CloudCmp: Shopping for a Cloud Made Easy

Ang Li Xiaowei Yang
Duke University

Srikanth Kandula Ming Zhang
Microsoft Research

Abstract – Cloud computing has gained much popularity recently, and many companies now offer a variety of public cloud computing services, such as Google AppEngine, Amazon AWS, and Microsoft Azure. These services differ in service models and pricing schemes, making it challenging for customers to choose the best suited cloud provider for their applications. This paper proposes a framework called CloudCmp to help a customer select a cloud provider. We outline the design of CloudCmp and highlight the main technical challenges. CloudCmp includes a set of benchmarking tools that compare the common services offered by cloud providers, and uses the benchmarking results to predict the performance and costs of a customer's application when deployed on a cloud provider. We present preliminary benchmarking results on three representative cloud providers. These results show that the performance and costs of various cloud providers differ significantly, suggesting that CloudCmp, if implemented, will have practical relevance.

1. INTRODUCTION

Cloud computing has emerged as a revolutionary approach to provide computing services. It offers several key advantages over a traditional in-house computing model, including on-demand scaling, resource multiplexing, pay-as-you-go metered service, and high-speed network access. A cloud customer only needs to pay the usage costs of its applications, which are likely to be much smaller than the upfront investments of building its own infrastructure.

The advantages of cloud computing have enticed a number of companies to enter this market [2, 4, 5, 8, 9], including traditional online service providers such as Amazon, Google, and Microsoft, web hosting companies such as Rackspace and GoGrid, and new start-ups such as Flexiant and Heroku. A practical challenge thus arises. How can a potential cloud customer, *e.g.*, a small enterprise wishing to outsource its IT infrastructure, decide whether it should migrate some services to a cloud? And if so, which cloud provider should it choose?

Answering these questions is not straightforward, due to the diversity of cloud providers and the applications they support.

Different cloud providers may offer different service models, including platform as a service (PaaS) [13], in which a cloud customer builds its applications using the computing platform (*e.g.*, a Python sand-box) provided by a cloud, and infrastructure as a service (IaaS) [6], in which customers can run their applications within virtual machines of their chosen operating systems.

Different cloud providers also have different pricing models. For instance, Amazon AWS charges customers by the number of virtual instances a customer instantiates and how long it uses them, while Google's AppEngine charges by the number of CPU cycles a customer's application consumes.

Worse yet, applications can have drastically different workloads, *e.g.*, web service versus scientific computing. It is difficult to predict which cloud offers the best performance for a particular application without deploying it on each potential cloud provider. However, deploying an application on a cloud has significant overhead. For instance, to use a PaaS provider, a customer needs to port its application to the cloud platform's APIs. Even for an IaaS provider, deploying an application may involve database and configuration file migrations.

In this work, we propose a framework called CloudCmp that helps a potential cloud customer estimate the performance and costs of running a legacy application on a cloud without actually deploying the application. The framework first characterizes the services offered by various cloud providers into a set of common service interfaces, and benchmarks the performance and costs of these services. It then expresses an application's workload using the interfaces, and estimates the application's performance and costs based on the benchmarking results. This approach to performance prediction resembles the measurement-based prediction approaches widely adopted in evaluating different high performance computing systems [15, 18]. To the best of our knowledge, we are the first to generalize the approach to evaluate cloud providers.

As an initial step, this paper focuses on characterizing and benchmarking the common services offered by different cloud providers. We first identify a set of services shared by six representative cloud providers: Google AppEngine, Amazon AWS, Microsoft Azure, GoGrid, and Rackspace CloudServers/CloudSites. The providers range from PaaS (AppEngine and CloudSites) to IaaS (CloudServers and GoGrid) and a combination of the two (AWS and Azure). We then develop a set of benchmarking tools to measure and compare the performance and monetary costs of each type of services. It is our ongoing work to use these benchmarking results to estimate the performance and costs of an application on

different cloud providers, and we outline the challenges and possible solutions.

