This is the pre-peer reviewed version of the following article: Performance prediction of cloud applications through benchmarking and simulation by Antonio Cuomo; Massimiliano Rak; Umberto Villano, which has been published in final form in International Journal of Computational Science and Engineering and is available at http://www.inderscienceonline.com/doi/abs/10.1504/IJCSE.2015.071362

DOI: 10.1504/IJCSE.2015.071362
Performance Prediction of Cloud Applications through Benchmarking and Simulation

A. Cuomo
University of Sannio,
Benevento 82100, Italy
E-mail: antonio.cuomo@unisannio.it
*Corresponding author

M. Rak
Department of Engineering,
Second University of Naples,
Aversa 81031, Italy
E-mail: massimiliano.rak@unina2.it

Umberto Villano
Department of Engineering,
University of Sannio,
Benevento 82100, Italy
E-mail: villano@unisannio.it

Abstract: As the cloud paradigm gains widespread adoption, the performance evaluation and prediction of cloud applications remain daunting tasks, not yet fully accomplished. Nevertheless, reliable performance figures are the key to take the cloud to the next step, in which it will be possible to predict the maintenance cost of the applications and to introduce richer Service Level Agreements between service providers and consumer.

In this paper we propose a methodology based on benchmarking and simulation that aims at predicting the performance of cloud applications developed through the mOSAIC framework. We prove the efficacy of the methodology on a real case study, showing how it is possible to predict performance indexes (throughput, message queue length, . . . ) under a generic workload, using pre-acquired benchmark results and simple simulation models.

Keywords: Discrete-event Simulation; Cloud Computing; Concurrency; Platform-as-a-Service; Process-oriented Simulation

1 Introduction

The emerging cloud computing paradigm is based on a pay-per-use model, where resource provisioning is delegated to the network (behind the scenes, to one or more remote cloud providers). Resources are acquired only when actually needed, and they are charged on the basis of their actual usage. In addition, as pointed out in the NIST (Mell and Grance, 2011) definition, the resources are offered in self-service modality, since no human involvement is needed other than the request from the service user.

As the cloud paradigm grows in maturity, in research and enterprise clouds new building blocks can ease the development of cloud applications. Founded on higher level abstractions as messaging and data storage systems, cloud applications are designed independently of the specific provider, and then executed on platforms deployed on resources leased from different cloud providers. Applications are able to acquire new cloud resources (i.e., components and virtual machines) in a (semi) automated way, as service demand grows. The mOSAIC project (mOSAIC Project, 2010) is one of the first examples of this approach to cloud application development.

It is well known that cloud-based services tend to exhibit good scalability as the number of user requests grows and additional resources are elastically acquired. However, at the state of the art, in the literature very few results are available on the prediction and evaluation of cloud application performance. In our opinion, this is a research problem that is worth of further investigation. A reliable prediction of the performance that a given application can obtain when executed in a cloud enables a more thorough evaluation of business risks (as performance is directly linked to management and maintenance costs, due to the use of the pay-per-use paradigm), and the introduction of richer Service Level Agreements (SLAs) for the negotiation of the quality of service between provider and consumer.
In this paper, we propose a simple methodology that allows to obtain performance predictions of mOSAIC-based cloud applications. The proposed approach is based on the derivation of a set of custom benchmarks from the target application. These benchmarks are used to measure the performance parameters that make it possible to tailor the simulation model to the set of available resources. The execution of the simulation model leads to the performance prediction of the application. The mOSAIC API includes a framework (mOSAIC benchmarking framework) composed of predefined components and configuration files that help to build custom benchmarking applications. These can be run on-the-fly on the target resources (i.e., exploiting the cloud services provided by cloud providers). The figures obtained through the benchmarks are used to determine the parameters of simulation models of the cloud components. These models are described through JADES, a Java-based library (Cuomo et al., 2012a) that allows to specify and evaluate process-oriented discrete-event simulations.

