C-Meter: A Framework for Performance Analysis of Computing Clouds

Nezih Yigitbasi, Alexandru Iosup, and Dick Epema
Delft University of Technology
{M.N.Yigitbasi, A.Iosup, D.H.J.Epema}@tudelft.nl

Simon Ostermann
University of Innsbruck
simon@dps.uibk.ac.at

***Abstract*—Cloud computing has emerged as a new technology that provides large amounts of computing and data storage capacity to its users with a promise of increased scalability, high availability, and reduced administration and maintenance costs. As the use of cloud computing environments increases, it becomes crucial to understand the performance of these environments. So, it is of great importance to assess the performance of computing clouds in terms of various metrics, such as the overhead of acquiring and releasing the virtual computing resources, and other virtualization and network communications overheads. To address these issues, we have designed and implemented *C-Meter*, which is a portable, extensible, and easy-to-use framework for generating and submitting test workloads to computing clouds. In this paper, first we state the requirements for frameworks to assess the performance of computing clouds. Then, we present the architecture of the C-Meter framework and discuss several cloud resource management alternatives. Finally, we present our early experiences with C-Meter in Amazon EC2. We show how C-Meter can be used for assessing the overhead of acquiring and releasing the virtual computing resources, for comparing different configurations, and for evaluating different scheduling algorithms.**

I. Introduction

Cloud computing has emerged as a new technology that lets users deploy their applications in an environment with a promise of good scalability, availability, and fault tolerance. As the use of cloud computing environments increases [1], it becomes crucial to understand the performance of these environments in order to facilitate the decision to adopt this new technology, and to understand and resolve any performance problems that may appear. In this paper, we present C-Meter, which is a framework for generating and submitting test workloads to computing clouds. By using C-Meter, users can assess the overhead of acquiring and releasing the virtual computing resources, they can compare different configurations, and they can evaluate different scheduling algorithms.

Many big vendors like Amazon, Google, Dell, IBM, and Microsoft are interested in cloud technology, and they invest billions of dollars in order to provide their own cloud solutions [1]; for an overview of the existing cloud providers, see [2]. The cloud providers are responsible for maintaining the underlying computing and data infrastructure while at the same time reducing the administration and maintenance costs for the users. This is commonly known as *Infrastructure as a Service* (IaaS).

Today's cloud environments make use of virtualization technologies for both the computing and networking resources. Virtualization is an abstraction between a user and a physical resource which provides the illusion that the user interacts directly with the physical resource [3]. These resources, which have access to large and efficient data storage centers, are interconnected together and are provisioned to the consumers on-demand. The cloud computing environment is open to its users via well defined interfaces over well known Internet protocols enabling anytime and anywhere access to the resources, similar to other common utilities like electricity and telephony. The users can deploy their software by creating customized virtual machine images and running these on the resources in the cloud.

To assess the performance of computing clouds, we have designed and implemented C-Meter, which is a portable, extensible and easy-to-use framework for generating and submitting workloads and analyzing the performance of cloud computing environments. C-Meter is designed as an extension to GrenchMark [4], which is a framework for generating and submitting synthetic or real workloads to grid computing environments. Our contribution is threefold:

1) We state the requirements for frameworks that can assess the performance of computing clouds (Section II).
2) We design and implement the C-Meter framework which satisfies these requirements (Section III).
3) We show how C-Meter can be used in practice through several experiments with Amazon EC2 (Section IV).

II. Problem Statement and Background

In this section, we state the requirements that we have identified for frameworks to assess the performance of computing clouds. After that, we give an overview of Amazon EC2, which we have used in our experiments. Finally, we present an overview of GrenchMark, within which we have implemented C-Meter.

A. Requirements for Cloud Performance Analysis Frameworks

To assess the performance of cloud computing environments researchers need to have a framework which should satisfy various requirements. We have identified three main requirements for framework to assess the performance of computing clouds:

1) The framework should be able to generate and submit both real and synthetic workloads. It should gather the results and extract various statistics specific to

978-0-7695-3622-4/09 $25.00 © 2009 IEEE
DOI 10.1109/CCGRID.2009.40

computing clouds such as the detailed overheads of resource acquisition and release. It should also provide performance analysis reports to the users as well as a database of obtained statistics in order for the users to perform offline analysis for their own needs.