Our preliminary benchmarking results indicate that cloud providers differ significantly in both performance and costs of the services they provide, and one provider is unlikely to ace all services. For example, provider X is able to execute the CPU intensive tasks in less than half of the time used by provider Y, while only charging $<10\%$ more per task. However, provider X's storage service is slower than that of others when the data set is large, and has larger latency variation. In another example, provider Z has good performance in computation, but its scaling speed is much slower than that of the other providers. The results suggest that choosing the best suited cloud platform for a particular application is a non-trivial problem, and a cloud comparison framework will be valuable to guide cloud selection.

We note that our measurements represent a snapshot in time. Changes in the load offered by customers as well as changes to the software, hardware, and network infrastructure or the pricing model offered by a provider can qualitatively impact the results. Our goal is to design a comprehensive benchmark that closely approximates customer use-cases. By not being specific to today's cloud offerings, we conjecture that such a comparison framework will help clarify customer choice as cloud services evolve.

2. CLOUD COMPUTING SERVICES

In this section, we summarize the common services offered by six representative cloud providers: Amazon AWS [2], Microsoft Azure [8], Google AppEngine [5], GoGrid [4], and Rackspace CloudServers/CloudSites [9]. The providers have different service models: AppEngine and CloudSites are PaaS providers, whereas CloudServers and GoGrid are IaaS ones. AWS and Azure provide both PaaS and IaaS functionality. The IaaS providers (AWS, Azure, CloudServers, and GoGrid) support native application code, with most of them further supporting different guest operating systems such as Linux and Windows. In contrast, AppEngine and CloudSites only provide sandbox environments to run managed code written in languages such as Java and Python.

We find that the services provided by different cloud providers do not fully overlap. As a preliminary step, we focus on the set of common services that support *web applications*, which is the only type of applications supported by all six cloud providers, and is also one of the most popular types of applications supported by today's cloud providers. Extending to other types such as backup and computation services (*e.g.*, HPC, MapReduce, and Dryad) remains future work.

A cloud provider typically offers the following services for web applications:

- **Elastic compute cluster.** The cluster includes an elastic number of virtual instances that run the application's code and process incoming requests.
- **Persistent storage.** The storage service stores application data in a similar fashion as traditional databases.
- **Intra-cloud network.** The network inside a cloud that connects the virtual instances of an application, and connects them to cloud-provided services such as the persistent storage service.
- **Wide-area delivery network.** The wide-area delivery network of a cloud delivers an application's contents to the end hosts from multiple geographically distributed data centers of the cloud.

Table 1 summarizes the services offered by the six cloud providers. Next, we describe how the cloud providers differ in terms of performance and costs for each type of service.

2.1 Elastic Compute Cluster

A compute cluster provides virtual instances that host and run a customer's application code. The virtual instances among various cloud providers differ in at least three aspects: the underlying hardware, the virtualization technologies, and the hosting environment. These differences determine how efficient an application can run and how many concurrent requests an instance can serve. They may also affect the cost to host an application, as the customer may need more instances to serve the same amount of workload if each instance is weaker.

The compute cluster is also "elastic" in the sense that a customer can dynamically and efficiently scale up and down the number of instances to match the application's workload. An application with low scaling latency can absorb workload surges without introducing additional processing latency, and requires fewer always-running instances. For example, assume an application's workload increases at a maximum rate of 100 requests per second, and each instance can handle 50 concurrent requests. With an unrealistic (short) one second scaling latency to allocate new instances, the customer only needs to maintain two always-running instances in addition to those necessary to support the current load. Since two new instances can be allocated every second, the scaling speed of the cluster keeps up with the workload increase speed and ensures that requests are never bottlenecked for processing. In contrast, if the scaling latency is 10 seconds, to maintain the same guarantee, the customer would have to maintain at least 20 additional always-running instances.

Cloud Provider	Elastic Compute Cluster	Persistent Storage Service	Intra-cloud Network	Wide-area Delivery Network
AWS	Xen-based VM	SimpleDB	Proprietary	3 data center locations (2 in US, 1 in Europe)
Azure	Azure VM	Azure Table		6 data center locations (2 in each of US, Europe, and Asia)
AppEngine	Sandbox	DataStore		Unpublished number of Google data centers
CloudServers	Xen-based VM	N/A		2 data center locations in US
CloudSites	Sandbox	MySQL/MSSQL		2 data center locations in US
GoGrid	Xen-based VM	N/A		1 data center location in US

Table 1: The services offered by the cloud providers.