The main drawback of such approach is, obviously, that benchmarking and simulation are resource-consuming, in that they affect the overall application resource usage. However, benchmarks and simulation can be executed at application start-up, or when the application is not under stress. The overhead introduced is likely to be negligible as compared to the savings obtained through the optimization of resource usage made possible by the availability of performance predictions. A main goal of future research is surely to limit the resource overhead introduced by benchmarking and simulation. The remainder of the paper is structured as follows. Section 2 provides background material on the mOSAIC platform. Section 3 describes the proposed performance evaluation methodology. The following section shows how to apply the proposed methodology through a simple case study. The paper ends with a section dedicated to related work and with our conclusions and plans for future work.
2 mOSAIC: Development of Cloud Applications

As discussed in the introduction, the main goal of this paper is to provide a technique that enables developers to predict the performance of applications developed using mOSAIC. The presentation of the performance prediction technique requires the knowledge of a few basic concepts on mOSAIC and mOSAIC application development which are summarized here for completeness’ sake.

mOSAIC (Petcu et al., 2011; Craciun et al., 2011) is a framework that provides an API to develop cloud applications, which are thereafter executed in a leased environment provisioned and controlled by the mOSAIC run-time. Hence, the target user for the mOSAIC solution is the application developer (mOSAIC user).

In mOSAIC, a cloud application is structured as a set of components running on cloud resources (i.e., on resources leased by a cloud provider) and able to communicate with each other. Cloud applications are often provided in the form of Software-as-a-Service, and can also be accessed/used by other users than the mOSAIC developer (i.e., by final users). In this case, the mOSAIC user acts as service provider for final users.

A mOSAIC application is built up as a collection of interconnected mOSAIC components. Components may be (i) core components, i.e., predefined helper tools offered by the mOSAIC platform for performing common tasks, (ii) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (iii) cloudlets. Basic mOSAIC components, and the symbols used to represent them, are described in Figure 1. Queues and Key Value Stores provide support for inter-component communication and storage of data. Cloudlets are the programmable components that encapsulate the core logic of the specific application. The mOSAIC API, through which cloudlets are described, promotes an event-driven programming style, which results in an asynchronous execution model that allows higher scalability. Cloudlets are executed in special containers and are able to self-scale and to interact with any kind of cloud resource.

A cloud application is described as a whole in a file named Application Descriptor, which lists all the application components (cloudlets), the cloud resources (queues and key-value stores) and the details of their interconnections. A mOSAIC developer has the role both of developing new components and of writing application descriptors that connect them. All the mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the Platform Manager, which makes it possible to manage a set of virtual machines hosting the mOS as a virtual cluster, on which the mOSAIC components are independently managed. It is possible to increase or to decrease the number of virtual machines dedicated to the mOSAIC Application, which will scale in and out automatically.
3 Performance evaluation of mOSAIC applications

In the previous section we have described the mOSAIC approach to component-based application development in the cloud. A key point is that a mOSAIC application is fully described in its components and interconnections through the Application Descriptor. Using the descriptor as a recipe for instantiating the components on the mOSAIC platform, the resulting application can then be provided in the form of Software-as-a-Service, to be accessed by final users.

Moving from these premises, we propose a methodology to generate from the Application Descriptor a performance model of the application that can be used to predict its performance and to tune its execution. It should be noted that in this paper we will show how to build benchmark application by hand. However, the procedure can be fully automated, avoiding any human intervention for the generation of the benchmarks and of the simulation model.

The methodology is made of the following steps (Figure 2):

1. Derive the benchmarking applications from the Application Descriptor: Benchmark applications stress a well-defined portion of the system, like single components, to obtain performance figures as throughput or response time.

2. Derive the simulation model from the Application Descriptor: The model is an abstracted representation of the mOSAIC application, in which every component is replaced by a simulation process that mimics its flow of activity. The interconnected simulation processes model the timing behavior of the applications. Models are parameterized using the figures obtained running the benchmarks, so that predictions are bound to the actual execution platform:

3. Execute the benchmarks and collect the results: The benchmark applications are launched in the target mOSAIC cloud, obtaining performance figures for every component on the actual execution platform;

4. Obtain a performance prediction by executing the simulation model: The simulation model can be executed with multiple synthetic workloads, using benchmark results as parameters, obtaining performance predictions for different scenarios of interest.

In the following, we will describe each step in detail.

