CACSS: TOWARDS A GENERIC CLOUD STORAGE SERVICE

Yang Li, Li Guo and Yike Guo*
Department of Computing, Imperial College London, UK
{yl4709,liguo,yg}@doc.ic.ac.uk

Keywords: Cloud Computing, Cloud Storage, Amazon S3, CACSS

Abstract: The advent of the cloud era has yielded new ways of storing, accessing and managing data. Cloud storage services enable the storage of data in an inexpensive, secure, fast, reliable and highly scalable manner over the internet. Although giant providers such as Amazon and Google have made a great success of their services, many enterprises and scientists are still unable to make the transition into the cloud environment due to often insurmountable issues of privacy, data protection and vendor lock-in. These issues demand that it be possible for anyone to setup or to build their own storage solutions that are independent of commercially available services. However, the question persists as to how to provide an effective cloud storage service with regards to system architecture, resource management mechanisms, data reliability and durability, as well as to provide proper pricing models. The aim of this research is to present an in-depth understanding and analysis of the key features of generic cloud storage services, and of how such services should be constructed and provided. This is achieved through the demonstration of design rationales and the implementation details of a real cloud storage system (CACSS). The method by which different technologies can be combined to provide a single excellent performance, highly scalable and reliable cloud storage system is also detailed. This research serves as a knowledge source for inexperienced cloud providers, giving them the capability of swiftly setting up their own cloud storage services.

1. INTRODUCTION

For many IT professionals, researchers and computer owners, finding enough storage space to hold data is a real challenge. Some people invest in ever larger external hard drives, storing files locally in order to deal with their data growth. However, protecting against the many unexpected things that can happen to computers or hard drives is not a trivial task. Similar issues exist for a number of large and geographically diverse enterprises; they store and process their rapidly growing data in several data centres and may adopt multiple storage systems. Finding and managing files in this increasingly large sea of data is extremely difficult. Although modern distributed computing systems provide ways of accessing large amounts of computing power and storage, it is not always easy for non-experts to use. There is also a lack of support in user defined file metadata in these systems. This has led most users to store a lot of data that strongly relates to the file content in relational databases. This separation of the file data and semantic metadata can create issues of scalability, interoperability and furthermore may result in low performance.

Unlike local storage, cloud storage relieves end users of the task of upgrading their storage devices constantly. Cloud storage services enable inexpensive, secure, fast, reliable and highly scalable data storage solutions over the internet. Leading cloud storage vendors, such as (Amazon) and (Google) , provide clients with highly available, low cost and pay as you go based cloud storage services with no upfront cost. Recently, Amazon announced that it currently holds more than 566 billion objects and that it processes more than 370,000 requests per second at peak times (Barr, 2011). Even so, many enterprises and scientists are still unable to shift into the cloud environment due to privacy, data protection and vendor lock-in issues. In addition, the high read/write bandwidths that are demanded by I/O intensive operations, which occur in many

* Please direct your enquiries to the communication author Professor Yike Guo

different scenarios, cannot be satisfied by current internet connections.

These reasons provide an incentive for organisations to set up or build their own storage solutions, which are independent of commercially available services and meet their individual requirements. However, knowledge of how to provide an effective cloud storage service with regards to system architecture, resource management mechanisms, data reliability and durability, as well as proper pricing models, remains untapped.

In order to reveal this *secret knowledge* behind cloud storage services and thereby a generic solution, we present CACSS, a generic computational and adaptive cloud storage system that adapts existing storage technologies to provide efficient and scalable services. Through a demonstration of CACSS, full details can be given of how a proper cloud storage service can be constructed, in consideration of its design rationale, system architecture and implementation. We also show that our system not only supports the mainstream cloud storage features —such as well supported scalability, sufficient replications and versioning— but that it also supports large scale metadata operations such as metadata searching. These features provide a generic basis that enables potential storage service providers, both internet wide and intranet wide, to setup their "private S3". Furthermore, CACSS is designed at petabyte scale and can be deployed across multiple data centres in different regions around the world. We describe in detail how we manage the separation of file content and file metadata, including user defined metadata. We also demonstrate the implementation of enabling multi-region support for this cloud storage system. This paper demonstrates how different technologies can be combined in order to provide a single and highly superior generic solution.

