DOI 10.12694/scpe.v14i1.825
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. 33–45. http://www.scpe.org © 2013 SCPE

THE MOSAIC BENCHMARKING FRAMEWORK: DEVELOPMENT AND EXECUTION OF CUSTOM CLOUD BENCHMARKS

GIUSEPPE AVERSANO[*], MASSIMILIANO RAK[†], AND UMBERTO VILLANO[‡]

Abstract. A natural consequence of the pay-per-use business model of Cloud Computing is that cloud users need to evaluate and to compare different cloud providers in order to choose the best offerings in terms of trade-off between performance and cost. But at the state of the art, in cloud environments no real grants are offered by providers about the quality of the resources offered and no clear ways exists to compare two different offerings. Moreover, the high elasticity of cloud resources (virtual machines can be added or removed in few minutes) makes the evaluation of such systems a hard task. In this paper we propose to build ad-hoc benchmark applications, whose behavior is strictly related to user needs and which can be used to compare different providers. The proposal is based on the use of the mOSAIC framework, which offers a deployable platform and an API for building provider-independent applications. Due to such independence, we are able to compare directly multiple cloud offers. The paper details the proposed approach and the framework architecture implemented in order to apply it. Simple case studies illustrate how the framework works in practice. Moreover the paper presents a detailed analysis of the state of the art and of the problem of benchmarking in cloud environment.

Key words: Cloud computing, Benchmark, Cloud Performance, mOSAIC, Platform as a Service

1. Introduction.

The emerging cloud computing paradigm owes most of its success to the pay-per-use business model. One of the key idea of the cloud is that all resources, even the ones that usually are physically managed, are offered *as a service* exploiting virtualization techniques. The charges for their use depends only on their actual real usage of the resources acquired. This approach reduces maintenance costs and helps to optimize resource usage, as they are dynamically acquired only when really needed.

As a natural consequence, cloud users need to evaluate and compare different cloud providers in order to choose the best offerings in terms of the trade-off between performance and price. But, at the state of art, techniques and tools to perform this comparison are still lacking in cloud environments. As a matter of fact, providers offer no clear grants about the quality of the resources offered, and no clear ways exist to compare two different offerings. The only support offered by providers is the adoption of Service Level Agreements (SLAs). These are contracts between service consumer and provider that assure the level of quality granted, in the form of natural language contracts with very few quantitative references.

In this context, the most common approach to evaluate the quality of cloud services is the use of monitoring tools. They constantly evaluate the state of resources acquired, as the bandwidth available to users, or the actual throughput of services. Sometimes such services are offered by providers, as Amazon CloudWatch, while in other cases third-party services have to be used. However, it should be noted that monitoring can be exploited only *after* that the resources have been actually acquired. Benchmarking techniques, instead, aim at predicting and describing in a synthetic way the behavior of resources to be (possibly) acquired, performing off-line evaluations.

In order to point out clearly the difference among monitoring and benchmarking, in the following we will refer to benchmarking activities when we aim at performing static evaluations of performance indexes, while we will refer to monitoring when measurements are performed on the software *in production*, i.e., when it is subject to a workload due to real user requests. As such, a benchmark is a program that generates a well-known workload on the system under test (SUT) and enables the expert to measure a set of predefined performance indexes. In our work, we exploit benchmarking techniques to compare alternative cloud solutions.

In this paper, an extended version of [24], we tackle the benchmarking problem in the context of provider-independent cloud frameworks, as mOSAIC[20, 22], Contrail[19, 23] or Optimis[15], which enable their users to develop applications independently of the target providers. In such frameworks, developers create their solution using local resources (or even cloud platforms) and choose the provider where the developed services are to be offered in a second step, comparing the offerings on the basis of the software developed. In such a case, the availability of a benchmarking solution that can help the end user to perform a clear comparison among different offerings is of great interest.

[*]Department of Engineering, University of Sannio, Piazza Roma 21, 82100 Benevento, Italy (giuseppe.aversano@unisannio.it).
[†]Department of Information Engineering, Second University of Naples (SUN), Via Roma 29, 81031 Aversa, Italy (massimiliano.rak@unina2.it).
[‡]Department of Engineering, University of Sannio, Piazza Roma 21, 82100 Benevento, Italy (villano@unisannio.it).

We propose a novel approach to benchmarking, based on the idea of building custom benchmark applications, able to measure indexes strictly related to cloud user needs. We adopted the mOSAIC framework, which is a cloudware which helps a cloud developer to build provider-independent applications. Thanks to the proposed benchmarking framework, a mOSAIC user can easily generate new applications from its mOSAIC-based ones. These applications can successively used can be run, thanks to mOSAIC, on many different providers. The result is that the cloud user, with minimal cost and effort, is made able to run benchmarks dynamically, collecting comparable results from multiple providers.

The remainder of this paper is organized as follows. The next section briefly outlines the state of art in cloud benchmarking, while section 3, briefly summarizes the mOSAIC framework concepts. Section 4 illustrates the approach adopted and the possible different benchmarking scenarios for a mOSAIC application. Section 5 summarizes the framework organization and the main mOSAIC components offered. Section 6 illustrates how to build a mOSAIC application which manages the benchmark life cycles while Section 7 describes the benchmark applications adopted for some key components and for a simple full application. Finally, section 8 describes the results obtained and outlines our future work.

2. Related Work.

This section aims at offering a clear and global view of the state of art about benchmarking in cloud environment. Its goal is to outline the new challenges in cloud environment, the currently available solution and the new proposals. The roles of benchmarking and of monitoring of services are sometimes misunderstood. In the following we will refer to benchmarking activities when we aim at performing static performance evaluations of indexes. A benchmark is a program that generates a well-known workload on the system under test (shortly, *sut*) and enables the expert to measure a set of predefined performance indexes. At the state of the art, few solutions exist that tackle the problem of benchmarking in cloud environments, even if a lot of different approaches exists in different contexts. This section will firstly present benchmark results and products in many different contexts (benchmark in HPC systems, comparison between IaaS providers, existing frameworks, etc.), Then, a brief critical analysis of the solution presented is performed, outlining the open research problems.

