CloudCmp: Comparing Public Cloud Providers

Ang Li Xiaowei Yang
Duke University
{angl, xwy}@cs.duke.edu

Srikanth Kandula Ming Zhang
Microsoft Research
{srikanth, mzh}@microsoft.com

ABSTRACT

While many public cloud providers offer pay-as-you-go computing, their varying approaches to infrastructure, virtualization, and software services lead to a problem of plenty. To help customers pick a cloud that fits their needs, we develop CloudCmp, a systematic comparator of the performance and cost of cloud providers. CloudCmp measures the elastic computing, persistent storage, and networking services offered by a cloud along metrics that directly reflect their impact on the performance of customer applications. CloudCmp strives to ensure fairness, representativeness, and compliance of these measurements while limiting measurement cost. Applying CloudCmp to four cloud providers that together account for most of the cloud customers today, we find that their offered services vary widely in performance and costs, underscoring the need for thoughtful provider selection. From case studies on three representative cloud applications, we show that CloudCmp can guide customers in selecting the best-performing provider for their applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: General—*measurement techniques, performance attributes*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*network monitoring, public networks*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications, distributed databases*

General Terms

Measurement, Design, Performance, Economics.

Keywords

Cloud computing, comparison, performace, cost.

1. INTRODUCTION

Internet-based cloud computing has gained tremendous momentum in recent years. Cloud customers outsource their computation and storage to public providers and pay for the service usage on demand. Compared to the traditional computing model that uses dedicated, in-house infrastructure, cloud computing offers unprecedented advantages in terms of cost and reliability [22, 27]. A cloud customer need not pay a large upfront cost (*e.g.*, for hardware purchase) before launching services, or over-provision to accommodate future or peak demand. Instead, the cloud's pay-as-you-go charging model enables the customer to pay for what she actually uses and promises to scale with demand. Moreover, the customer can avoid the cost of maintaining an IT staff to manage her server and network infrastructure.

A growing number of companies are riding this wave to provide public cloud computing services, such as Amazon, Google, Microsoft, Rackspace, and GoGrid. These cloud providers offer a variety of options in pricing, performance, and feature set. For instance, some offer platform as a service (PaaS), where a cloud customer builds applications using the APIs provided by the cloud; others offer infrastructure as a service (IaaS), where a customer runs applications inside virtual machines (VMs), using the APIs provided by their chosen guest operating systems. Cloud providers also differ in pricing models. For example, Amazon's AWS charges by the number and duration of VM instances used by a customer, while Google's AppEngine charges by the number of CPU cycles consumed by a customer's application.

The diversity of cloud providers leads to a practical question: *how well does a cloud provider perform compared to the other providers?* Answering this question will benefit both cloud customers and providers. For a potential customer, the answer can help it choose a provider that best fits its performance and cost needs. For instance, it may choose one provider for storage intensive applications and another for computation intensive applications. For a cloud provider, such answers can point it in the right direction for improvements. For instance, a provider should pour more resources into optimizing table storage if the performance of its store lags behind competitors.

Despite the practical relevance of comparing cloud providers, there have been few studies on this topic. The challenge is that every provider has its own idiosyncratic ways of doing things, so finding a common ground needs some thought. A few efforts have characterized the performance of one IaaS provider (Amazon AWS) [24, 34]. Some recent blog posts [6, 15, 36] compare Amazon AWS with one other provider each. These measurements are limited in scope; none of them cover enough of the dimensions (e.g., compute, storage, network, scaling) to yield meaningful conclusions. Further, some of the measurement methodologies do not extend to all providers, e.g., they would not work for PaaS providers.

In this paper, we consider the problem of systematically comparing the performance of cloud providers. We identify the key requirements for conducting a meaningful comparison (§2), develop

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IMC'10, November 1–3, 2010, Melbourne, Australia.
Copyright 2010 ACM 978-1-4503-0057-5/10/11 ...$10.00.

a tool called CloudCmp, and use CloudCmp to evaluate a few cloud providers (§3– §4) that differ widely in implementation but together dominate the cloud market. Our results (§5) provide a customer with the performance-cost trade-offs across providers for a wide set of metrics. For providers, the results point out specific areas for improvement in their current infrastructures.

Several technical challenges arise in realizing a comparator for cloud providers. The first is the choice of what to measure. Rather than focusing on the nitty-gritty such as which virtualization technology a provider uses or how it implements its persistent storage, we take an end-to-end approach that focuses on the dimensions of performance that customers perceive. Doing so has the advantage that the measurement methodology remains stable even as the implementations change over time or differ widely across providers. To this end, we identify a common set of services offered by these providers, including elastic computing, persistent storage, and intra-cloud and wide-area networking (§3.2).

The second challenge is the choice of how to measure customer perceived performance of these services. For each service, we focus on a few important metrics, *e.g.*, speed of CPU, memory, and disk I/O, scaling latency, storage service response time, time to reach consistency, network latency, and available bandwidth (§3.3). We leverage pre-existing tools specific to each metric. However, when applying the tools, we had to be careful along several axes, such as using variable number of threads to test multi-core, piecing apart the interference from colocated tenants and the infrastructure itself, and covering the wide set of geographically distributed data centers offered by the providers. The individual tools *per se* are simple, but this specific collection of them that enables comparing cloud providers is novel.

Third, as with all significant measurement efforts, we trade off development cost to the completeness of the study. We skip functionality that is specific to small classes of applications but are comprehensive enough that our benchmark results allow predicting the performance of three representative applications: a storage intensive e-commerce web service, a computation intensive scientific computing application, and a latency sensitive website serving static objects. By deploying these applications on each cloud, we demonstrate that the predictions from CloudCmp align well with the actual application performance. CloudCmp enables predicting application performance without having to first port the application onto every cloud provider.

Finally, unlike other measurement efforts, we are constrained by the monetary cost of measuring the clouds and the acceptable use policies of the providers. We note that the results here were achieved under a modest budget by judicious choice of how many and how often to measure. CloudCmp complies with all acceptable use policies.

We used CloudCmp to perform a comprehensive measurement study over four major cloud providers, namely, Amazon AWS, Microsoft Azure, Google AppEngine, and Rackspace CloudServers. We emphasize that the infrastructure being measured is ephemeral. Providers periodically upgrade or regress in their software or hardware and customer demands vary over time. Hence, these results are relevant only for the time period in which they were generated. To keep the focus on the value of this comparison method and its implications rather than rank providers, our results use labels $C_1 - C_4$ instead of provider names.

From the comparison results, we find that the performance and price of the four providers vary significantly with no one provider standing out (§5). For instance, while the cloud provider C_1 has the highest intra-cloud bandwidth, its virtual instance is not the most cost-effective. The cloud provider C_2 has the most powerful virtual instances, but its network bandwidth is quite limited. C_3 offers the lowest wide-area network latency, but its storage service is slower than that of its competitors. We highlight a few interesting findings below:

- Cloud instances are not equally cost-effective. For example, while only 30% more expensive, C_4's virtual instance can be twice as fast as that of C_1.
- C_2 in our study allows a virtual instance to fully utilize the underlying physical machine when there is no local resource competition. Hence, an instance can attain high performance at low cost.
- The performance of the storage service can vary significantly across providers. For instance, C_1's table query operation is an order of magnitude faster than that of the others.
- The providers offer dramatically different intra-datacenter bandwidth, even though intra-datacenter traffic is free of charge. For instance, C_1's bandwidth is on average three times higher than C_2's.

