SIMSON GARFINKEL

commodity grid computing with Amazon's S3 and EC2

Simson L. Garfinkel is an Associate Professor at the Naval Postgraduate School and a Fellow at the Center for Research on Computation and Society at Harvard University. He is also a consulting scientist at Basis Technology Corp., which develops software for extracting meaningful intelligence from unstructured text, and a founder of Sandstorm Enterprises, a computer security firm that develops advanced computer forensic tools used by businesses and governments to audit their systems.

simsong@acm.org

AMAZON.COM RECENTLY INTRODUCED two new storage and computing services that might fundamentally change the way that we provision equipment for both e-commerce and high-performance computing. I learned about these services a few days after I had started looking for quotes to purchase a multiblade server with 10–20 TB of storage to further my own research in computer forensics. Rather than moving ahead with that purchase, I decided to evaluate whether or not I could use Amazon's offering for this real-world problem. My conclusion is that Amazon's offering is good enough for my research and will probably save me tens of thousands of dollars, but the system isn't yet ready for hosting serious e-commerce customers.

Amazon's Simple Storage Service (S3) and Elastic Compute Cloud (EC2) break up Amazon's awesome computer infrastructure into tiny little pieces that the company can incrementally rent out to any individual or business that needs e-commerce or high-performance computing infrastructure. Like Google and Yahoo!, Amazon has built computing clusters around the world, each with tens of thousands of computer systems. The theory behind these services is that economies of scale allow Amazon to run and rent out these services to businesses at a cheaper price than businesses can provide the services to themselves.

Amazon's Simple Storage System

Amazon announced S3 back in March 2006. The service allows anyone with a credit card to store information on Amazon's redundant and replicated storage network. Storage costs are 15 cents per gigabyte per month; data transfer is 20 cents per gigabyte. You may store an unlimited amount of information, and there is no setup fee.

The best way to think about S3 is as a globally available distributed hash table with high-level access control. You store data as a series of name/value pairs. Names look just like UNIX filenames; the value can be any serialized object between 0 and 5 GB. You can also store up to 4K of metadata with each object.

All objects in Amazon's S3 must fit in the same global namespace. The namespace consists of a "bucket name" and an "object name." Bucket names are available from Amazon on a first-come, first-serve basis. You can't have "foo" because it's already been taken, but you could probably get a bucket name with your last name, and certainly with your last name followed by a random seven-digit number. Bucket names are reasonably secure: You can list the names of the buckets that your account has created, but you can't list the buckets belonging to other people. You can only have 100 buckets per account, so don't go crazy with them.

Access control is based on these buckets. You can make your bucket readable by up to 100 Amazon Web Service Account IDs and read/write for up to another 100 IDs. You can also make a bucket world readable, although this is a lot like handing the world a blank check, since downloads do cost 20 cents per gigabyte. Near the end of this article we'll see how this cost compares with existing hosting services.

You send data to S3 using a relatively straightforward SOAP-based API or with raw HTTP "PUT" commands (a technique that has taken on the name "REST," short for Representational State Transfer). Data can be retrieved using SOAP, HTTP, or BitTorrent. In the case of BitTorrent, the S3 system operates as both a tracker and the initial seed. There is also a program called JungleDisk that lets you treat storage on S3 as if it were a remote file system; JungleDisk runs on Linux, Mac OS, and Windows.

After delving into the needless complexity of SOAP, I gave up and decided to use the pure HTTP/REST API. I'm using S3 to store images of hard drives that I have acquired or developed during the course of my research in computer forensics. With REST, I can store raw data without having to first base-64 encode it. I also found it much easier to code up a simple S3 REST implementation than to deal with all of the overhead required for the SOAP client.

S3 Performance and Security

I tested S3's performance with the REST API from networks at MIT, Harvard, and my house. Both universities have ridiculously fast connections to multiple Internet carriers. Despite this speed, both my upload and download speeds averaged between 1 and 2 MB per second, depending on the time of day. I saw similar performance from my house, where I have a 30 megabit per second Verizon FiOS connection. Based on the feedback in the Amazon developer forums, these performance figures are at the upper end of what others are seeing. One developer in Germany reported seeing between 10 and 100 kilobytes per second, depending on the time of day. Although this speed is simply not fast enough for doing serious computation, it is good enough for backups and for using S3 to deliver Web objects. Clearly, though, performance is an area that needs work for all S3 users.