2. PROBLEM ANALYSIS

2.1 Generic Cloud Storage Features

A cloud storage system should be designed and implemented with consideration of several generic features:

- High scalability and performance: storage demand has increased exponentially in recent years and as such, it is necessary for cloud storage systems to be able to seamlessly and rapidly scale their storage capability for both file content and file metadata. Traditionally, metadata and file data are managed and stored by the same file system and most of the time on the same device. In some modern distributed file systems, to improve scalability and performance, metadata is stored and managed separately by one or more metadata servers (Gibson and Van Meter, 2000). However, many of these still suffer from bottlenecks at high concurrent access rates (Carns et al., 2009). The metadata of a petabyte scale file system could contain in excess of billions of records, consuming terabytes of space or more. Therefore, the efficient metadata management of cloud storage systems is crucial for their overall storage system performance.
- Data durability: a far more common event than hardware failure or a disaster scenario is end user error, by which data is unintentionally deleted or overwritten. This demands that cloud storage systems have sophisticated replication, versioning and recovery mechanisms to restore data.
- Support of various pricing models: the traditional pricing model for software has been a one-time payment for unlimited use. Cloud pricing models, such as pay as you go and pay monthly are very similar to the usage charges of utility companies. In order to accommodate this, it is necessary for cloud storage systems to have an efficient monitoring framework to track all kinds of resource usage, including network data transfer, I/O requests, amounts of data stored (file content and file metadata) and resources consumed for various computations.
- Security models: various security models need to be implemented to ensure that documents and files are accessible at the correct time, location and by the right person, providing adequate and accurate security control over the data without compromising performance.
- Interoperability: no specific standards that enable interoperability between cloud storage vendors currently exist, and this has created problems in the moving of data from one cloud to another. Similar issues arise in converting existing applications that are built on traditional file systems to the cloud. The ideal cloud storage system must offer a level of abstraction, portability and ease of use that enables the consuming of storage services with minimal support and development overhead.

2.2 Cloud Storage for Different Domains

Enterprises and scientists use cloud storage services for various purposes, and files are in different sizes and formats. Some use cloud storage for large video and audio files, and some use it for storing relatively small files that are large in quantity; the variety and range is vast. The different purposes of using cloud storage services give rise to a significant diversity of patterns of access to stored files. The nature of these stored files, in terms of features such as size and format, and the way in which these files are accessed are the main factors that influence the quality of cloud storage services that are eventually delivered to the end users. Some common domains for future cloud storage are as follows:

- Computational storage: many applications in science and enterprise are becoming increasingly demanding in terms of their computational and data requirements. Some applications store terabytes of data and carry out intensive I/O operations. Examples include bioinformatics analysis and log processing applications. Improving the overall performance of such applications often demands a cloud storage system that can bring computational power closer to the data. Amazon Elastic MapReduce service (Amazon) adapts a hosted Hadoop framework running on the infrastructure of Amazon EC2, in order to provide an on-demand service to setup computational tasks and process data that are stored on Amazon S3.
- Small file storage: some large online ecommerce companies and social networking websites store enormous numbers of small files; these are mostly image files, and their numbers are constantly growing. Every second, some of these files are requested at a high rate by public users. As the metadata of a small file may well occupy more space than the file content itself, large concurrent accessing of a small file may cause excessive and redundant I/O operations due to metadata lookups (Beaver et al., 2010). This scenario can sometimes create a bottleneck.
- Metadata operation intensive storage: metadata is the information that describes data files. Common metadata attributes include the time of an event, author's name, GPS location and captions. Many scientists record information about their experimental configuration, such as temperature, humidity and other data attributes; for many it has become an integral part of storage. Accurate identification and adequate support for metadata queries of these metadata would bring additional values to the stored files and ensure that analyses and computations are carried out correctly and efficiently. Unfortunately most storage systems are not capable of efficiently searching file metadata, especially those which are user defined on a large scale. Leading cloud storage providers such as Amazon S3, Cloud Files (Rackspace) and Google Cloud Storage (Google) offer cloud storage services for object data combined with a key-value style of user defined metadata. These metadata can be retrieved and used by user applications. However, none of these cloud storage providers yet support any object search or query based on the object metadata. Some research has already been done on the topic of metadata indexing and searching services (Singh et al., 2003, Leung et al., 2009)

The present study addresses only some of the common domains to which a cloud storage system can be applied; indeed there are more domains that are yet to be studied. In general, it is not easy to make a system that can work for absolutely any domain. However, we can learn through generic features and the challenges that exist in those domains already studied, and eventually we can build towards a more adaptive and generic cloud storage system that can serve different purposes without too much effort.

3. BACKGROUND

Amazon Simple Storage Service (Amazon S3) is an online storage service that aims to provide reliable and excellent performance at a low cost. However, neither its architecture nor its implementation has yet been made public. As such, it is not available for extension in order to develop the capability of creating private clouds of any size.

Amazon S3 is the leading *de facto* standard of bucket-object oriented storage services. Successive cloud storage vendors, such as (Rackspace) and (Google) all adopt s3's style of bucket-object oriented interface. This style hides all the complexities of using distributed file systems, and it has proven to be a success (Barr, 2011). It simply allows users to use the storage service from a higher level: an object contains file content and file metadata, and it is associated with a client assigned

key; a bucket, a basic container for holding objects, plus a key together uniquely identify an object.

4. CACSS DESIGN

Following earlier discussion on key characteristics of the generic private cloud storage system, we now present in detail the design rationale of a CACSS system.

