent experiment three realistic user scenarios with 50, 200 and 400 users sending HTTP requests to the server were used, in order to test the application behavior. An example of the derived *HTTPAgent* profile obtained by considering 50 users is represented as vector in Fig. 9.

As far as the benchmark profiling is concerned, the benchmark applications were installed in a Linux VM in VMware player. In order to automate the process of the benchmarks installation and execution, *Benchmarking Suite* and benchmarking tools (Filebench, DaCapo and YCSB) were used. The total number of the benchmark

Table 2
Time consumed and cost for each phase of the proposed approach.

Phases of the proposed approach	Role	Time consumed	Cost($)
Indicative average time and cost for the entire benchmarking suite for one run in m1.small, m1.medium & m1.large VMs of AWS	Benchmark provider	132.5 min in m1.small 110.5 min in m1.medium 100.57 min in m1.large	0.097 m1.small 0.160 m1.medium 0.2933 m1.large
Indicative average time and cost for the proposed weekly scenario of benchmarking execution in AWS	Benchmark provider	795 min in m1.small 663 min in m1.medium 603,42 min in m1.large	0.583 m1.small 0.960 m1.medium 1.76 m1.large
Application profiling	Application owner	50 min	Local usage of pc resources
Benchmark profiling	Application owner	1350 min	Local usage of pc resources
Classification of the application	Application owner	ms	Local usage of pc resources
Service efficiency calculation	Benchmark provider	Depends on database size (usually ms)	Depends on Benchmark provider, number of users, amount charged per user

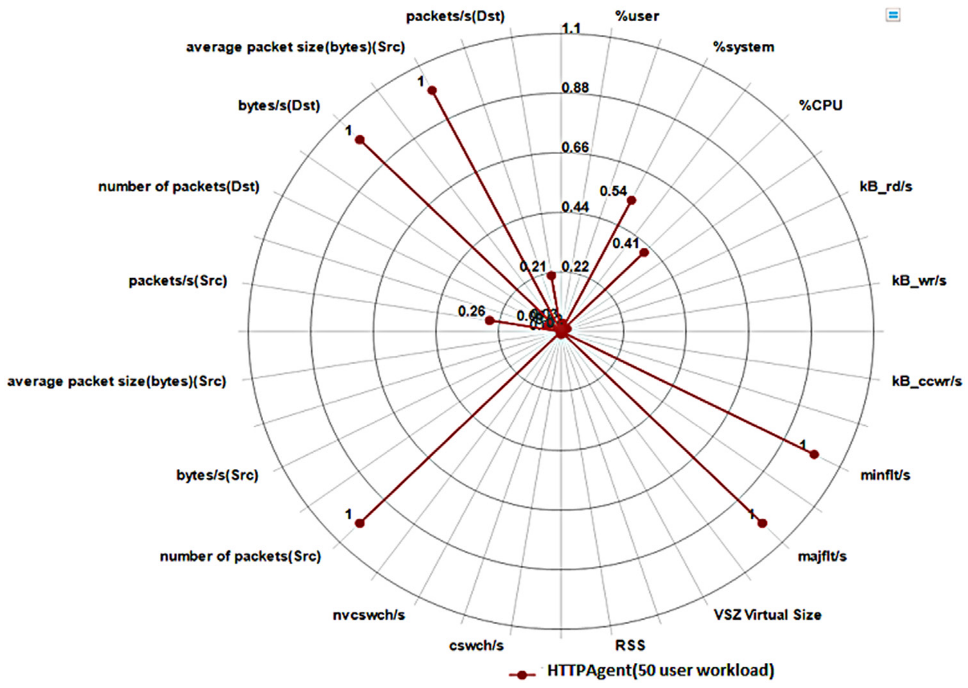

Fig. 9. HTTPAgent application profiling vector.

application workloads used in the profiling process was eighteen. However, in order to increase the accuracy and the reliability of the classification process, each of the benchmark workload was profiled several times. In each iteration a result vector was produced, consisted of the computational traces generated by Pidstat and Tshark. The aforementioned process resulted in a set of 180 vectors, used as input for the *Classification Tool*.