2.1. HPC Benchmarking.

Even if recently cloud providers have started to offer HPC-oriented services (like the Amazon HPC service eAmazon, HPC Services, available at http://aws.amazon.com/hpc-applications/), HPC users started to be interested to cloud paradigm from its birth (as shown by the incredible amount of discussions comparing GRID and Cloud). In HPC systems, performance benchmarking is a well known practice. Many stable benchmarking suites of codes exist, among which NAS Parallel Benchmarks (NPB) [4], SkaMPI[5], LinPack [3] (just to mention the most famous ones).

Such benchmarks were adopted to compare HPC systems, like the ones in the Top500 list [9]. An interesting survey of HPC-related problems can be found in [11].

The first results in the context of cloud benchmarking can be found in the tentative of adapting such benchmarks to cloud based clusters, in order to understand the real applicability of solutions based on virtual machines in this context. Probably the most relevant paper in this field is the one by Walker [27] that tries to outline the real performance of cloud based clusters. The paper results are now outdated, but the approach followed is interesting.

Papers as [14, 28] propose the systematic benchmarking of clusters of virtual machines, obtained from IaaS cloud providers, and offer some interesting consideration about the performance perceived by users.

2.2. Benchmark Standard Solutions.

It is a matter of fact that, at the state of the art, there exists no standard for cloud benchmarking. Here, we focus on two of the most important benchmark consortiums that are addressing cloud-related benchmarking: SPEC[8] and TPC[10]. SPEC has developed through the years some solutions that can be adopted in the context of cloud environments. From March 2011 a sub-committee dedicated to cloud has been formed, and works on cloud-related benchmarking problems. Currently the offerings of SPEC that can be considered of interest in the cloud-oriented contexts are SPECvirt[7] and SPA SOA [6]. Moreover, the SPECweb benchmark, which focuses on web servers/application server benchmarking, is also of high interest. The TPC [10] is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. In the context of clouds, which are born to offer transactional services, TPC benchmarks are of great interest. TPC can be considered as application-level benchmarks in cloud environments and they are a basis for the evaluation of the actual performance offered by standard transactional software on the top of IaaS-delivered machines.

2.3. Benchmarking Services. The adoption of standard benchmarks, as described above, is interesting to compare the offerings from different cloud providers, mainly at IaaS level. The main drawback of the standard benchmarks is that they are not designed to cope with the elasticity of cloud offerings. They just provide an evaluation of static systems, and are not able to manage systems that dynamically may change over time, with highly variable performance. A possible solution is to benchmark the systems after any elastic mutation, for example by adopting the ubiquitous as-a-service approach. This is the solution adopted in [18, 25], where the use of a benchmarking service is proposed, that starts up the benchmark execution on any newly-delivered machine. CloudHarmony [1] is probably the most interesting solution of this kind. It offers a very large collection of customizable benchmarks that is continuously executed on remote resources. The solution is very interesting and looks similar to the ones adopted for monitoring. In fact, in this case, it is hard to define clearly the difference between monitoring and benchmarking. While CloudHarmony is built on the top of standard benchmarks, CloudSleuth [2] can build up a cloud application benchmark, which generates a standard workload on target services in order to perform comparisons.

2.4. Cloud-Oriented Benchmarks. Currently only few cloud-specific proposals for benchmarking exist in the literature. The contribution can be summarized by mentioning the following Cloudstone[26] and CloudCmp[17] projects. These are academic projects, which aim at offering flexible frameworks able to build up custom workloads to be run in cloud environments. Cloudstone is an academic open source project from the UC at Berkeley. It provides a framework for testing realistic performance. The project does not publish as of yet comparative results across clouds, but provides users with the framework that allows them to get their measurements. The reference application is a social Web 2.0-style application. The Cloudstone stack can be divided into three subcategories: web application, database, and load generator. When running benchmarks, the load generator generates load against the web application, which in turn makes use of the database. The Web 2.0 application used in Cloudstone is Olio, an open source social-event application with both PHP and Ruby implementations. Cloudstone uses the Ruby implementation. Nginx and HAProxy are used to load balance between the many Rail servers (i.e., thin servers). Cloudstone supports the use of either MySQL or PostgreSQL. Replication when using MySQL is supported using the built-in master-slave replication features. Replication when using PostgreSQL is supported using PGPool-II, a type of load balancer for PostgreSQL database servers. Cloudcmp instead focuses on typical Cloud-based IaaS services (storage and virtual machines) for which a set of custom kernel benchmark were proposed that aims at comparing key features of the systems. The academic project, also supported by some private companies, was applied to compare few common cloud providers. The main limit of this kind of benchmarks is that they do not take into account the variability of results of the same provider in different conditions. Moreover, the results, even if offer a general idea of provider cloud behavior, can hardly be used by a service provider to make a real choice, taking into account how its application behaves.

2.5. A Critical Analysis. The above presented solutions clearly state the main problem of benchmarking in cloud environments. These are: how to represent with a single (or a few) indexes the dynamicity of a cloud environment? how useful is to perform few static measurement? As a matter of fact, at the state of the art, the main direction for performance evaluation of cloud offerings is oriented to the use of monitoring. Benchmarking solutions, like the ones proposed commercially by CloudHarmony or CloudSleuth, try to adapt benchmarking solutions to typical monitoring approaches. Moreover, currently it is impossible to identify a single workload representing the incredibly large variety of cloud services offered. The direction adopted is to reuse benchmarks targeted at specific services, and to try to deliver them in a more flexible way, as in [16], which uses TPC-W to evaluate the cloud services delivered. It should be noted that benchmark results are not reliable in cloud environments. The same request of resource may lead to completely different behavior, as shown in [12, 13]. This means that benchmarking should be applied on the single specific resource, more than on generic resources offered by a given cloud provider. This approach, on the other side, has a high impact on the costs. It is questionable whether it is acceptable to pay for a resource usage which is just targeted to execute a benchmark?