3.1 Benchmark Generation

The Benchmark generation step aims at producing a set of benchmarking kernels that stress the components of the mOSAIC application, collecting performance indexes. The mOSAIC environment provides a built-in framework for the development of benchmark applications, which has been described in (Rak and Aversano, 2012). In this work, we rely on the mOSAIC benchmarking facilities to test each component of the whole application in isolation.

Our initial focus is on benchmarking the basic mOSAIC components, namely Queue Servers, KV Stores and Filter Cloudlets (i.e., cloudlets that receive messages, elaborate them and send out or store the result). Extending the benchmarking to more complex components, e.g. to cloudlets that do not operate as a filter, is part of our plans for future work.

For each class of components, we build up a dedicated benchmark application, which stresses the component and measures its performance indexes. These
Performance indexes will be used as parameters for the simulation models described in following sections. The benchmarking applications can be executed on the actual target resources, leased from a cloud provider, and stress the components while they are executed, varying their load (e.g., number of concurrent requests, number and length of concurrent SET/GET operations, ...). The results of benchmarking application execution are then collected to feed simple regression models that make it possible to obtain performance parameters for the simulation models.

Figure 3 illustrates the architecture of the Queue benchmark application. The main goal of this application is to measure the waiting time of each message in the queue. We assume that for each message, such waiting time depends on the total number of messages that are already waiting for being processed when the message is put in the queue. As a consequence, it is possible to stress the Queue component through a set of different experiments, each of which submits a fixed number of concurrent messages, and evaluating the average response time.

Figure 4 illustrates the architecture of the KV Store benchmark application. In this case, the application stresses the KV store component by submitting a fixed number of concurrent requests to the storing component. The stressing cloudlets perform a sequence of SET/GET operations on the system. The application evaluates the average number of completed operations per second.

The last application stresses a cloudlet, which act in reaction to the receipt of a message. When the input has been processed, the component sends out a new message on the queue. Figure 5 illustrates the architecture of such benchmark application, which measures the processing time (i.e., the time spent from the moment in which the request is sent on the queue and the time when the result is received by the consumer.

It should be noted that all the benchmark application have a very similar architecture, due to the exploitation of of the mOSAIC benchmarking framework, which...
offers base components as the benchmark controller and the workload generator.

### 3.2 Simulation Generation

In the second step, the Application Descriptor is used to generate a simulation model of the application. The simulation model predicts the performance behavior (i.e., response time, resource consumption, throughput, ...) of the whole application based on the performance figures obtained from the benchmarks.

In our approach, the simulation of a mOSAIC application is achieved by the following procedure:

- for each class of components, we have implemented a simulation object that mimics the behavior of that kind of component. A concrete example is shown in Listing 1, which contains a relevant portion of code of the mOSAIC Queue component model;
- the interconnections between application components are mapped into corresponding connections between the simulated counterparts;
- the timing behavior is modeled by simple regression models derived from the results of the benchmarks.

In order to develop the simulated components, as shown in Listing 1, we rely upon the JADES simulation library (Cuomo et al., 2012b), which allows the description of process-oriented simulations in Java. To model a mOSAIC application, every component is represented by a simulation process that describes its behaviour. A simulated mOSAIC component can perform two basic operations: communication with other components and processing.

Communication is represented by simulated resources, provided by the library, that allow a simulation process to send and to receive messages from/to another simulation process. Processing is simulated through the JADES library hold function, that delays the process for a given amount of time. As anticipated, this amount of time is determined by a regressive model built using the results from the benchmarks. For example, the Queue component processing time depends on the size of the message and on the number of messages currently in the Queue. The benchmark of the Queue stresses the Queue by sending a variable number of messages and measuring the response time corresponding to the size of the message and to the number of messages in the Queue.

A noteworthy feature of the solution devised is that the simulation models expressed through the JADES library can be easily evaluated through the mJADES platform (Cuomo et al., 2012a). mJADES is a system developed recently that supports the distribution of multiple JADES simulations on cloud resources. The mJADES simulation system is founded on a Java-based modular architecture. The mJADES simulation manager produces simulation tasks from simulation jobs, and schedules them to be executed concurrently on multiple instances of the simulation core. This is a process-oriented discrete-event simulation engine based on the JADES simulation library. The outputs from the runs are handed on to a simulation analyzer, whose task is to compute aggregates and to generate reports for the final user. mJADES has been developed as a collection of mOSAIC components (cloudlets and resources) as described in Section 2, and so the evaluation of the models can be conducted on any mOSAIC platform, whether it is the one used by the application under study, or a different one.