6.4. HTTPAgent classification and selection of the cloud provider

A far as the classification of the *HTTPAgent* application is concerned, the results of benchmarking and profiling are used as an input for the *Classification Tool*. Once the input files have been selected from the GUI, the classification process starts. *kNN* Classifier, one of the main components of the classification process, classifies the *HTTPAgent* application by calculating the cosine distance between the application and the benchmark profiles. In order to address the curse of dimensionality issue and produce a reliable prediction, since each of the profiling vectors is consisted of 23 features, we used Pearson correlation coefficient [24] for reducing this number. The mapping of the application profile to a benchmark profile is achieved by selecting the minimum distance detected by the *kNN* algorithm. The purpose of this algorithm is to use a dataset (benchmark profiles) in which the data points are separated into several separate classes to predict the classification of a new sample point (application profile). In our case, the *HTTPAgent* application profile is classified by a majority vote of its neighbors with the application profile being assigned to the benchmark profile, which is the most common among its k-nearest neighbors. In case one or more benchmark profiles appear with the same highest frequency, the selection of the “winning” benchmark profile is based on the minimum average value of the cosine similarity between the application profile and the benchmark profile.

According to the Classification Tool, the computational profile of *HTTPAgent* application in case of a light workload (50 users) behaves similar to the tomcat benchmark application (from DaCapo Suite), while in case of heavier workloads (200 and 400 users), the *HTTPAgent* behaves similarly the eclipse benchmark application, also from the DaCapo Suite. At this point, it is interesting to highlight the double behavior of HTTPAgent application depending up the workload type. In order for our approach to handle multiple classifications, as in HTTPAgent case, the application owner should be aware in advance of the application actual usage and provide the related workloads during the profiling phase. Note that, the set of VM type identified by the classification stage are then used as candidates by the optimization

Table 3
Service efficiency score of tomcat application benchmark for various instance types.

Benchmark application	VM instance type	Normalized service efficiency
tomcat DaCapo	amazonm1.small	**0.3644**
	flexiant2 Gb-2CPU	0.226
	amazonm1.medium	0.2169
	azureA1	0.2134
	azureA2	0.1913
	amazonm1.large	0.1732
	flexiant1 Gb-1CPU	0.1882
	flexiant4 Gb-4CPU	0.1722
	flexiant4 Gb-3CPU	0.1699

Table 4
Service efficiency score of eclipse application benchmark for various instance types.

Benchmark application	VM instance type	Normalized service efficiency
eclipse DaCapo	amazonm1.medium	**0.2586**
	flexiant1 Gb-1CPU	0.2451
	azureA1	0.2401
	flexiant2 Gb-2CPU	0.2387
	amazonm1.small	0.2059
	azureA2	0.1987
	amazonm1.large	0.1854
	flexiant4 Gb-4CPU	0.1623
	flexiant4 Gb-3CPU	0.1306

components. D-Space4Cloud then takes also into account the QoS constraints and workload variation during the 24 h and identifies a single VM type for the minimum cost deployment. Extracting the SE metric based on a 50%–50% weighted decision between performance and cost, we concluded that the optimal VM instance type for the 50 users workload is m1.small from Amazon EC2, while for the heavier workloads is the m1.medium instance type from the same cloud provider. Regarding the cost of the Service Efficiency calculation phase (using the Classification Tool), we suggest that this process should be provided as a service by a Benchmark Provider where the time consumed will depend on the database size which includes the stored benchmark results and the prices.

6.5. Validation of the approach

The purpose of the validation process was to demonstrate that the implemented mechanism operates efficiently and provides accurate results.