In conclusion, benchmarking tools should be more flexible than the those commonly adopted in other environments. The adoption of a single performance index is not acceptable and workload definition should be customizable by the benchmarker user, according to its specific needs.

3. The mOSAIC Framework. mOSAIC aims at providing a simple way to develop cloud applications [20, 21, 22]. Hence, the target user for the mOSAIC environment 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 from 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 users other than the mOSAIC developer (i.e., by *final users*). In this case, the mOSAIC user acts as service provider for final users.

The mOSAIC framework is composed of a few stand-alone components. Among them, the most important roles are played by the *Platform* and the *Cloud Agency*. The first one (mOSAIC Platform) enables the execution of applications developed using the mOSAIC API. The second one (Cloud Agency) acts as a provisioning system, brokering resources from a cloud provider, or even from a federation of cloud providers.

mOSAIC can be useful in three different scenarios:

- when a developer wishes to develop an application not tied to a particular cloud provider;
- when an infrastructure provider aims at offering "enhanced" cloud services in the form of SaaS;
- when a final user (e.g., a scientist) wishes to execute his own application in the cloud because he needs processing power.

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* developed using the mOSAIC API and running in a *Cloudlet Container*. mOSAIC cloudlets are stateless, and developed following an event-driven asynchronous approach [20, 21].

The mOSAIC platform offers ready-to-use components such as queuing systems (*Rabbitmq* and *zeroMQ*), which are used for component communications, or an HTTP gateway, which accepts HTTP requests and forwards them to application queues, NO-SQL storage systems (as KV store and columnar databases). 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 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.

A cloud application is described as a whole in a file named *Application Descriptor*, which lists all the components and the cloud resources needed to enable their communication. A mOSAIC developer has both the role of developing new components and of writing application descriptors that connect them together.

4. Benchmarking Problem Analysis. The problem we aim at solving is to offer facilities for off-line evaluation of mOSAIC applications and their behavior when running on resources from many different cloud providers. This analysis will be of help to the developer to design better his solutions, and to compare different design choices.

The benchmarking problem can be faced as a typical performance evaluation problem. Benchmarking tools will be a clear definition of the execution conditions for the target application to be evaluated. In any performance evaluation problem the key to success is the identification of the goals of the performance analysis. This section aims at stating the different goals that the benchmarking procedure may assume. In order to identify such goals it is important to make some key assumptions: the target of our benchmarks are always mOSAIC entities (modules, components, resources), and the benchmarking users are mOSAIC actors.

We have identified a few main benchmarking scenarios that represent possible conditions under which a benchmarking procedure should take place. We describe each of these scenarios through three elements: Target User, Main Goal, Target Resource.

The Target User is the user interested in performing the benchmarking. He could be the mOSAIC User (Developer) or a Final User (i.e., the one that uses applications developed with mOSAIC). The Main Goal is the expected result of the benchmarking procedure (e.g., comparing different providers).

The Target Resource is the subject of the Benchmarking procedure, i.e., what the Benchmark aims at stressing (the system under test).

The main scenarios we focus on are summarized in Table 4.1. Each scenario is enriched with a brief description of the goal to be achieved by the performance analysis.

4.1. The benchmarking framework approach. The analysis presented outlined the main requirements of the benchmarking framework for the mOSAIC platform. The first aspect to be taken into account is that the component-based approach adopted in mOSAIC has two side effects on benchmarking. The positive one is that it is possible to evaluate independently the single pieces of a solution. This helps in defining a benchmarking framework composed of independent benchmarks that can be composed in order to fulfill the different requirements of each scenario. On the bad side, the easiness in changing the configurations of the mOSAIC

Table 4.1
Summary of Benchmarking Scenarios

Target User	**Goal**	**Target Resource**	**Goal Description**
Developer	Compare Providers	Cloud resources and/or services	The goal is to enable a developer to choose among services offered by different providers
Developer	Evaluate mOSAIC	mOSAIC Components	The goal is to enable a developer to choose among different components and evaluate how each component behaves
Developer	Compare Applications configurations	Mosaic applications	The goal is enable the mOSAIC developer to compare two different mOSAIC applications in order to configure them at best
Developer	Offer Benchmark as a service	Cloud providers	The goal is to help developers which aim at developing benchmark as a service applications
Developer	Predictions	mOSAIC applications, components and resources	The goal is to help developer to build up benchmarks whose goal is to predict the behavior of a target application
Final User	Compare Applications	mOSAIC applications	The goal is to enable the final user to know the performance offered by the target applications for comparison with other offerings (SLA-related problem)

platform at run-time may imply a huge number of different tests to be performed and difficulties in interpreting evaluation results. Moreover, the analysis outlines the need to customize workloads and to organize them, in preference to the definition of complex workloads. Workloads can be identified for some of the key components of the framework, while mOSAIC users should be able to easily enrich the set of workloads according to their needs. The component approach enables the reuse of the workloads defined.

In conclusion, the main requirements of the benchmarking framework are the capability of offering tools that help mOSAIC user to build up custom solutions and to execute targeted benchmarks for predefined goals. In addition to this, the benchmarking framework should help the developer to identify the systems (components, resources, etc) to be benchmarked, to isolate them and to start up a benchmarking procedure on them.

4.2. Benchmarking Methodology. The Benchmarking framework founds on the idea of using mOSAIC modularity and its component-based approach to implement all the above presented concepts. The proposed scenarios put in evidence (except the scenario 4) that in any case when applying a benchmark to a system, we go through mOSAIC components and interfaces. So it is possible to build up the full framework as a collection of mOSAIC components connected with each other.