Pricing: All providers except AppEngine and CloudSites charge customers based on how many instances are allocated and how long each instance is used, regardless of whether the instances are fully utilized. AppEngine charges based on the number of CPU cycles a customer's code consumes; an idle application incurs no cost. It also offers 6.5 free CPU hours per application per day. CloudSites charges a monthly fee which includes a fixed amount of CPU cycles the application can consume. It also charges any over-limit fee based on how many extra cycles are used beyond the quota.

We do not consider the differences in the customizability of a virtual instance, because it does not directly impact the performance or costs of running an application. We also do not consider the engineering cost in porting an application to a cloud, because such cost is hard to measure by a technical method. Our experience suggests that the more complex the application and the ecosystem supporting the application (dependencies on specific databases such as Oracle or specific libraries) the greater the cost. We have manually ported our benchmarking tasks to run on both PaaS and IaaS providers. We had to make minimal modifications to code in this process, whereas the main monetary costs for this research came from running the benchmarking tasks over extended periods.

2.2 Persistent Storage Service

All Cloud providers except CloudServers and GoGrid offer persistent storage services that store the dynamic application data in lieu of the traditional database backends in legacy web applications. Most of the services provide SQL-like interfaces but with limited support for complex operations such as *select*. Compared to traditional databases, the storage services offered by cloud providers are highly scalable and robust against failures via replication [1].

The storage services have similar interfaces across providers, but are implemented with different proprietary technologies. Consequently, they vary in performance, as suggested by our benchmarking results in § 4.

Pricing: Amazon's SimpleDB and CloudSites charge a customer based on the CPU cycles consumed to execute each storage operation. Therefore, a complex operation can incur a higher cost than a simple one. AppEngine's DataStore also charges based on each operation's CPU consumption, but offers 60 free CPU hours per application per day. Azure Table Storage currently charges a customer only based on the number of operations, regardless of each operation's complexity.

2.3 Intra-cloud Network

The network infrastructure inside a cloud provider's data centers connects the virtual instances of an application among themselves and with the shared cloud-provided services. All providers promise high bandwidth network paths (typically from hundreds of Mbps to Gbps), approximating a private data center network. The actual network infrastructures of all the cloud providers we studied are proprietary.

Pricing: Currently intra-cloud network is offered at no cost by all the examined providers.

2.4 Wide-area Delivery Network

The wide-area delivery network serves an application's content from geographically dispersed locations. Each end user's request can be directed to the instances close to the user to minimize the wide-area network latency. Here we focus on the delivery network for *dynamic content*, because static objects can be served by a third-party CDN, decoupled from a cloud platform.

Almost all cloud providers provide a handful of geographically distributed data centers to host an application. Different clouds have data centers at different locations. For example, Azure currently has data centers in US, Europe, and Asia, while CloudServers only has data centers in US. Such differences can affect the application response time observed by an end user, which in turn has significant impact on the application's revenue [16]. This is particularly important as today's web applications face a much broader and geographically more diverse user base than before.

Pricing: Charges for using the wide-area delivery network are based on the amount of data delivered through the cloud boundaries to the end users. Currently all the providers we studied have similar prices for this service.

3. CloudCmp FRAMEWORK

The ultimate goal of the CloudCmp framework is to estimate the performance and costs of a legacy appli-

cation if it is deployed on a chosen cloud provider. A potential cloud customer can use the results to compare different providers and decide whether it should migrate to the cloud and which cloud provider is best suited for its applications.

In this section, we outline a preliminary design of the CloudCmp framework. CloudCmp estimates the performance and costs of an application via three steps: service benchmarking, application workload collection, and performance prediction. For each step, we describe the challenges in realizing it and the possible solutions.

3.1 Service Benchmarking

The goal of service benchmarking is to generate a cloud service's performance and costs profile which includes the time to finish a task and the associated cost.