2) The framework should let users compare computing clouds with other environments such as clusters and grids. It should also let the users perform experiments with different configurations such as with different types and amounts of resources.
3) Since currently no resource management components and no middleware exist for accessing and managing cloud resources, the framework has to provide basic resource management functionalities. The framework should be extensible in the sense that new resource management algorithms and support for new cloud environments can easily be added. It should also be platform independent and easy to use.

B. Amazon EC2

Amazon EC2 (Elastic Compute Cloud) is a web service that opens Amazon's cloud computing infrastructure to its users [5]. It is elastic in the sense that it enables the applications to adapt themselves to their computational requirements either by launching new virtual machines or by terminating virtual machines which are running. EC2 uses its own image format called AMI (Amazon Machine Image) for virtual machine images allowing the users to create their own virtual computing environments containing their software, libraries and other configuration items. After an AMI is launched on a physical computing resource, the resulting running system is called an *instance*. By using the EC2 web service, users can launch, monitor and terminate instances.

There are many instance types in Amazon EC2 environment which are grouped into two families: the standard and High-CPU [5]. Standard CPUs are suitable for general purpose applications whereas High-CPU instances have more computational resources and so are more suitable for computationally intensive applications. The users pay per hour according to the instance type they have used. Table I shows the types of instances available in Amazon EC2 with their costs and computing capacities in terms of EC2 Compute Unit (ECU).

TABLE I
AMAZON EC2 INSTANCE TYPES.

Instance Type	Category	Capacity (EC2 Compute Unit)	Cost (US $/hour)
m1.small	Standard	1 (1 virtual core with 1 ECU)	0.10
m1.large	Standard	4 (2 virtual core with 2 ECU)	0.40
m1.xlarge	Standard	8 (4 virtual core with 2 ECU)	0.80
c1.medium	High-CPU	5 (2 virtual core with 2.5 ECU)	0.20
c1.xlarge	High-CPU	20 (8 virtual core with 2.5 ECU)	0.80

C. GrenchMark

GrenchMark is a framework for generating and submitting synthetic or real workloads to grid computing environments. Over the past three years, GrenchMark has been used in over 25 testing scenarios in grids (e.g., Globus based), in peer-to-peer systems (e.g., BitTorrent-based), and in heterogeneous computing environments (e.g., Condor-based). By using GrenchMark, users can perform functionality and performance tests, system tuning, what-if analysis, and compare various grid settings. GrenchMark supports unitary and composite applications and allows the users to determine the job interarrival time distribution, letting them generate various workloads for their analysis. GrenchMark can also replay real traces taken from various grid environments by converting them into the standard workload format, hence it can help to perform realistic test scenarios on a large scale. However, GrenchMark can not satisfy the requirements stated in Section II-A since it does not have support for managing cloud resources and hence it can not be used for experiments with computing clouds. So, to satisfy these requirements, we have designed and implemented C-Meter as an extension to GrenchMark.

III. THE C-METER FRAMEWORK

In this section we present the architecture of the C-Meter framework. Then we explain the C-Meter experimentation process and describe the various alternatives for resource management in clouds.

A. The Architecture of C-Meter

C-Meter is a portable, extensible and easy-to-use framework for generating and submitting both real and synthetic workloads to analyze the performance of cloud computing environments. It is designed as an extension to GrenchMark. It is portable in the sense that it is implemented in Python, which is a platform-independent programming language. It is extensible in the sense that it can be extended to interact with many cloud computing environments and it can also be extended with different scheduling algorithms.

The architecture of C-Meter as part of GrenchMark is illustrated in Figure 1. C-Meter consists of three subsystems. The *Core* subsystem is responsible for providing the core functionalities of C-Meter and it consists of three modules. The *Listener* module is responsible for listening for job submissions from the workload generator and commands from the user such as a command to terminate the experiment. After receiving the job descriptions from the workload generator of GrenchMark, these descriptions are queued in the *Job Queue* until some resources become available for submitting these jobs. The *Job Submission* module is responsible for copying the executables and stage in files of the job to an HTTP which is installed at the submission host, and transferring an *execution agent* to a virtual resource in the computing cloud. This agent downloads the executables and stage in files from the HTTP server, executes the job, and reports the statistics back to C-Meter.