In order to implement a benchmark in mOSAIC the approach adopted is the following. Once the System under Test (SUT), i.e., the target of the benchmarking, has been identified, a Benchmarking Model is defined around it. This represents the actual benchmark and it will be implemented as a mOSAIC application.

Once the Benchmark Model application (BM application from now on) has been designed and all its components are available, a mapping of it on the top of the resources will be defined. This represents the definition of the testing condition.

In order to control (start, stop, monitor) the Benchmarking procedure, the BM application is enriched with dedicated components that, among other features, offer a simple interface to final users. These components can be seen as an additional and complementary application, named *The Benchmarker*. This application resides on resources different from the ones adopted for the BM application, and it has the role of controlling the benchmark evolution. To summarize, the Benchmarker will be build by the mOSAIC User (the Developer) through a set of dedicated components, that solve intermediate problems. Examples of such components are:

- Monitor of the Benchmarking procedure
- Manager of Benchmarking results
- Manager of Benchmark tests
- User interfaces

4.3. Pros and Cons of the approach. The above proposed approach enables a clear distinction among Workloads, Benchmark Target and Benchmark conditions. Moreover, it helps in reducing the overhead due to data collection (by defining the role of the Benchmarker and its mapping on resources). When evaluating mOSAIC components, mOSAIC applications and cloud resources, this solution clearly identifies the effect of each module and leads to a clear understanding of the offered performance.

The main drawback of the approach adopted is that it may imply additional costs in the benchmarking procedure. Resources must be acquired in order to be evaluated. Moreover, the Benchmarker consumes additional resources.

5. The Benchmarking Framework. The approach described in the above section leads to a global vision of the benchmarking framework shown in Figure 5.1. The picture puts in evidence that Benchmarking Components are organized in few modules:

- **Benchmark Models**: this module collects the components needed for building up a specific benchmark model. The benchmarking framework offers mainly the components needed to build up custom benchmark workloads. It is up to the developer to use them, along with all the other mOSAIC API to build up a dedicated Benchmark model.
- **Benchmarker**: this module collects all the components dedicated to control the benchmarking evolution. Mainly it offers a set of controllers, which start, stop and manage the execution of Benchmarking Models, Monitoring and Analyzers components, which collect data results and perform the analysis on results collected to produce the required performance indexes.
- **Benchmark GUI**: this module collects all the components needed to offer a simple interface to the Benchmarker user (developer or possibly final user). This module reuses existing components from mOSAIC (as the HTTP gateway) and offers some examples of HTTP backend, i.e., the components adopted to send messages to the controller.

The Benchmarking Models represent the description of the workload and of the system under test. It is up to the framework user (typically a developer) to build up such models on the basis of the goals of his analysis.

Benchmark Models are composed of two key elements: The System under test and the workload generators. The System Under test can be any of the target resources identified in the previous section (cloud resources, mOSAIC components, mOSAIC application, mOSAIC modules). The Workloads represents the real benchmark to be applied. This module offers facilities which help in the development of such generators. These components can be customized by the mOSAIC User in order to build up custom solutions.

While the Benchmarker GUI is a simple customization of common mOSAIC components dedicated to web-based interfaces, the *Benchmarker* is the key of the benchmarking procedure. If the Benchmark Model represents *what* to benchmark, the Benchmarker is the implementation of *how* benchmarking takes place. The core of the Benchmarker is the *Controller*, which controls the benchmark executions on the basis of messages received on a dedicated queue.

6. Benchmark Control Components. As outlined before, the benchmark applications are composed of two main block: *Benchmarker* and *Benchmark Models*. In this section we describe the main components devoted to control the benchmark execution (*Benchmarker*).