6.5.1. Validation of the profiling and classification tools

As already mentioned, using the *Classification Tool*, we have ranked the cloud offerings according to the SE metric, which uses the stored benchmarking performance measurements. In particular, knowing that the *HTTPAgent* computational profiles are similar to the tomcat and eclipse benchmark applications (part of the Classification process), we have identified that the best SE for these benchmarks was provided by m1.small and m1.medium respectively. Tables 3 and 4 present all the measured values of SE for tomcat and eclipse benchmark applications run on various instances in Amazon EC2, Microsoft Azure and Flexiant cloud environments, verifying that Amazon EC2 provides the best SE results. In order to validate the outcomes from our generalized mechanism, the most straightforward way was to actually deploy the *HTTPAgent* component on various instance types of Amazon EC2. Regarding the Apache Jmeter, it was also installed on Amazon EC2 on a m1.large VM, in order to avoid bottlenecks and to ensure that the required resources can be allocated for all the created threads. When the deployment was completed, the application was tested with the same workload user scenarios used in the *HTTPAgent* application profiling process. The performance results from the aforementioned procedures were used to calculate the normalized SE for each cloud offering. We avoided using a normalization interval including 0 as this in some cases may lead to infinite values. Finally, the hourly prices for the instance types used in SE formula were provided on the official website of Amazon EC2.

Table 5
Highest service efficiency score for the m1.small and m1.medium instance type.

Benchmark application	VM instance type	Normalized service efficiency
tomcat DaCapo	m1.small	0.3644
fop DaCapo	m1.small	0.3759
h2 DaCapo	m1.small	0.4067
fileserver Filebench	m1.small	0.6704
varmail Filebench	m1.small	0.6934
videoserver Filebench	m1.small	0.8419
c YCSB	m1.medium	0.1919
f YCSB	m1.medium	0.1961
b YCSB	m1.medium	0.2041
d YCSB	m1.medium	0.2098
a YCSB	m1.medium	0.2351
eclipse DaCapo	m1.medium	0.2586
pmd DaCapo	m1.medium	0.2908
xalan DaCapo	m1.medium	0.2914
avrora DaCapo	m1.medium	0.3024
e YCSB	m1.medium	0.3024
jython DaCapo	m1.medium	0.3095
webproxy Filebench	m1.medium	0.3622
webserver Filebench	m1.medium	0.4175

However, in order to perform an accurate and complete validation, we had to calculate the SE for all the benchmark workloads (Table 5) used in profiling and classification processes and select these benchmark applications that perform the highest SE score for the m1.small in 50 users workload case and m1.medium in 200 and 400 users workload cases (Table 6). In turn, we proceeded with comparing them with the SE of each *HTTPAgent* user scenario.

From the classification process we had concluded that for low workload (50 users) the application resembles the tomcat benchmark, while for average and high workloads (200 and 400 users) to the eclipse one. From Table 5, we would have been led to the decision to use a small VM for the 50 user workload and a medium VM for the other two cases. In order for the approach to be validated, we deployed the respective workloads in the three potential sizes per case. By measuring the SE index in Table 6 we can conclude that the validation process was successful since

Table 6
Normalized service efficiency and predicted optimal solution for HTTPAgent application.

HTTPAgent workload	VM instance type	Normalized service efficiency	Predicted optimal VM from profiling and classification process
50 users	m1.large	0.2005	m1.small
	m1.medium	0.2818	
	m1.small	**0.2961**	
200 users	m1.large	0.1875	m1.medium
	m1.medium	**0.2751**	
	m1.small	0.2026	
400 users	m1.large	0.2019	m1.medium
	m1.medium	**0.2818**	
	m1.small	0.2005	

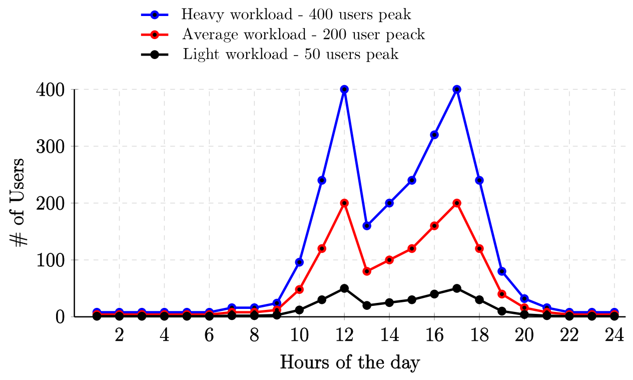

Fig. 10. Workload adopted for the experiment.

