The final publication is available at:
http://link.springer.com/chapter/10.1007/978-3-319-04519-1_7

An Efficient and Performance-Aware Big Data Storage System

Yang Li, Li Guo and Yike Guo

Department of Computing, Imperial College London, UK
{yl4709,liguo,yg}@doc.ic.ac.uk

Abstract. Recent escalations in Internet development and volume of data have created a growing demand for large-capacity storage solutions. Although Cloud storage has yielded new ways of storing, accessing and managing data, there is still a need for an inexpensive, effective and efficient storage solution especially suited to big data management and analysis. In this paper, we take our previous work one step further and present an in-depth analysis of the key features of future big data storage services for both unstructured and semi-structured data, and discuss how such services should be constructed and deployed. We also explain how different technologies can be combined to provide a single, highly scalable, efficient and performance-aware big data storage system. We especially focus on the issues of data de-duplication for enterprises and private organisations. This research is particularly valuable for inexperienced solution providers like universities and research organisations, and will allow them to swiftly set up their own big data storage services.

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

1 Introduction

The truth is that data growth is rapidly outpacing our ability to store, process and analyse the data we are collecting. Cloud storage relieves end users of the task of constantly upgrading their storage devices. Cloud storage services offer inexpensive, secure, fast, reliable and highly scalable data storage solutions over the internet. Many enterprises and personal users with limited budgets and IT resources are now outsourcing storage to cloud storage service providers, in an attempt to leverage the manifold benefits associated with cloud services. Leading cloud storage vendors, such as Amazon S3 [1] and Google Cloud Storage[2] , provide clients with highly available, low cost and pay-as-you-go based cloud storage services with no upfront cost. A variety of companies have outsourced at least a portion of their storage infrastructure to Amazon AWS, including SmugMug[3], ElephantDrive[4], Jungle Disk[5] and 37signals[3]. Recently, Amazon announced that as of June 2012 it currently holds more than a trillion objects, and the service has so far been growing exponentially [6]. 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. An Amazon S3

storage service outage in 2008 left many businesses that rely on the service offline for several hours and resulted in the permanent loss of customer data, [7, 8], an incident that led many to question the S3's "secret" architecture.

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 large quantities of relatively small files; 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. Another challenge to the data storage community is how to effectively store data without taking the exact same data and storing it again and again in different locations and storage devices. Data de-duplication and other methods of reducing storage consumption play a vital role in affordably managing today's explosive growth of data. However, no much research has been done on how to efficiently apply these methods to big data services.

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 efficient big data storage service with regards to system architecture, resource management mechanisms, data reliability and durability, as well as how to utilise all the resources, reduce storage consumption, costs of backup and improve the quality of the services remains untapped.

Taking one step beyond our previous work [9] to target large-scale data de-duplication for enterprises and private organisations, we present the new CACSS, an efficient and performance-aware big data storage system offering not only mainstream cloud storage features, but global object data de-duplication and data caching services specifically suited to big data management and analysis. A thorough demonstration of CACSS can offer full details on how to construct a proper big data storage service, including design rationale, system architecture and implementation. This paper demonstrates how different technologies can be combined in order to provide a single and highly superior generic solution.

2 Related Work and Problem Analysis

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 [11] and Google Cloud Storage [2] 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 [12]. 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 to uniquely identify an object.

The cloud provides a new way of storing and analysing Big Data because it is both elastic and cost-efficient. Additional computational resources can be allocated on the fly to handle increased demand and organizations only pay for the resource that they need. However, companies that work with big data have been unable to realize the full potential of the cloud due to the Internet connections used to move big data in, out and across cloud infrastructures are not quite as elastic. In addition, the high read/write bandwidths that are demanded by I/O intensive operations, which occur in many different Big Data scenarios, cannot be satisfied by current internet connections [13, 14].

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

The Openstack [15] 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.

Walrus [16] 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.

pWalrus [17] 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 [18] 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) [19] 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 [20] 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 [21] [22].

For the past four decades, disk-based storage system performance has not improved as quickly as its capacity. As a result, many large-scale web applications are keeping a lot of their data in RAMs, and the role of RAM in storage systems has steadily increased over recent years. For example, as of 2008 Facebook used over 28

terabytes of memory[23], and major Web search engines such as Google and Yahoo keep their search indexes entirely in memory[24]. Google's Bigtable storage system [25] allows entire column families to be loaded into memory where they can be read without disk accesses. RAMCloud[26] is a DRAM-based storage system that provides inexpensive durability and availability by recovering quickly after crashes.

Data de-duplication is a data compression technique for eliminating duplicate copies of redundant data. The de-duplication technology has been widely applied in disk-based secondary storage systems to improve cost-effectiveness via space efficiency. It is most effective in storage systems where many duplicates of very similar or identical data are stored. Many studies on block-level and file-level data de-duplication have been carried out. One of the challenges facing large-scale de-duplication enabled storage systems is duplicate-lookup created bottlenecks due to metadata and actual file data which is stored separately and the large size of the data index, which limits the de-duplication throughput and performance[27-33].