From a conceptive level, the architecture of CACSS has five main components: the access interface, which provides a unique entry point to the whole storage system; the metadata management service, which manages the object metadata; the object operation management service, which handles a wide range of object operation requests; the metadata storage space, which stores all of the object metadata; and the object data storage space, which stores all of the object content data (Figure 1).

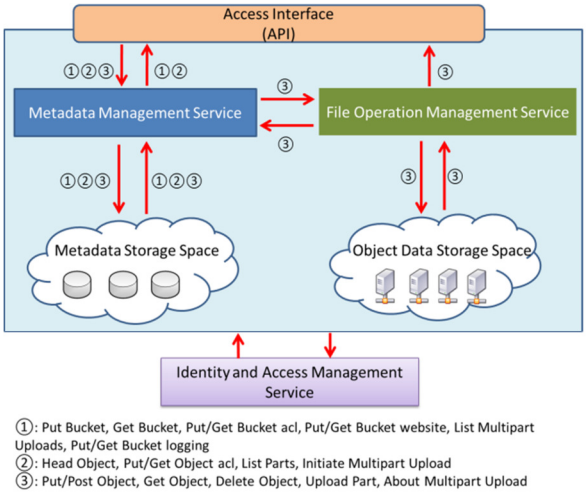

Figure 1 CACSS Architecture

4.1 Access Interface

CACSS offers a web-based interface for managing storage space and searching for objects. The current implementation supports Amazon's S3 REST API, the prevailing standard commercial storage cloud interface.

4.2 Identity and Access Management service

IAM is a separated service that provides authorization and access control of various resources. It offers sub user, group management and precise permission control of which operations a user can perform and under what conditions such operations can be carried out.

4.3 Metadata Management

To achieve high performance in metadata access and operation, CACSS's object metadata and content are completely separated. Each object's metadata—including its system metadata such as size, last date modified and object format, together with user defined metadata—are all stored as a collection of blocks addressed by an index in CACSS's Metadata Storage Space (MSS). MSS keeps all of the collections' data sorted lexicographically by index. Each block is akin to a matrix which has exactly two columns and unlimited rows. The values of the elements in the first and second columns are block quantifiers and block targets, respectively. All of the block quantifiers have unique values in each block:

$$\text{Block}_A=[a_{i,j}]\ 1\le i\le m,\ 1\le j\le 2, \text{ for any } k,s \in m, \text{ where } k\neq s,\ a_{k,1}\neq a_{s,1}$$

E.g. an index of W maps to a collection:

$$\left(\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \\ \vdots & \vdots \end{bmatrix} \cdots \begin{bmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \end{bmatrix}\right)$$

4.3.1 Metadata Management Service

MMS manages the way in which an object's metadata is stored. In such a system a client will consult the CACSS MMS, which is responsible for maintaining the storage system namespace, and they will then receive the information specifying the location of the file contents. This allows multiple versions of an object to exist.

MMS handles requests as follows. First, it checks if a request contains an access key and a signed secret key. CACSS consults AIM and MSS to verify whether the user has the permission to perform the operation. If they do have permission, the request is authorized to continue. If they don't, error information is returned. If a request does not contain an access key or a signed secret key, MMS is looked up to verify if the request to the bucket or object is set as publicly available to everyone. If it is set as public, then the request continues to the next step. All the requests are logged, both successful and failed. The logging data can be used by both the service provider and storage users for billing, analysis and diagnostic purposes.

Differing from traditional storage systems that limit the file metadata which can be stored and accessed, MMS makes metadata more adaptive and

comprehensive. Additional data regarding file and user-defined metadata can be added to the metadata storage, and these data can be accessed and adopted on demand by users or computational works at any time. Searching via metadata is another key feature of CACSS.

4.3.1.1 Buckets

To reduce interoperability issues, CACSS adopts the *de facto* industry standard of buckets as basic containers for holding objects.

Unlike some traditional file systems, in which a limited number of files can be stored in a directory, there is no limit to the number of objects that can be stored in a CACSS bucket. CACSS has a global namespace—bucket names are unique and each individual bucket's name is used as the index in MSS. We use various block quantifiers and block targets to store a variety of information, such as properties of a bucket or an object, permissions and access lists for a particular user, and other user defined metadata.

For example, for a bucket named "bucket1", an index "bucket1" should exist, which maps to a collection of data such as:

$$\left(\begin{array}{c}\begin{bmatrix} pp{:}key & \text{bucket1} \\ pp{:}owner & userid1 \\ pp{:}region & uk1 \\ pp{:}web & page.html \\ pp{:}type & bucket \end{bmatrix} \\ [pm{:}userid2 \quad READ;\ READ\_ACP;] \\ [um{:}info \quad this\ is\ a\ bucket]\end{array}\right)$$

4.3.1.2 Objects:

The index of each object is comprised of a string, which has the format of the bucket name together with the assigned object key. As a result of this nomenclature, objects of the same bucket are naturally very close together in MSS; this improves the performance of concurrent metadata access to objects of the same bucket.