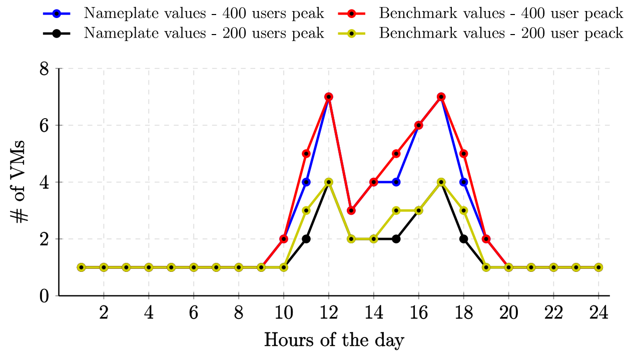

Fig. 11. Results considering variable workloads and C1 constraint.

the originally selected types demonstrated the optimal SE per workload. Another conclusion is that from the comparable SE indexes of the *HTTPAgent* deployment (benchmarking SE indexes cannot be directly compared), the globally optimal VM size and workload distribution (from the examined scenarios) is for 50 users per small VM.

6.5.2. *Validation of the optimization step*

This section is aimed at evaluating the repercussions of more precise information about the performance of cloud resources on the design-time assessment and optimization methodology and embodied in *SPACE4Cloud*. For this reason, we devised a specific two-phase experiment; in the first phase we execute *SPACE4Cloud* to optimize, under different workloads, the deployment configuration (namely, type and number of VMs over a daily horizon) for the *HTTPAgent* application on Amazon EC2 as cloud service; in this phase only nameplate (nominal) performance values available on the on-line catalogs for m1.small and m1.medium VMs have been considered. In the second phase the performance values obtained from the benchmarking process are used to update the cloud resource database used by SPACE4Cloud and the optimization process is repeated as in phase 1. Eventually, the results of the two phases are harvested and compared.

As said, different workloads have been considered for the experiment, in particular, we considered three constant workloads where the number of users is set at 50, 200 and 400, respectively, and three bi-modal workloads with (50, 200 and 400) users at its peaks, located in the central part of the day (see Fig. 10). All the considered workload belong to the "closed" type and are characterized by a *Think time* of 10 s.

For the experiment, an Extended PCM model for *HTTPAgent* application (a more comprehensive version of the one presented in Fig. 6) has been realized and the resource demands for the exposed functionality has been determined. The model is available at [15].

Furthermore, the experiment has been realized considering two Quality of Service (QoS) constraints: the first (C1) limits to 200 ms the average response time of the `partialRead` functionality, while the second (C2) set to 300 ms the 90th percentile of same the response time. The optimization time required to analyze the two scenarios mentioned above took between 5 and 10 min.

Results of this analysis for variable workloads and constraint C1 are depicted in Fig. 11. We can notice that, as was to be expected, all traces follow the trend defined by the workload. Moreover, in the analysis performed using the benchmark values on average a higher number of machines is needed to fulfill the QoS requirement (C1) with respect to the nominal case, resulting in a cost increment ranging from 5% to 8%, circa. This behavior is attributable to the gap existing between the nameplate and actual performance for the m1.small VMs. For sake of readability we removed from the picture the lines referring to the workload with 50-user peaks, as they are constant and contain only one machine per hour.