The measurement data we collected is available at `http://www.cloudcmp.net`.

We believe that this is the first study to comprehensively characterize the performance and cost of the major cloud providers in today's market. Though we present results for four providers in this paper, we believe the techniques in CloudCmp can be extended to measure other providers. In future work, we plan to build performance prediction models based on CloudCmp's results to enable fast and accurate provider selection for arbitrary applications (§7).

2. GOALS AND APPROACH

In this section, we highlight the design goals of CloudCmp and briefly describe how we meet them.

1. **Guide a customer's choice of provider:** Our primary goal is to provide performance and cost information about various cloud providers to a customer. The customer can use this information to select the right provider for its applications. We choose the cost and performance metrics that are relevant to the typical cloud applications a customer deploys. These metrics cover the main cloud services, including elastic computing, persistent storage, and intra-cloud and wide-area networking.
2. **Relevant to cloud providers:** We aim to help a provider identify its under-performing services compared to its competitors. We not only present a comprehensive set of measurement results, but also attempt to explain what causes the performance differences between providers. This enables a provider to make targeted improvements to its services.
3. **Fair:** We strive to provide a fair comparison among various providers by characterizing all providers using the same set of workloads and metrics. This restricts our comparative study to the set of common services offered by all providers. The core functionality we study suffices to support a wide set of cloud applications. However, we skip specialized services specific to some applications that only a few providers offer. While support for functionality is a key decision factor that we will consider in future work, our focus here is on the performance-cost trade-off.
4. **Thoroughness vs. measurement cost:** For a thorough comparison of various cloud providers, we should measure all cloud providers continuously across all their data centers.

Provider	Elastic Cluster	Storage	Wide-area Network
Amazon AWS	Xen VM	SimpleDB (table), S3 (blob), SQS (queue)	3 DC locations (2 in US, 1 in EU)
Microsoft Azure	Azure VM	XStore (table, blob, queue)	6 DC locations (2 each in US, EU, and Asia)
Google AppEngine	Proprietary sandbox	DataStore (table)	Unpublished number of Google DCs
Rackspace CloudServers	Xen VM	CloudFiles (blob)	2 DC locations (all in US)

Table 1: The services offered by the cloud providers we study. The intra-cloud networks of all four providers is proprietary, and are omitted from the table.

This, however, incurs significant measurement overhead and monetary costs. In practice, we periodically (*e.g.*, once an hour) measure each provider at different times of day across all its locations. The measurements on different providers are loosely synchronized (*e.g.*, within the same hour), because the same measurement can take different amount of time to complete in different providers.

5. **Coverage vs. development cost:** Ideally, we would like to measure and compare all cloud providers on the market. Achieving this goal, however, can be cost and time prohibitive. We cover a representative set of cloud providers while restricting our development cost. We choose the cloud providers to compare based on two criteria: popularity and representativeness. That is, we pick the providers that have the largest number of customers and at the same time represent different models such as IaaS and PaaS. Our measurement methodology, however, is easily extensible to other providers.
6. **Compliant with acceptable use policies:** Finally, we aim to comply with cloud providers' use policies. We conduct experiments that resemble the workloads of legitimate customer applications. We do not overload the cloud infrastructures or disrupt other customer applications.

3. MEASUREMENT METHODOLOGY

In this section, we describe how we design CloudCmp to conduct a fair and application-relevant comparison among cloud providers. We first show how we select the providers to compare, and discuss how to choose the common services to ensure a fair comparison. Then for each type of service, we identify a set of performance metrics that are relevant to application performance and cost.

3.1 Selecting Providers

Our comparative study includes four popular and representative cloud providers: Amazon AWS [2], Microsoft Azure [12], Google AppEngine [7], and Rackspace CloudServers [14]. We choose Amazon AWS and Rackspace CloudServers because they are the top two providers that host the largest number of web services [19]. We choose Google AppEngine because it is a unique PaaS provider, and choose Microsoft Azure because it is a new entrant to the cloud computing market that offers the full spectrum of computation and storage services similar to AWS.

3.2 Identifying Common Services

Despite the complexity and idiosyncrasies of the various cloud providers, there is a common core set of functionality. In this section, we focus on identifying this common set, and describe the experience of a customer who uses each functionality. We defer commenting on the specifics of how the cloud achieves each functionality unless it is relevant. This allows us to compare the clouds from the end-to-end perspective of the customer and sheds light on the meaningful differences. The common set of functionality includes:

- **Elastic compute cluster.** The cluster includes a variable number of virtual instances that run application code.
- **Persistent storage.** The storage service keeps the state and data of an application and can be accessed by application instances through API calls.
- **Intra-cloud network.** The intra-cloud network connects application instances with each other and with shared services.
- **Wide-area network.** The content of an application is delivered to end users through the wide-area network from multiple data centers (DCs) at different geographical locations.

These services are offered by most cloud providers today because they are needed to support a broad spectrum of applications. For example, a web application can have its servers run in the elastic compute cluster, its data stored in the persistent storage, and its content delivered through the wide-area network. Other cloud applications such as document translation, file backup, and parallel computation impose different requirements on these same components. A few providers offer specialized services for specific applications (e.g., MapReduce). We skip evaluating these offerings to focus on the more general applications. Table 1 summarizes the services offered by the providers we study.

3.3 Choosing Performance Metrics

For each of these cloud services, we begin with some background and describe the performance and cost metrics we use to characterize that service.

3.3.1 Elastic Compute Cluster

A compute cluster provides virtual instances that host and run a customer's application code. Across providers, the virtual instances differ in their underlying server hardware, virtualization technology, and hosting environment. Even within a provider, multiple tiers of virtual instances are available, each with a different configuration. For example, the instances in the higher tier can have faster CPUs, more CPU cores, and faster disk I/O access. These differences do impact the performance of customer applications.

The compute cluster is charged per usage. There are two types of charging models among the providers we study. The IaaS providers (AWS, Azure, and CloudServers) charge based on how long an instance remains allocated, regardless of whether the instance is fully utilized or not. However, the PaaS provider (AppEngine) charges based on how many CPU cycles a customer's application consumes in excess of a few free CPU hours per application per day.

The compute cluster is also "elastic" in the sense that a customer can dynamically scale up and down the number of instances it uses to withstand its application's varying workload. Presently, there are two types of scaling mechanisms: opaque scaling and transparent scaling. The former requires a customer herself to manually change the number of instances or specify a scaling policy, such as creating a new instance when average CPU usage exceeds 60%. The latter automatically tunes the number of instances without customer intervention. AWS, Azure, and CloudServers support opaque scaling whereas AppEngine provides transparent scaling.

Service	Operation	Description
Table	get	fetch a single row using the primary key
	put	insert a single row
	query	lookup rows that satisfy a condition on a non-primary key field
Blob	download	download a single blob
	upload	upload a single blob
Queue	send	send a message to a queue
	receive	retrieve the next message from a queue

Table 2: The operations we use to measure the performance of each storage service.