For example, considering an object with the user-assigned key "object/key.pdf" in bucket "bucket1", an index of "bucket1- object/key.pdf" should exist, which maps to the following collection of data:

$$\left(\begin{array}{c}\begin{bmatrix} pp{:}key & object/key.pdf \\ pp{:}owner & userid1 \\ pp{:}loc & hdfs://cluster1/bucket1/uuid\ldots \\ pp{:}type & object \end{bmatrix} \\ [pm{:}userid1 \quad FULL\_CONTROL;] \\ \begin{bmatrix} um{:}author & some\ author \\ um{:}year & 2011 \end{bmatrix}\end{array}\right)$$

4.3.2 Object Versioning:

When versioning setting is enabled for a bucket, each object key is mapped to a core object record. Each core object record holds a list of version IDs that map to individual versions of that object.

For example, for an object with a predefined key "object/paper.pdf" in bucket "versionbucket", an index of $\text{"versionbucket} - \text{object/paper.pdf"}$ should exist, which maps to the collection data:

$$\left(\begin{array}{c}\begin{bmatrix} pp{:}key & object/paper.pdf \\ pp{:}owner & userid1 \\ pp{:}type & object \end{bmatrix} \\ \begin{bmatrix} ver{:}lastest & uuid1 \\ ver{:}(versionbucket) & uuid1 \\ ver{:}(versionbucket) & uuid2 \end{bmatrix}\end{array}\right)$$

Similarly, the object's version record with row key "versionbucket-object/paper.pdf-uuid1" maps to the collection data:

$$\left(\begin{array}{c}\begin{bmatrix} pp{:}loc & nfs://cluster1/versionbucket/uuid... \\ pp{:}type & version \\ pp{:}replicas & 2 \end{bmatrix} \\ [pm{:}userid2 \ \ READ;]\end{array}\right)$$

4.4 Object Data Management

CACCS stores all the unstructured data, such as file content, in the Object Data Storage Space (ODSS). ODSS is intentionally designed to provide an adaptive storage infrastructure that can store unlimited amounts of data and that does not depend on underlying storage devices or file systems. Storage service vendors are able to compose one or multiple types of storage devices or systems together to create their own featured cloud storage system based on their expertise and requirements in terms of level of availability, performance, complexity, durability and reliability. Such implementation could be as simple as NFS (Sandberg et al., 1985), or as sophisticated as HDFS (Borthakur, 2007), PVFS (Carns et al., 2000) and Lustre (Schwan, 2003).

CACSS's File Operation Management Service (FOMS) implements all ODSS's underlying file systems' API, so that it can handle a wide range of file operation requests to the ODSS. FOMS works like an adapter that handles the architectural differences between different storage devices and file systems. It works closely with MMS to maintain the whole namespace of CACSS. FOMS also possesses the capability of utilising all the available resources by performing various object allocation strategies, which are based on factors such as objects' types, sizes and even their previous usage patterns.

5. IMPLEMENTATION

After considerable research and experimentation, we chose HBase as the foundational MSS storage for all object metadata. HBase is highly scalable and delivers fast random data retrieval. Its column-orientation design confers exceptional flexibility in the storing of data.

We chose Hadoop DFS (HDFS) as the foundational storage technology for storing object data in ODSS. Hadoop also supports MapReduce framework (Apache) that can be used for executing computation tasks within the storage infrastructure. Although there is a single point of failure at the NameNode in HDFS's original design, many research studies have been carried out in order to build a highly available version of HDFS NameNode, such as AvatarNode (Borthakur, 2010). Every file and block in HDFS is represented as an object in the NameNode's memory, each of which occupies about 150 bytes. Therefore the total memory available on NameNode dictates the limitation of the number of files that can be stored in the HDFS cluster. By separating object metadata and object data, CACSS is able to construct an adaptive storage infrastructure that can store unlimited amounts of data using multiple HDFS clusters, whilst still exposing a single logical data store to the users (Figure 3).

5.1 Multi-region support

The design and architecture of CACSS are based on the principles of scalability, performance, data durability and reliability. Scalability is considered in various aspects including the overall capacity of multi-region file metadata and file storage, as well as throughput of the system. Taking another perspective, the implementation of CACSS consists of a region controller and multiple regions (Figure 2).