Tables 7 and 8 summarize the outcomes of the experiment. For each possible setting (workload type × considered constraint × Nominal vs. Benchmark performance values) the following information are reported: the VM type selected, the average number of machines per hour and the daily leasing cost. It is worth to spent some more time examining the effect of C2 on the VM selection: as a matter of fact, imposing the 90th percentile of the response time to remain under the threshold of 300 ms results is the selection of a more powerful machine, even when a light constant workload has to be served. This choice is eventually reflected on the daily cost, which is largely increased even if the average number of VMs is reduced. Furthermore, the use of larger VMs can lead to a potential under-use of resources when serving a given workload and this makes *SPACE4Cloud* to produce identical results when nominal and for benchmark performance values are considered. This is particularly true when the workload is relatively small and so is the difference between the nominal and benchmark values, as in the case under study.

6.6. *Discussion*

Identifying cloud services that best match the characteristics of an application to be migrated to the cloud providing QoS

Table 7
Results for variable workloads.

	Nameplate values			Benchmark value		
	Light	Average	Heavy	Light	Average	Heavy
	Constraints: C1 & C2					
VM type	m1.medium	m1.medium	m1.medium	m1.medium	m1.medium	m1.medium
Avg. # of VMs	1.00	1.46	2.21	1.00	1.46	2.21
Cost ($)	27.84	40.49	61.27	27.84	40.49	61.27
	Constraints: C1					
VM type	m1.small	m1.small	m1.small	m1.small	m1.small	m1.small
Avg. # of VMs	1.00	1.54	2.38	1.00	1.67	2.50
Cost ($)	13.92	21.43	32.98	13.92	23.14	34.72

Table 8
Results for constant workloads.

	Nameplate values			Benchmark value		
	Light	Average	Heavy	Light	Average	Heavy
	Constraints: C1 & C2					
VM type	m1.medium	m1.medium	m1.medium	m1.medium	m1.medium	m1.medium
Avg. # of VMs	1.00	2.00	4.00	1.00	2.00	4.00
Cost ($)	27.84	55.44	110.88	27.84	55.44	110.88
	Constraints: C1					
VM type	m1.small	m1.small	m1.small	m1.small	m1.medium	m1.small
Avg. # of VMs	1.00	4.00	7.00	1.00	2.00	7.00
Cost ($)	13.92	55.44	97.2	13.92	55.44	97.2

guarantees is a very challenging task. The results we achieved so far demonstrate that the performance advertised by cloud providers has to be used carefully, only as guidelines for the choice of the VM type, whereas it can be important to estimate the application performance in the cloud relying on an accurate and sound benchmarking procedure. Our results show, however, that the optimal solution depends on many factors including application characteristics, its workload and the target QoS constraints to be fulfilled.

The initial benchmarking step helps in two directions: (i) evaluating accurately how an application service demand changes across different cloud providers and VM types, (ii) filtering the set of cloud providers and VM types to be considered as candidate for the optimization step and final target deployment.

When the application is finally migrated to the cloud and put in production other challenges may arise. Indeed, cloud resources may fail or the application demand might change because the user behavior changes (e.g., the incoming workload deviates from the design-time forecast or the implemented services are subject to inputs different to the ones considered during the profiling activities).

To cope with such issues our design-time methodology has to be complemented by a run-time framework counterpart, which continuously monitors application resource consumption and incoming workload and periodically re-optimizes VMs allocation.

In [25,26], we proposed and evaluated a cloud middleware based on the receding-horizon control techniques for the run-time management of cloud applications.

A comparison between the allocation predicted at design-time and the real allocation enacted at run-time was performed. We noticed that the run-time allocation appears to be temporally shifted ahead with the respect to the design-time plan, in which when a new VM is required it is assumed to become available instantly. We explained such behavior through the observation of two factors: During the ramp up phase of new VMs a delay is introduced resulting by various delays, (e.g., the time required to have a new instance up and running and the time required by the load balancer to start splitting the workload considering also the new instantiated VM). On the other side, during the ramp down phase, if a VM instance results to be not required anymore, but it is still available for free until the end of the hourly billing period, it will be left running since the application has already been charged for it. We demonstrated that our overall approach is effective since *HTTPAgent* violates QoS constraints only in 2% of time intervals at a 10 s control period granularity (see [27] for further details).