We use three metrics to compare the performance of the compute clusters: benchmark finishing time, cost per benchmark, and scaling latency. These metrics reflect how fast an instance can run, how cost-effective it is, and how quickly it can scale.

Benchmark finishing time. Similar to conventional computational benchmark metrics for computer architectures [18], this metric measures how long the instance takes to complete the benchmark tasks. The benchmark has tasks that stress each of the main compute resources (CPU, memory, and disk I/O).

Cost. This is the monetary cost to complete each benchmark task. Because we use the same tasks across different instances provided by different clouds, customers can use this metric to compare the cost-effectiveness of the instances regardless of their prices and charging models. Together with the above metric, this provides customers with a view of the performance-cost trade-offs across providers. These metrics correspond to the criteria that customers use when choosing, such as best performance within a cost budget or lowest cost above a performance threshold.

Scaling latency. This is the time taken by a provider to allocate a new instance after a customer requests it. The scaling latency of a cluster can affect the performance and cost of running an application. An application can absorb workload spikes more quickly and can keep fewer number of instances running continuously if it can instantiate new instances quickly. With this metric, a customer can choose the compute cluster that scales the fastest or design better scaling strategies. She can also make more nuanced decisions based on what it would cost to provide good performance when the workload of her application varies.

There are a few other metrics, such as the customizability of a virtual instance and the degree of automation in management, that capture vital aspects of cloud providers. However, these are harder to quantify. Hence, we focus on the performance and costs of running an application and defer considering other metrics to future work.

3.3.2 Persistent Storage

Cloud providers offer persistent storage for application state and data. There are currently three common types of storage services: table, blob, and queue. The table storage is designed to store structural data in lieu of a conventional database, but with limited support for complex queries (*e.g.*, table join and group by). The blob storage is designed to store unstructured blobs, such as binary objects, user generated data, and application inputs and outputs. Finally, the queue storage implements a global message queue to pass messages between different instances. Most storage services are implemented over HTTP tunnels, and while not standardized, the usage interfaces are stable and similar across providers.

The cloud storage services have two advantages over their conventional counterparts: scalability and availability. The services are well-provisioned to handle load surges and the data is replicated [1] for high availability and robustness to failures. However, as a trade-off, cloud storage services do not offer strong consistency guarantees [25]. Therefore, an application can retrieve stale and inconsistent data when a `read` immediately follows a `write`.

There are presently two pricing models for storage operations. The table services of AWS and AppEngine charge based on the CPU cycles consumed to run an operation. Thus, a complex query costs more than a simple one. Azure and CloudServers have a fixed per-operation cost regardless of the operation's complexity.

We use three metrics to compare the performance and cost of storage services: operation response time, time to consistency, and cost per operation.

Operation response time. This metric measures how long it takes for a storage operation to finish. We measure operations that are commonly supported by providers and are popular with customers. Table 2 summarizes these operations. They include the basic `read` and `write` operations for each storage service. For table storage service, we also use an SQL-style `query` to test the performance of table lookup. In §6.1, we show that these operations account for over 90% of the storage operations used by a realistic e-commerce application.

Time to consistency. This metric measures the time between when a datum is written to the storage service and when all reads for the datum return consistent and valid results. Such information is useful to cloud customers, because their applications may require data to be immediately available with a strong consistency guarantee. Except for AppEngine, cloud providers do not support storage services that span multiple data centers. Therefore, we focus on consistency when the reads and writes are both done from instances inside the same data center.

Cost per operation. The final metric measures how much each storage operation costs. With this metric, a customer can compare the cost-effectiveness across providers.

3.3.3 Intra-cloud Network

The intra-cloud network connects a customer's instances among themselves and with the shared services offered by a cloud. The performance of the network is vital to the performance of distributed applications. Within a cloud, the intra-datacenter network often has quite different properties compared to the inter-datacenter network. Providers vary in the type of network equipment (NICs, switches) as well as in their choice of routing (layer 2 vs. layer 3) and configuration such as VLANs. All providers promise high intra-datacenter bandwidth (typically on the order of hundreds of Mbps to Gbps), approximating a private data center network.

To compare the performance of intra-cloud networks, we use path capacity and latency as metrics. We use TCP throughput as a measure of path capacity because TCP is the dominant traffic type of cloud applications. Path capacity impacts data transfer throughput and congestion events can lead to errors or delayed responses. Path latency impacts both TCP throughput [30] and end-to-end response time. Together, these metrics provide insight into how a provider's intra-cloud network is provisioned.

None of the providers charge for traffic within their data centers. Inter-datacenter traffic is charged based on the volume crossing the data center boundary. Since all providers charge similar amounts, comparing cost of network transfers becomes a moot point.

3.3.4 Wide-area Network

The wide-area network is defined as the collection of network paths between a cloud's data centers and external hosts on the Internet. All the providers we study offer multiple locations to host

customer applications. Requests from an end user can be served by an instance close to that user to reduce latency. Uniquely, AppEngine offers a DNS-based service to automatically map requests to close-by locations. The others require manual configuration.

We use the *optimal wide-area network latency* to compare providers' wide-area networks. The optimal wide-area network latency is defined as the minimum latency between a vantage point and any data center owned by a provider. We use locations of PlanetLab nodes as vantage points. The more the data centers offered by a provider and the closer they are to population centers, the smaller the optimal network latency. The metric is useful for customers because it corresponds to the network latency an application may experience given an ideal mapping. For AppEngine, which provides automatic mapping of requests to locations, we also measure how close its automatic mapping is to the optimal mapping.

4. IMPLEMENTATION

In this section, we describe the implementation details of CloudCmp and highlight the practical challenges we address.

4.1 Computation Metrics

Benchmark tasks. As described above, we would like a suite of benchmark tasks that stresses various aspects of the compute infrastructure offered by cloud providers. In traditional computation performance measurement, any benchmark suite, such as the SPEC CPU2006 benchmarks [16], would fit this bill. However, the context of cloud computing poses new constraints. For example, AppEngine only provides sand-boxed environments for a few cross-platform programming languages, and applications have to be singled-threaded and finish within limited time.

To satisfy those constraints and be fair across different providers, we modified a set of Java-based benchmark tasks from SPECjvm2008 [17], a standard benchmark suite for Java virtual machines. We choose Java because it is supported by all cloud providers. The benchmark suite includes several CPU intensive tasks such as cryptographic operations and scientific computations. We augment it with memory and I/O intensive tasks. Each benchmark task runs in a single thread and finishes within 30 seconds so as to be compatible with all providers.

Benchmark finishing time. We run the benchmark tasks on each of the virtual instance types provided by the clouds, and measure their finishing time. Some instances offer multiple CPU cores for better parallel processing capability. For these instances, we also evaluate their multi-threading performance by running instances of the same benchmark task in multiple threads simultaneously, and measuring the amortized finishing time of the task. The number of threads is set to be equivalent to the number of available CPU cores.

Cost per benchmark. For cloud providers that charge based on time, we compute the cost of each benchmark task using the task's finishing time and the published per hour price. For AppEngine that charges by CPU cycles, we use its billing API to directly obtain the cost.