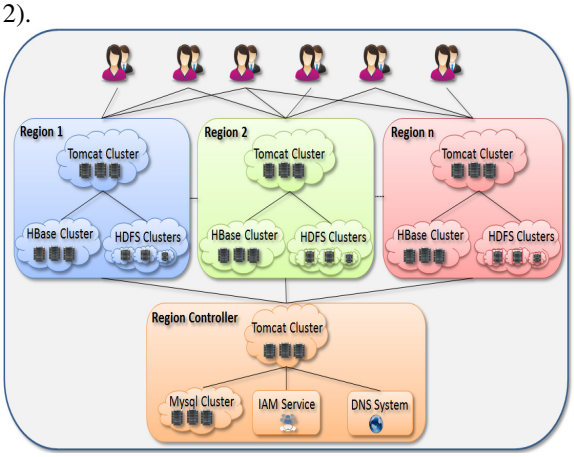

Figure 2: Implementation of CACSS

A Tomcat cluster is used as the application server layer in each region. It is easy to achieve high scalability, load balancing and high availability by using a Tomcat cluster and configuring with other technologies such as HAProxy and Nginx (Doclo, 2011, Mulesoft).

The region controller has a MySQL cluster for storing various data such as user account information and billing and invoice details.

A bucket can be created in one of the regions. At the same time, a DNS A record is also inserted into the DNS server. This mapping ensures that clients will send a hosted-style access request of the bucket and the object to the correct region. Each region is consistent with a Tomcat cluster, an HBase cluster and a set of HDFS clusters. The object data is stored in one of the HDFS clusters in the region. The object key and metadata are stored in the region's HBase cluster. It is always important to consider that any access to a bucket or object requires access rights to be checked. In CACSS, each request goes through its region first; if the requested bucket or object is set to be public, there is no need to communicate with the region controller. If it is not set as public, it consults the region controller to perform the permission check before making a response. The region controller, which includes a MySQL cluster, keeps records of all the requests and maintains user accounts and billing information. A DNS system (such as Amazon Route 53 (Amazon)) serves to map the bucket name to its corresponding region's Tomcat cluster IP. The region controller can also connect to the existing IAM service to provide more sophisticated user and group management.

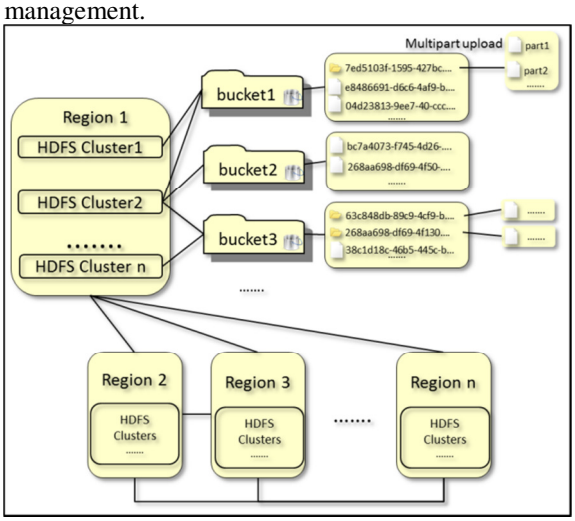

Figure 3: Implementation multi-region HDFS clusters for storing buckets and contents of objects

CACSS also adopts other useful features of HDFS such as no explicit limitation on a single file size and no limitation on the number of files in a directory. In CACSS, most of the objects are stored in a flat structure in HDFS. Each object's file name under HDFS is a generated UUID to ensure uniqueness.

The implementation of CACSS does not need to rely solely on HDFS. The separation of file metadata entirely from file content enables CACSS to adapt to one or even multiple file systems, such as GPFS or Lustre. It is now deployed as a service under IC-Cloud platform (Guo and Guo, 2011), and is expected to work with a variety of distributed file systems through POSIX or their APIs without much effort.

6. EXPERIMENTS

We performed our experiments on top of Amazon EC2 instances, to enable the comparison of CACSS and Amazon S3 under similar hardware and network environments. We used (JetS3t), an open source Java S3 library, configuring it with our experiment code to evaluate the performance of CACSS.

We used one m2.xlarge instance, with 17.1GB of memory and 6.5 EC2 Compute Units, to run MySQL, HDFS NameNode, HBase Hmaster and Tomcat with the CACSS application. Three m1.large instances, each with 7.5GB memory and 4 EC2 Compute Units ran HDFS DataNodes and HBase Regionservers. Each of these instances was attached with 100GB volumes of storage space. Another two m1.large instances were configured with the same experiment code but different S3 end points. We refer to these two instances as "S3 test node" and "CACSS test node".

To evaluate the performance of CACSS, we ran a series of experiments on both Amazon S3 and CACSS. The evaluation of the performance of Amazon EC2 and S3 has been carried out previously by (Garfinkel, 2007). A similar method was adopted here to evaluate the overall throughput of CACSS.

Figure 4 and Figure 5 illustrate respectively the write and read throughputs of Amazon EC2 to Amazon S3, and of EC2 to CACSS, based on our experiments. Each graph contains traces of observed bandwidths for transactions of 1Kbyte, 1Mbyte, 100Mbyte and 1Gbyte. Both Amazon S3 and CACSS perform better with larger transaction sizes, because smaller size files would experience more transaction overhead. For files larger than 1Mbyte, the average speed of transaction of CACSS is higher than Amazon S3; this is probably due to underlying hardware differences between Amazon EC2 and Amazon S3, such as hard drive RPM and RAID levels.