Challenges: The benchmarking tasks need to be carefully chosen to represent generic customer applications and to avoid measurement bias. For example, to unbiasedly benchmark the performance of a compute cluster, we develop a set of standard CPU, memory, and I/O benchmarking tasks written in Java (some are based on the tests in SPECjvm2008 [11], an existing standard benchmarking suite for Java Virtual Machines). We chose Java to ensure fair comparison and mitigate the bias introduced by different compilers/interpreters, because Java is one of the two languages supported by all cloud providers we study (the other language is Python, which is not as widely used as Java in web applications). This design allows us to run the same Java bytecode on different cloud providers.

We also design benchmarks to measure the performance and costs of other services (i.e., besides CPU/memory/disk benchmarks). Some preliminary benchmarking results are shown in § 4. Due to space limitation, we omit the design details of those tools.

3.2 Application Workload Collection

The second step toward performance/costs prediction is to obtain workload representative of an application. To do so, we propose to collect the application's request traces and each request's execution path. A request's trace should include the source IP of the request, the time when the request arrives, the size of the request, etc.

Challenges: It is a non-trivial task to collect a request's execution path, as it involves recording the sequence of events (*e.g.*, queries/answers sent to/from a database backend) triggered by the request and inferring their causal relationships. It may be necessary to infer the causal relationships between events across layers to predict the impact on an application's performance from changes at one layer.

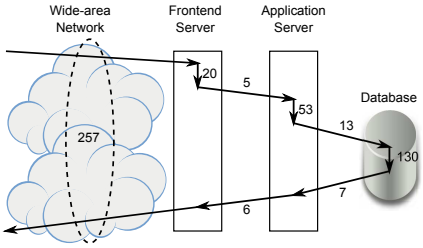


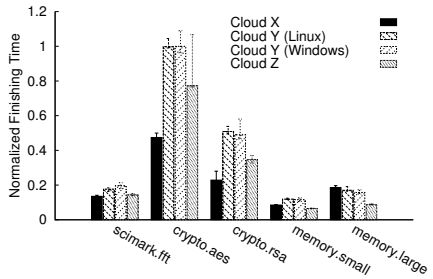
Figure 1: An example of a request in a standard three-tier web application. The execution path of the request is shown and the time (ms) is marked at each component on the path. The processing time of each component is computed from the service performance profiles, and the overall processing time (491ms) is the sum of each component's processing time as the execution path is single-threaded.

Fortunately, there has been much study in obtaining execution paths annotated with causal relationships. One technique of particular interest is vPath [19]. It exploits the synchronous intra-thread communication pattern of the thread-based programming model and extracts the causality of events within the same thread. vPath can infer the precise request execution path in multi-threaded programs without modifying the application. Another technique //trace [17] uses throttling to infer event causality: if throttling a parent event leads to the absence of a child event, it indicates a causal relationship between the two events. In our design, we propose to use a vPath-like technique to obtain a request's execution path if the application uses the common thread programming model. For applications that do not conform to the model, we will use the throttling technique as in //trace to infer inter-event dependencies.

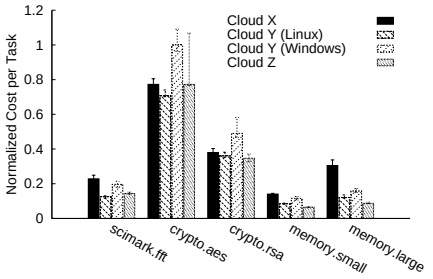
3.3 Performance Prediction

In this step, we will try to predict the processing time and the costs of each application request we collect by using the pre-generated performance and costs profile of each cloud service. We first use the profiles to estimate the time and costs spent at each component on the execution path, and then combine the individual processing time and costs together to obtain the overall time and costs, as shown in Figure 1.

Challenges: To compute the overall processing time of a request, we need to identify the critical path that takes the longest time among all branches in the execution path. This can be obtained by simulating how a request is processed. The simulated processing time at each component is estimated from the service performance profiles. We can simplify the performance prediction step if a request is processed in a single thread, *i.e.*, the execution path contains only one branch, because in this case the overall processing time equals the sum of each component's processing time.



(a) Normalized finishing time



(b) Normalized cost per task

Figure 2: The benchmarking results of the compute clusters. Figure (a) shows the finishing time of each benchmarking task. The time is normalized using the longest finishing time in all tests. Figure (b) shows the normalized cost per task.