The *Cloud Interaction* subsystem is responsible for interacting with the cloud environment under test. This subsystem consists of two modules. The *Resource Management* module is responsible for acquiring, managing and releasing the virtual resources from the cloud. The scheduling algorithms are also provided by this module. The user configures the resource specification in the configuration file and the *Resource*

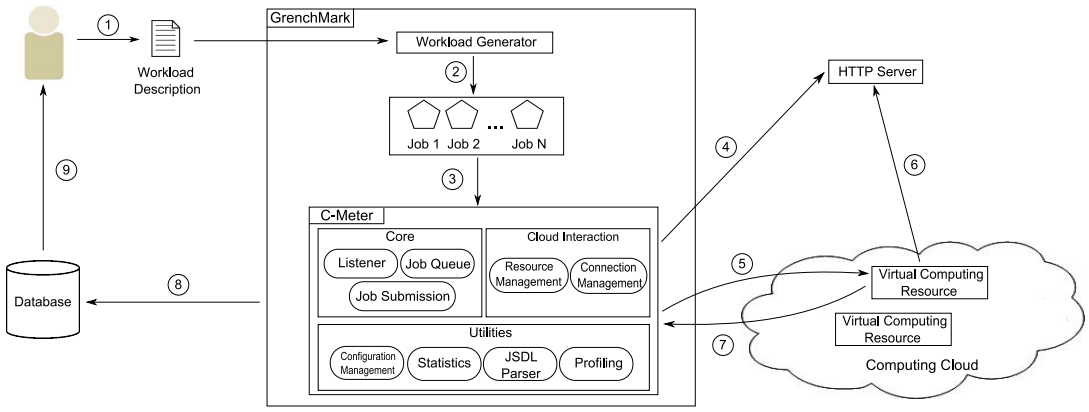

Fig. 1. The architecture of C-Meter and the flow of a typical experiment.

Management module interprets the specification and allocates resources accordingly. The *Connection Management* module is used to establish connections to the cloud environment in order to submit the jobs by the *Job Submission* module. By using functionalities provided by this subsystem, C-Meter provides basic resource management functionalities which means that C-Meter satisfies Requirement 3. Users can submit the same workload to different environments like grids or clusters, by using the extensible architecture of GrenchMark, and as a result, they can compare the performance of various different architectures. By using the resource management capabilities of C-Meter, users can also perform experiments with different numbers of homogeneous and heterogeneous resources, and they can compare different configurations for computing clouds. With these functionalities, C-Meter also satisfies Requirement 2.

The *Utilities* subsystem is responsible for providing basic utilities and consists of four modules. The *Configuration Management* module is responsible for the configuration of the experiments. The user can define parameters such as the number and type of resources that should be allocated from the cloud and the credentials needed for authentication. The other modules are the *Statistics* module, the *JSDL Parser* module and the *Profiling* module. The functionalities of these modules are obvious from their names. Thanks to the workload generation capabilities of GrenchMark and functionalities provided by this subsystem, C-Meter satisfies Requirement 1 of Section II-A.

B. C-Meter Experimentation Process

The flow of a typical experiment when using the C-Meter framework is illustrated in Figure 1 (the numbers below correspond to the numbers in the figure):

1) The C-Meter user prepares a workload description file in the format of the GrenchMark workload description format. In this workload description file it is possible to define the type of the jobs (e.g., sequential or MPI jobs) and the statistical distribution for the job interarrival times.
2) The workload generator of GrenchMark uses this workload description for generating the workload in JSDL (Job Submission Description Language) format [6].
3) The workload is submitted to C-Meter.
4) C-Meter parses the job descriptions and copies the necessary executable and stage-in files to the HTTP Server.
5) After copying the files, C-Meter launches an execution agent on the virtual resource which is responsible for running the executable.
6) The agent downloads the executable and stage-in files from the HTTP Server.
7) The agent runs the executable and reports the gathered statistics to C-Meter.
8) C-Meter stores these statistics in the results database.
9) Finally, when the user concludes the experiments C-Meter generates performance analysis reports for the experiment, and stores them in the results database.