Scaling latency. We write our own scripts to repeatedly request new virtual instances and record the time from when the instance is requested to when it is available to use. We further divide the latency into two segments: a provisioning latency and a booting latency. The former measures the latency from when an instance is requested to when the instance is powered on. The latter measures the latency from the powered-on time to when the instance is ready to use. The separation of the two is useful for a cloud provider to pinpoint the performance bottleneck during instance allocation.

4.2 Storage Metrics

Benchmark tasks. Along with each storage service, comes an API to get, put or query data from the service. Most APIs are based on HTTP. To use the APIs, we wrote our own Java-based client based on the reference implementations from the providers [3, 9, 21]. The client has a few modifications over the reference implementations to improve latency. It uses persistent HTTP connections to avoid SSL and other connection set up overheads. It also skips a step in some implementations in which a client first sends a request header and waits an RTT until the server returns an HTTP 100 (Continue) message before proceeding with the request body. For comparison, we also tested other non-Java-based clients such as *wget* and C#-based clients. To avoid the potential impact of memory or disk bottlenecks at the client's instance, our clients mimic streaming workload that processes data as it arrives without retaining it in memory or writing it to disk.

Regarding the benchmark workload, we vary the size of the data fetched to understand the latency vs. throughput bottlenecks of the storage service. We vary the number of simultaneous requests to obtain maximum achievable throughput as well as measuring performance at scale. We vary the size of the working sets to observe both in- and out-of-cache performance. Because performance will be impacted by load on the client and in the network, we repeat each experiment at different times across different locations to get representative results. We also study the impact of the different client implementations described above.

Response time. The response time for an operation is the time from when the client instance begins the operation to when the last byte reaches the client.

Throughput. The throughput for an operation is the maximum rate that a client instance obtains from the storage service.

Time to Consistency. We implement a simple test to estimate the time to consistency. We first write an object to a storage service (the object can be a row in a table, a blob, or a message in a queue). We then repeatedly read the object and measure how long it takes before the read returns correct result.

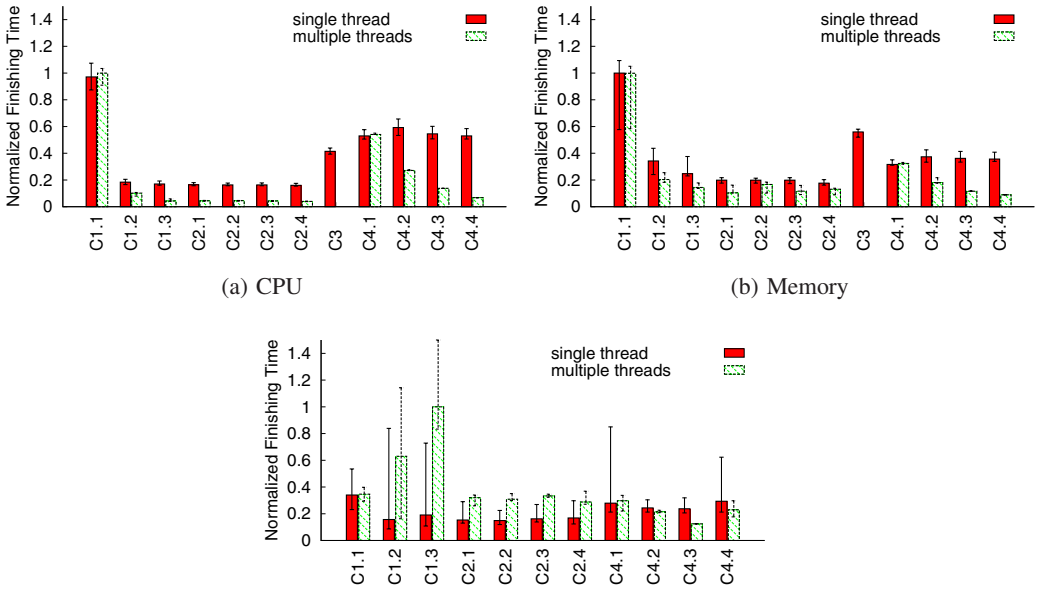
Cost per operation. Similar to cost per benchmark task, we use the published prices and billing APIs to obtain the cost per storage operation.

4.3 Network Metrics

We use standard tools such as *iperf* [8] and *ping* to measure the network throughput and path latency of a provider. To measure intra-cloud throughput and latency, we allocate a pair of instances (in the same or different data centers), and run those tools between the two instances. Some providers further divide instances within the same data center into zones for a finer-grained control of instance location (e.g., not placing all instances in the same failure domain). In this case, we also deploy inter-zone and intra-zone instances respectively to measure their throughput and latency.

To prevent TCP throughput from being bottlenecked by flow control, we control the sizes of the TCP send and receive windows. Our measurements show that with a 16MB window, a single TCP flow is able to use up the available capacity along the paths measured in this paper. Larger window sizes do not result in higher throughput. For comparison, we also measure the throughput obtained by TCP clients that use the default window size configured by the instance's operating system.

To measure the optimal wide-area network latency, we instantiate an instance in each data center owned by the provider and ping these instances from over 200 vantage points on PlanetLab [13].



(a) CPU

(b) Memory

(c) Disk I/O

Figure 1: The finishing time of benchmark tasks on various cloud instances. The time values are normalized using the longest finishing time to emphasize each provider's relative performance. We show both single-threaded and multi-threaded results. The multi-threaded and I/O results are missing for C_3 because it does not support multi-threading or accessing the local disk.

For each vantage point, the optimal latency to a provider is the smallest RTT to a data center of the provider. AppEngine automatically replicates our application to multiple instances at different data centers. By querying the DNS name corresponding to our application from each of the PlanetLab vantage points, we collect the IP addresses of the instance that AppEngine's automatic mapping service maps the request from each vantage point. Each of these IP addresses, we conjecture, corresponds to the virtual IP address of the front-end load balancer at a data center. We then ping all of these IP addresses from each of the PlanetLab vantage points to identify the best mapping that AppEngine might have achieved.

5. RESULTS

In this section, we present the comparison results between the four providers: AWS, Azure, AppEngine, and CloudServers. Due to legal concerns, we anonymize the identities of the providers in our results, and refer to them as C_1 to C_4. The data was collected over a two-month period from March to May 2010. We have made the data available for download from the project website [4].

5.1 Elastic Compute Cluster

We first measure the computation performance of different types of instances offered by cloud providers. Our naming convention refers to instance types as $provider.i$ where i denotes the tier of service with lower numerals corresponding to lower cost and computational speed. For instance, $C_1.1$ is the cheapest and slowest instance type offered by C_1.