4. BENCHMARKING RESULTS

In this section, we present some preliminary benchmarking results of the common services offered by three popular and representative cloud providers. Due to legal concerns, we anonymize the names of the providers we study, and refer to them as cloud X, Y, and Z.

The goal of the measurement study is to examine whether the performance of the cloud services varies across different providers. It is our future work to further understand the causes of the performance differences. The results in this section show a snapshot of the performance and costs of each cloud provider at the provider's default data center over a two-week period from March 13th to March 25th, 2010. We are currently extending the measurements to run continuously in all the data centers offered by these cloud providers. More results will be available on the project's website [3].

We stress that the specifics of the results will change as providers evolve their software, hardware and network. Our goal is to formulate a methodology to compare providers and we use these results to only demonstrate the value of such a comparison tool.

4.1 Elastic Compute Cluster

We first compare the performance of the elastic compute clusters. We benchmark both the efficiency of a virtual instance and the scaling latency to allocate a new instance. To benchmark a virtual instance's efficiency, we develop a set of standard benchmarking tasks in Java and run them on different compute clusters. For each provider, we choose to benchmark the default instance type. For cloud X, we run the tasks on its default sandbox environment. For Y, we run the tasks on both Linux (Ubuntu 9.04) and Windows (Server 2008) based instances to test whether different OSes would affect the benchmarking results. For Z, we run the tasks on Windows Server 2008.

Figure 2(a) shows the results. We include the finishing time of five tasks, with three CPU intensive ones (scimark.fft, crypto.aes, and crypto.rsa) and two memory intensive ones (memory.small and memory.large). We can see from the figure that for CPU intensive tasks, cloud X has the best performance, followed by Z, while Y has the worst performance. For cryptographic computations, cloud X instances are almost twice as fast as Y instances. However, for memory intensive tasks, X's performance does not show significant advantages. In fact, for the task with a large memory allocation size (memory.large) that does not fit into the L2 cache, X performs the worst. This result suggests that X instances might have faster CPUs but slower memory controllers. The two types of Y instances (Linux and Windows-based) have similar performance. This suggests that the benchmarking results are independent of the guest OSes. Therefore, they are likely to reflect the differences in the underlying hardware and the virtualization technologies used by different cloud providers.

Cost We further measure the cost of running each task. Cloud Y and Z charge a customer based on how long each virtual instance is used. We thus compute the cost per task using the task's finishing time and the per hour price for them. X charges a customer based on each task's CPU consumption. We compute the task's cost by multiplying the CPU time with the per CPU second price for X. The pricing information is obtained from each provider's public website.

Figure 2(b) shows the results. Despite having the worst performance in the previous test, the Linux-based cloud Y instance turns out to have the lowest cost for the first two tasks. This is because the Linux-based Y instance has the lowest per hour price among the three instance types charged by usage time (Linux/Windows-based Y and Z instances). The cost per task of the Linux-based Y instance is even lower than that of X, although the task takes less time to finish on X. The results suggest that the metric of cost per task captures the cost-effectiveness of each service regardless of its pricing scheme.

Scaling We then benchmark the scaling service by measuring the latency between when a new instance is re-

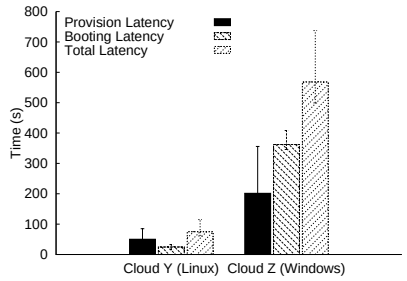


Figure 3: The scaling latency to allocate a new virtual instance.