C. Managing Cloud Resources

Since the user has complete control over the cloud resources there is a need to devise resource management schemes for effective management of these virtual resources. We have identified four alternatives for managing these virtual resources, based on whether there is a queue between the submission

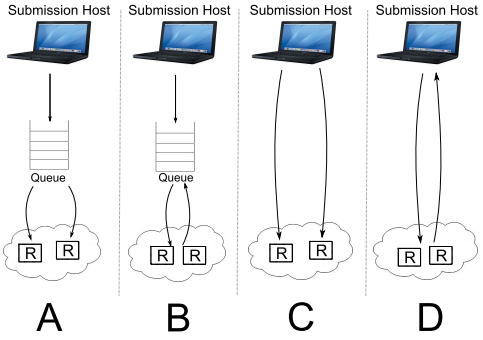

Fig. 2. Resource management alternatives for computing clouds. (R denotes the virtual resources in the cloud.)

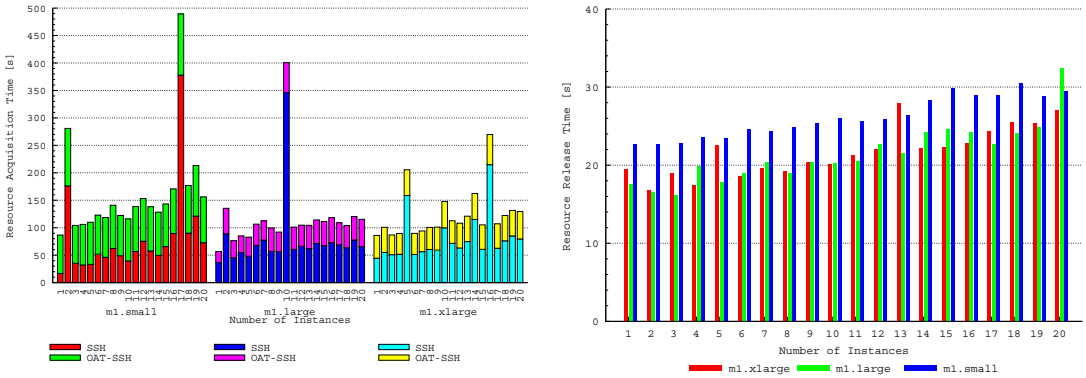

Fig. 3. Resource acquisition time (left) and resource release time (right) with different number of instances of standard instance types.

host and the computing cloud and whether the resources are acquired and released for each submitted job. The queue can actually reside in the submission host or in another host. The alternatives are shown in Figure 2.

In alternative A, a queue resides between the submission host and the cloud, and it is managed by a resource management component responsible for managing the virtual resources. The resource manager acquires all the resources at the beginning of the experiment and only releases them at the end. In alternative B, a queue resides between the submission host and the cloud, but this time the resources are acquired and released for each job submission, causing overhead for each job. In alternative C, there is no queue in the system and the application to be submitted is responsible to acquire the resources at startup and to release them at termination. Finally, in alternative D, there is no queue in the system but the resources are acquired and released for each job submission. Although alternatives B and D seem inefficient, they may still be useful in practice, for example, when there is a possibility of jobs with malicious behavior (e.g., programs with bugs or intentional malicious behavior) which is more likely in production environments. After executing such a job on a resource, the resource can be left in an inconsistent state, but since a new resource is acquired for each job, the next job will not be affected by the previous malicious jobs. Although not depicted in Figure 2, the resources acquired from the cloud can be managed on a time-shared or a space-shared basis. Assessing the effectiveness of resource management schemes for computing clouds is in itself an interesting research topic, but falls outside the scope of this paper.

IV. Experimental Results

In this section we present our early experiences with the C-Meter framework on the Amazon EC2. C-Meter manages the virtual resources in a time-shared basis. Three sets of experiments are performed. In the first set of experiments we have analyzed the resource acquisition and release overhead for three different standard instance types, namely the *m1.small*, *m1.large* and *m1.xlarge* instance types (see Table I). In the second sets of experiments, we have analyzed the performance of different configurations with different numbers and types of resources. In the last set of experiments we have analyzed the performance of two scheduling algorithms, the round robin scheduling algorithm and a predictive scheduling algorithm.

We have performed experiments with at most 20 instances, which is the default stated limit of the EC2 environment without human intervention [5]. During the experiments we have used a workload of one thousand jobs consisting of sequential applications with a Poisson arrival process with mean 500 ms. The runtimes of the jobs range from 0.01 s to 50.6 s on an m1.small instance. As performance metrics we have used the times spent waiting in the queue, the response times, the bounded slowdowns with a threshold of 1 second [7], and the execution times (wall-clock time) of the jobs submitted during the experiment. The total cost of the experiments is around $150 including the retries for some experiments and the learning period of the EC2 platform.

