An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS

An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS

Simson L. Garfinkel

TR-08-07

Computer Science Group
Harvard University
Cambridge, Massachusetts

Technical Report TR-08-07: An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS

Simson L. Garfinkel
Center for Research on Computation and Society
School for Engineering and Applied Sciences
Harvard University
Cambridge, MA
simsong@acm.org

Abstract

Amazon.com's Elastic Compute Cloud (EC2), Simple Storage Service (S3) and Simple Queue Service (SQS) offer enterprise-class computing, storage and coordination facilities to any organization or individual in the world with a valid credit card. This paper details our experience working with these commodity grid computing services between November 2006 and May 2007, including an analysis of the overall system's API and ease-of-use; an analysis of EC2's management and security facilities; an end-to-end performance analysis of S3's throughput and latency as observed from Amazon's EC2 cluster and other locations on the Internet; and an analysis of the SQS operation and performance. We conclude with a report of our experience moving a large-scale research application from dedicated hardware to the Amazon offering. We find that this collection of Amazon Web Services (AWS) has great promise but are hobbled by service consistency problems, the lack of a Service Level Agreement (SLA), and a problematic Web Services Licensing Agreement (WSLA).

1 Introduction

In 2006, Amazon Web Services (AWS), a subsidiary of Amazon.com, began offering three web services that allow organizations and individuals to use Amazon's enterprise-class computing infrastructure on an as-needed basis and at commodity prices. Amazon's Elastic Compute Cloud (EC2) rents Linux virtual machines at 10 cents per CPU hour; users may rent dozens, hundreds, or even thousands of CPUs simultaneously. Amazon's Simple Storage Service (S3) allows users to store data at a cost of 15 cents per gigabyte per month, with bandwidth priced at between 10 and 18 cents per gigabyte transferred outside Amazon's network. Amazon's Simple Queue Service (SQS) is a reliable messaging service, making it relatively straightforward to coordinate a cluster of computers on a parallelizable large-scale problem, at a cost of 10 cents per thousand messages. All of these services are sold without startup fees and with no minimum pricing, potentially making them attractive for individuals, universities and corporations alike.

1.1 Utility Computing with AWS

Businesses, universities and government users may consider cluster computing as a potential solution to their computing needs. But building and operating even a relatively small cluster can be a formidable undertaking, requiring not just money but physical space, cooling, power, and management resources. Planning and building a cluster requires considerable investment. Once built, it can be a challenge to keep the cluster both sufficiently utilized to justify the expense and sufficiently under-utilized so that there is room for existing projects to grow their resource requirements as necessary.

Utility computing has been proposed as an alternative to this conundrum. In the utility computing model, a large organization (the "utility") builds a large infrastructure and then rents out computation, storage and bandwidth on an as-needed basis, just as power and water utilities rent the use of their infrastructure. Utility computing promises to cut costs for all but the largest users because the utility can achieve dramatic savings through economies of scale.

Amazon's S3, EC2 and SQS services are among the first utility computing services to be offered at a large scale. "Success Stories" on Amazon's website profile customers using the S3 service: Webmail.us uses S3 to host 350,000 paid mailboxes with 10 gigabytes of mail per user; YouOS uses S3 to allow 60,000 customers to store 12 GB of storage for its collaborative web operating system; SmugMug uses S3 to store 10 terabytes of new customer images each month.[2]

Amazon says its services are designed to be scalable, reliable, fast, inexpensive, and simple. In par-

ticular, Amazon writes that S3 can "support an unlimited number of web-scale applications;" that stored data has "99.99% availability;" and that the system is "fast enough to support high-performance applications" with "server-side latency [that is] insignificant relative to Internet latency."[24] But Amazon has declined to provide specific information about its architecture and its the limits of its scalability. There is no Service Level Agreement (SLA).

This paper presents an original evaluation of EC2, S3 and SQS based on a series of experiments conducted between November 2006 and May 2007, as well as our personal experience in moving a large scale computer forensics project to the Amazon system. We also present an analysis of the AWS security model and terms in the AWS License Agreement that could be used to justify a service interruption. Overall, we find that Amazon largely delivers on its availability promises, although performance can suffer in unpredictable ways. Nevertheless, the combination of EC2, S3 and SQS appears to be a stable, reliable alternative for universities and other organizations considering the creation of their own clusters.

Section 2 of this paper presents related work. Section 3 describes the AWS user-visible architecture, its APIs, and analyzes a variety of security implications. Section 4 describes our experience with EC2; Section 5 describes our experience with S3; and Section 6 describes our experience with SQS. Section 8 concludes.

2 Related Work

Because the quality of a single commercial Internet service can literally affect hundreds of millions of individuals, there is a small but important academic tradition that measures such services.

Paxson conducted a large-scale study of end-to-end routing behavior on the Internet.[17] Paxson's study is significantly different from ours, in that Paxson studied performance between many different end points, whereas we have confined our study to a single service studied from a small number of locations. However, Paxson noted the advantage conducting measurements according to a Poisson distribution of delays between measurements: any sampling of the measurements will have the same statistical properties as the whole. We follow Paxson's sampling methodology for the measurements reported in this paper.

Pang *et al.* performed a large-scale study of DNS availability, usage and deployment characteristics [16]. They found that DNS servers were highly available and that there was a large amount of diversity in the way that organizations deployed DNS servers. We similarly found that the availability of the domains used by Amazon for S3 to be high. Changes in the DNS name resolutions appear to be the result of intentional decisions made by Amazon, and not the result of system failures.

Strauss *et al.* examined bandwidth estimation tools and introduced a new tool (Spruce), that can accurately estimate available bandwidth using a small amount of probe traffic [21]. Instead of using a bandwidth estimation tool such as Spruce, we directly measure the throughput that S3 can deliver.

EC2 is similar to PlanetLab[4], the global research overlay network, in that both provide access to distributed computing services. But EC2 is different from PlanetLab in several important ways:

- EC2's servers are housed in a relatively small number of clusters, while PlanetLab's servers are geographically distributed all over the world.
- EC2 is paired with rapid interconnect to S3, which allows the servers to store significantly more data than possible with PlanetLab.
- PlanetLab is only available to individuals at institutions that have contributed resources to PlanetLab, while EC2 is available to anyone with a functioning credit card.
- PlanetLab does not give users root access to cluster "slices."
- Planetlab is designed for experiments of limited duration, and slices must be continually renewed by the experimenter. Users who use large amounts of computing or network resources risk being shut down for abusing the service.

3 Understanding S3 and EC2

This section examines the AWS API and discusses several limitations that we have experienced.

3.1 The AWS Interfaces

There are three primary interfaces to Amazon Web Services:

- Customers create their Amazon Web Services account and sign up for individual services using a web-based dashboard. Access to the dashboard requires an email address and Amazon.com password.
- Specific functions on EC2, S3 and SQS such as starting up virtual machines on EC2 and accessing data on S3 can be performed using standard HTTP `GET`, `PUT` and `DELETE` commands sent over HTTP or HTTPS, a so-called "REST"[5] API.

- Remote procedure calls can also be executed using a traditional SOAP[22] API.

Amazon distributes a variety of code libraries that make use of these APIs.

3.2 AWS Security

AWS supports two different schemes for identification and authorization. One uses a 40-character secret key and the HMAC[14] algorithm to sign each request, the second is based on X.509 certificates. Amazon makes each user's secret key and X.509 keypair and allows them to be downloaded from the dashboard. Access to the dashboard is granted to anyone with an Amazon username (an email address) and a password. AWS also users may also upload their own X.509 certificates that they may have from other sources; these certificates are then trusted.