Security is another important part of any storage system. Amazon's S3 has impressive support for privacy, integrity, and short-term availability. The long-term availability of the service is unknown, since it ultimately depends upon Amazon's internal level of commitment. Surprisingly, the weakest part of S3 is the service's authentication architecture. I'll discuss each of these issues next.

Data privacy is accomplished through the use of encryption and access control. If you want your data to be encrypted, encryption must be done before the data is sent to S3. You can protect names of the objects and other metadata by communicating with Amazon's Web servers using SSL

with HTTPS on port 443. In my testing with a 2-GHz Intel Core Duo MacBook on MIT's network, downloading data from S3 over SSL took roughly 10% longer than downloading the same data over raw HTTP. This minor overhead demonstrates that the computational cost of encrypting data these days is really minor compared to other costs; nonetheless, encryption still isn't free.

Integrity for stored data is accomplished with an end-to-end check using the MD5 cryptographic hash as a checksum. When an object is stored to S3, Amazon's system computes the MD5 of that object and returns that hash with its response. My S3 implementation compares Amazon's computed hash with a hash that I computed locally. If the two don't match, my implementation resends the data. Although my implementation will send objects of any size, my code never sends objects larger than 16 MB.

Short-term availability is a reflection of Amazon's connectivity, the load on its servers and network fabric, and even the reliability of its code. In my testing I found that somewhere between 0.1% and 1% of all PUTs had to be retried because the PUT did not complete successfully. Normally PUTs succeeded on the second retry, but sometimes I needed to retry three or four times. An Amazon employee posting in one of the developer forums recommended implementing an exponential back-off for failed writes [1], but my implementation just retries as soon as it receives an error. After writing more than a terabyte to S3, I never experienced a failure that required more than four retries.

Long-term availability is a bigger question, unfortunately. Once the data is actually stored at S3, it's Amazon's responsibility to ensure that it remains available for as long as the customer pays the bills. Amazon claims that the data is stored on multiple hard drives in multiple data centers. Unfortunately, Amazon doesn't back up this claim with any sort of Service Level Agreement (SLA). There is also no backup or recovery service in the event that you accidentally delete some important data. As a result, it's important to maintain a backup of any important data stored inside S3.

The *authentication strategy* of Amazon Web Services (AWS) looks quite robust at first. Unfortunately, it's really a steel bunker built on a foundation of quicksand.

AWS supports a simple authentication strategy based on the SHA1-HMAC algorithm. Every AWS account has an Access Key ID and a Secret Access Key. The Access Key ID is a 20-character string that's used to uniquely identify your account; the Secret Access Key is a 41-character string that's used to digitally sign SOAP and REST requests. To sign a request, you simply compute the HMAC of the request parameters using the Secret Access Key as the key for the HMAC. This HMAC is sent along with the request. Amazon's servers, which know your Secret Access Key, compute the same HMAC. If the two HMACs match, then the request is authorized. Requests include a timestamp to prevent replay attacks.

The HMAC approach is fast, efficient, and pretty secure. The underlying weakness is that the credentials are downloaded from the AWS Web site. This means that anyone who knows your Amazon username and password can download your Secret Access Key. Since Amazon allows the password to be reset if you can't remember it, by simply clicking on a link that's sent to the account's registered email address, anyone who has control of your email system can effectively delete all of the information you have stored in S3. Amazon will have to rethink this authentication architecture before organizations can trust it with mission-critical information.

The other real problem with S3 is the cost structure: Currently it costs nearly as much to upload and download a piece of information as it costs to store that same data for three months. Although this may be a dramatic demonstration that the cost of storage is dropping much faster than the cost of bandwidth, these bandwidth charges make S3 simply unaffordable for many projects. Unfortunately, Amazon's pricing made the S3 service completely unusable for me until the company introduced its second grid-computing offering—a high-performance computing utility that let me move my computation close to my data.

The Elastic Compute Cloud

Amazon's Elastic Compute Cloud (EC2) makes S3's pricing strategy far easier to manage by eliminating the bandwidth charges for moving data between storage and computation.

As its name implies, EC2 lets you rent time on a "cloud" of computers. These computers are all the equivalent of 1.7-GHz Xenon servers with 1.25 GB of RAM and 160 GB of local disk. The cost of these machines is 10 cents per CPU per hour. As with S3, it costs 20 cents per gigabyte to move data between the rest of the Internet and EC2. However, there is no charge to move between EC2 and S3. According to Amazon, each virtual machine has 250 megabits per second of bandwidth, although how that translates to speed between EC2 and S3 depends upon a variety of factors.