7. Related work

Our work is related mainly with four research areas: cloud benchmarking, cloud applications performance assessment, application performance prediction on cloud and cloud applications design space exploration. With relation to benchmarking of cloud services, CloudHarmony [28] and CloudSleuth [29] are performance measurement tools that archive the test results and make them available for access through a web API. The former offers a vast number of customizable benchmarks and provides various performance metrics with focus on application, CPU, Disk I/O, Memory I/O etc. for various cloud providers online. However, there is one aspect that could be ameliorated with regard to this approach. Since a large number of benchmarking tests is included in the list it would be desirable to limit the scope of tests to interesting shift in measurements. CloudSleuth can built up a cloud application benchmark which provides availability and response time of various cloud providers online by continuously monitoring a sample application running on top cloud computing providers. However, the focus is only on web-based applications.

With regard to performance frameworks, PerfKit Benchmarker [30] is a living open source tool for benchmarking cloud, allowing developers to get a transparent view of application throughput, latency, variance and overhead. This framework includes popular benchmarking workloads that can be executed across multiple cloud providers. However, PerfKit tools are currently supporting only Amazon AWS, Microsoft Azure and Google Compute Engine. Finally, Skymark [31] is an extensible and portable performance analysis framework for IaaS clouds. It enables the generation and submission of real or synthetic complex workloads across IaaS cloud environments and it can analyze the impact of individual provision and allocation policy specified by the user, prior to the initiation of the experiment. Through the accumulation of statistical information regarding the workload execution, the framework is able to carry out a performance analysis of the underlying IaaS systems.

Concerning the prediction of a non-cloud application performance if migrated to a cloud infrastructure, CloudProphet [32] is a trace-and-replay tool which traces the workload of an application when running locally, and replays the same workload in the cloud for predicting the performance and costs. However, a major practical restriction is that it demands multiple runs of the application to acquire the appropriate workload to replay, and this overhead can be prohibitively high if the application has many synchronization events. Moreover, CloudProphet targets only at web applications while our approach covers as much as possible the most prominent application types. CloudCmp [33] is also similar to our work. CloudCmp provides a methodology and has as goal estimating the performance and costs of a non-cloud application when it is deployed on a cloud provider. A potential cloud customer can use the results to compare different providers and decide whether she should migrate to the cloud and which cloud provider is best suited for her applications. CloudCmp identifies the common services for various cloud providers, and then for each service identifies a set of performance metrics relevant to application performance and cost, develops a benchmarking task for each metric and runs the tasks on different providers. Though CloudCmp has a utility similar to our approach, however it does not define a common framework for all the benchmark tasks.

As far as quality modeling and assessment is concerned, the Object Management Group (OMG) introduced for this purpose two UML profiles specially tailored to model QoS, called Schedulability, Performance and Time (SPT) [34] and Modeling and Analysis of Real-Time and Embedded Systems MARTE [35]. These profiles allow to express some performance characteristics but still lack the proper support to model the heterogeneity of the cloud infrastructure. A similar approach led to PCM [8], a language that can be used to model an application and its non-functional properties. Once an application is fully described performance models can be automatically derived and solved in order to obtain a prediction on the application behavior. However, since the space of design alternatives for a single application can be very large, the task of finding the most suitable one is often arduous and time demanding; for this reason solutions able to guide the user have been proposed. The majority of them leverage particular algorithms to efficiently explore the design space in seeking for solutions that optimize particular quality metrics. Examples of techniques usually adopted are evolutionary algorithms and integer linear programming. Both ArcheOpterix [36,37] and PerOpterix framework [2,38] use genetic algorithms to generate candidate solutions. Other work presents an efficient tabu search (TS) heuristic [39] that has been used to derive component allocation in the context of embedded systems considering availability constraints. The SASSY [40] framework 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. Frey et al. [41] proposed a combined metaheuristic-simulation approach based on a genetic algorithm to derive deployment architecture and run-time reconfiguration rules to move a legacy application to the cloud.