Amazon S3's List Objects operation only supports a maximum of 1000 objects to be returned at a time, so we could not properly evaluate its object metadata service performance. However, we were able to run some tests to see how CACSS's metadata management performs. We ran a List All Objects operation after every 1000 Put Object operations. All of the operations were targeted to the same bucket. Each Put Object was done using an empty file, because we were only interested in the performance of the metadata access in this experiment. Figure 6 shows a scatter graph of the response time of each Put Object, with respect to the total number of objects in the bucket. The result shows an average response time of 0.007875s and a variance of 0.000157s for each Put Object operation. This indicates that the response time is pretty much constant no matter how many objects are stored in the bucket. Figure 7 illustrates the response time of each List All Objects operation with respect to the total number of objects contained in the bucket. There are several peaks in the graph which have been marked with a red circle. These peaks are caused by sudden network latency between Amazon EC2 instances during that time. Otherwise, the overall result shows a linear relation between the response time and the total number of objects.

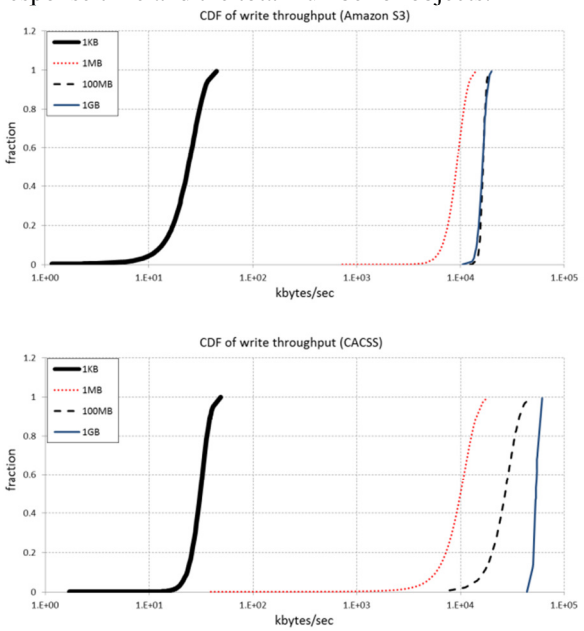

Figure 4: Cumulative Distribution Function (CDF) plots for writing transactions from EC2 to Amazon S3 and CACSS of various sizes.

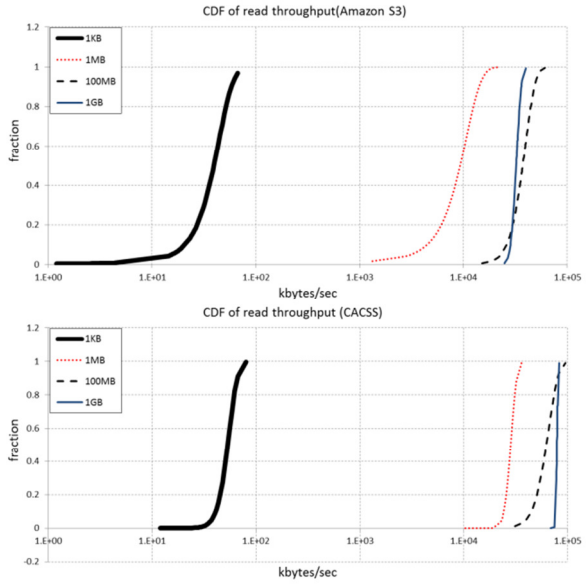

Figure 5: CDF plots for reading transactions from EC2 to Amazon S3 and CACSS of various sizes.

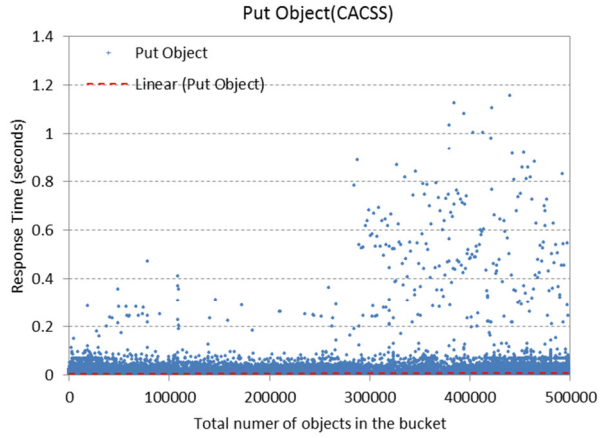

Figure 6: Put Object requests

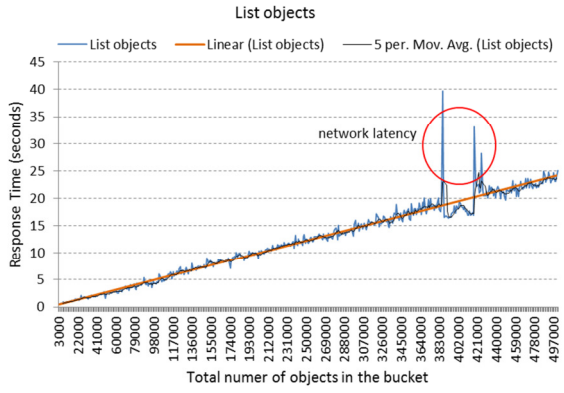

Figure 7: List all objects requests

7. RELATED WORK AND DISCUSSION

Besides Amazon S3, there have been quite a few efforts in cloud storage services, including the following.