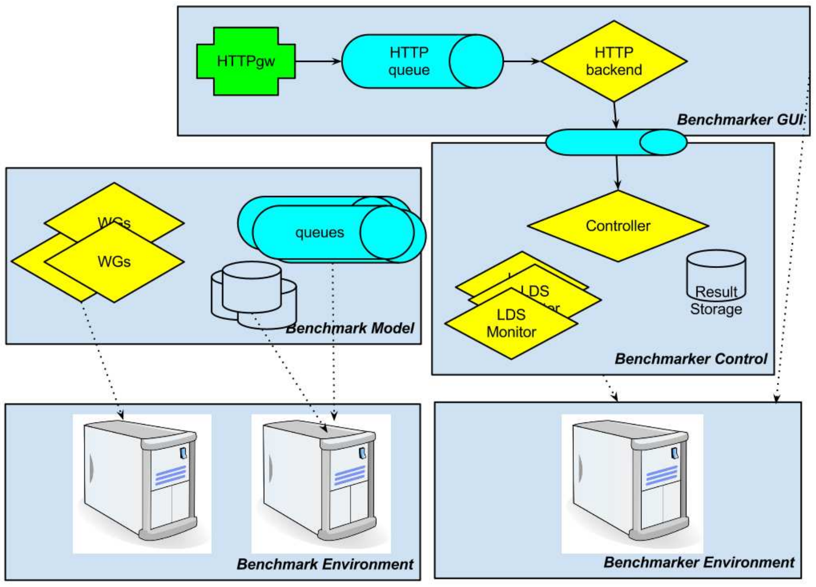

FIGURE 5.1. *A view of the benchmarking framework*

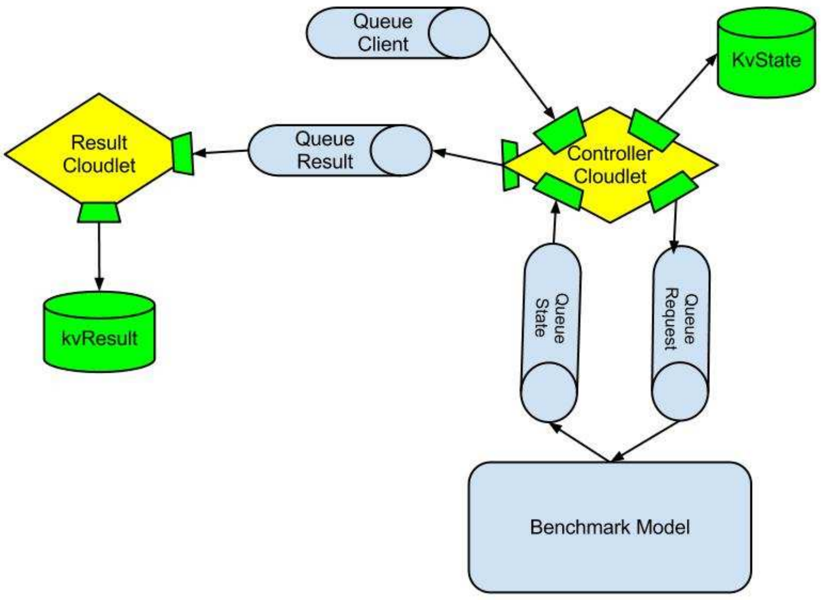

FIGURE 6.1. *The Benchmark Controller Components*

The mOSAIC benchmarking framework offers mainly two key components: the Controller Cloudlet and the Result Cloudlet. A typical mOSAIC Benchmark Control application is structured as shown in figure 6.1. The Result Cloudlet has simply the role of collecting the benchmark results and tostore them in a key-value (KV) Store; the Controller is devoted to maintain the full benchmark life cycle. The Controller Cloudlet maintains state information in a KV store resource and evolves during the benchmark execution according to the user requests received from the *Queue Client*. The Controller is embedded with a Workload generator that sends out messages that activate the Benchmark model and stresses it through the request and State queues. Moreover, it forwards the results to the Result cloudlet when they are available.

The mOSAIC Benchmarking framework offers a set of simple Workload Generators that can be extended by

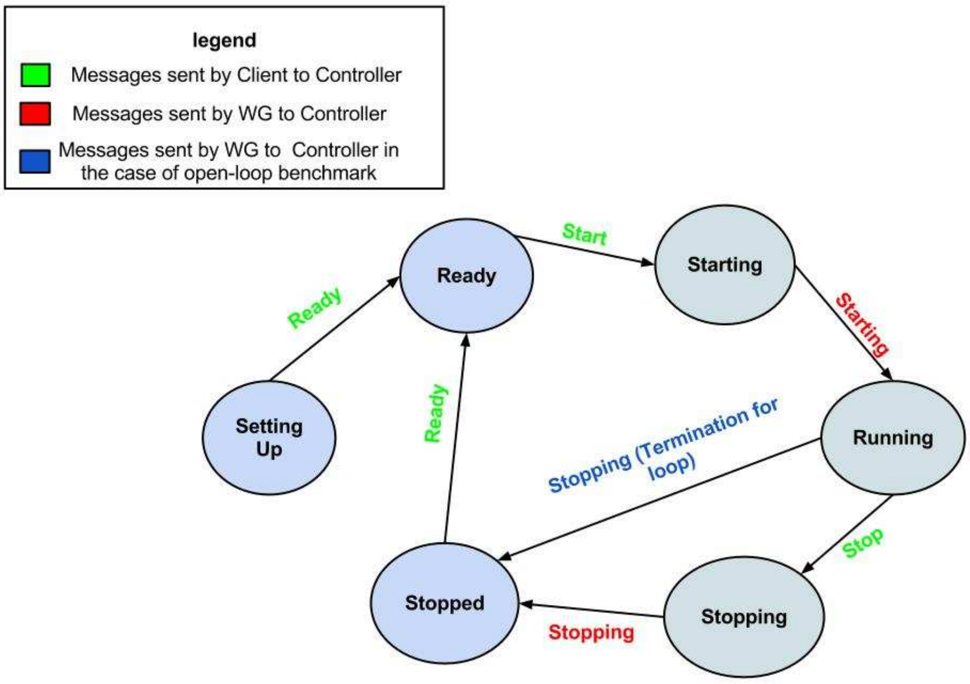
legend
🟩 Messages sent by Client to Controller
🟥 Messages sent by WG to Controller
🟦 Messages sent by WG to Controller in the case of open-loop benchmark
Setting Up
Ready
Ready
Start
Starting
Starting
Running
Stopping (Termination for loop)
Ready
Stop
Stopped
Stopping
Stopping

Figure 6.2. *The Benchmark Control States*

mOSAIC Developers in order to build up custom workloads. At the state of the art, mOSAIC offers a generator that sends out one or more bursts of concurrent messages, a sequence of different messages, or generates a fixed set of messages with known delays. In the future, we aim at offering generators that stress the application with other type of workloads.

It should be noted that all the controller components can be deployed on dedicated machines in order to avoid the sharing of resources among components dedicated to benchmark control and the benchmark model (i.e., what is to be benchmarked). All the benchmark applications can be used using the same pattern. They behave according to the state diagram presented in figure 6.2. The events that enable the state transitions are linked to messages that End users can send to the benchmark application, or to messages generated internally by the application.

The Benchmarking states have the following meaning:

- **SettingUp**: The benchmark application is starting (not all the components are available, the End user should avoid to interact with application).
- **Ready**: The benchmark application is ready to receive requests from the End user. All components correctly started.
- **Starting**: A Start request was sent by the End user to Controller (through the web interface).
- **Running**: The benchmarking application is running some benchmark tests. End users should not interact with the application (in any case, any End user message is intercepted by the controller that does not deliver it to the application).
- **Stopping**: The benchmarking application has received a Stop message and it is waiting that the tests are completed, without sending more messages. The Stop message is not needed in the case of workloads with fixed amount of messages.
- **Stopped**: The benchmarking application has terminated the tests. Note that in this case it is not ready to start a new set of tests (i.e., to accept a Start message). New tests can be started only in the Ready state (reachable through a Ready message).

