Research Paper by David Smelser

Data Storage – Amazon S3 vs. Google Datastore

The primary purpose of this document is to outline Amazon's and Google's competitive storage engines currently available in the market. Through comparing and contrasting each engine, this document will provide a framework for making a decision on which technology to use. The metrics used in the comparison are cost, reliability, scalability, and security.

The majority of the background information was found on the associated website of each provider. Additional research materials were pulled from community-based blogs, websites, and Wikipedia. Thought and consideration to the source of these materials was taken into account prior to inclusion in this document.

Readers should have basic understanding of file storage and database systems when applied to a distributed system.

Background

Since technology changes so quickly, we should quickly examine the background of file storage, databases, and the reasons why distributed technologies and scalability are such important issues to be considered in a modern data storage system.

To give a baseline, let's consider the first "mass storage" device that closely resembled a modern day hard drive. In 1957, IBM released the RAMAC 350, which required 50 24-inch disks to store 5 MB and cost a staggering $35,000 per year to lease [1]. A quick Google search for hard drives today reveals a 1.5 TB external hard drive that costs $99.99. This is a 31 million % increase in data storage for .3% of the cost.

Although these numbers are staggering, they neglect the most import part of current data storage requirements: availability. Not only do we require an extremely high reliability rate, but also secure access to this information from anywhere at any time, which leads us to the internet and the distributed file systems to be discussed. To put the scope of possible scalability needs into perspective, youtube.com delivers 25 Petabytes per month of data with over 100 million downloads per day [2]. Over the past few decades, particularly along with the advent of the internet, the potential needs for data storage have increased from what now seems like tiny amounts of locally stored data to needs and/or requirements to make data available to millions throughout the entire world.

Amazon S3

The Amazon S3 storage engine was released in 2006. The S3 system was designed with a minimal set of functionality to provide a simple-to-use system.

The Amazon S3 is a simple web services interface for storing data. The intent of the S3 design was to be fast, reliable, secure, secure, inexpensive, and highly scalable. This is achieved by using a relatively low-cost infrastructure distributed to multiple locations around the world.

S3 Features

Basic Structure:

S3 utilizes something similar to a hash table implementation for their storage engine. Each entry is given an id referencing the "bucket" that the data is stored in. Each bucket is comprised of one or more objects stored as simple blobs (binary data). These objects are binary data that can range in size from 1 byte to 5 Terabytes.

Simple Commands:

S3 uses a minimal instruction set for simplicity and reliability. The S3 uses four basic operations, the get(), put() , copy(), and delete(). Since the buckets for data storage only require a bucket id, these methods permit simple data access. In addition, this simplified API makes implementing the technology as a developer simple and allows for greater reliability and simplicity in implementation for the S3 framework.

Speed:

S3 provides several features which are ideal for high speed and scalability for traffic. By providing regional storage in multiple locations around the world, S3 can deliver content to targeted users from the closest position possible. Currently, there are six regional locations around the world and an additional U.S. location targeted at servicing governmental agencies. Reducing distance to users reduces latency and address resolution.

In addition to geography, S3 provides REST – a rest interface. This will improve cacheing by networks and backbones to reduce latency and therefore speed. There is another side affect of cacheing in less data transfer and thus reduced costs to users of S3.

Interfaces and Standardization:

In addition to the REST interface previously described, the S3 provides SOAP. This is a standard XML-based communication mechanism. Since SOAP is a standard communication mechanism, this provides an easy and flexible mechanism for communication in nearly any coding language. Most languages have SOAP classes for simplifying this communication.

Security:

S3 provides two basic mechanisms for security, which are familiar to most developers. The ACL (Access Control Lists) and IAM (Identity Access Management) allow for fairly fine grain permissions. This fine grain control allows for setting up permissions on a per user level as low as per object or as high as a group to bucket.

S3 also includes a robust mechanism for communication security in SSL. Although SSL is a typical and expected security mechanism, S3 takes this a step further by providing several ways of implementing keys. Developers can use their own keys or Amazon's keys. The implementation for utilizing Amazon is as simple as setting a flag in the request header during upload.

Reliability:

Although the architecture and implementation details are not made public, Amazon S3 boasts 99.999999999% durability and 99.99% availability of objects over a given year [3]. S3 also provides redundancy synchronized with "put() and copy()" commands by storing data across multiple facilities prior to returning a "success." To learn more about the S3 service level agreement, see http://aws.amazon.com/s3-sla/.

Since reliability can have a reduction in performance, S3 addresses this with their Reduced Redundancy Storage or RRS for short. Objects that are not critical for this mechanism of storage can be designated as such. Although the name may suggest decreased reliability, S3 still stores this data across multiple storage facilities and has 400 times the durability of a typical SSD [3]. This mechanism merely reduces the typical S3 storage mechanism.