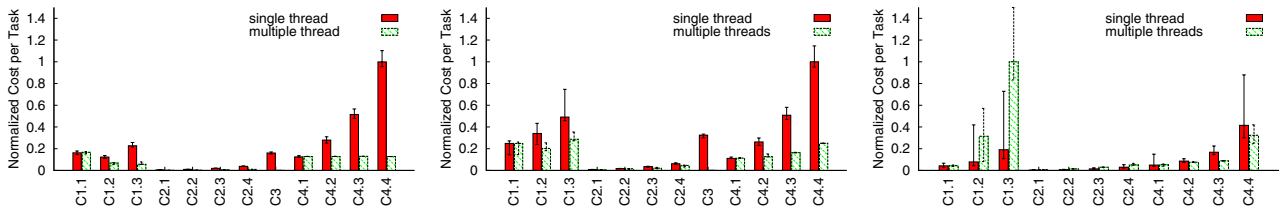
Table 3 summarizes the instances we measure. We test all instance types offered by C_2 and C_4, and the general-purpose instances from C_1. C_1 also provides specialized instances with more memory or CPU or high bandwidth network, which we do not include in this study, as they have no counterparts from other providers. C_3 does not offer different instance types, so we use its default environment to run the benchmark tasks.

Cloud Provider	Instance Type	Number of Cores	Price
C_1	$C_1.1$	< 1	\$0.085 / hr
	$C_1.2$	2	\$0.34 / hr
	$C_1.3$	4	\$0.68 / hr
C_2	$C_2.1$	4	\$0.015 / hr
	$C_2.2$	4	\$0.03 / hr
	$C_2.3$	4	\$0.06 / hr
	$C_2.4$	4	\$0.12 / hr
	$C_2.5$	4	\$0.24 / hr
	$C_2.6$	4	\$0.48 / hr
	$C_2.7$	4	\$0.96 / hr
C_3	default	N/A	\$0.10 / CPU hr
C_4	$C_4.1$	1	\$0.12 / hr
	$C_4.2$	2	\$0.24 / hr
	$C_4.3$	4	\$0.48 / hr
	$C_4.4$	8	\$0.96 / hr

Table 3: Information of the cloud instances we benchmark. With C_3, the first six CPU hours per day per application are free.

Some providers offer both Linux and Windows instances with the latter being slightly more expensive due to licensing fees. For experiments that depend on the type of OS, we compare instances from both OSes. For others, we choose Linux instances to reduce the cost of our experiment.

Figure 1 shows the finishing time of a CPU intensive task, a memory intensive task, and a disk I/O intensive task. Each bar shows the median and the 5th/95th percentiles of the measured samples. The same convention is used for all other figures without special notice. We omit the results of other benchmark tasks that show similar trends. For each instance type, we instantiate 10 instances and repeat each task 20 times per instance, i.e., a total of 200 samples per task per instance type. We only show the first four instance types of C_2 because the others have similar performance for the reason we soon describe. The I/O and multiple-threaded re-



(a) CPU (b) Memory (c) Disk I/O

Figure 2: The per-task monetary cost on each type of cloud instance.

sults are not available for C_3 because it does not support local disk access or multi-threading.

From the results, we can see that price-comparable instances offered by different providers have widely different CPU and memory performance. For example, C_4.1 and C_1.1 are in the same pricing tier with the former being only 30% more expensive per hour per instance, but twice as fast as the latter. Alternatively, C_1.2 and C_1.3 offer better performance than their counterparts from C_4 and are on average 50% more expensive.

Note that across providers the instance types appear to be constructed in different ways. For C_1, the high-end instances (C_1.2 and C_1.3) have shorter finishing times in both single and multiple threaded CPU/memory tests. This is perhaps due to two reasons. First, besides having more CPU cores, the high-end instances may have faster CPUs. Second, the low-end instance may suffer from higher resource contention, due to high load and poor resource multiplexing techniques (e.g., CPU time sharing) that open a tenant to interference from other colocated tenants.

In contrast, for C_4, the finishing times do not improve significantly for the single threaded tests when we alter the instances from low-end to high-end, while the amortized running times of the multi-threaded tests are greatly reduced. This suggests that all the C_4 instances might share the same type of physical CPU and they either have similar levels of resource contention or are better at avoiding interference.

Interestingly, instances of C_2 have the same performance regardless of their prices. This might be explained by the work-conserving CPU sharing policy of C_2, where a virtual instance can fully use all physical CPUs on a machine if there is no contention, and when colocated instances compete for CPUs, the high-end instances are given larger weight in the competition. Under such policy, we expect to observe interference and poor performance at times of high load. However, we found this to never happen in our experiments, suggesting that C_2's data centers were lightly loaded throughout our experiment period.

Unlike CPU and memory intensive tasks, the disk I/O intensive task exhibits high variation on some C_1 and C_4 instances, probably due to interference from other colocated instances [22]. Further, the multi-threaded I/O performance is worse than the single-threaded performance, perhaps because interleaved requests from multiple threads are harder to optimize than requests from the same thread. On the contrary, instances from C_2 are much more stable perhaps due to better I/O scheduling techniques or lightly loaded physical machines.

5.1.1 Performance at Cost

Figure 2 shows the monetary cost to run each task. We see that for single-threaded tests, the smallest instances of most providers are the most cost-effective compared to other instances of the same providers. The only exception is C_1.1, which is not as cost-effective as C_1.2, because the latter has much higher performance due to faster CPU or lower contention.

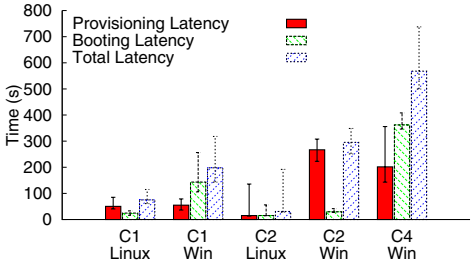


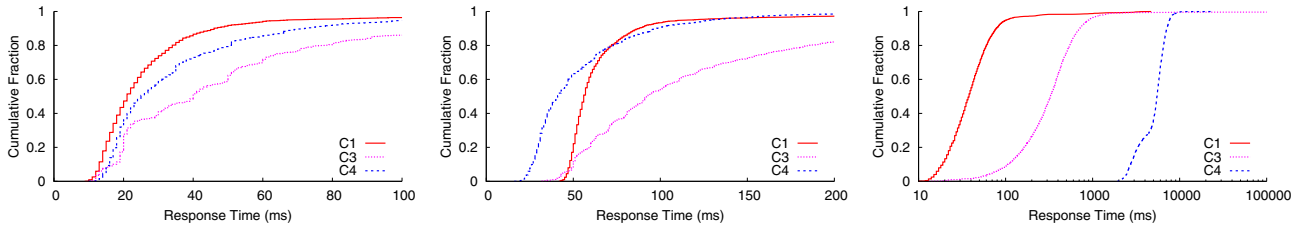
Figure 3: The scaling latencies of the lowest end instance of each cloud provider.

Surprisingly, for multi-threaded tests the high-end instances such as C_1.3 and C_4.4 with more CPU cores are not more cost-effective than the low-end ones. There are two possible reasons. First, the prices of high-end instances are proportional to the number of CPU cores, and thus do not provide any cost advantage per core. Second, although high-end instances are assigned more CPU cores, they still share other system resources such as memory bus and I/O bandwidth. Therefore, memory or I/O intensive applications do not gain much by using high-end instances as long as the applications do not run out of memory or disk space. This suggests that for parallel applications it might be more cost-effective to use more low-end instances rather than fewer high-end ones.