Because all authentication credentials can be downloaded from the AWS web dashboard, any individual with access to an account's password has full access to all of the resources of that account. For example, an attacker with a stolen password can delete all of the information that the account has stored, or even disable the service entirely. Because Amazon allows lost passwords to be reset by email, any individual who controls an email system used by an AWS account or who can passively eavesdrop on the network through which a password-reset email would pass can effectively seize control of an AWS account. (See Garfinkel[8] for a discussion of the potential risks of E-Mail Based Identification and Authentication.)

None of the AWS services provide for snapshots, versioning of user data or backups. Instead, users that require redundancy are encouraged to develop their own systems—for example, running mission-critical services on multiple EC2 machines and storing mission-critical data in multiple S3, controlling this infrastructure with multiple AWS accounts, and making sure that each AWS is linked to an email address at a different email provider. (Ironically, one of the most celebrated cases of network eavesdropping involved mail messages sent from Amazon to its customers that both ran an Internet email system and sold books in competition with Amazon. The so-called Councilman Case ([6]) demonstrates that unencrypted email-based communications are vulnerable in practice, and not just in theory.)

The AWS protocols use signed timestamps to prevent replay attacks. SSL provides the only realistic defense against a man-in-the-middle attack: instead of connecting the port 80 using HTTP, the client can elect to connect to the server on port 443 using TLS. AWS secures authenticates its TLS server using a 1024 bit RSA certificate signed by VeriSign.

Amazon does not guarantee the privacy of information stored in S3, but instead advises organizations requiring security to encrypt their data. We go further: Given Amazon's weak security model, users should employ some kind of data authentication technology to assure themselves that data returned by S3 is the same as the data that was stored there. Technology such as an HMAC or a digital signature would protect users from both accidental data modification by Amazon and from malicious modification by third parties who managed to crack a password or intercept a password-reset email message.

3.3 AWS and DNS

Based on our testing and on comments that Amazon has made in its user forums, it is clear that Amazon uses multiple data centers to provide its EC2, S3 and SQS services. Each data center has one or more external IP addresses which are advertised in the Amazon DNS. Amazon warns developers that DNS resolutions should not be cached for an extended period of time: "Amazon cannot guarantee the validity of an IP address at any given time. It is necessary to change IP addresses from time to time for a variety of reasons, including hardware changes. Amazon does, however, guarantee that the URL resource name for a given service will not change without public notification."[1]

The IP addresses connect to load balancers which then distribute requests with Amazon's infrastructure. We recorded DNS resolutions and saw a variety of different IP addresses being made visible at different times to different hosts at different locations on the Internet. Although we think that it is likely that Amazon is using DNS for some form of load balancing in addition to its load balancers, we were unable to discern any pattern.

During the course of our study we saw two problems that could be attributed to the load balances. In November 2006 an apparent load balancer failure caused S3 to be unavailable to many users on the Internet; the problem was corrected with Amazon removed the load balancer's IP address from the `s3.amazonaws.com` DNS announcement. (A separate ongoing problem with the load balancers causes them to terminate any TCP/IP connection that contains more than 2^{31} bytes. This means that objects larger than 2GB must be stored to S3 in several individual transactions, with each of those transactions refering to different byte ranges of the same object.)

3.4 EC2 Virtualization and Tools

EC2's servers are Linux-based virtual machines running on top of the Xen virtualization engine[25]. The virtual machines "predictably provides the equivalent of a system with a 1.7Ghz x86 processor, 1.75GB of

RAM, 160GB of local disk, and 250Mb/s of network bandwidth."[23] Amazon calls these virtual machines "instances." Instances are billed at 10 cents per instance hour, irrespective of the load placed on the CPU.

EC2 instances boot from an "Amazon Machine Image" (AMI) that is digitally signed and stored in S3. Amazon provides a number of base images that can be used as-is or customized by the user and then saved in S3. Users may also create their own images from scratch, although all EC2 images must use the Linux 2.6 kernel.

Amazon has created a collection of programs written in Java that control EC2 through SOAP commands sent over SSL to computers at Amazon. The tools can list available AMI images, boot an instance from an image, display the instances that the user is running, display console output, terminate an instance, etc.

EC2 instances have two access control mechanisms. Amazon's EC2 tools can write an ssh public key into the AMI image, allowing a user who has the matching private key to log into the superuser account. Amazon also provides a firewall that can be used to restrict network connectivity to a user's instances based on IP address and TCP port number.

3.5 The S3 Storage Model

Amazon's Simple Storage Service (S3) stores data as named "objects" that are grouped in named "buckets." Buckets must be explicitly created before they can be used; each user may create up to 100 buckets. Bucket names are global; an S3 user attempting to create a bucket named "foo" will be told that the bucket name is already in use. The S3 API allows buckets to be created, deleted, and listed. Each bucket has an access control list allowing read or read/write permission to be given to other specific AWS users or to the world.

Objects may contain any byte string between 1 and 5 GBytes. Object names are essentially URI [3] pathnames. The API allows objects to be stored and retrieved in their entirety or by specific byte ranges. For each object S3 maintains a name, modification time, an access control list, and up to 4 Kbytes of user-defined metadata. Amazon has stated that S3 is designed to store large objects [18]. Our testing confirms that S3 delivers dramatically faster throughput on large objects than small ones due to a high per-transaction overhead.

3.5.1 Retries and Errors

S3 is designed to quickly fail requests that encounter problems; it is the client's responsibility to retry failed requests until they succeed. This is different than traditional web servers, which implement a "best effort" policy for satisfying web requests.

Two kinds of write failures can occur when data is being stored to S3:

- The service can fail when an HTTP PUT is issued, forcing a retry. We call this a Write Retry.
- S3 computes the MD5 of every object that is written and returns the MD5 as part of the write acknowledgment in a special field called `ETag`. Clients are supposed to compute the MD5 of objects that they write and compare this MD5 with the returned `ETag`. If the two values do not match, then the object was corrupted during transmission or storage. In this case, the object should be re-written. We call this a Write Error.

  Although we have tested for write errors, in our experience we have never seen one.

Two kinds of failures can occur when data is read from S3:

- Instead of returning the requested data, S3 can generate some kind of error, forcing a retry. We call this a Read Retry.
- When S3 returns a data object, the MD5 of the object is returned in the `ETag` field. If the MD5 does not match the original value stored, a Read Error has taken place.

  Although we have tested for read errors, in our experience we have never seen one.

Because web browsers do not automatically retry their requests, web content served directly from S3 may fail to load or appear as "broken" images more often than content served from traditional web servers. A two-tiered system involving a caching proxy on EC2, backed by storage on S3 would overcome this problem. This is not to say that S3 is less reliable than traditional web servers: it simply implements a retry policy which is not strictly compatible with today's web browsers.

3.5.2 API Limitations

S3 supports PUT, GET, and DELETE primitives, but there is no way to copy or rename an object, move an object to a different bucket, or change an object's ownership. Although these primitives can be implemented by combining PUT, GET and DELETE operations, it can take days to move a few terabytes of data from one bucket to another using successive PUTs and GETs. Furthermore, moving data via this strategy can result in significant data transfer charges unless the GETs and PUTs are done from EC2. The EC2 option was not available to many developers when this paper was written, as EC2 was in a period of "limited beta."

Currently, S3 only supports writing using SOAP or the HTTP PUT command. Users in Amazon's developer forums have repeatedly requested the ability to upload data using a standard HTTP POST, but, as of this writing, S3 does not offer this feature.

3.6 The SQS Storage Model

Amazon's Simple Queue Service (SQS) allows users to create one or more named queues. SQS supports three basic operations. A named message consisting of up to 256K of data and 4K of metadata can be written into a queue; one or more messages can be read from a queue; and one or more named messages can be deleted from a queue.

When a message is read from SQS the reading process specifies a time lock. While the message is locked, no other read request will return the message. The reading process must delete the message before its time lock expires, otherwise another concurrent process may read the message instead.

Despite the name "queue," messages stored in SQS appear to be only loosely ordered.