The "machines" that Amazon delivers with EC2 are actually virtual machines, each running on top of the Xen platform. You create a virtual machine by storing a disk image inside S3 using special tools that Amazon provides and then running a Java program that instantiates the virtual machine. A second Java program lets you monitor the progress of the machine's creation; when it is ready, the script displays the computer's hostname. Obviously, the image that you instantiated should have an account that lets you log into the machine.

Because EC2 is based on Xen, it should support any Linux distribution as well as NetBSD, FreeBSD, Plan 9, and other operating systems. In practice, though, EC2 is largely based on the RedHat Fedora Core operating system, although there are instructions on the Internet for using it with Ubuntu distributions. I found this disappointing, because FreeBSD and Darwin, followed by Ubuntu, are my preferred operating systems.

Amazon makes no promises about the reliability of the EC2 computers: Each machine can crash at any moment, and they are not backed up. In my experience these machines don't crash, but, remember, computers do fail. If you want reliable storage, you can run two or more EC2 machines as a cluster. A better approach, though, is to have the EC2 machines store information in S3, which is sold as a reliable, replicated service.

What's really neat about EC2 is that you can build a small system and expand it as it becomes more popular, by simply bringing up more virtual computers. In fact, you could even bring up virtual machines on Thursdays and Fridays, if those are your busy days, and shut those machines down during the rest of the week.

The EC2 security model is similar to that of S3, except that commands are signed with an X.509 private key. Unfortunately, you download your private key from the AWS Web site, so the security still fundamentally depends on the AWS username and password. That private key can be used to start up machines, shut them down, and configure the "firewall" that

protects your virtual machines on the EC2 infrastructure. The firewall allows you to control which IP addresses and ports on the Internet can reach which of your virtual machines. By default all ports are closed; you'll probably want to open the firewall to allow port 22 (ssh) through, at the least. Machines that run Web servers should probably have port 80 opened. And, of course, you'll probably want to configure the firewall so that your virtual machines can communicate with each other, at least on some ports.

Amazon had an early security problem with EC2: The company was neglecting to wipe the computer's virtual disk drives before switching them from one customer to another. That problem has since been corrected.

s3_glue: A C++ implementation of the S3 REST API

As I already mentioned, I've been using S3 and EC2 for my research in computer forensics. As part of my research I've created an open source system for imaging hard drives and storing the results in highly compressed but random-access disk images [2]. This October I added support for S3 to the library so that images could reside on the local computer or on Amazon S3.

Amazon provides code samples for S3 in C#, Java, JavaScript, Perl, Python, and Ruby. Although these examples are instructive, my disk-imaging system is written in C++ for performance reasons. To make the code usable for others I separated out the basic S3 implementation from the code that is specific to my forensics library. The implementation can be downloaded from http://www.simson.net/s3/. It uses libcurl [3] for HTTP.

Recall that S3 objects are all given object names and that these objects are in turn placed into buckets. The REST API turns object and bucket names into URLs of the form http://s3.amazonws.com/bucket-name/object-name. Data is downloaded with an HTTP GET and uploaded with an HTTP PUT. There is also provision for setting up a virtual host (e.g., http://bucket.s3.amazonws.com/object-name), which makes it somewhat easier to have S3 directly serve Web content to browsers.

S3 requests are authenticated through additional terms that are added to the query section of the URL. The "Signature=" term includes the HMAC of the requests headers represented in a canonical form and the user's AWS Secret Access Key. The "Expires=" term allows you to specify when the query will expire. Finally, the "AWSAccessKeyId=" term specifies the requestor. Remember, authentication isn't needed for buckets that are world-readable or world-writable.

HTTP 1.1 allows a client to request a range of bytes; S3 implements this part of the protocol, allowing you to request a few bytes of a very large object. S3 limits an object overall to 5 GB, although a bug in Amazon's load balancers means that objects are effectively limited to 2 GB in size. I store disk images larger than 16 MB as multiple pages, each of which is 16 MB in length before being compressed, so the 2GB limitation wasn't a problem for me.

My S3 implementation provides simple and efficient C++ functions for listing all buckets that belong to a user, making a new bucket, deleting a bucket, selectively listing the contents of a bucket, getting an object, saving an object, and removing an object. The code supports arbitrary name/value pairs for metadata on an object. This metadata needs to be stored with an object but can be independently retrieved.