A. Overhead of Resource Acquisition and Release

In this experiment we have simultaneously acquired and released 1 to 20 instances of the standard instance types. After an instance is launched, first it completes its boot process, and then it becomes ready to accept a job submission. The resource acquisition times, defined as the earliest time when all the instances are available, are shown in Figure 3. *SSH* is the time for polling a resource using SSH to see whether it is ready to accept a job submission; it is not enough to look for the state of the instance and check whether it is 'running'. *OAT* is the overall acquisition time for the resource. The outliers in the results have a higher SSH overhead, which is due to latency of the virtual machine boot or network latency between the EC2 and S3 to transfer the AMI.

Releasing resources is a cheap operation that just turns off the virtual machine and triggers other possible housekeeping and cleanup operations. Acquiring resources is quite expensive since it involves transferring the AMI from the S3 store to the EC2 environment and booting the virtual machine. The results show that C-Meter is capable of measuring the detailed overheads of acquiring and releasing virtual resources in the computing cloud.

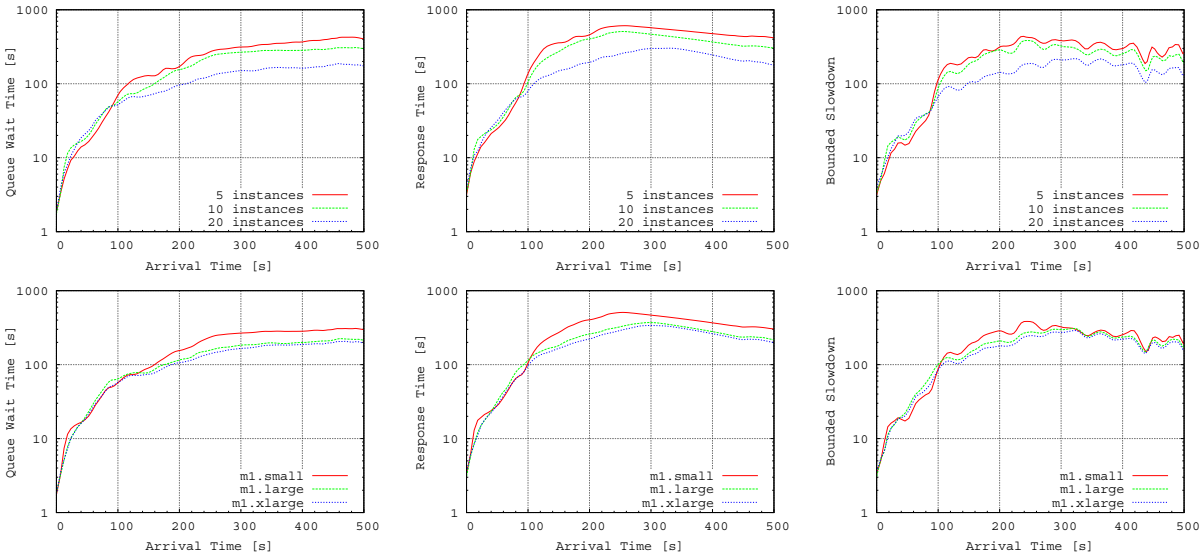

Fig. 4. The waiting time in queue, the response time, and the bounded slowdown with a threshold of 1 second for 5,10, and 20 instances of type m1.small (top), and 10 instances of type m1.small, m1.large, and m1.xlarge (bottom). (Arrival times are relative to the first job submission and the vertical axis has a logarithmic scale. Also note that the data is plotted with smoothed curves.)

B. Performance of Different Configurations

The user can configure C-Meter to acquire both homogeneous and heterogeneous resources by using the resource specification fields of the configuration file. We have performed two experiments with in total six configurations. Each configuration consists of a specific number and type of resources. In the first experiment, we submit the workload to three configurations which consist of 5, 10 and 20 resources of type m1.small respectively. The results of this experiment are shown in Figure 4 and are also summarized in Table II.

In the second experiment we submit the workload to three configurations which consist of 10 instances with m1.small, m1.large and m1.xlarge instance types, respectively. Figure 4 illustrates the results of our second experiment; these results are summarized in Table III.