3.7 Pricing

Amazon charges separately for computer resources consumed and for bandwidth. The pricing model underwent a significant change on June 1st, 2007. This section presents both pricing models.

Amazon's pricing for EC2 is 10 cents per hour for each instance, with fractional hours rounded up. Instances must be shut down with the ec2-terminate-instances command. Instances that have crashed and not automatically reboot continue to acquire charges.

Storage for S3 is charged on a flat basis of 15 cents per gigabyte stored per month, with the amount of data stored being calculated twice each day. Starting June 1st Amazon has also charged a per-transaction fee of 1 cent for every 1,000 PUT or LIST requests, and 1 cent for every 10,000 GET requests. There is no charge for deleting objects.

Use of SQS is charged at 10 cents for every thousand messages sent.

Originally pricing for bandwidth for the AWS services was charged on a flat basis of 20 cents per gigabyte transferred in or out of the Amazon network. Under the new pricing model Amazon charges 10 cents per gigabyte sent from the Internet to Amazon. Bandwidth from Amazon is charged at 18 cents per gigabyte transferred for the first 10 TB of data transferred per month, 16 cents per gigabyte for the next 40 TB transferred, and 13 cents for each gigabyte transferred thereafter. There is no charge for moving data between EC2, S3 and SQS.

3.8 Legal Issues

All uses of Amazon Web Services are covered by the Amazon Web Services Licensing Agreement (WSLA) [19]. Although an in-depth analysis of the agreement is beyond the scope of this paper, the agreement has several clauses that have direct bearing on the issue of service availability.

Specifically, Amazon's WSLA allows the company to terminate service to any web services customer at any time for any reason. The agreement further contains a covenant not to sue Amazon or its affiliates as the result of any damages that might arise out of the use of Amazon Web Services. In the absence of these rights, organizations wishing to protect themselves against a failure on the part of Amazon could rely on other techniques such as redundant backups, business continuation insurance, or even relying on multiple utility computing providers.

The specific language that gives Amazon these broad rights appears in three places in the WSLA:

- Section 1.A.7 prohibits the use of "other information obtained through Amazon Web Services for the purpose of direct marketing, spamming, contacting sellers or customers."

- Section 7.B.4.ii prohibits AWS from being used to store any content that is "obscene, libelous, defamatory or otherwise malicious or harmful to any person or entity."

- Section 7.B.4.iv prohibits S3 from being used "in any way that is otherwise illegal or promotes illegal activities, including without limitation in any manner that might be discriminatory based on race, sex, religion, nationality, disability, sexual orientation or age."

Amazon's spokesperson told us that these terms were to prohibit S3 from being used in ways that are illegal[10]. Nevertheless, many of the acts prohibited are not criminal in nature. For example, under US law, accusations of libel are resolved by litigation between private parties, and not through the criminal courts. To do otherwise would be a violation of the principle of freedom of expression. Discrimination based on sexual orientation is actually permitted under federal law, although Executive Order 13087 signed by President Clinton on May 28, 1998 reaffirmed "the Executive Branch's longstanding policy that prohibits discrimination based upon sexual orientation within Executive Branch civilian employment."[15] But this is an executive order, not a law, and it is not legally binding on Amazon.

4 Experience with EC2

We signed up for Amazon's EC2 "beta" program in September 2006. Over the following months we created and used multiple EC2 instances, we built our own AMI, and we used EC2 as a test bed for working with S3. During this time EC2 was in a "limited beta" and not available for general use. Our account was initially approved to create 10 simultaneous instances, although Amazon raised this limit to 100 when we made a request that included a reasonable justification.

We found EC2 instances to be fast, responsive, and very reliable. Using Amazon's provided `ec2-run-instances` program we could start a new instance in less than two minutes. During approximately one year of instance time we experienced one unscheduled reboot and one instance freeze. We lost no data during the reboot, but we were not able to recover any data from the virtual disks of the frozen instance.

All of our EC2 instances were allocated from one of two Amazon clusters: `usma1` and `usma2`. We do not know how many clusters Amazon operates, but there does not appear to be a corresponding `usma3` cluster as of this writing.

Each of our EC2 instances ran a Linux 2.6.16-xenU kernel with a minimal install. Linux reported that the host CPU was an "i686 athlon" with 980MB of high memory and 727MB of low memory. Except when our experiment was running, the instances all reported a load average of "0.00." Although our instances did not run `nntp`, their clocks were highly synchronized to UTC; in all likelihood, time synchronization was being performed on the host operating system and simply passed through to the virtual machine.

5 Evaluating S3

This section describes our performance testing of the S3 storage network.

Although Amazon distributes implementations of the S3 REST API in C#, Java, PHP, Perl Python and Ruby, Amazon does not distribute an implementation in either C or C++. To use S3 for an unrelated project[7] we wrote our own S3 REST client implementation in C++ using libcurl [20].

Using our S3 implementation we wrote a throughput testing program which measured the elapsed time to upload one, two or three same-sized objects to S3 using the `gettimeofday` system call. The objects were then deleted. Next the program downloaded one, two or three same-sized but different objects that had been previously loaded into S3, again measuring the elapsed time. Downloading different objects minimized the possible effects of caching: Each read request must be satisfied by reading the object from the disk, unless the object was still in the S3 cache from a previous invocation of the test program.

5.1 Throughput and TPS

We set up an S3 bucket explicitly for testing and stocked it with 15 objects, three each in sizes 1 byte, 1 Kbyte, 1 Mbyte, 16 Mbytes and 100 Mbytes.

Because our throughput testing program required that each transaction be successfully resolved before the next was initiated, the measured throughput for the 1-byte objects is a direct measurement of S3's transaction speed. That is, an observed throughput of 50 bytes/sec with 1-byte probes from a particular host means that users on that host could execute a maximum of 50 non-overlapping (*ie,* without the use of a pipeline) S3 transactions-per-second (TPS)

At the other end of the spectrum, our 100 Mbytes probes are direct measurements of the maximum data throughput that the S3 system can deliver to a single client thread.

We used our program to measure end-to-end performance under two scenarios. Our first scenario consisted of a series of successive probes from EC2 to S3. These probes were separated in time by a random delay where the length of the delay followed a Poisson distribution. Our second scenario performed repeated queries to the S3 service with no delay between queries. In this second scenario, we experimentally varied the number of threads on each CPU issuing queries every 10 minutes, with between 1 and 6 threads executing at any moment. We called this our "surge experiment."

5.2 The Distributed Testbed

In addition to our EC2 server, we ran the bandwidth and TPS probe experiments from six other locations, including two computers at Harvard University (Harvard 1 & 2), two computers at MIT (MIT 1 & 2), a shared server at an ISP in Los Angles (ISP LA), a shared server at an ISP in Pittsburgh (ISP PIT), a home server in Belmont, MA, and a server in The Netherlands. We conducted a pre-test in November and December 2006 using EC2, Harvard 1 & 2, MIT 1 & 2, ISP LA, ISP PIT, and Belmont. We then conducted a more carefully controlled test in March and April 2007 using Harvard 1, MIT 1, ISP PIT, and Netherlands.

While it might appear that comparing the throughput at the various hosts tells us more about the network connections between the hosts and Amazon than it does about S3, one of the selling points of S3 is that the data

is stored at multiple data centers with excellent connectivity to the rest of the Internet. Testing the performance and availability of S3 from multiple locations allowed us to both measure this claim and infer the kind of performance that others will experience when they use S3 from outside the Amazon network.

5.3 Experimental Results

We conducted a total of 137,099 probes in 2007. Of these probes, 32,748 tested performance from a single EC2 instance to S3, 74,808 were probes for our surge experiment, and the remaining 39,269 probes were from our distributed test bed. Probes from EC2 consistently experienced the best S3 performance of all our hosts. This is not surprising, as both S3 and EC2 are within Amazon's network. Since there is furthermore no bandwidth charges for data sent between S3 and EC2, we focused the majority of our testing and data analysis on the probes from EC2 to S3.