5.1.2 Scaling Latency

Finally, we compare the scaling latency of various providers' instances. To save cost, we only measure the scaling latency for the smallest instance of each provider. For providers that support both Linux and Windows instances, we test both choices to understand how different OSes affect the scaling latency, especially the booting latency. We run Ubuntu 9.04 for Linux instances and Windows Server 2008 for Windows ones. For each cloud and each OS type, we sequentially allocate 20 instances and measure the time between the request for a new instance and when that instance becomes reachable. We attribute the kernel uptime of the instance once it becomes available to be the time to boot and the remaining latency as the time to provision or otherwise set up the VM. We drop C_3 here because it does not allow manual requests for instances.

Figure 3 shows the scaling latencies for three clouds and different OS types. All cloud providers can allocate new instances quickly with the average scaling latency below 10 minutes. C_1 and C_2 can even achieve latency within 100 seconds for Linux instances. The latency of C_4 is larger. We see that across providers, Windows instances appear to take longer time to create than Linux



(a) Get (b) Put (c) Query

Figure 4: The cumulative distribution of the response time when using the large table with 100K entries. Note that for the query operation, the x-axis is in a logarithmic scale, due to the significant performance gaps between different services.

ones, but for different reasons. For C_1, the provisioning latency is similar for both instances, but the Windows ones have larger booting latency, possibly due to slower CPUs of the smallest instances. Windows Server 2008 R2 instances based on the Win7 code base can boot faster. For C_2, the booting latency is similar, while the provisioning latency of the Windows instances is much larger. It is unclear but likely that C_2 may have different infrastructures to provision Linux and Windows instances.

We note that a few other factors impacting scaling agility have not been considered here. Rather than focusing just on allocating instances quickly, some providers make it easy to update active instances or provide extensive monitoring and automatic recovery from faults during running. Another common goal is to create instances consistently within an SLA deadline even when they are brought up in large batches.

5.2 Persistent Storage

We measure and compare three types of storage services: table, blob, and queue, offered by various cloud providers.

5.2.1 Table Storage

We first compare three table storage services offered by C_1, C_3, and C_4. C_2 does not provide a table service. Here we only show the results obtained using our Java-based client, as other non-Java clients achieve similar performance, because the table operations are lightweight on the client side. For each table service, we test the performance of three operations: `get`, `put`, and `query` (see Table 2). Each operation runs against two pre-defined data tables: a small one with 1K entries, and a large one with 100K entries. The `get/put` operations operate on one table entry, while the `query` operation returns on average 10 entries. For storage benchmarks, unless otherwise specified, we use instance types that occupy at least one physical core to avoid excessive variation due to CPU time sharing.

Figure 4 shows the distributions of response time for each type of operation on the large table. We repeat each operation several hundred times. The results of using the small table show similar trends and are omitted to save space. From the figure, we can see that all table services exhibit high variations in response time. For example, the median response time of the `get` operation is less than 50ms, while the 95th-percentile is over 100ms for all services. The three services perform similarly for both `get` and `put` operations, with C_3 slightly slower than the other two providers. However, for the `query` operation, C_1's service has significantly shorter response time than the other two. C_4's service has a very long response time, because unlike the other providers, it does not appear to maintain indexes over the non-key fields in a table. In contrast, C_1 appears to have an indexing strategy that is better than the others.

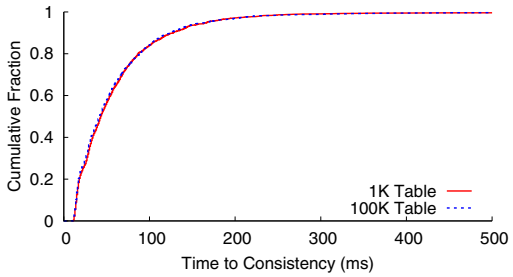


Figure 5: The cumulative distribution of the time to consistency for C_1's table service.

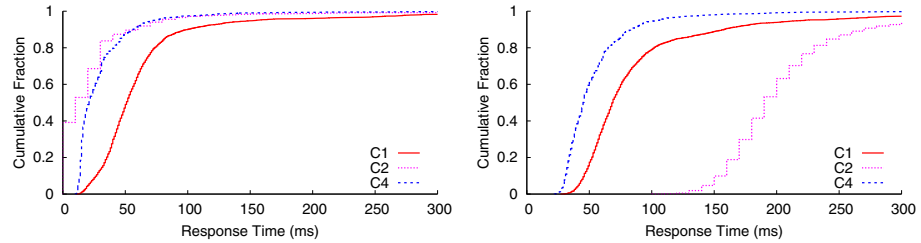
Provider	Cost per Operation (milli-cents)		
	get	put	query
C_1	0.13	0.31	1.47
C_3	0.02	0.23	0.29
C_4	0.10	0.10	0.10

Table 4: The average cost per operation for all three table services. The cost is in the unit of milli-cent, i.e., one-thousandth of a cent.

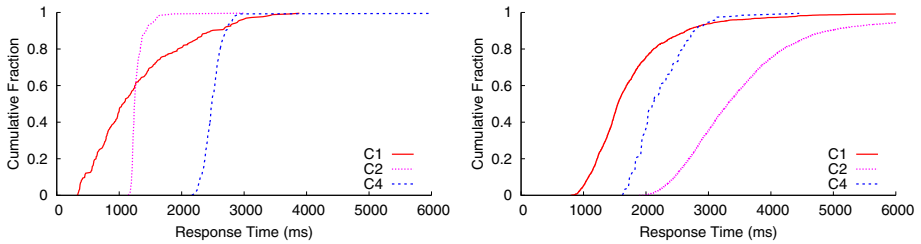
We also measure how well each table service scales by launching multiple concurrent operations. None of the services show noticeable performance degradation when up to 32 operations are issued at the same time. This suggests that all these table storage services appear to be reasonably well provisioned. Testing at higher scale is deferred to future work.

We then evaluate the time to reach consistency for the table services by using the mechanism described in §4.2. We discover that around 40% of the `get` operations in C_1 see inconsistency when triggered right after a `put`, and do not return the entry just inserted by the `put`. Other providers exhibit no such inconsistency. Figure 5 shows the distribution of the time to reach consistency for C_1. From the figure, we see that over 99% of the inconsistencies are resolved within 500ms, with the median resolved within 80ms. The long duration of the inconsistency opens up a sizable window for race conditions. C_1 does provide an API option to request strong consistency but disables it by default. We confirm that turning on this option eliminates inconsistencies and surprisingly does so without much extra latency for `get` or `put`. We conjecture that the performance overhead of enforcing consistency might be visible only when more users request it or during failure cases because otherwise C_1 would have enabled it by default.

Finally, we compare the cost per operation for different table services in Table 4. We can see that, although the charging models of the providers are different, the costs are comparable. Both C_1 and C_3 charge lower cost for `get/put` than `query`, because the former



(a) Download 1KB Blob (b) Upload 1KB Blob



(c) Download 10MB Blob (d) Upload 10MB Blob

Figure 6: The cumulative distribution of the response time to download or upload a blob using Java-based clients.

operations are simpler and consume less CPU cycles to serve. C_4 charges the same across operations and can improve its charging model by accounting for the complexity of the operation. This is an example of how CloudCmp's benchmarking results can help the providers make better choices.