TABLE II
SUMMARY OF THE RESULTS FOR THE EXPERIMENT WITH 5,10, AND 20 INSTANCES OF TYPE M1.SMALL (FIGURE 4 TOP).

Metric	5 instances	10 instances	20 instances
Avg. Wait Time in Queue [s]	242.39	189.10	114.12
Avg. Response Time [s]	396.40	313.92	187.27
Avg. Bounded Slowdown	321.87	257.15	155.62

From Figure 4 (please note that the vertical axis has logarithmic scale and the data is plotted with smoothed curves) and Table II we conclude that as the number of instances increases, the relative performance also increases. This happens because the load is distributed to more resources, therefore decreasing the queue wait times and also decreasing the network communications bottleneck. Hence, we can say that the cloud environment is *horizontally scalable*, which means that we can increase the performance of the system by acquiring more resources. From Figure 4 and Table III it can be seen that acquiring more powerful resources causes the queue wait time, bounded slowdown and response time to decrease. We conclude that the cloud environment is also *vertically scalable*, which means that we can increase the performance of the system by acquiring more powerful resources.

TABLE III
SUMMARY OF THE RESULTS FOR THE EXPERIMENT WITH 10 INSTANCES OF TYPE M1.SMALL, M1.LARGE, AND M1.XLARGE. (FIGURE 4 BOTTOM)

Metric	m1.small	m1.large	m1.xlarge
Avg. Wait Time in Queue [s]	189.10	137.69	126.37
Avg. Response Time [s]	313.92	228.12	207.40
Avg. Bounded Slowdown	257.15	203.10	184.85

C. Performance of Different Scheduling Algorithms

In C-Meter we have already implemented two different scheduling algorithms. The first scheduling algorithm is the round robin scheduling algorithm, and the second algorithm is a simple heuristic that selects the resource with the minimum predicted response time. The response time is predicted by a simple time-series prediction method which uses the average of the last two response times of that resource as the prediction [8].

TABLE IV
SUMMARY OF THE WAITING TIME IN THE QUEUE, THE RESPONSE TIME , THE BOUNDED SLOWDOWN AND THE EXECUTION TIME FOR 5 INSTANCES OF TYPE M1.SMALL WITH TWO DIFFERENT SCHEDULING ALGORITHMS.

Metric	Predictive	Round Robin
Avg. Wait Time in Queue [s]	150.02	145.29
Avg. Response Time [s]	427.19	396.40
Avg. Bounded Slowdown	283.98	321.87
Avg. Job Execution Time [s]	13.30	2.77

In this experiment we have used 5 instances of type m1.small and submitted the workload using the two scheduling algorithms. The results for this experiment are presented in Figure 5 together with a summary in Table IV. Although the average response time with the predictive scheduling algorithm

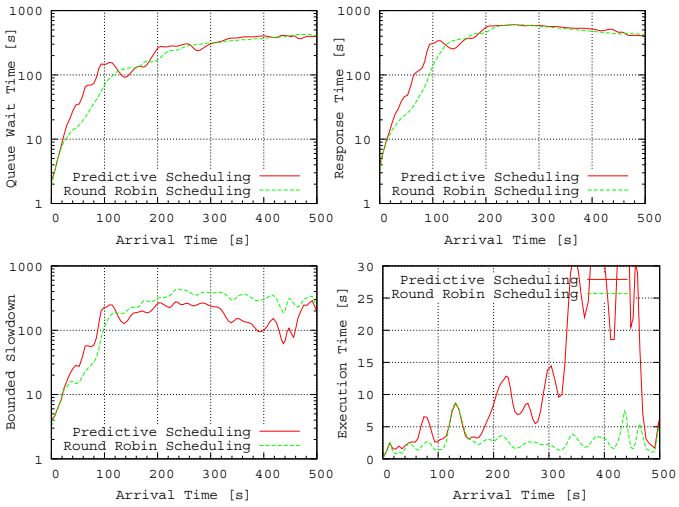

Fig. 5. The waiting time in the queue, the response time, the bounded slowdown with a threshold of 1 second, and the execution time for 5 instances of type m1.small with two different scheduling algorithms.(Arrival times are relative to the first job submission and the vertical axis has a logarithmic scale except for the Execution Time graph. Also note that the data is plotted with smoothed curves.)