The following messages are enabled:

- **Ready**: This message indicates that the client aims at starting a new set of benchmarks. The message moves the benchmarking app to Ready from SettingUp or Stopped states to Ready (it is discarded in other cases).

- **Start**: This message starts the benchmark procedure. It is accepted only when the application is in Ready state (otherwise it is discarded).
- **Stop**: This message ends the benchmark procedure, i.e., it advise the controller to end sending out messages. It is accepted only in Running state. It is not needed when the workload assumes a fixed (or limited) number of messages.
- **Plot**: This message makes available the full set of results produced by the benchmark procedure. The message is accepted only in Stopped state (otherwise it is discarded). Note that it does not materially plot the results, unless the Web UI is opportunely configured.
- **Check**: This message makes available the state of the system.

The Controller discards all the messages received from the user that are not explicitly managed in the above described state diagram. This implies that if an user sends many Start messages together, only one is considered and all the other are discarded. Moreover, once a benchmark has been completed, the user needs to ask explicitly (through the *Ready* message) to make it available for a new set of tests.

Such Controlling applications are usually interfaced by means of the mOSAIC HTTP gateway, which offers a Web-based interface and accepts the above described messages through a simple REST interface, using the benchmark messages as content of the HTTP message:
`(URL: http://<ip> :1337, METHOD: POST).`

It is possible to read the state of the benchmarking process through a dedicated REST call, which depends on the benchmark Model to use:
`URL: http://<ip>:1337/raw/<bucket>/<key>, METHOD: GET`
Moreover, another REST call lets the user retrieve the benchmark results:
`URL: http://<ip>:1337/raw/<bucket>/<key>, METHOD: GET`
A Set of simple HTTP clients and scripts are offered in order to automate the benchmarking process.

7. Benchmark Examples in mOSAIC. In this section we focus on how to use the mOSAIC benchmarking framework to make an evaluation of a mOSAIC cloud application. We consider the first three scenarios described in section 4, i.e., the comparison of cloud providers (the stress of Cloud Resources described in the next two subsections), the comparison of mOSAIC components (the stress of cloudlets described in the third subsection) and the evaluation of a full mOSAIC Applications (last subsection). We executed the tests described hereafter on a private cluster based on OpenNebula, enabled in order to run the mOSAIC platform. In the mOSAIC repositories (`https://bitbucket.org/mosaic`) it is possible to retrieve the code of the benchmarking framework and bundle versions of mOSAIC that enable its execution on every kind of machine and cloud provider (it was tested on OpeNebula, Openstack, Cloudsigms, Deltacloud, Flexiscale, Amazon and others).

7.1. Benchmarking Cloud Resources: Queue. The main goal of applications that stress a single queue is to evaluate the response time needed for each single message. The main goal of the benchmarking process is to offer a reference to developers in order to predict the time needed in their application to deliver a message from a component to another. The performance of a Queue depends heavily on a lot of parameters that are hardly controllable in a cloud environment (for example, using mOSAIC RabbitMQ COTS component it depends on the VMs on which they run, and on the number of other components sharing CPU and memory). The goal of this benchmark is to create a simple repeatable evaluation that can be used to tune the application. Tests can be repeated (consuming resources) online, in order to perform a new evaluation in different conditions. We offer three different Benchmark applications: QueueBenchmark, QueueBenchmarkBlock, QueueBenchmarkStep, which generate three different loads on the queue. The first one produces a set of concurrent requests. The second one repeats the first one a known number of times, the third generates a sequence of concatenated messages. Repeating these tests with a variable number of messages and repetitions can help the developer to understand how the queue performs in different load conditions. The results, reported in table 7.1, show that Response time depends on the Number of concurrent messages (the measurement with a low number of concurrent messages, instead, is unreliable).

7.2. Benchmarking Cloud Resources: KV Store. The main goal of applications that stress a Key-value Store is to evaluate how many sequences of SET/GET operations can be completed in a fixed interval of time. The main goal of the benchmarking process is to offer a reference to developers in order to predict the time needed in their application to access the Key-value Store. The performance of a KV store heavily depends on a lot of parameters that are hardly controllable in a cloud environment (as an example, using the mOSAIC Riak COTS component, it depends on the VMs on which they run, and on the number of other components

Table 7.1
Mean Burst Response time (ms) varying the number of concurrent messages (nM) and message dimension (Dim)

Message Dim vs number of Messages	$nM = 1$	$nM = 10$	$nM = 100$
Dim=4Kb	25ms	164ms	2194ms
Dim=8Kb	32ms	331ms	5018ms
Dim=12Kb	25ms	542ms	14044ms

Table 7.2
Mean Burst Response time (ms) varying the number of concurrent sequences (nM) and file dimension (Dim)

Message Dim vs number of Messages	$nM = 1$	$nM = 10$	$nM = 100$
Dim=4Kb	92ms	529ms	6785ms
Dim=8Kb	100ms	1285ms	12345ms
Dim=12Kb	96ms	2006ms	19199ms

sharing CPU and memory). The goal of this benchmark is to create a simple repeatable evaluation that can be used to tune the application. Tests can be repeated (consuming resources) on-line, in order to perform a new evaluation in different conditions. We offer three different Benchmark applications: BenchmarkKvStore, BenchmarkKvStoreVariant, BenchmarkErrorKvStore that generate different loads on the KV store: the first one issues a (parameterized) sequence of SET/GET operations which are submitted concurrently to the KV store, while the second one performs GET operations only when all SET operations are completed. Both applications measure the overall response time and the mean number of SET/GET operations executed per second. The last one generates the same load of the first application, but checks the coherence of results (i.e., evaluates the effect of *eventual consistency* typical of KV stores).

Repeating these tests with a variable number of messages and repetitions helps the developer to understand how the KV performs in different conditions. Table 7.2 summarizes an example of results for the first benchmark on our testbed.