A comparison with the compute costs in Table 3 indicates that the cost of table storage is comparable to that of compute instances for workloads that trigger about 1000 operations per instance per hour. Applications that use storage at a lower rate can choose their provider based on their computation costs or performance.

5.2.2 *Blob Storage*

We compare the blob storage services provided by C_1, C_2, and C_4. C_3 does not offer a blob store.

Figure 6 shows the response time distributions for uploading and downloading one blob measured by our Java-based clients. We consider two blob sizes, 1KB and 10MB, to measure both latency and throughput of the blob store. The system clock of C_2's instances have a resolution of 10ms, and thus its response time is rounded to multiples of 10ms. We see that the performance of blob services depends on the blob size. When the blob is small, C_4 has the best performance among the three providers. When the blob is large, C_1's average performance is better than the others. This is because blobs of different sizes may stress different bottlenecks – the latency for small blobs can be dominated by one-off costs whereas that for large blobs can be determined by service throughput, network bandwidth, or client-side contention. Uniquely, C_2's storage service exhibits a 2X lower performance for uploads compared to downloads from the blob store. It appears that C_2's store may be tuned for read heavy workload.

Figure 7 illustrates the time to download a 10MB blob measured by non-Java clients. Compared to Figure 6(c), in every provider, non-Java clients perform much better. Markedly C_4's performance improves by nearly 5 times because it turns out that the Java implementation of their API is particularly inefficient.

We then compare the scalability of the blob services by sending multiple concurrent operations. As per above, we use non-Java clients to eliminate overheads due to Java. Figure 8 shows the

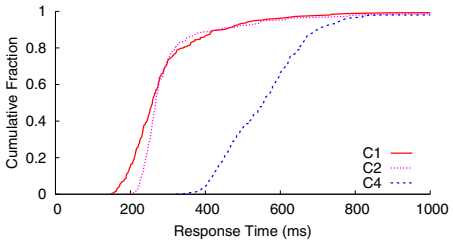


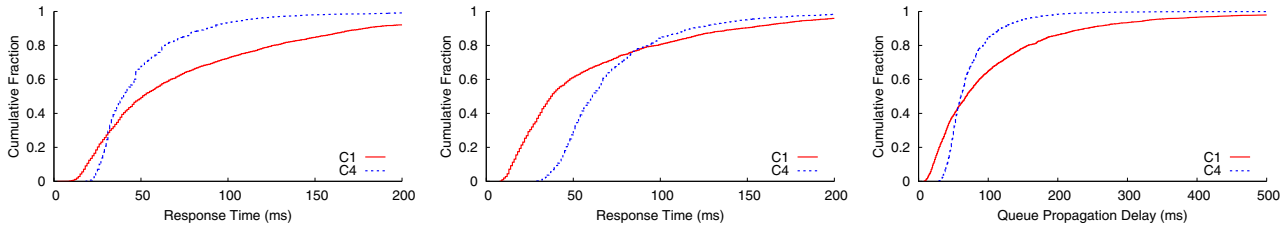
Figure 7: The cumulative distribution of the time to download a 10MB blob using non-Java clients.

Provider	Maximum throughput achieved by one instance (Mbps)	
	Smallest instance with at least one core	Largest instance from a provider
C_1	773.4	782.3
C_2	235.5	265.7
C_4	327.2	763.3

Table 5: The maximum throughput an instance obtains from each blob service when downloading many 10MB blobs concurrently.

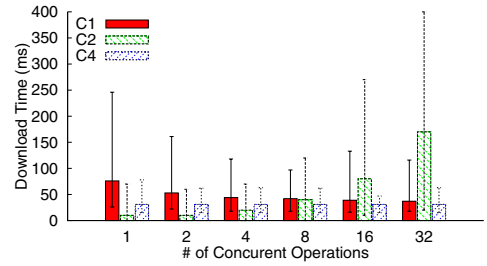
downloading time with the number of concurrent operations ranging from 1 to 32. We omit the results for uploading because they are similar in trend. When the blob size is small, all services except for C_2 show good scaling performance. This suggests that C_1 and C_4's blob services are well-provisioned to handle many concurrent operations. When the blob is large, C_4 and C_1 continue to scale better though all three providers show certain scaling bottlenecks. The average time to download increases with the number of concurrent operations, illustrating a throughput bottleneck.

We compute the maximum throughput of the blob service that one instance can obtain by increasing the number of simultaneous operations until the throughput stabilizes. Table 5 shows the results for two types of instances – the smallest instance that has at least one full core and the largest instance from each provider. The former data point eliminates CPU time sharing effects [35] while the latter minimizes contention from colocated VMs. We report

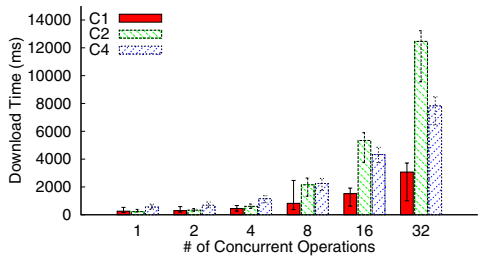


(a) Send (b) Retrieve (c) Propagation delay

Figure 9: The cumulative distributions of the response time to send or retrieve a message from a queue, and the propagation delay of a queue.



(a) 1KB Blob



(b) 10MB Blob

Figure 8: The blob downloading time from each blob service under multiple concurrent operations. The number of concurrent requests ranges from 1 to 32. Note that the x-axes are on a logarithmic scale.

results with non-Java clients to eliminate overheads due to client side inefficiency. As we will soon show, C_1 and C_2's blob service throughput is close to their intra-datacenter network bandwidth (see Figure 10), suggesting that the bottleneck in throughput is unlikely within the blob service itself. This is also true for C_4's blob service throughput of a large instance, which more than doubles that of the single core instance that we were using earlier. It appears that the impact on an instance's throughput due to other VMs colocated on the machine is non trivial in C_4.

In summary, we observe that client implementation and contention from other VMs or along network paths significantly impact the perceived storage service performance (across the three cloud platforms we study). Therefore, a more efficient client implementation or less contention may improve the results in this paper.

We tested the consistency property of the blob services, and did not find inconsistency problems in the three providers. The charging models are similar for all three providers and are based on the number of operations and the size of the blob. We omit these results for brevity.

5.2.3 *Queue Storage*

We compare the queue services of C_1 and C_4. Similar to table services, we only show the results from our Java-based client.

Cloud Provider	Data Center Name	Location	Region
C_1	$C_1.DC_1$	North Virginia	US
	$C_1.DC_2$	North California	US
	$C_1.DC_3$	Ireland	Europe
C_2	$C_2.DC_1$	Dallas/Fort Worth, Texas	US
	$C_2.DC_2$	Chicago, Illinois	US
C_4	$C_4.DC_1$	Chicago, Illinois	US
	$C_4.DC_2$	Amsterdam, Netherlands	Europe
	$C_4.DC_3$	San Antonio, Texas	US
	$C_4.DC_4$	Singapore	Asia
	$C_4.DC_5$	Dublin, Ireland	Europe
	$C_4.DC_6$	Hong Kong	Asia

Table 6: The geographical location of the cloud data centers.