5.4 Overall Throughput from EC2

Figure 1 shows the average daily read throughput for GETs of 100MB-sized objects as measured from EC2. The error bars indicate the 5th and 95th percentile for each day's throughput measurement.

The graph clearly shows that our EC2 instance saw a marked decrease in available bandwidth from EC2 between April 9th and April 11th. In consultation with Amazon we learned that Amazon made a minor change to its network topology on April 10th. The result of this change was to introduce an additional 2ms delay between the EC2 and S3 systems. Because Amazon's load balancers did not support TCP window scaling [11] in April 2007, the maximum window size supported by these systems is 65,535 bytes. With such a small TCP window, even a relatively minor increase in round-trip-time can have a dramatic decrease on throughput, as shown in Figure 1. Thus, what this figure really shows is the impact that a poorly-tuned TCP stack can have on overall system performance. Sadly, there is no way around this poor tuning with Amazon's currently deployed architecture.

Because of this decrease, we have restricted the remainder of our analysis in this section to data collected between March 21st and April 9th. However, the averages in this date range show a clear downwards trend, so we feel that the sudden decrease in performance is consistent with the rest of our data.

We also examined the DNS resolutions seen from our EC2 instance during the period of the experiment. In total, we saw 10 different IP addresses for the `s3.amazonaws.com` hostname. We have seen Amazon' use different DNS resolutions both for load balancing between different S3 gateways and to re-route traffic when an S3 gateway failed.

Figures 2 and 3 show the average hourly throughput in megabytes per second and transactions per second (TPS). These graphs indicate that the average performance is more-or-less constant, but that the system does suffer significant slow downs—particularly in terms of the number of sequential transactions per second that it can handle—at certain times of the day. A two-tailed t-test for independent variables comparing the samples between 1000 and 1100 UTC with those between 1600 and 1700 UTC and found that this difference was statistically significant ($p < .0001, n_1 = 126, n_2 = 112$), indicating that the differences between these two periods is probably not the result of chance.

5.5 Performance Distribution

Figure 4 contains two cumulative distribution function plots. The top plot shows the distribution of observed bandwidths for reads from EC2 to S3, while the bottom plot shows the distribution of observed write bandwidths. In this section we will discuss the implications of these graphs.

Each graph contains five traces showing the distribution of observed bandwidths for transactions of 1 byte, 1Kbyte, 1MByte, 16MBytes and 100MBytes. As previously discussed, the observed bandwidth for 1 byte transactions is dominated by S3's transaction processing overhead, while the 100MByte transactions is presumably dominated by bulk data transfer. Not surprisingly, S3 performs better with larger transaction sizes. However, there is little discernible difference between the 16MB and 100MB transactions on reads or writes, indicating that little is gained by moving from 16MB data objects to 100MB data objects. It is therefore doubtful that moving to larger transactions would result in significantly increased performance gains.

The top and bottom graphs are on the same scale, making it easy to compare the performance differences between reads and writes.

Comparing 1-byte reads with 1-byte writes, we see most read and write transactions have performance between 10 and 50 TPS. However, writes are more likely to have performances of less than 10 TPS than reads are — roughly 40% of writes are slower than 10 TPS, while approximately 5% of reads have similar performance. This is not a result of our S3 implementation's retry logic: we observed only 6 write retries *in total* from the EC2 host during the course of the experiment, and only 4 read retries. A plausible explanation is that writes must be committed to at least two different clusters to satisfy Amazon's S3 data reliably guarantee, whereas reads can be satisfied from any S3 data cluster where the data happens

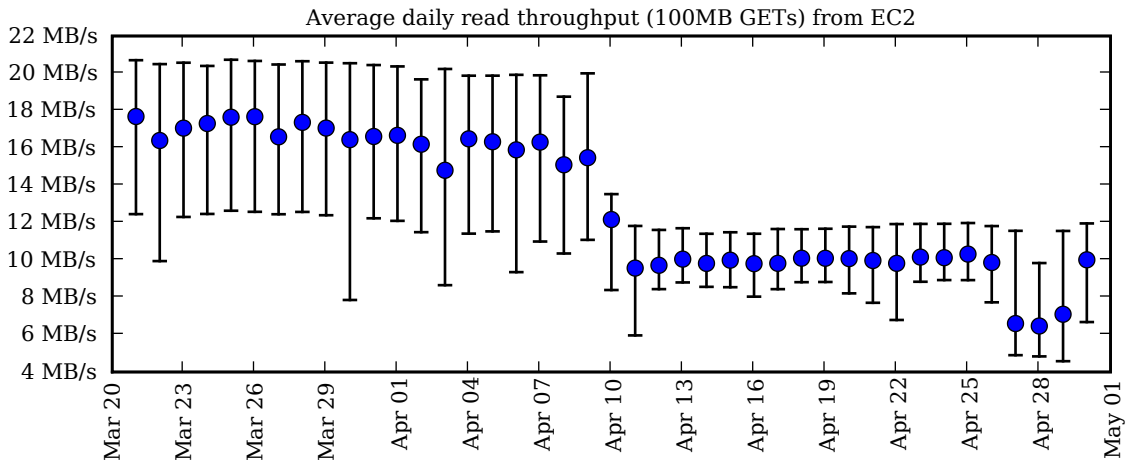

Figure 1: Average daily throughput as measured by 100MB GET operations from EC2. Error bars show the 5^{th} and 95^{th} percentile for each day's throughput measurement.

Hostname	IP Address	Count	First Seen	Last Seen
s3-1.amazonaws.com	72.21.202.37	526	2007-03-20 23:21:13	2007-04-30 04:12:04
s3-1.amazonaws.com	72.21.202.66	547	2007-03-20 23:22:29	2007-04-30 06:06:31
s3-1.amazonaws.com	72.21.203.129	522	2007-03-21 07:03:49	2007-04-30 05:25:53
s3-1.amazonaws.com	72.21.206.171	487	2007-03-20 22:44:51	2007-04-30 04:12:04
s3-1.amazonaws.com	72.21.206.184	505	2007-03-21 04:14:22	2007-04-30 03:58:25
s3-1.amazonaws.com	72.21.207.241	499	2007-03-20 23:22:29	2007-04-30 04:12:04
s3-1.amazonaws.com	72.21.211.210	500	2007-03-20 22:54:40	2007-04-30 05:38:45
s3-1.amazonaws.com	72.21.211.225	511	2007-03-21 00:43:51	2007-04-30 04:12:04
s3-1.amazonaws.com	72.21.211.241	565	2007-03-21 01:15:41	2007-04-30 04:09:01
s3-2.amazonaws.com	207.171.181.225	1	2007-04-06 13:58:25	2007-04-06 13:58:25

Table 1: DNS resolutions for the hostname s3.amazona2s.com as seen from our EC2 instance.

to reside.

Looking at the 1 megabyte transfers, we see that the median read bandwidth is a megabyte per second, while the median transfer of 1 megabyte writes is roughly 5 times faster. A plausible explanation for this is that the write transactions are being acknowledged when the data is written to the disk cache controller at two data centers, rather than when the data is safely on the disk. Many modern disk controllers provide for battery-backed up cache, so writing to the cache is a reliable operation, even if the computer system's power fails before the transaction is committed to the disk. But our experiment is designed so that the read transactions cannot be satisfied from RAM, based on the presumable background transaction rate that S3 must be experiencing. Thus, the read transactions can only be satisfied by actually reading data off the disk.

Caches on RAID controllers tend to be smaller than 100MB. Thus, once the transaction raises to that size it is no longer possible to satisfy the write request by writing solely into the cache, and the factor of 5 difference between read and write speeds vanishes.