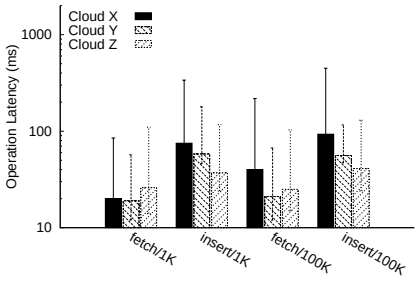


Figure 4: The time it takes to insert/fetch a random entry to/from a data table in a storage service. The x-axis label shows the operation type (fetch or insert) and the size of the table (1K entries or 100K entries). The y-axis is in a logarithmic scale.

quested and when the instance is ready. We do not consider X because its scaling is done automatically and cannot be triggered manually. The scaling latency is further split into a provisioning latency, during which the cloud is preparing to launch a new instance, and a booting latency, during which the instance's operating system is booting.

Figure 3 shows the result. It show that cloud Y has a much shorter scaling latency than cloud Z. This suggests that the two cloud providers perhaps use different strategies to manage virtual instances. Furthermore, the booting time of Linux on Y appears shorter than that of Windows 2008 on Z, suggesting that choosing the right combination of the guest OS and the virtualization layer can be important for scaling performance.

4.2 Persistent Storage Service

We benchmark the performance of the persistent storage services by measuring the latency to insert (or fetch) a random entry to and from a pre-defined data table in each service. We run the tests against two tables with 1K and 100K entries respectively, and Figure 4 shows the results. All the storage services exhibit significant variations in latency across requests (indicated by the lengths of the error bars whose ends show the 5th and 95th percentiles respectively). For fetch operations, Y performs best under both table sizes, while for insert operations, Z outperforms the other two. X's storage service performs well when the table size is small, but its latency increases significantly with a large variance when the table size increases. The results suggest that different storage services can offer different performance at different data scales with different operations. Hence, it is important to design benchmarking tools that cover a comprehensive set of operations.

Cloud Provider	TCP throughput (Mbps)		
	Median	5th Percentile	95th Percentile
Y	630	465	742
Z	626	15.1	687

Table 2: The TCP throughput between two instances in each cloud.

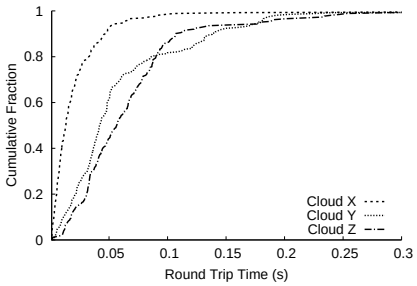


Figure 5: The distributions of the round trip time from 260 PlanetLab vantage points to their optimal data centers.

4.3 Intra-cloud Network

The intra-cloud network performance is benchmarked by measuring the available bandwidth between two instances in the cloud. We use the `iperf` tool [7] to measure the average TCP throughput between two instances, and repeat the experiment on many different pairs of instances. We do not consider cloud X because it does not allow direct communication between instances. Table 2 shows the results. From the table, we see both providers have an average throughput higher than 600Mbps. Cloud Z's throughput has a large variation, ranging from 10Mbps to around 700Mbps. We speculate that it might be caused by network congestion inside Z's data center.

4.4 Wide-area Delivery Network

We measure the latency of each cloud's delivery network by sending ping packets to the application instances deployed at a cloud provider's data centers. We send the ping packets from geographically distributed vantage points. For each vantage point, we choose the data center with the minimum round trip time as the optimal data center selected by a perfect geographic load balancing algorithm.

Figure 5 shows the distributions of the round trip time measured from 260 randomly chosen vantage points on PlanetLab to their optimal data centers. We see from the figure that X has the lowest wide-area network latency distribution among the three providers. This is

not surprising because X also offers popular online services globally, and is known to have a large number of data centers around the world. Y and Z have similar latency distributions and Y has slightly more vantage points with latencies greater than 100ms. Closer examination of the data reveals that those vantage points are mostly in Asia and South America, where cloud Y does not have any data center presence.

5. RELATED WORK

Garfinkel [14] and Walker [20] investigate the performance of one specific cloud provider: Amazon's AWS. In contrast, this work focuses on comparing the performance of multiple cloud providers. It serves as an initial step toward a framework that can assist a customer to choose a cloud provider based on its cost and performance requirements. A few industry reports perform simple pair-wise evaluations on the computation performance of different cloud instances [10, 12]. This work compares a broader range of cloud services including computation, scaling, storage, and networking services. Ward compares the performance of a public cloud provider AWS with that of a private cloud software suite: Ubuntu Enterprise Cloud [21], while this work focuses on comparing the performance and costs of the public cloud providers.