Figure 9(a) and 9(b) show the distributions of the response time of sending and retrieving a message. The queue services are designed to transfer only small messages up to 8KB. Thus, we choose a message size of 50B in our measurement. The results show that both services have large variations in response time. C_4 is slightly faster at sending messages while C_1 is faster at retrieving messages. We test the scalability of the queue services, up to 32 concurrent messages, and no significant performance degradation is found, mostly due to the small message size.

It is interesting to note that the response time of the queue service is on the same order of magnitude as that of the table and blob services. This suggests that although the queue service is simpler and designed to be more efficient compared to the other storage services, the performance gain is insignificant. One may use the table or blob service to implement a simple signaling framework with similar functionality as the queue service without much performance degradation.

We then measure the propagation delay of a queue. The delay is defined as the time between when a message is sent to an empty queue and when it is available to be retrieved. A queue with shorter propagation delay can improve the responsiveness of an application. Figure 9(c) shows the distribution of the propagation delay of the two services. We see that roughly 20% of the messages for C_1 take a long time (>200ms) to propagate through the queue, while C_4's queue service has a similar median propagation delay but lower variation. Finally, both services charge similarly–1 cent per 10K operations.

5.3 Intra-cloud Network

In this section, we compare the intra-cloud network performance of different providers. As of April 27th, when we conducted this measurement, we found a total of 11 data center locations from three providers: 6 for C_4, 3 for C_1, and 2 for C_2. Similar to instance types, we name each data center as provider.DC_i. Table 6 summarizes the geographic locations of the data centers. We do not

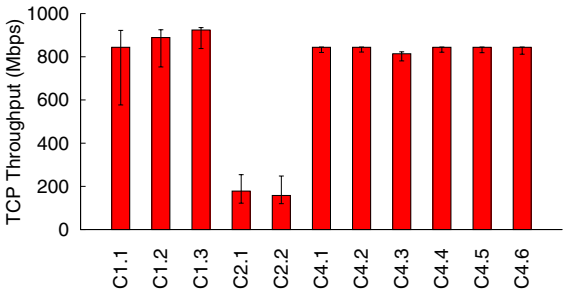


Figure 10: The intra-datacenter TCP throughput between two instances in all data centers we measure.

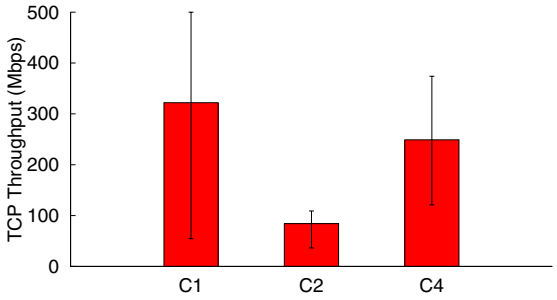


Figure 11: The TCP throughput between two different US data centers of a cloud provider.

consider C_3's intra-cloud network performance because it does not allow direct communication between instances.

When measuring the intra-cloud bandwidth and latency, we choose the instance types that can at least fully occupy one CPU core. This is to avoid the network performance degradation due to CPU time sharing introduced by virtualization [35].

5.3.1 Intra-datacenter Network

Figure 10 shows the TCP throughput between two instances in the same data center. In rare cases, a pair of instances are colocated on the same machine and obtain a throughput larger than 1Gbps which is the speed of the NIC. We filter out such pairs as they do not measure actual network performance. We combine the results for intra-zone and inter-zone cases, because the difference between them is not significant. From the figure, we see that the network bandwidth of providers differs significantly. C_1 and C_4 provide very high TCP throughput which is close to the limit of the NIC (1Gbps) and the variation is low. C_2 has much lower throughput, probably due to throttling or under-provisioned network.

In terms of latency, all data centers achieve low round trip time ($<$ 2ms) for all pairs of instances we test. This result is not surprising, because the instances located in the same data center are physically proximate. We omit the results to save space.

5.3.2 Inter-datacenter Network

Next, we show the performance of network paths between data centers of the same provider. Because most providers focus on the US market (and some only have US data centers), we only show the results for data centers within the US. Figure 11 shows that the throughput across datacenters is much smaller than that within the datacenter for all providers. Both C_1 and C_4 have their median inter-datacenter TCP throughput higher than 200Mbps, while

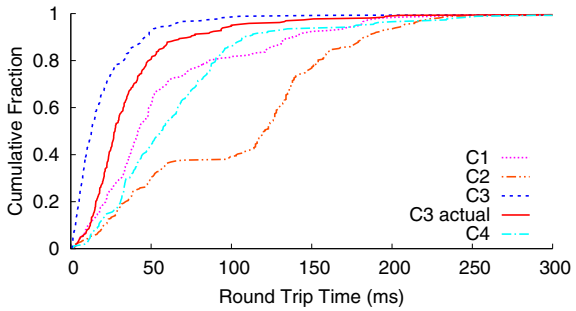


Figure 12: This figure shows the cumulative distribution of the optimal round trip time (RTT) to the instances deployed on a cloud provider from 260 global vantage points. For C_3 we also show the actual RTT from a vantage point to the instance returned by the cloud's DNS load balancing.

C_2's throughput is much lower. Further, the variation in throughput across datacenters is higher since the wide-area traffic may have to compete with much other traffic.

We also compare the above result with the inter-datacenter throughput obtained by a vanilla client that uses one TCP flow with the default send and receive window sizes configured by the provider. All providers see a smaller throughput with this vanilla client. This is because the network paths across data centers have a high bandwidth-delay product (*e.g.*, 50ms $\times$ 800Mbps = 5MB) and the default window sizes are not configured appropriately. We find that the degradation is less with Linux instances, since modern Linux kernels auto-tune the size of the TCP receive buffer [11] which can grow up to 4MB in the default setting. However the degradation is far worse in the non-Linux instances since either auto-tuning is not turned on or is configured poorly.

We find that the latencies between data centers largely correspond to the geographical distance between the data centers. The latencies across providers are incomparable because their data centers are at different locations. Hence, we omit these results.

5.4 Wide-area Network

In this section, we compare the wide area network performance of different providers. Figure 12 shows the distribution of the optimal wide-area latency observed from a diverse set of vantage points. We also show the actual latency of C_3 which is the latency to the instance returned by its DNS load balancing system. From the figure, we can see that both the optimal and actual latency distributions of C_3 are lower than that of other providers. This could be explained by the provider's widely dispersed presence – we observe 48 unique IP addresses simultaneously serving our test application. These IP addresses likely correspond to the virtual IPs of front-end load balancers at distinct locations. Furthermore, the gap between the optimal and actual latency of C_3 is less than 20ms, suggesting that its load balancing algorithm works very well in practice.

C_1 and C_4 have similar latency distributions: they are worse than C_3, but much better than C_2. A main difference is that C_1 has a larger fraction of vantage points that have an optimal latency higher than 100ms. Closer examination of our data reveals that these high latency vantage points are mostly in Asia and South America, where C_1 does not have a presence (Table 6).

C_2 has the worst latency distribution because it has the smallest number of data centers. The flat curve between 50ms and 100ms corresponds to the latency differences between two groups of vantage points: the North American nodes and those in other conti-