7.3. Benchmarking a Simple Cloudlet. Cloudlet benchmarking heavily depends on the target cloudlet to be evaluated. In any case it is possible to generalize the approach, considering that the majority of the cloudlets have similar behavior (for example: receive a message, make evaluations and send results, or receive a message, access a KV store. make evaluations and send results). We can classify this behaviors and produce target benchmarks of general use for each different pattern. At the state of the art, we focused on "Filter" cloudlets, which operate as filters, receiving messages, evaluating the content and generating a new message on the basis of evaluations. An alternative pattern is the one the accessing KV stores. In this case, the application proposed to benchmark KV store can be used as a template to generate the new custom benchmark application. Benchmarking Filter cloudlets needs the same control application adopted for Queues and it is possible to reuse the same client that automates the benchmark. Table 7.3 contains the results of such a benchmark on a typical cloudlet we use as test. This cloudlet analyzes XML documents and reports the result in an output message.

7.4. Full Application benchmarking. In this section we aims at showing how to benchmark a complete mOSAIC application, evaluating its behavior under well known stressing workloads. Differently from benchmarking mOSAIC resources (i.e. KV stores, Queues) or single cloudlets, a full application is composed of a lot of different components that interact in different ways and react in different ways to stressing conditions. Full Application benchmarking aims at identifying the application bottlenecks and critical points, in order to fine tune them on cloud environments.

Full Application benchmarking implies two different aspects:

- Evaluate the application as a whole under stressing loads which are known to the developer
- Evaluate the application components in order to understand how each of them behaves and reacts to different working conditions.

The Full Application can be evaluated using a benchmark application whose controlling part is the same of the Filter cloudlet, substituting the user interface (usually HTTPgw or mHTTPgw) with the controlling

Table 7.3
Mean Burst Response time (ms) varying the number of concurrent requests (nM) and file dimension (Dim)

Message Dim vs number of requests	$nM = 1$	$nM = 10$	$nM = 100$
Dim=4Kb	49ms	239ms	4659ms
Dim=8Kb	67ms	679ms	83595ms
Dim=12Kb	74ms	910ms	16435ms

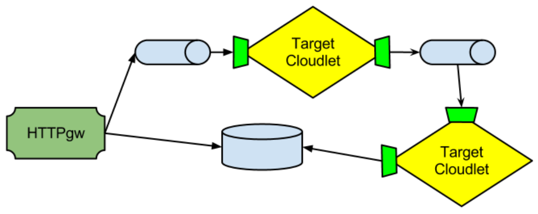

Figure 7.1. *A simple application to be benchmarked*

benchmark application. In this case the benchmark submits the stressing load to the application which generate its own results.

In order to evaluate the single components, thanks to the mOSAIC modularity, it is possible to extract each cloudlet of the application and test it independently (see benchmarking cloudlets), evaluating it under specific stressing condition.

In order to illustrate how the approach works, instead of detailing all the pieces of the framework, we will use a simple example, showing how to build a benchmark application starting from a mOSAIC application.

Figure 7.1 sketches the structure of the target mOSAIC application. For simplicity's sake, it is a toy application which receives messages from the mOSAIC web interface (through a queue). It performs an elaboration on messaages content, and forwards the result to a second application component which stores them in a Key-Value store.

Even if this application is very simple, its pattern can be adopted on many different real applications. In this paper our purpose is just to illustrate how to derive a benchmark application.

The first step of benchmarking procedure is to identify *what* to evaluate, the global application or single components of it. In this example, the core of the application is the *Target Cloudlet*, which performs the most complex work and may be a bottleneck of the application. The goal of the benchmark application will be to evaluate the conditions under which the component should be replicated, in order to grant application responsiveness.

As a result we can derive a new mOSAIC application, whose main architecture is described in figure 7.2. The new application is enriched with the GUI, the controller and a set of other components which produce a fixed, well known workload.

The derived application can be run, independently of the starting application, on any cloud provider through the mOSAIC Framework, collecting the results of execution in different working conditions.

As an example table 7.4 summarizes the results of the execution of the target cloudlet under the same requests, varying the number of machines acquired and the number of cloudlet instances started. This analysis will help a developer to evaluate the effects of application scalability.

One of the most interesting aspects is that the benchmarking application was developed, *without any additional code*. In fact, we just reused the set of components offered in mOSAIC, composing them in order to stress the target cloudlet.

8. Conclusions and Future Work. Cloud benchmarking is an hot topic, in which few effective solutions are available at the state of the art. In this paper we proposed a benchmarking framework, focused on the mOSAIC platform, which helps cloud application developers to build up custom benchmarking application, which can be used to compare different cloud provider offerings. We have illustrated the main concepts adopted in the framework and illustrated their adoption in different scenarios. We have shown that it is possible to

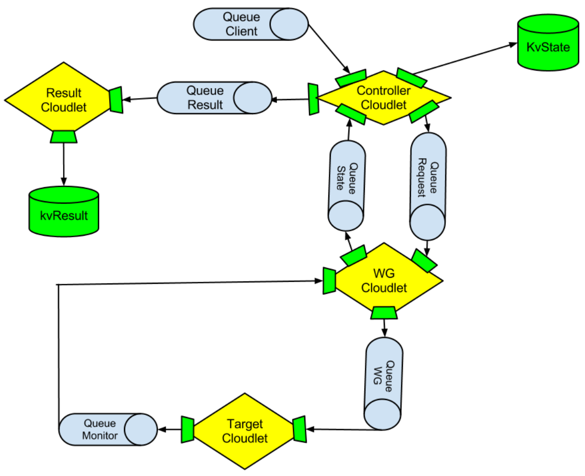

FIGURE 7.2. *The derived benchmarking application*

TABLE 7.4
Mean Cloudlet Response time (ms) varying the number of Virtual Machines (VM) and of Cloudlet instances (nC)