Finally, this paper is an extended version of work published in [22]. In the current work we extend the existing mechanism by including *Profiling* and *Classification Tools* which are responsible for identifying the computational footprint of an arbitrary application component and finding the optimal cloud service respectively. Furthermore, we encompass a more extended validation by including and performing analysis on a real application.

8. Conclusions

Understanding how an arbitrary application component uses the compute resources is critical for its migration to cloud environments. Furthermore, the ability to measure cloud services on a variety of different application types enabled us to abstract the process of service measurement and selection, avoiding repeating such analysis for each and every individual application component one needs to deploy. However, the optimal selection for a specific application deployment of an application to be migrated to the Cloud may include the selection from a multitude of providers and consider the performance of the individual VM types available.

In this paper, we present a methodology and a tool chain consisting of three tools, i.e., *Profiling Tool*, *Classification Tool* and SPACE4Cloud. Our tool chain in conjunction with benchmark results obtained from a benchmarking process identifies the profile of an application and enables the ability to apply the optimal decision on the entire application chain level, while taking under consideration user interests in terms of cost and performance constraints. This has been demonstrated by the Constellation use case where different resources have been selected.

As future work, we intend to include more cloud providers in our study and extend the validation phase to include more cloud providers such as Flexiant and Microsoft Azure, so as to gain a complete insight of the efficiency of the implemented methodology.

Acknowledgments

The research leading to these results is partially supported by the European Community Seventh Framework Programme (FP7/2007–2013) under grant agreement n. 317859, in the context of the ARTIST Project and under grant agreement n. 318484 in the context of the MODACLOUDS project.

References

[1] Gartner Group. Hype Cycle for Cloud Computing, https://www.gartner.com/doc/2807621/hype-cycle-cloud-computing-, 2014.
[2] A. Koziolek, H. Koziolek, R. Reussner, PerOpteryx: Automated application of tactics in multi-objective software architecture optimization, in: QoSA 2011 Proc, QoSA-ISARCS'11, ACM, New York, NY, USA, 2011, pp. 33–42.
[3] G. Kousiouris, G. Giammatteo, A. Evangelinou, N. Galante, E. Kevani, C. Stampoltas, A. Menychtas, A. Kopaneli, K. Ramasamy Balraj, D. Kyriazis, T. Varvarigou, P. Stuer, L. Orue-Echevarria Arrieta, A Multi-cloud framework for measuring and describing performance aspects of cloud services across different application types, in: Proc. of MultiCloud 2014.
[4] G. Kousiouris, A. Evangelinou, D7.1- Definition and extension of performance stereotypes, 2014.
[5] N. Ferry, A. Rossini, F. Chauvel, B. Morin, A. Solberg, Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems, in: IEEE CLOUD 2013 Proc, IEEE Computer Society, 2013, pp. 887–894.
[6] D. Franceschelli, D. Ardagna, M. Ciavotta, E. Di Nitto, Space4cloud: a tool for system performance and cost evaluation of cloud systems, in: Proc. MultiCloud'13, ACM, New York, NY, USA, 2013, pp. 27–34.
[7] M.A. Almeida da Silva, D. Ardagna, N. Ferry, J.F. Pérez, Model-Driven Design of Cloud Applications with Quality-of-Service Guarantees: The MODAClouds Approach, MICAS Tutorial, in: MICAS-SYNASC 2014 Workshops Proceedings, pp. 3–10.
[8] S. Becker, H. Koziolek, R. Reussner, The palladio component model for model-driven performance prediction, J. Syst. Softw. 82 (1) (2009) 3–22.
[9] J. Rolia, K. Sevcik, The method of layers, IEEE Trans. Softw. Eng. 21 (8) (1995) 689–700.
[10] J. Perez, G. Casale, Assessing SLA compliance from palladio component models. in: SYNASC 2013 Proc., Sept 2013, pp. 409–416.