Figures 5 and 6 show a histogram of the bandwidth for 1 byte and 100 MByte GET queries. Although these graphs contain a subset of the information of what's in Figure 4, certain characteristics are easier to see. In particular, Figure 6 shows that there are five distinct groupings of response rates—a structure that is not evident in Figure 5.

5.6 Query Variance

Figure 7 presents two scatter plots showing the speed of successive 1 Byte and 100 Megabyte S3 requests. Recall that our experiment involves a series of two successive queries for two different but same-sized data objects for each test probe. Because DNS lookups are not cached by our test program, successive queries may or may not go to the same IP border address.

If the times for each query were the same, then the scatter plots would show a lines with 45-degree slopes. Instead, these scatter plots show a cloud of points, indicating that there can be significant variance between successive queries.

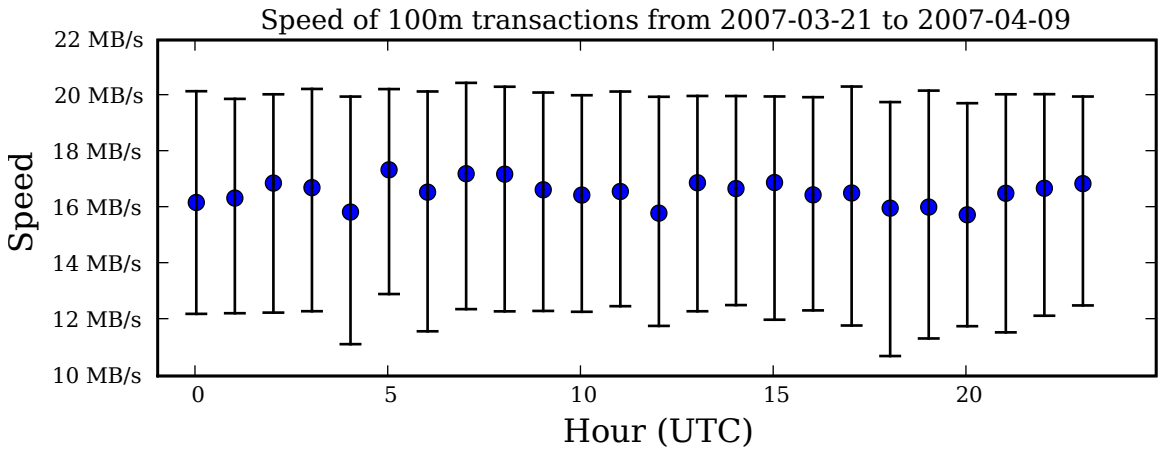

Figure 2: Average hourly performance as measured by 100MB GET operations from EC2.

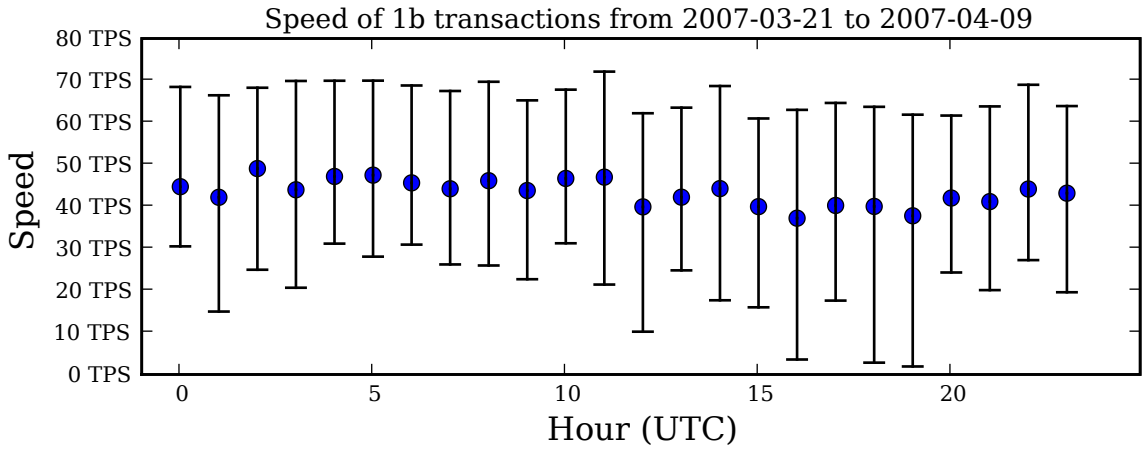

Figure 3: Average hourly TPS as measured by 1 byte GET operations from EC2.

For the 1 byte transactions, we found a correlation coefficient of $r = 0.22$ ($p << .001$). This relatively weak correlation is born out by the side lobes of the plot. In practical terms, this is support for the supposition that if an S3 read request does not quickly respond, it may be faster for the calling application to issue a second request than to wait for the first to finish. Alternatively, it might be appropriate for time-critical applications to issue two simultaneous requests.

For the 100 MByte transactions we observed a much higher correlation $r = 0.85$ ($p << .001$). This implies that if two simultaneous requests are issued and both begin to respond with data, the other request should simply be dropped, since both requests, once they are providing data, are likely to take similar amounts of time.

5.7 Concurrent Performance

If Amazon's "Success Stories" are to be believed and there really are hundreds of thousands of users simultaneously accessing the infrastructure, then it is doubtful that we could actually test the depth of S3's ability to serve simultaneous users without bringing up thousands of virtual machines. Lacking the budget to do this, we have decided to pass on such a test.

What we have decided to test, however, is whether a client can improve its S3 performance by issuing concurrent requests to the same bucket. Amazon advertises that both EC2 and S3 have 250 megabits per second of bandwidth, although they do not indicate to where that bandwidth goes. It seems reasonable to assume that if the bandwidth goes anywhere, it is available for data trans-