S3 is pretty powerful as far as it goes, but there is a lot of functionality missing. There is no way to rename an object, for example. There is no way to search—you can't even search for objects of a particular length or that have a particular metadata field in their headers. In this way, S3 is a lot like Berkeley DB or the Python "dictionary" data structures: You can store data, get it back, and iterate. Anything else is up to you. Because objects can be listed and retrieved in lexical sort order, I expect that many applications will encode a lot of information inside the file name. That's what I did.

In addition to my S3 implementation, I've also created a command-line utility called "s3." This program is mostly for testing the S3 library and maintenance of the S3 system. It implements UNIX-like commands for listing the contents of a bucket, copying the contents of an object to standard output, deleting an object, deleting a set of objects, and managing buckets. This program is also available from my Web site.

Crunching the Numbers

In my research I have been running programs that take literally a month to run on a workstation with a terabyte hard drive. With Amazon's EC2 and S3 I can split the task up and run it on 30 virtual computers over the course of a day, for roughly $72. Or I can run it on 60 virtual computers for 12 hours, again for $72. This simple example demonstrates the big advantage of renting time on another organization's grid over building your own. Unless you have enough work to occupy your grid 100% of the time, every hour that a computer isn't working is an hour that you paid for but received nothing in return.

There are other alternatives to Amazon's offerings. Dreamhost, an ISP that I use for some of my personal work, just dramatically lowered the cost of its Web-hosting plans. For just $9.95/month you can have 200 GB of storage (automatically increasingly by 1 GB each week) and 2 TB a month of bandwidth. Amazon would charge $30 for the same storage but a whopping $400 for that much bandwidth. Unfortunately, Dreamhost had significant reliability problems this past summer.

Pair.com, a premium Web-hosting company at the other end of the cost/performance spectrum, charges $9.95/month for a basic Web-hosting account with 500 MB of disk storage and 40 GB per month of bandwidth. Amazon would charge 7.5 cents for the storage and $8 for the same bandwidth. Pair.com will rent you a dedicated 2.8-GHz Celeron computer with 512 MB of RAM and an 80-GB hard drive with 600 GB per month of traffic for $249/month. Amazon's EC2 machines are faster, have twice the RAM and twice the local disk, and cost just $72/month, although that 600 GB per month of bandwidth will cost you another $120. On the downside, Pair will provide 24/7/365 server monitoring and support, whereas the Amazon servers can crash and there is no support other than what's available in the developer forums. But Pair won't let you bring up 50 machines after lunch and then shut them down when you go home for dinner.

Whereas Amazon's EC2 is an automated provisioning system for virtual machines, another approach is being pursued by 3Tera, a small company in Aliso Viejo, California. 3Tera has developed an operating system for grid computing that allows a single application to be deployed across multiple machines in an automated fashion. As of this writing 3Tera has licensed its technology to UtilityServe, which will run AppLogic-based applications for between 75 and 99 cents per RAM-GB hour; bandwidth is $1.49 to $1.99

per GB; the company includes between 100 and 4000 GB of storage in its base packages, and it sells additional storage for $99 per 50 GB.

Conclusions

S3 and EC2 are obviously both young and immature services: They are tantalizing in what they promise, but Amazon needs to address the issues of authentication, availability, and long-term stability before businesses should seriously rely on this offering. I wouldn't trust my business to S3 or EC2 without a signed contract in place that clearly outlined Amazon's obligations and my recourse against Amazon if those obligations were not met.

At the same time, I think that S3 and EC2 are a taste of the kinds of computer utility services that will be available in the not-so-distant future. High-quality storage, computation, and bandwidth will be available at commodity prices. With any luck other companies will reimplement the server side of Amazon's APIs, making it possible to move a service easily from one provider to another. With these kinds of services, I can spend my time using a computer utility, rather than building one from blade servers and RAID boxes. I can then devote my time to worrying about algorithms instead of rack space, electricity bills, and cooling.

Because it is running so many computers, Amazon can run them a lot cheaper than I can. Assuming that the company can make good on its implicit availability and bandwidth commitments, this is going to be a very compelling offering.

REFERENCES

[1] http://developer.amazonwebservices.com/connect/thread.jspa?messageID=46813.

[2] http://www.afflib.org/.

[3] http://curl.haxx.se/.