Walrus (Nurmi *et al.*, 2009) is a storage service included with Eucalyptus that is interface-compatible with Amazon S3. The open source version of Walrus does not support data replication services. It also does not fully address how file metadata is managed and stored.

The Openstack (Openstack) project has an object storage component called Swift, which is an open source storage system for redundant and scalable object storage. However, it does not support object versioning at present. The metadata of each file is stored in the file's extended attributes in the underlying file system. This could potentially create performance issues with a large number of metadata accesses.

pWalrus (Abe and Gibson, 2010) is a storage service layer that integrates parallel file systems into cloud storage and enables data to be accessed through an S3 interface. pWalrus stores most object metadata information as the file's attributes. Access control lists, object content hashes (MD5) and other object metadata are kept in .walrus files. If a huge number of objects are stored under the same bucket, pWalrus may be inefficient in searching files based on certain metadata criteria; this factor can cause bottlenecks in metadata access.

Cumulus (Bresnahan *et al.*, 2010) is an open source cloud storage system that implements the S3 interface. It adapts existing storage implementations to provide efficient data access interfaces that are compatible with S3. However, details of metadata organisation and versioning support are not fully addressed.

Hadoop Distributed File System (HDFS) (Borthakur, 2007) is a distributed, reliable, scalable and open source file system, written in Java. HDFS achieves reliability by replicating data blocks and distributing them across multiple machines.

(HBase) is an open source, non-relational, versioned, column-oriented distributed database that runs on top of HDFS. It is designed to provide fast real time read/write data access. Some research has already been done to evaluate the performance of HBase (Carstoiu et al., 2010) (Khetrapal and Ganesh, 2006).

Table 1 shows a comparison of the support features of Amazon S3, Google Cloud Storage and CACSS. Many features are shared, with CACSS having the additional capability of metadata and user defined data searching.

Table 1

	Amazon S3	Google Cloud Storage	CACSS
Bucket region support	Yes	Yes	Yes
Security control	ACL; access key with signature; policy	ACL; OAuth access	ACL; access key with signature
Large file upload	Multi part upload	Resumable upload	Multi part upload
Object immutable	Yes	Yes	Yes
Host static website	Yes	No	Yes
Versioning	Yes	No	Yes
Access logs	Yes	Yes	Yes
Random read support	Yes	Yes	Yes
Random write support	No	No	No
Search support	Object key only	Object key only	Object key, system metadata and user-defined data
Pricing	Storage space, network usage and number of requests	Storage space, network usage and number of requests	N/A
SLA	99.9% uptime and 99.999999999 % durability	99.9% uptime	N/A

8. CONCLUSIONS AND FUTURE WORK

We have presented the design and implementation of CACSS, a cloud storage system based on the generic principles of scalability, performance, data durability and reliability. CACSS not only enables users to access their data through the S3 interface, the *de facto* industry standard, but also provides support for additional features that make the storage service more comprehensive. These features include user defined metadata and object metadata searching. The storage model we propose offers service providers considerable advantage in combining existing technologies to compose a single customized cloud storage system. Furthermore, CACSS performance was found to be comparable to Amazon S3 in formal tests, with similar read/write capabilities. Although other features were difficult to compare directly, CACSS performance was highly adequate in terms of Put Object and List All Object requests.

There are several directions of potential future studies. In any science and enterprise environment, data experiences rapid growth. Some of these data are very active in that they need to be retrieved very frequently, and conversely much other data are hardly accessed. In light of this, a potentially useful research direction is the extension of current work to enable a more application-aware cloud storage service. This would include a data movement framework that enables users or applications to automatically and manually archive inactive data in lower cost storage space, whilst upgrading highly active data to higher performance storage space. This should enhance capacity on demand cloud storage services to a higher level, towards performance on demand cloud storage services.

Further study could also address the issue that current cloud storage systems do not offer much support for "in-house" data-centric services, such as processing and converting existing stored data on top of the storage infrastructure. This drives users into an inefficient workflow pattern of downloading data from cloud storage, dealing with the data externally, and then uploading the processed data back to cloud storage. Although parallel computing (Kumar, 2002) and MapReduce (Dean and Ghemawat, 2008) provide ways of effectively executing computational tasks distributed on a large scale, these implementations are not always easy for non-experts to setup or use.