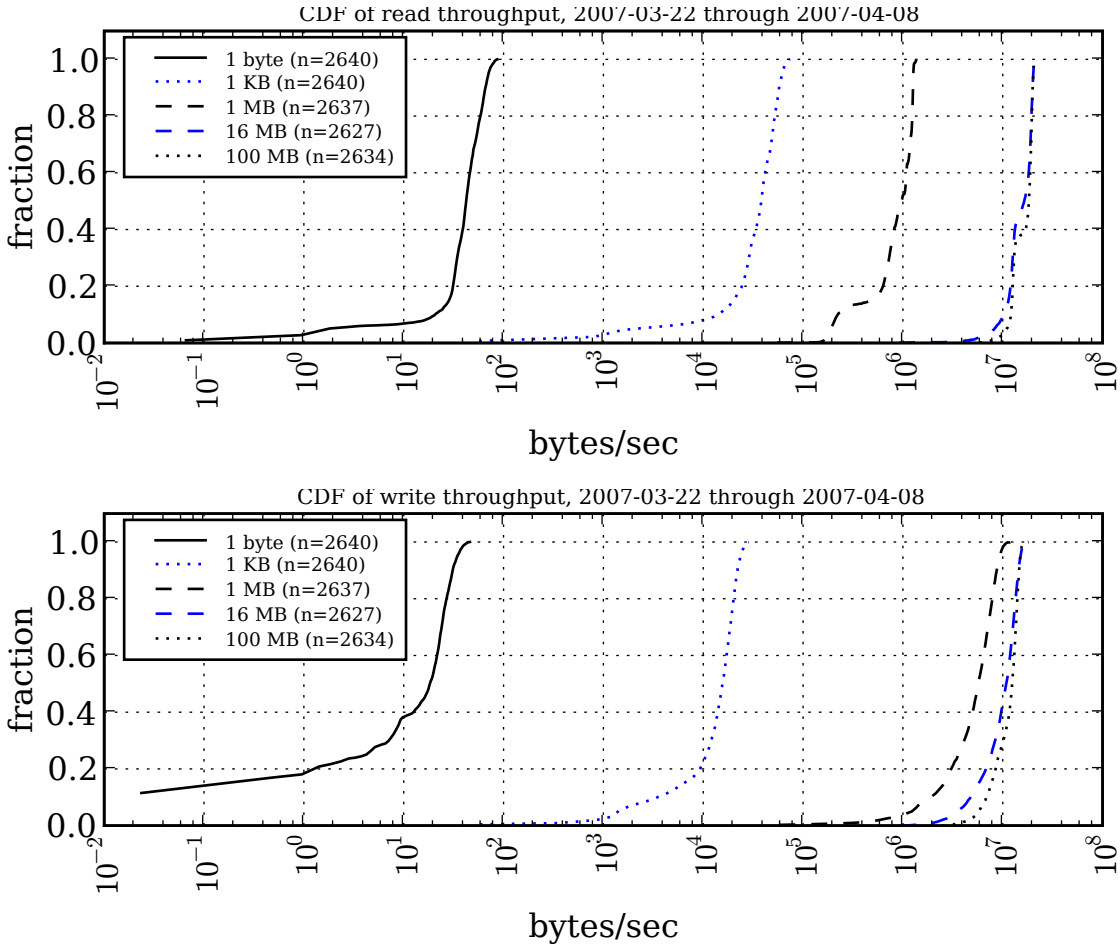

Figure 4: Cumulative Distribution Function (CDF) plots for transactions from EC2 to S3 for transactions of various sizes.

fers between EC2 and S3.

Our "surge" experiment involved two virtual machines, one running on EC2 cluster `usma1` and the second running on cluster `usma2`, accessing the same bucket with repeated 100M GET and PUT operations. The virtual machines were coordinated, with each virtual machine executing 1 thread for 10 minutes, then 2 threads, then 3 threads, and so on up to 6 threads, at which point the experiment reset back to 1 thread. This experiment was repeated for 11 hours.

Our results are presented in Figure 8. The graph on the left shows the per-thread performance for each virtual machine as the number of threads increase from 1 to 6. The graph on the right shows how the total aggregate bandwidth changes as the number of threads increase from 1 to 6.

The decreasing performance of each thread as the number of threads increases is likely the result of processor contention on each virtual machine. As the load raises from 1 to 6 threads, the amount of bandwidth that each thread receives is roughly cut in half. But as the graph on the right shows, this means that six threads have roughly three times the aggregate bandwidth of 1 thread, or roughly 30 megabytes per second. This is very close to the maximum bandwidth that Amazon states users are allowed on EC2 and S3.

5.8 Availability

In addition to measuring the clock time for the upload and download transfers, our testing program also recorded the number of retries and errors encountered (as defined in Section 3.5.1). Our program retried each request a maximum of 5 times, then declared a failure.

Observed failure rates were quite low. In a total of 107,556 non-surge tests from EC2 in 2007, each one consisting of multiple read and write probes, we encountered only 6 write retries, 3 write errors, and 4 read retries.

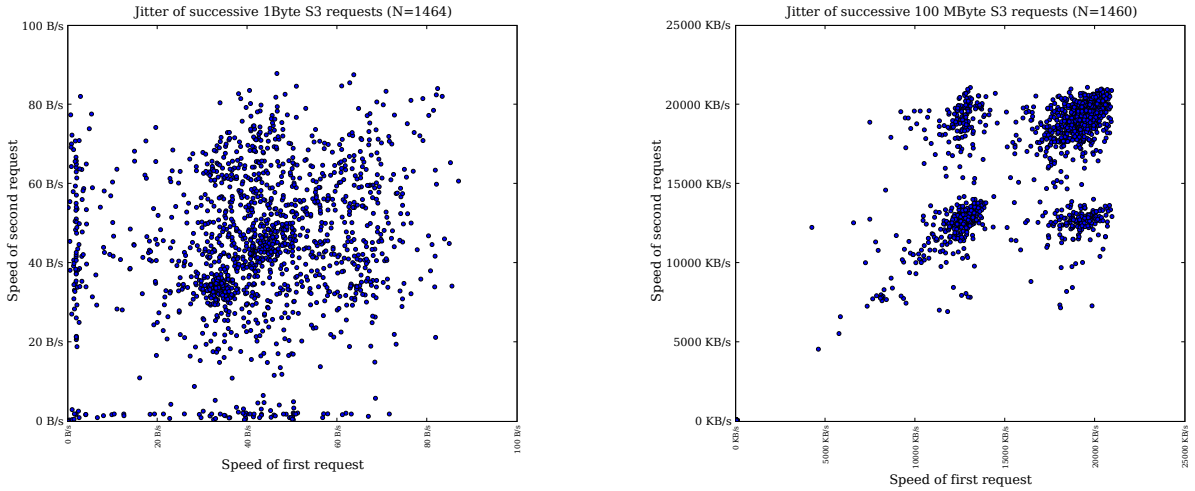

Figure 7: Scatter plots of bandwidth successive S3 GET requests for 1 Byte (left) and 100 Megabyte (right) transactions. The X axis indicates the speed of the first request, while the Y axis indicates the speed of the second.

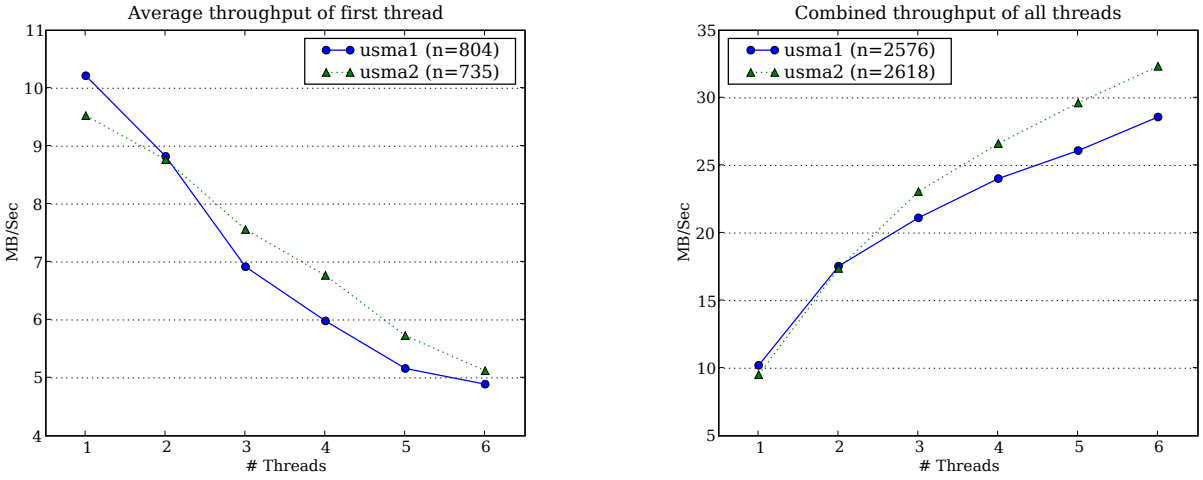

Figure 8: Performance of 100MB GETs from S3 for one thread (left) and combined threads (right), as the number of threads increases.

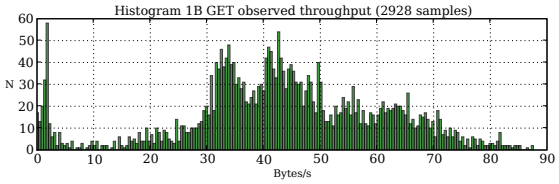

Figure 5: Histogram of 1 byte GET throughput, March 20 through April 7.

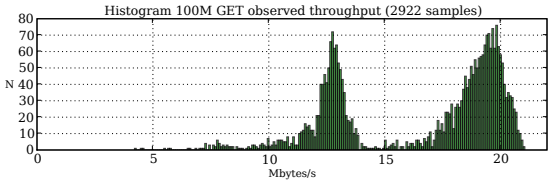

Figure 6: Histogram of 100Mbyte GET throughput, March 20 through April 7.