Implementation:

Amazon does not disclose their architecture, but does, however, describe its proprietary software used for implementing S3. The S3 system utilizes Amazon's Dynamo featured in an article released by their CTO for the Symposium on Operating Systems Principles (SOSPS) [4]. The key feature of Dynamo is its ability to be flexible with storage sizes. Consider storing a "huge" object of (3TB) and then increasing that object size to (5TB). The likelihood of being able to contain this object on the original disk on a distributed system is low. The Dynamo system resolves these types of issues and allows tremendous flexibility and speed.

Google App Engine – Background

Google App Engine, or GAE, was released in 2008 as a cloud platform for hosting web applications using Google managed data centers [5]. Any "cloud" system is some form of distributed computing over multiple servers and data store devices. This allows for greater flexibility and scalability. Google App Engine is larger in scope than Amazon's S3 so we will focus on the data storage components of Google's App Engine.

Google Datastore

Datastore is a schema less datastore, which was designed to be reliable, scalable, and secure. The datastore product is available in two versions: Master/Slave and High Replication Datastore (HRD). The schema less design makes these products extremely comparable to Amazon's S3 product.

Despite some of the benefits, Google products have a downside; their documentation is oftentimes incomplete or hidden in the details. It should be noted that during my research of the Google

Datastore, I uncovered many articles outlining limitations that I did not find directly in the Google documentation. Unfortunately, these articles were not published within the past six months and therefore were not included in this research.

Datastore Features

Basic Structure:

Google Datastore is a schema less object datastore that provides query-based interface and atomic transactions. The datastore distributes content over multiple data centers designed to sustain outages in multiple data centers without any downtime.

The datastore objects are defined as Entities by Google. All Entities have a key, which is comprised of a *kind* and entity ID provided by the datastore. The *kind* attribute is used to facilitate searches and filtering. The *kind* attribute should not be confused with the data type, as entities can have different data types, but the same *kind* [6].

Unlike Amazon S3, each entity, or object, can contain multiple value types that are considered properties of the entity [7]. These value types resemble typical SQL data types in many ways and are designed to make querying simpler and quicker. However, the underlying implementation of the entity is hidden by datastore through either Java or Python. This reduces flexibility in applications and forces at a minimum a layer of Java or Python in any application built using Datastore.

Limits and Restrictions:

Compared to Amazon S3, Google's datastore implementation is rather complicated. In an attempt to add flexibility in pricing and "throttling" for your application, Google has built in the antithesis of scalability. Datastore bills by Datastore API calls, Data Sent to Datastore API, Data Received from Datastore API, and Datastore size. In addition, these all have limitations on a monthly, daily, and per minute basis. This complicated hierarchy starts at the Google Application layer and disseminates to your datastore restrictions, making understanding your limitations very difficult.

Beyond making the process of understanding how you will be billed or what costs may be involved complicated, these limitations are hard. Meaning if you exceed your limits, your application will get shut down. This does not sound like scalability. As stated in Google documentation, *"App Engine by default returns an HTTP 403 Forbidden status code for the request instead of calling a request handler"* [8]. This seems like a painful response for a functioning application that has been too successful at driving traffic.

Speed:

The most notable difference of the Google App Engine is the ability to use a Memcache for storing data that is being retrieved on a regular basis. Through Google's Java and Python API, data can be written to cache. This is a nice feature, but it has very restrictive constraints of 1MB per entry with a maximum of 32MB for entries broken into multiple "buckets" [9].

Interfaces and Standardization:

Datastore features the use of DataNucleus Access Platform [10] and Python and Java APIs. DataNucleus is an Open Source API under the Apache 2 License for standardizing communication between multiple datastores. Standard Java, JDO and JPA, and custom Python package, GQL (Google Query Language), are available for performing queries.

Unfortunately, GQL is a proprietary language created by Google. It would seem unfortunate to create an SQL-like language and not just a subset of SQL. This makes porting code in the future difficult or impossible.

Security:

Typical HTTPS communication protocol is accepted. No other information regarding security was available.

Reliability:

There are two options for Datastore: HRD and Master/Slave. HRD is now the default setup and boasts a 100% up time in 2012 [13]. However, Google only guarantees a 99.95% up time in their SLA [14].

For data retrieval, the Datastore uses transactional queries. However, many limitations are present regarding what transactions can be performed. All transactions must be of the same entity group or for Cross-group Transactions (XG), have a limit of 5 entity groups in a given transaction. Making updates to an entity group locks that entire group during write time. This can be a major limitation if performing XG transactions [15].