[11] G. Franks, T. Al-Omari, M. Woodside, O. Das, S. Derisavi, Enhanced modeling and solution of layered queueing networks, IEEE Trans. Softw. Eng. 35 (2) (2009) 148–161.
[12] IBM ILOG CPLEX optimization studio. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/.
[13] D. Ardagna, G. Gibilisco, M. Ciavotta, A. Lavrentev, A multi-model optimization framework for the model driven design of cloud applications, in: SBSE 2014 Proc. 2014.
[14] F. Glover, Tabu search: part i, ORSA J. Comput. 1 (3) (1989) 190–206.
[15] G. Casale, M. Tribastone, P.G. Harrison, Blending randomness in closed queueing network models, Perform. Eval. 82 (2014) 15–38.
[16] G.P. Gibilisco, A Methodology and a Tool for QoS-Oriented Design of Multi-Cloud Applications. Ph.D. thesis dissertation, 2016.
[17] SOFTEAM. Modelio. The open source modeling environment, https://www.modelio.org. 2015.
[18] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, IEEE Trans. Softw. Eng. 33 (6) (2007) 369–384.
[19] DaCapo Benchmarking Suite: http://www.dacapobench.org.
[20] B.F. Cooper, et al., Benchmarking cloud serving systems with YCSB, in: Proceedings of the 1st ACM Symposium on Cloud Computing, ACM, 2010, pp. 143–154.
[21] Filebench Benchmarking Suite: http://filebench.sourceforge.net.
[22] A. Evangelinou, M. Ciavotta, G. Kousiouris, D. Ardagna, A joint Benchmark-Analytic approach for design-time assessment of multi-cloud applications, Procedia Comput. Sci. 68 (2015) 67–77. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.
[23] ARTIST Deliverable D7.4 Classification methods and tools, ICCS and other partners, March 2015.
[24] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search: Framework and applications, in: Handbook of Metaheuristics, Springer, 2010, pp. 363–397.
[25] D. Ardagna, M. Ciavotta, R. Lancellotti, A Receding Horizon Approach for the Runtime Management of IaaS Cloud Systems, in: SYNASC 2014, pp. 445–452.
[26] M. Guerriero, M. Ciavotta, G.P. Gibilisco, D. Ardagna, A Model-Driven DevOps Framework for QoS-Aware Cloud Applications, in: SYNASC 2015, pp. 345–351.
[27] N. Ferry, A. Solberg, P. Jamshidi, R. Osman, W. Wang, S. Seycek, V. Gligor, R. Sucasa, A. Abherve, MODACLOUDS evaluation report Final version, http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D3.7.2_MODACloudsEvaluationReportFinalVersion1.pdf.
[28] CloudHarmony.com TM. Cloudharmony services. Online, July 2015, available at: http://cloudharmony.com/services/.
[29] Compuware Cloudsleuth Platform, available at: https://cloudsleuth.net/.
[30] PerfKitBenchmark, available at: https://github.com/GoogleCloudPlatform/PerfKitBenchmarker.
[31] A. Antoniou, Performance evaluation of cloud infrastructure using complex workloads (Master's thesis), Delft University of Technology, 2012.
[32] Ang Li, et al., CloudProphet: towards application performance prediction in cloud, ACM SIGCOMM Comput. Commun. Rev. 41 (4) (2011) ACM.
[33] Ang Li, et al., CloudCmp: comparing public cloud providers, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, ACM, 2010.
[34] OMG. UML Profile for Schedulability, Performance, and Time Specification, 2005.
[35] OMG. A uml profile for marte: Modeling and analysis of real-time embedded systems, 2008.
[36] A. Aleti, S. Björnander, L. Grunske, I. Meedeniya, Archeopterix: An extendable tool for architecture optimization of aadl models, in: Proc. of Workshop MOMPES 2009.
[37] I. Meedeniya, B. Buhnova, A. Aleti, L. Grunske, Architecture-driven reliability and energy optimization for complex embedded systems, in: QoSA 2010.