None of these errors was fatal: We never saw more than 2 consecutive write errors and 1 consecutive write retries. Thus, for applications that properly retried according to Amazon's specification, we saw 100% availability of the S3 service.

Like Amazon's claim of 99.99% availability, the availability numbers in the previous paragraph make no mention of throughput. Restricting the analysis to the 19,630 of 1 Mbyte and greater, we found 0 probes where throughput for write requests was less than 10 KB/s, and 6 where the throughput was less than 100 KB/s—poor performance in comparison to other throughput numbers.

5.9 Other Locations

Table 2 shows performance from several other monitoring locations on the Internet between 2007-03-29 and 2007-05-03. We believe that these numbers are representative of what other US Amazon customers are likely to experience when using S3, for several reasons. None of the hosts outside of Amazon appeared to be limited by the speed of their own Internet connections, as we were able to achieve much higher data transfers from locations other than Amazon. However, the observed performance of S3 may be the result of poor peering arrangements between those network and Amazon's network. Alternatively, the poor performance may be the result of other factors.

Figure 9 shows the same data plotted as a CDF.

6 Evaluating SQS

In our experience SQS is appropriate for coordinating relatively large jobs executed on EC2 that store data on S3. Performance of SQS is sufficient for scheduling tasks that take tens of seconds to minutes or hours, but not sufficient for coordinating many tasks that are faster than a second or slower than several hours. But this isn't really a problem: if a task requires many tasks of a second each, these could be combined into tens of tasks requiring a minute. Tasks that would take longer than a few hours could be broken up into several dependent tasks. This would also protect against system failures.

We performed an experiment in which 10,000 messages were stored sequentially in an SQS queue. The total clock time required was approximately 2230 seconds in one trial and 2334 seconds in a second, or roughly 4 messages per second. We used an SQS implementation in Python based on the REST implementation. Only one message can be inserted into the queue at a time using our interface.

Messages can be removed from a queue in batches of between 1 and 256 objects at a time. Removing the 10,000 objects from the queue 1 message at a time took 3855 seconds, or roughly 2.5 messages per second. Removing the messages 256 at a time required 1919 seconds, or roughly 5 messages per second. The queues appeared to be loosely ordered. That is, while we loaded a queue with messages `1` through `10000` consecutively, the messages were read from the queue with the sequence `2 15 16 7 26 25 6 65 24 83 8 40 18 1 46 14 23 28 27 56 10 32 72 4 77 13 70 102 5 29...`. Although it might be possible to learn a lot about the structure of SQS by examining these sequence over successive attempts, we decided against conducting these experiments.

Overall we found that SQS lived up to its name. It provides a simple API for managing work that can be readily parallelized. We found it easy to integrate SQS into our workflow.

7 Using EC2, S3 and SQS in Research

Our interest in the Amazon Web Services offerings was sparked by a large-scale computer forensics project. Over the past 8 years we have purchased more than 1000 used hard drives on the secondary market. Although our original motivation in this project was to evaluate the sanitization practices of the hard drives' previous owners, in recent years our research has shifted to the development of new computer forensic techniques and tools using this corpus.

As part of our project we have developed a file format and access layer so that we can store disk images in

Host	Location	N	Read Avg	Read top 1%	Read Stdev	Write Avg	Write top 1%	Write Stdev
Netherlands	Netherlands	1,572	212	294	34	382	493	142
Harvard	Cambridge, MA	914	412	796	121	620	844	95
ISP PIT	Pittsburgh, PA	852	530	1,005	183	1,546	2,048	404
MIT	Cambridge, MA	864	651	1,033	231	2,200	2,741	464
EC2	Amazon	5,483	799	1,314	320	5,279	10,229	2,209

Units are in bytes per second

Table 2: Measurements of S3 read and write performance in KBytes/sec from different locations on the Internet, between 2007-03-29 and 2007-05-03.

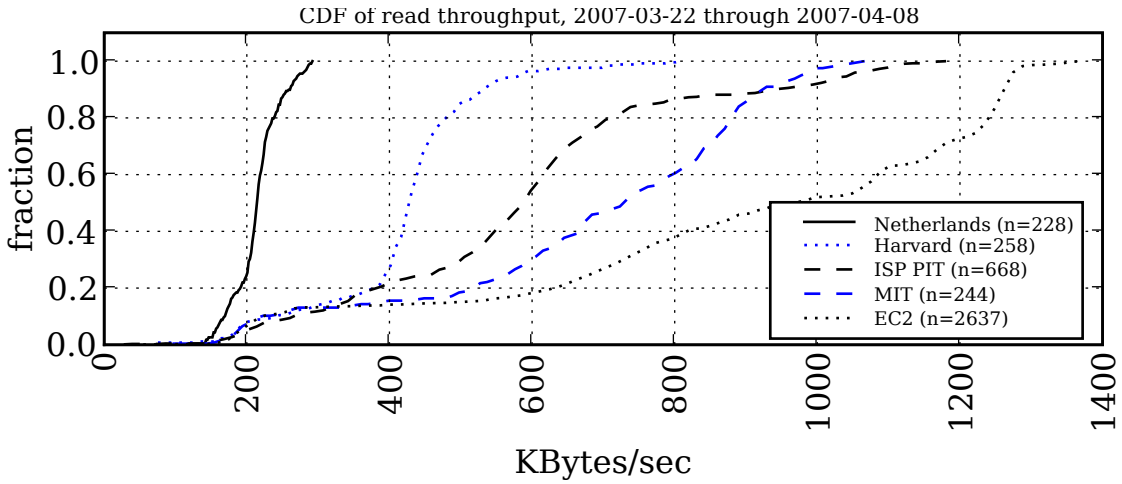

Figure 9: Cumulative Distribution Function (CDF) plots for 1MB GET transactions from four locations on the Internet and from EC2.

a highly compressed manner [9]. Compressed, the entire corpus now requires approximately 1.2 terabytes to store. This translates to approximately 5 terabytes of uncompressed data. As our project has grown in size, the task of managing this data on a tight budget has become increasingly onerous. Furthermore, much of our research is characterized by the development of a new technique and then the desire to run a tool over the entire data set. Simply reading and decompressing all of the data can take more than a week.

When we learned about the S3 offering we first saw it as an attractive backup alternative, as we had recently had a $400 hard drive fail and had spent more than a week of clock time reconstructing the drive from backup. But the data bandwidth charges made S3 impractical for use in our research. When Amazon introduced EC2 we immediately realized that we could improve our throughput on experiments by leveraging the combination of EC2, S3 and SQS.

Over the past six months we have developed a working prototype that allows us to perform forensic experiments using the combination of EC2, S3 and SQS. Our first step was to modify the access layer of the forensic library so that disk images could be transparently stored either on a computer's local hard drive or on Amazon S3. Our access layer already had support for several different kinds of disk image formats; we added S3 as a new disk image format. As a result, our access layer now treats file names that begin *s3:/bucket//image.aff* as a reference to an image called *image.aff* stored in the S3 bucket *bucket*.

Having written the implementation, we then copied our 1.2TB archive to Amazon S3. This process took several weeks. (Amazon informed us that customers needing to upload dozens or hundreds of terabytes may send the data by courier to Amazon's site for faster uploading.) Once uploaded, we spent several months familiarizing ourselves with the use of EC2 and SQS.

With the work that we have done it is now possible to develop forensic tools that work either on a local computer at our lab with a disk image stored on a local hard drive, or that will run on one or more EC2 instances using S3. To run an experiment we load the identifier for each disk image into an SQS queue; programs running on one or more EC2 instances will then read the identifiers out of the queue, process the disk image with the tool, and write the results into a database stored on another server. Additional EC2 instances can be activated as necessary to meet conference deadlines. And, perhaps best of all, researchers at other institutions can be given read-only access to the S3 data from their own Amazon

AWS accounts.