is higher, the average slowdown is lower than with the round robin algorithm. We attribute this phenomenon to increased job execution times. This increase is due to the fact that the predictive algorithm schedules many jobs to the same instance causing an uneven load distribution which results in higher loads and network latencies for that instance. The ratio of the numbers of jobs scheduled to the most and the least loaded instance is roughly 4.5 for the predictive scheduling algorithm, whereas this ratio is 1 for the round robin scheduling algorithm. The average accuracy of the prediction method, which is calculated by using the definition in [8], is 80%, which is quite high. This is an important result as it shows that better accuracy does not imply better performance. This experiment shows how C-Meter can be used to evaluate different scheduling algorithms.

V. Related Work

Recently, cloud computing has been attracting the attention of the research community. These research efforts generally focus on the performance assessment of IaaS architectures [9], [10], [11], [12], [13]. However, there is still a lack of tools for performing performance assessment experiments. Closest to our work, in [11] the performance of EC2 for high-performance scientific applications is studied by using micro and macro benchmarks, in [9] Amazon EC2, S3 and SQS are evaluated in terms of the ease of use of the APIs, management facilities and end-to-end performance, in [10] the security, performance and availability of Amazon EC2 and S3 are discussed, and in [12] the performance of Eucalyptus and Amazon EC2 in terms of instance throughput and network latency are assessed. In contrast to these studies, our work focuses on designing and implementing a framework, C-Meter, for performance assessment studies of computing clouds, and assessing the performance of Amazon EC2 with synthetic workloads by using C-Meter.

VI. Conclusion and Future Work

In this paper we have presented C-Meter, which is a portable, extensible and easy-to-use framework for performance analysis of cloud computing environments. It is portable in the sense that it is implemented in Python which is a platform-independent programming language, and it is extensible in the sense that it can be extended to interact with many cloud computing environments and it can also be extended with different scheduling algorithms. We have also presented our early experiences with C-Meter on Amazon EC2 and performed various experiments and analyzed the resulting performance in terms of response time, waiting time in queue, bounded slowdown with a threshold of 1 second and job execution time with the standard instance types.

As future work we plan to perform more extensive experiments with Amazon EC2, and to extend C-Meter with different resource management architectures. We also plan to incorporate C-Meter into a complete resource management framework for computing clouds. For such a resource management framework, many other problems should be addressed, such as job monitoring and control, and also important issues like fault tolerance.

References

[1] The Economist, "A Special Report on Corporate IT," October 2008, "http://www.economist.com/specialReports/showsurvey.cfm?issue=20081025".

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," *CoRR*, vol. abs/0808.3558, 2008.

[3] T. Killalea, "Meet the virts," *Queue*, vol. 6, no. 1, pp. 14–18, 2008.

[4] A. Iosup and D. Epema, "GrenchMark: A Framework for Analyzing, Testing, and Comparing Grids," in *Proc. of the 6th IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid06)*. IEEE Computer Society, 2006, pp. 313–320, an extended version can be found as Technical Report TU Delft/PDS/2005-002, ISBN 1387-2109.

[5] Amazon, "http://aws.amazon.com/ec2/," 2008.

[6] Open Grid Forum, "Job Submission Description Language (JSDL) Specification, Version 1.0," 2005.

[7] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, vol. LNCS 1291, 1997, pp. 1–34.

[8] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather Than User Runtime Estimates," *IEEE Trans. Parallel & Distributed Syst.*, vol. 18, no. 6, pp. 789–803, Jun 2007.

[9] S. L. Garfinkel, "An Evaluation of Amazons Grid Computing Services: EC2, S3 and SQS," Center for Research on Computation and Society School for Engineering and Applied Sciences, Harvard University, Tech. Rep. TR-08-07, 2007.

[10] S. Garfinkel, "Commodity Grid Computing with Amazon's S3 and EC2," *;login*, vol. 32, no. 1, pp. 7–13, 2007.

[11] E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing," *;login*, vol. 33, no. 5, pp. 18–23, 2008.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Eucalyptus : A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems," Department of Computer Science, University of California, Santa Barbara, Tech. Rep. 2008-10, 2008.

[13] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?" in *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*. New York, NY, USA: ACM, 2008, pp. 55–64.