#### Listing 1: The Queue Simulation Process

```java
public class Queue extends it.unisannio.ing.perflab.jades.core.Process {
    private Mailbox inputMailbox;
    private Mailbox outputMailbox;

    public Queue(String name, double beta0, double beta1, double beta2) {
        super(name);
        this.inputMailbox = new Mailbox(name + " inputMailbox");
        this.outputMailbox = new Mailbox(name + " outputMailbox");
    }

    public void send(Object m) {
        this.inputMailbox.send(m);
    }

    public void run() {
        while (true) {
            int msgSize = (Integer) inputMailbox.receive();
            int msgInQueue = inputMailbox.msg_in_queue()+1;
            hold(beta2 * msgSize + beta1 * msgInQueue + beta0);
            outputMailbox.send(msgSize);
        }
    }

    public Object receive() {
        return outputMailbox.receive();
    }
}
```

### 4 A Case Study: Performance Evaluation of an XML Document Analyzer

In order to show the practical use of the proposed methodology, we will deal here with the performance evaluation of a mOSAIC-based XML document analyzer. This is a simple application, but one that allows to study the interaction of three main mOSAIC components (Queue, Cloudlet, KV Store). The structure of the target application is shown in Figure 6. The XML Analyzer
clouddlet checks for the presence of incoming XML files in the input queue. For each received file, the XML Analyzer clouddlet counts the number of occurrences of every tag inside the file and stores the results as a pair \(<\text{filename}, \text{tagcounts}>\) (where \(\text{tagcounts}\) is a collection of \(<\text{tag}, \text{count}>\) pairs) into the KV store.

The first step for the application of the proposed performance evaluation methodology is the generation of benchmark codes for all the components of the application, namely the Queue, the KV Store and the main XML Analyzer Cloudlet, which has the typical Input/output behavior described in the previous section. The structure of the benchmarks for these components has been shown in section 3.1. Then, the simulation model must be generated. As has been said in section 3.2, each component is modeled by a JADES simulation process which communicates with other components and simulates delays, whose length are computed by regression formulas taking into account the results obtained by executing the benchmarks. The simulation model is made up of component simulation processes, plus a simulation driver that instantiates them and launches their execution. The code for the process simulating the Queue has been presented in section 3.2. The codes for the XML Analyzer process and the KV Store process are similar in structure, and so they will not be shown here for brevity’s sake.

The final steps of the methodology involve the execution of the benchmarks and the collection of the performance parameters that are used during the execution of the simulator to evaluate the delays of each elementary operation.

### 4.1 Experimental Results

To perform an initial experimentation of the performance evaluation methodology and to validate the simulation results, we have considered three types of workload. Each workload consists of a series of analysis requests, and each request is relative to one of three sample XML files of different size (4, 8 and 12 KB), chosen with equal probability. All the three workloads used in our experimentation (Figures 7(a), (c) and (e)) are similarly shaped, in that they are made up of a sequence of phases low workload \(\rightarrow\) peak workload \(\rightarrow\) low workload. All of the requests in each elementary phase have a Poisson distribution; the phases are characterized by different duration and mean inter-arrival time, as shown in Table 1.

For simulation purposes, every workload is represented by a file containing a sequence of \(<\text{nextwaittime}, \text{nextmessagesize}>\). The workload files are used to drive the simulator, suitably parameterized by the results of the benchmarks.

A mOSAIC testbed has been set up to perform experimentation. The testbed consists of a mOSAIC environment over an OpenNebula cloud infrastructure. The details about software and hardware configuration are synthetically reported in Table 2.

We have first executed on this testbed the benchmark applications described before to collect performance figures for each component of the target application in isolation. These figures have been used to derive the regressive parameters for the simulation model. After running the simulation of the three workloads, we have validated the obtained results, feeding the actual application with the same workloads and collecting performance measurements.