6. CONCLUSION AND FUTURE WORK

Cloud computing has gained much momentum in recent years due to its economic advantages. Various cloud providers offer different types of services with different pricing schemes, posing a practical challenge on how to choose the best performing cloud provider for an application. This paper proposes a framework to estimate and compare the performance and costs of deploying an application on a cloud. This framework characterizes the common set of services a cloud provides, including computation, storage, and networking services, benchmarks each service's performance and costs, and combines the benchmarking results with an application's workload to estimate the application's processing time and costs. As an initial step toward building such a framework, we benchmark and compare the cloud services provided by three representative cloud providers to support web applications. The results show that the performance and costs of different cloud providers can differ significantly, suggesting that a systematic cloud comparison framework is highly relevant.

In the future, we plan to extend the benchmarking tools to cover more cloud providers, more types of cloud services (*e.g.*, the different tiers of instances offered by the same cloud provider and the services that support other types of applications), and more data center locations. We also plan to provide continuous rather than snapshot benchmarking results. We can then build a complete CloudCmp system to predict application performance and costs based on the benchmarking results we collect.

Acknowledgements

This work was funded in part by an NSF CAREER Award CNS-0845858 and Award CNS-0925472. We thank the anonymous reviewers for their helpful feedback and suggestions.

7. REFERENCES

[1] Amazon SimpleDB. `http://aws.amazon.com/simpledb/`.
[2] Amazon Web Service. `http://aws.amazon.com`.
[3] CloudCmp Project Website. `http://cloudcmp.net`.
[4] GoGrid Cloud Hosting. `http://gogrid.com`.
[5] Google AppEngine. `http://code.google.com/appengine`.
[6] Infrastructure-as-a-Service (IaaS). *Gni, Tech. Rep.*
[7] Iperf. `http://iperf.sourceforge.net`.
[8] Microsoft Windows Azure. `http://www.microsoft.com/windowsazure`.
[9] Rackspace Cloud. `http://www.rackspacecloud.com`.
[10] Rackspace Cloud Servers versus Amazon EC2: Performance Analysis. `http://www.thebitsource.com/featured-posts/rackspace-cloud-servers-versus-amazon-ec2-performance-analysis/`.
[11] SPEC Java Virtual Machine Benchmark 2008. `http://www.spec.org/jvm2008/`.
[12] VPS Performance Comparison. `http://journal.uggedal.com/vps-performance-comparison`.
[13] D. Cheng. PaaS-onomics: A CIOs Guide to using Platform-as-a-Service to Lower Costs of Application Initiatives While Improving the Business Value of IT. *Longjump, Tech. Rep.*
[14] S. Garfinkel. An Evaluation of Amazon s Grid Computing Services : EC2 , S3 and SQS. *Harvard University, Tech. Rep. TR-08-07*.
[15] S. Hammond, G. Mudalige, J. Smith, S. Jarvis, J. Herdman, and A. Vadgama. WARPP: a toolkit for simulating high-performance parallel scientific codes. In *International Conference on Simulation Tools and Techniques*, page 19, 2009.
[16] R. Kohavi, R. M. Henne, and D. Sommerfield. Practical guide to controlled experiments on the web. In *ACM KDD*, volume 2007. ACM Press, 2007.
[17] M. P. Mesnier, M. Wachs, R. R. Sambasivan, J. Lopez, J. Hendricks, G. R. Ganger, and D. O'Hallaron. //trace: parallel trace replay with approximate causal events. In *USENIX FAST*, 2007.
[18] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. Pace–A Toolset for the Performance Prediction of Parallel and Distributed Systems. *International Journal of High Performance Computing Applications*, 14(3):228, August 2000.
[19] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang. vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities. In *USENIX ATC*, 2009.
[20] E. Walker. Benchmarking amazon EC2 for high-performance scientific computing. *USENIX Login*, 2008.
[21] J. S. Ward. A Performance Comparison of Clouds: Amazon EC2 and Ubuntu Enterprise Cloud. SICSA DemoFEST, 2009.