As a possible solution to this lack of in-house services, a cloud storage system could be developed with a framework that enables different storage oriented services and operations to be "plugged in". This framework should implement sophisticated data and computing resource sharing mechanisms in order to allow stored data to be used by those plugged in services in a controlled and safe manner. In such a way, it is possible to create a cloud storage environment for a user community. For example, a user can publish an application, describing the specific computational task in an application repository; another user is then able to process the same task in a way that requires little or no knowledge about the computation, simply by specifying which published application to execute.

REFERENCES

ABE, Y. & GIBSON, G. pWalrus: Towards better integration of parallel file systems into cloud storage. 2010. IEEE, 1-7.

AMAZON. *Amazon Elastic MapReduce* [Online]. Available: http://aws.amazon.com/elasticmapreduce/.

AMAZON. *Amazon Simple Storage Service (S3)* [Online]. Available: http://aws.amazon.com/s3/.

AMAZON. *Route 53* [Online]. Available: http://aws.amazon.com/route53/.

APACHE. *Hadoop MapReduce* [Online]. Available: http://hadoop.apache.org/mapreduce/.

BARR, J. 2011. Available from: http://aws.typepad.com/aws/2011/10/amazon-s3-566-billion-objects-370000-requestssecond-and-hiring.html.

BEAVER, D., KUMAR, S., LI, H. C., SOBEL, J. & VAJGEL, P. 2010. Finding a needle in Haystack: Facebook's photo storage. *Proc. 9th USENIX OSDI*.

BORTHAKUR, D. 2007. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*.

BORTHAKUR, D. 2010. Hadoop avatarnode high availability. Available from: http://hadoopblog.blogspot.com/2010/02/hadoop-namenode-high-availability.html.

BRESNAHAN, J., KEAHEY, K., FREEMAN, T. & LABISSONIERE, D. 2010. Cumulus: an open source storage cloud for science. *SC10 Poster*.

CARNS, P., LANG, S., ROSS, R., VILAYANNUR, M., KUNKEL, J. & LUDWIG, T. Small-file access in parallel file systems. 2009. IEEE, 1-11.

CARNS, P. H., LIGON III, W. B., ROSS, R. B. & THAKUR, R. PVFS: A parallel file system for Linux clusters. 2000. USENIX Association, 28-28.

CARSTOIU, D., CERNIAN, A. & OLTEANU, A. Hadoop Hbase-0.20.2 performance evaluation. New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on, 11-13 May 2010 2010. 84-87.

DEAN, J. & GHEMAWAT, S. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM,* 51**,** 107-113.

DOCLO, L. 2011. Clustering Tomcat Servers with High Availability and Disaster Fallback. Available from: http://java.dzone.com/articles/clustering-tomcat-servers-high.

GARFINKEL, S. L. An evaluation of amazon's grid computing services: EC2, S3, and SQS. 2007. Citeseer.

GIBSON, G. A. & VAN METER, R. 2000. Network attached storage architecture. *Communications of the ACM,* 43**,** 37-45.

GOOGLE. *Google Cloud Storage Service* [Online]. Available: http://code.google.com/apis/storage/.

GUO, Y.-K. & GUO, L. 2011. IC cloud: Enabling compositional cloud. *International Journal of Automation and Computing,* 8**,** 269-279.

HBASE, A. Available: http://hbase.apache.org/.

JETS3T. *JetS3t* [Online]. Available: http://jets3t.s3.amazonaws.com.

KHETRAPAL, A. & GANESH, V. 2006. HBase and Hypertable for large scale distributed storage systems. *Dept. of Computer Science, Purdue University*.

KUMAR, V. 2002. *Introduction to parallel computing*, Addison-Wesley Longman Publishing Co., Inc.

LEUNG, A. W., SHAO, M., BISSON, T., PASUPATHY, S. & MILLER, E. L. 2009. Spyglass: fast, scalable metadata search for large-scale storage systems. *Proccedings of the 7th conference on File and storage technologies.* San Francisco, California: USENIX Association.

MULESOFT. Tomcat Clustering - A Step By Step Guide. Available from: http://www.mulesoft.com/tomcat-clustering.

NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L. & ZAGORODNOV, D. The eucalyptus open-source cloud-computing system. 2009. IEEE, 124-131.

OPENSTACK. Available: http://openstack.org.

RACKSPACE. *Cloud Files* [Online]. Available: http://www.rackspace.co.uk.

SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D. & LYON, B. Design and implementation of the Sun network filesystem. 1985. 119-130.

SCHWAN, P. Lustre: Building a file system for 1000-node clusters. 2003.

SINGH, G., BHARATHI, S., CHERVENAK, A., DEELMAN, E., KESSELMAN, C., MANOHAR, M., PATIL, S. & PEARLMAN, L. A metadata catalog service for data intensive applications. 2003. IEEE, 33-33.