Cloudlet vs Virtual Machines	$nM = 1$	$nM = 2$	$nM = 3$
nC=1	122432ms	115803	119315
nC=2	72604	70890	76916
nC=3	84336	55423	54044

make comparisons among different cloud providers with a set of ad-hoc benchmarks that stresses the mOSAIC components of an application and helps in making a clear evaluation of the quality of the services offered through mOSAIC-based applications. The solution we proposed enabled both a fine grained analysis of single components of a mOSAIC application and an application-oriented comparison of Cloud provider offerings. Even if the focus of the paper and of the framework is on mOSAIC based application the approach (and the framework) can be easily adapted even for non-mOSAIC Applications. The Approach proposed (i.e. subdivision among benchmark control and benchmark model) can be applied as a general model for building custom benchmarks. Moreover the *Benchmarker*, i.e. the application which controls the benchmark life cycle and generate loads against the benchmark model, can be used easily even with non-mOSAIC application interfacing them through queue or through a dedicated component. In Future works we aims at applying such solution in order to control general puropose benchmarks and to compare providers even for evaluating non-mOSAIC uses.

REFERENCES

[1] *Cloudharmony available at.* `http://cloudharmony.com/`.
[2] *Cloudsleuth available at.* `https://cloudsleuth.net/`.
[3] *Linpack available at.* `http://www.netlib.org/linpack/`.
[4] *Nas parallel benchmark available at.* `http://www.nas.nasa.gov/publications/npb.html`.
[5] *Skampi available at.* `http://liinwww.ira.uka.de/~skampi/`.
[6] *Spec soa subcommittee.* `http://www.spec.org/soa/`.
[7] *Spec virtualization benchmark 2010.* `http://www.spec.org/virt_sc2010/`.
[8] *Standard performance evaluation corporation.* `http://www.spec.org/`.
[9] *Top 500 supercomputing sites available at.* `http://www.top500.org/`.
[10] *Tpc, transaction processing performance council available at.* `http://www.tpc.org/`.
[11] R. Aversa, B. Di Martino, M. Rak, S. Venticinque, and U. Villano, *Performance Prediction for HPC on Clouds*, John Wiley &amp; Sons, Inc., 2011, pp. 437–456.
[12] J. Dejun, G. Pierre, and C.-H. Chi, *EC2 performance analysis for resource provisioning of service-oriented applications*, in Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, Nov. 2009.
[13] J. Dejun, G. Pierre, and C.-H. Chi, *Resource provisioning of web applications in heterogeneous clouds*, in Proceedings of the

2nd USENIX conference on Web application development, WebApps'11, Berkeley, CA, USA, 2011, USENIX Association, pp. 5–5.

[14] C. Evangelinos and C. N. Hill, *Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2*, Cloud Computing and Its Applications, Oct. 2008.

[15] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, *Optimis: A holistic approach to cloud service provisioning*, 2012-01-01 2012.

[16] D. Kossmann, T. Kraska, and S. Loesing, *An evaluation of alternative architectures for transaction processing in the cloud*, in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, SIGMOD '10, New York, NY, USA, 2010, ACM, pp. 579–590.

[17] A. Li, X. Yang, S. Kandula, and M. Zhang, *Cloudcmp: comparing public cloud providers*, in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC '10, New York, NY, USA, 2010, ACM, pp. 1–14.

[18] E. Mancini, M. Rak, and U. Villano, *Perfcloud: GRID services for performance-oriented development of cloud computing applications*, in 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, Groningen, The Netherlands, 29 June - 1 July 2009, Proceedings, S. Reddy, ed., IEEE Computer Society, 2009, pp. 201–206.

[19] C. Morin, *Open computing infrastructures for elastic services: contrail approach*, in Proceedings of the 5th international workshop on Virtualization technologies in distributed computing, VTDC '11, New York, NY, USA, 2011, ACM, pp. 1–2.

[20] D. Petcu, C. Craciun, M. Neagul, I. Lazcanotegui, and M. Rak, *Building an interoperability api for sky computing*, in 2011 International Conference on High Performance Computing &amp; Simulation, HPCS 2012, Istanbul, Turkey, July 4-8, 2011, W. W. Smari and J. P. McIntire, eds., IEEE, 2011, pp. 405–411.

[21] D. Petcu, C. Craciun, and M. Rak, *Towards a cross platform cloud api - components for cloud federation.*, in CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, Porto, Portugal, 18 - 21 April, 2012, F. Leymann, I. Ivanov, M. van Sinderen, and B. Shishkov, eds., SciTePress, 2011, pp. 166–169.

[22] D. Petcu, C. Crăciun, M. Neagul, S. Panica, B. Di Martino, S. Venticinque, M. Rak, and R. Aversa, *Architecturing a sky computing platform*, in Proceedings of the 2010 international conference on Towards a service-based internet, ServiceWave'10, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 1–13.

[23] G. Pierre, I. El Helw, C. Stratan, A. Oprescu, T. Kielmann, T. Schütt, J. Stender, M. Artač, and A. Černivec, *Conpaas: an integrated runtime environment for elastic cloud applications*, in Proceedings of the Workshop on Posters and Demos Track, PDT '11, New York, NY, USA, 2011, ACM, pp. 5:1–5:2.

[24] M. Rak and G. Aversano, *Benchmarks in the cloud: The mosaic benchmarking framework*, in Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on, 2012, pp. 415–422.

[25] M. Rak, A. Cuomo, and U. Villano, *A service for virtual cluster performance evaluation*, in Proceedings of the 2010 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 249–251.

[26] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, O. Fox, and D. Patterson, *Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0*, 2008.

[27] E. Walker, *Benchmarking Amazon EC2 for high-performance scientific computing*, LOGIN, 33 (2008), pp. 18–23.

[28] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, *C-meter: A framework for performance analysis of computing clouds*, in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 472–477.

Edited by: Marc Eduard Frîncu
Received: Mar 15, 2013
Accepted: Apr 8, 2013