Based on what we have now learned about the design of S3 we are considering a redesign of our forensic S3 implementation. We were not aware of the dramatic performance benefit to storing objects in the 16MB–100MB size; our system instead stores each disk image as many individual objects that range in size from 1 byte to 16 Kbytes and from 500 KBytes to 16 MBytes. The new design may combine all of the smaller objects into a single object that could be downloaded, modified, and uploaded as a single unit as necessary.

Because of Amazon's security model, we are adding the capability to digitally sign disk images to our forensic file system [9].

Overall, we found Amazon's protocols easy to work with, but we were frustrated by the lack of design guidance. We were also frustrated by Amazon's reluctance to release information about the S3, EC2 and SQS internals that would have allowed us to tune our implementation. Finally, we were frustrated by the lack of C/C++ example code; all of Amazon's code is in Java, perl, python, and Ruby.

8 Conclusions

We have presented a first-hand user report of Amazon's utility computing EC2, S3 and SQS services based on our analysis of Amazon's security model, our implementation of the S3 client API, out measurement of S3 performance from EC2, a series of end-to-end throughput measurements of S3 from various points on the Internet, and our integration of EC2, S3 and SQS into an unrelated research project.

We find that EC2 delivers on its promise of providing ready-to-go virtual machines at a reasonable cost, provided that users wish to run Linux on those virtual machines.

We find that S3, while certainly appearing to be scalable and available services, only delivers on Amazon's claimed level of throughput when applications execute data transfers that are 16MB of larger, and only when executing multiple simultaneous transactions. The inability of the system to provide peak throughput for small transactions appears to be the result of S3's high transaction overhead; we do not know why S3 is unable to deliver high performance for single-threaded applications. We also saw a large drop in system performance of S3 between April 9th and April 11th that we are unable to explain, and which Amazon has declined to explain.

We believe that most users will want to access S3 from EC2 rather than from elsewhere on the Internet, as there are no bandwidth charges between S3 and EC2, and because S3 delivered as much as a 5 times the performance to EC2 as it did to other locations that we measured on the Internet.

We find the SQS is an easy-to-use platform for coordinating work across multiple machines. Although we found that SQS is limited to delivering between 4 and 5 transactions per second per thread, we found a number of ways to code around this, including farming out work in larger units and executing queries from multiple threads.

Availability of the Amazon service was excellent. In no case were we unable to access our data during the period in which this test was run, although it was sometimes necessary to perform several retries to store or retrieve information.

In short, the Amazon services are innovative, useful, and represents a practical alternative for organizations interested in storing large amounts of data or making use of grid computing. Our primary concerns with the service are the risk and result of password compromise, and an extremely broad License Agreement which allows Amazon to terminate service for any reason, at any time, and without any recourse to the customer. We are unable to measure the significance of these risks, and they could easily be addressed by Amazon with an improved authentication model, the introduction of snapshots to protect accidentally deleted data, and more customer-friendly Web Services License Agreement.

Organizations thinking of using S3 to service real-time customer queries should carefully consider their data architecture, taking into account the long tails at the bottom of the CDF plots. Between 10% and 20% of all queries suffer decreased performance that is 5 times slower than the mean or worse. The fact that a similar percentage of requests see significantly faster performance is probably of little consolation for users having to wait for an S3 transaction to complete. In many cases it is almost certainly faster for the caller to abort an S3 transaction and initiate a new one, rather than waiting for a slow one to complete.

8.1 Acknowledgments

We gratefully thank Amazon for their decision to commercialize their S3 and EC2 technologies. We hope that these innovative services live up to their potential to positively impact academic and commercial computing.

We would specifically like to thank Drew Herdener at Amazon for answering many of our S3 and EC2 questions. We would also like to thank Navneet and Brian, who monitored Amazon's technical forums.

Our thanks to Steven Bauer, Rob Beverly, Jesper Johansson, M. Pannevis, Elisabeth Rosenberg and Charlie Wood for their review of this paper. We would also like to express our thanks to the anonymous reviewers of the previous versions of this paper.

References

[1] Amazon Web Services. Tech. note: Caching ip addresses, December8 2006. http://developer.amazonwebservices.com/connect/entry.jspa?externalID=193.

[2] Customer success stories: Amazon web services, May 2007. http://www.amazon.com/Success-Stories-AWS-home-page/b?node=182241011.

[3] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 3986: Uniform resource identifier (uri): Generic syntax, January 2005.

[4] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003. ISSN 0146-4833.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University Of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[6] First Circuit Court of Appeals (En Banc). US v. Councilman, 03-1383, August11 2005. http://www.eff.org/legal/cases/US_v_Councilman/councilman_decision.pdf.

[7] Simson Garfinkel. Forensic feature extraction and cross-drive analysis. *Digital Investigation*, August 2006. http://www.dfrws.org/2006/proceedings/10-Garfinkel.pdf.

[8] Simson L. Garfinkel. Email-based identification and authentication: An alternative to PKI? *Security & Privacy Magazine*, 1: 20–26, Nov. - Dec. 2003.

[9] Simson L. Garfinkel, David J. Malan, Karl-Alexander Dubec, Christopher C. Stevens, and Cecile Pham. Disk imaging with the advanced forensic format, library and tools. In *Research Advances in Digital Forensics (Second Annual IFIP WG 11.9 International Conference on Digital Forensics)*. Springer, January 2006.

[10] Andrew Herdener. Personal communication, 2006.

[11] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP extensions for high performance, May 1992. Obsoletes RFC1072, RFC1185 [12, 13]. Status: PROPOSED STANDARD.

[12] V. Jacobson and R. T. Braden. RFC 1072: TCP extensions for long-delay paths, October 1, 1988. Obsoleted by RFC1323 [11]. Status: UNKNOWN.

[13] V. Jacobson, R. T. Braden, and L. Zhang. RFC 1185: TCP extension for high-speed paths, October 1, 1990. Obsoleted by RFC1323 [11]. Status: EXPERIMENTAL.

[14] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104: HMAC: Keyed-hashing for message authentication, February 1997. Status: INFORMATIONAL.

[15] United States Office of Personnel Management. *Addressing Sexual Orientation Discrimination In Federal Civilian Employment: A Guide to Employees*. June 1999. http://www.opm.gov/er/address.pdf.

[16] Jeffrey Pang, James Hendricks, Aditya Akella, Roberto De Prisco, Bruce M. Maggs, and Srinivasan Seshan. Availability, usage, and deployment characteristics of the domain name system. In Alfio Lombardo and James F. Kurose, editors, *Internet Measurement Conference*, pages 1–14. ACM, 2004. ISBN 1-58113-821-0.

[17] Vern Paxson. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4 of *ACM SIGCOMM Computer Communication Review*, pages 25–38. ACM Press, New York, August 1996. ISBN 0-89791-790-1. citeseer.ist.psu.edu/article/paxson96endtoend.html.

[18] RolandPJ@AWS. Re: S3infinidisk, January 2007. http://developer.amazonwebservices.com/connect/message.jspa?messageID=51291#51291. Amazon says that S3 is designed for 10s of MB.

[19] Amazon Web Services. Web services licensing agreement, October3 2006.

[20] Daniel Stenberg. curl. http://curl.haxx.se/docs.

[21] Jacob Strauss, Dina Katabi, and M. Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Internet Measurement Comference*, pages 39–44. ACM, 2003. ISBN 1-58113-773-7.

[22] W3C. Soap version 1.2, June24 2003. http://www.w3.org/TR/soap/.

[23] Amazon web services. Amazon elastic compute cloud (amazon ec2) - limited beta, May2 2007. http://www.amazon.com/gp/browse.html?node=201590011.

[24] Amazon web services. Amazon simple storage service (amazon s3), May 2007. http://aws.amazon.com/s3.

[25] XenSource Inc. Xen. http://www.xensource.com/.