The metric that we have selected for evaluation is the number of messages in the Queue component: this is representative of the level of congestion in the application, an information useful to understand when to acquire new resources from providers in order to scale up or down the application.

For every workload, a plot of the measured versus the predicted performance is shown in Figures 7(b), (d) and (f). For all of the three workloads, the predictions follow closely the trend of the measurements on the real system, but are systematically slightly lower. The fact that all predictions appear to be optimistic is due to the simplifications of the complex interactions of the real application neglected to construct a simple and manageable model. This leads to an error in the estimated completion time (when the queue length drops finally to zero, as all the requests have been processed) which ranges from 10.87% to 12.5%. However, we think that the predictions obtained are a good compromise between accuracy and complexity of simulation. Similar accuracy has been obtained for the other performance parameters.

#### Table 2: Testbed Configuration

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU type</td>
<td>Intel Xeon E5430 2.66Ghz</td>
</tr>
<tr>
<td>Cpus per node</td>
<td>2</td>
</tr>
<tr>
<td>Cores per node</td>
<td>8</td>
</tr>
<tr>
<td>RAM per node</td>
<td>6 GB</td>
</tr>
<tr>
<td>Operating System</td>
<td>Ubuntu Linux</td>
</tr>
<tr>
<td>Kernel Version</td>
<td>3.0.0-12</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Xen 4.1.1</td>
</tr>
<tr>
<td>Cloud manager</td>
<td>OpenNebula 3.4</td>
</tr>
</tbody>
</table>
Table 1  Workload specification

<table>
<thead>
<tr>
<th>Workload</th>
<th>Phase 1</th>
<th>Phase 2</th>
<th>Phase 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Duration</td>
<td>Mean</td>
<td>Duration</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>0.2</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 7: Performance predictions for different application workloads
5 Related Work

The core of our proposal is the use of a simulation-based approach to application performance prediction, which finds its roots in the field of High Performance Computing (HPC). The use of simulation in the HPC context is widely discussed in a number of papers, like (Aversa et al., 1998; Di Martino et al., 2007). Recent efforts in this field are documented in (Achour et al., 2011) and (Clauss et al., 2011).

For component-based software systems (CBSS), most approaches available in the literature focus on integrating performance prediction and evaluation techniques at design time, i.e., when the system implementation is not available. An exception, which has some similarities with our approach, is the COMPAS system (Mos and Murphy, 2002). (Koziolek, 2010) presents a complete survey of performance strategies for CBSS.

CloudCMP (Li et al., 2010) is one of few examples of simulation-based performance predictions of cloud applications, where the goal is to compare different cloud providers and to select a suitable one. Another notable work in this area is CloudSim (Calheiros et al., 2011), which targets the simulation of the entire stack of software/hardware components in a cloud infrastructure.

6 Conclusions and Future Work

Cloud applications developed using provider-independent frameworks as mOSAIC have the advantage of being free from the vendor lock-in problem. On the minus side, due to the high number of abstraction layers involved, it is often very hard to tune and predict their performance. Cloud elasticity, which lets developers to vary dynamically the resources and the services requested to a cloud provider, makes even harder the resource optimization task. In this paper we have proposed a simulation-based methodology that helps the developer to predict the performance of his applications, given the actual resources used and the requests of the end users.

The approach proposed here is based on the use of performance simulations tuned by benchmarking. The benchmarks are based on ad-hoc applications that can be easily re-run to take into account possible variations of the resources acquired (e.g., when they are leased from a different provider). We have shown through a real-world case study, run on a private cloud, that it is possible to collect in an easy way benchmarking information and to use it to predict the behavior of a real cloud application, under different working conditions. The measurement results have shown that simulation predicts performance indexes with an error of about 15%. The predicted figures may be of help for developers who wish to optimize off-line their applications, to reduce the usage of cloud resources (and the related costs).

In future work we will extend the approach. Our goal is the development of cloud applications that self-tune themselves on the basis of the end-user requests, autonomously launching benchmarks and simulations, and making optimized decisions on the acquisition or the release of resources. In addition, we aim at integrating such functionality into the Service Level Agreement (SLA) framework offered in mOSAIC, in such a way that performance simulation could be used to negotiate the SLAs and to predict possible violations.

References