Implementation:

Google Datastore is implemented using two proprietary technologies, BigTable and Google File System [16]. Both of these systems are designed to host data across thousands of servers and over multiple data centers.

Compare & Contrast

At the initial construction of this document, it seemed that a one to one comparison could be made across all aspects. However, after detailed review of both technologies, I have found that this comparison is not straight forward and creates a grey area of which storage engine is best in cost, reliability, scalability, and security. Without having personally implemented any major application on these platforms, I cannot provide a complete picture of which technology is best given a specific application.

Below is a cost comparison between Amazon and Google. As might be expected, they are similarly priced. However, Amazon has a straight forward pricing structure that is easy to understand and interpret, whereas Google has a much more complicated structure that requires a detailed review and searching to determine the costs. There are additional costs that are not shown below because they are

not applicable to Amazon S3. For example, there is a minimum weekly cost of $2.10 for all Google paid applications.

Feature	Amazon S3	Google Datastore
Free Transfer	20,000 Put, 20,000 Get	32 GB
Paid Storage (approx.)	$ .125 per GB	$ .130 per GB (blob store)
Paid Transfer (approx.)	$ .120 per GB	$ .120 per GB
Write Cost	$ 10.00 per 100K operations	$ .10 per 100K operations
Read Cost	$ .10 per 100K operations	$ .07 per 100K operations
Incoming Bandwidth	Unlimited	32 MB / request

Figure 1 – Costs

When it comes to reliability and scalability, it is hard to beat either one. Anyone who has ran or managed their own server(s) knows these are impressive figures. In addition, each provides a mechanism for atomic and transactional insertions for data. The most probable downtime will come from Google when you go through either a monthly quota limit or their limited daily or per minute restrictions.

Feature	Amazon S3	Google Datastore
Reliability	99.99%	99.95% (Actual 100% in 2012)
Redundancy	Yes	Yes
Reduced Redundancy	Yes	Yes
Scalable (no limit on size*)	Yes	Yes

***** This refers to overall capacity and not individual objects

Figure 2 – Reliability

When it comes to features, both of these options are strong. Neither delivers the capabilities of a relational system, however, the payoff in scalability is the main purpose of these store engines. Google lacks a little in key features such as restricted access, regional data delivery, and HTTPS key flexibility. These features are critical components of any large scale web application. Regional delivery of content can have huge reductions in delivery time, which is one of the most important features of a good user experience.

Although Google offers a robust feature set and is typically on the forefront of technology, their lack of separation from application to datastore makes a developer tied to Google. Google's forced use of Java, Python, and GQL seems counter to key properties in software development of abstraction and portability.

Feature	Amazon S3	Google Datastore
REST	Yes	Yes (Partial)
Data Types	No	Yes
HTTPS	Yes	Yes
HTTPS Custom Keys	Yes	No
ACL	Yes	No (Google Accounts)
IAM	Yes	Yes (Google Accounts)

Cron Jobs	No	Yes
Stored Procedures	Yes	No
Logging	Yes	No
Language Independent	Yes	No
Relational	No	No
CRUD	Yes	Yes
Regional	Yes	No
Accessible Cacheing Mechanism	No	Yes

Figure 3 – Additional Features

After a review of all features, I would point to Amazon S3 as the preferred storage engine. Although the products are very similar, Amazon has done a better job of producing a product more friendly to a developer. Scouring the web for hours to find basic information regarding Google's Datastore has caused me to question the viability of trying to implement a complex application on their system. In addition, the application has to be implemented on their system through their APIs. This lack of flexibility, poor documentation, and complex explanations make Google a difficult choice.

References

[1] http://www.datarecoverygroup.com/articles/article3.htm

[2] http://willy.boerland.com/myblog/youtube_bandwidth_usage_25_petabytes_per_month

[3] http://aws.amazon.com/s3/

[4] http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

[5] http://en.wikipedia.org/wiki/Google_App_Engine

[6] http://code.google.com/appengine/docs/python/datastore/overview.html

[7] http://code.google.com/appengine/docs/java/datastore/entities.html

[8] http://code.google.com/appengine/docs/quotas.html#Safety_Quotas_and_Billable_Resources

[9] http://code.google.com/appengine/docs/java/memcache/overview.html

[10] http://www.datanucleus.org/products/accessplatform/

[12] http://code.google.com/appengine/docs/python/datastore/gqlreference.html

[13] http://googleappengine.blogspot.com/2012/01/happy-birthday-high-replication.html

[14] http://code.google.com/appengine/sla.html

[15] http://code.google.com/appengine/docs/python/datastore/transactions.html

[16] http://www.ctoedge.com/content/how-google-app-engine-datastore-works