CACSS is currently deployed on top of the IC-Cloud[34] infrastructure and is being used by over 200 internal students, especially those enrolled in the "Distributed Systems and Cloud Computing" course. Several assignments, individual and group projects rely heavily on the CACSS API to manage their data. Some other external collaborators are also using CACSS as their data backup space. By monitoring the data access patterns and analysing the actual data stored in our system, we discovered two important characteristics that might help improve our system's efficiency and performance. We discovered that while some files were used intensively over a very short period, much other data were hardly accessed. We also found over 20% duplicated objects with the same checksums stored in our system. This issue of redundancy is common and exists in many enterprises: a survey by AFCOM found that over 63% of IT managers surveyed have seen a significant increase in their storage costs. One of the main reasons for that dramatic increase is file sharing across different endpoint devices and collaboration tools creating large amounts of data duplication.

These discoveries have motivated us to determine how we can improve performance and make CACSS more efficient. Increasing the efficiency and effectiveness of storage environments helps organizations improve their competitiveness by removing constraints on data growth, improving their service levels, and maintaining better leverage over the increasing quantity and variety of data. While much research has been done on data de-duplication and data caching in traditional file storage systems, there is still a lack of research and evaluation for the big data environment in which security, performance and reliability are becoming more crucial. Therefore we decided to add in-line file-level de-duplication and object caching features to our cloud storage system and evaluate them from the real environment.

3 System Design

The architecture of CACSS is shown in **Fig. 1**. From a conceptive level, it consists of the following components:

- Access interface: provides a unique entry point to the whole storage system
- Metadata management service: manages the object metadata and permission controls.
- Metadata storage space: stores all of the object metadata and other related data.
- Object operation management service: handles a wide range of object operation requests.
- De-duplication controller: manages global inline data de-duplication.
- Object caching controller: provides data caching as a service.
- Object data storage space, global object storage space and object caching space,: store all of the object content data in different circumstances.

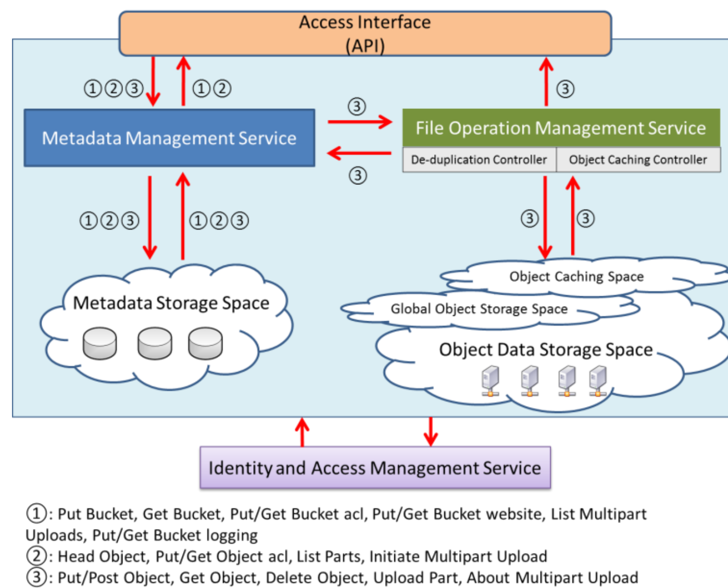


Fig. 1. CACSS Architecture

3.1 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.

3.2 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.

3.3 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:

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

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

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

3.4 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.

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 objects can be stored in a directory, a CACSS bucket has no limit. CACSS has a global namespace—bucket names are unique and each individual bucket's name is used as

the index in the 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:

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

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 in the same bucket.

For example, for 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:

$$\left(\begin{matrix} \begin{bmatrix} pp: key & object/key.pdf \\ pp: owner & userid1 \\ pp: loc & hdfs://cluster1/bucket1/uuid\ldots \\ pp: type & object \\ pp: bucket & bucket1 \\ pp: sha2 & 3c20dc4a766c6df7d20\ldots \\ pp: md5 & 20ad802b8df7c9fc\ldots. \end{bmatrix} \\ [pm: userid1 \quad FULL_CONTROL;] \\ \begin{bmatrix} um: author & some\ author \\ um: year & 2011 \end{bmatrix} \end{matrix}\right)$$

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 "versionbucket − object/paper. pdf" should exist, which maps to the collection data:

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

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

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

3.5 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 [35], or as sophisticated as HDFS [19], PVFS [36] and Lustre [37].

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 various storage devices and file systems. It works closely with MMS to maintain the whole namespace of CACSS.

File Level Data De-duplication.
To enable data storage efficiency, CACSS has introduced a De-duplication Controller (DDC) and a Global Object Storage Space (GOSS) into the design. Currently, the DDC is only implemented to use a global file-level de-duplication method, which manages how and where duplicated objects should be stored in the GOSS. If a bucket is configured to enable data de-duplication, all the objects in this bucket will be stored in the GOSS. It is extremely unlikely to have a collision between two files with different content but the same SHA-256 checksum [38]. Therefore, CACSS uses SHA-256 hash function to calculate the checksum of each incoming storing object, and if the checksum does not exist in the MSS, a new de-duplication object metadata record with $ddes and the hash value as the key will be inserted. The physical file content will be saved into the GOSS as a new file. If the checksum already exists in the MSS, the record will be updated to attach this object's bucket name, object key and the user id (**Fig. 2**).

$$\text{e.g. Row key "}\$ddes - D3F4D74F814D55EE80\ldots\text{" maps to records:}$$

$$\left(\begin{bmatrix} pp{:} < bucket1 >< user1 > key1 & \\ pp{:} < bucket2 >< user1 > key2 & \\ pp{:} < bucket3 >< user2 > key3 & \\ & \\ pp{:}filesize & 19751324 \end{bmatrix}\right)$$

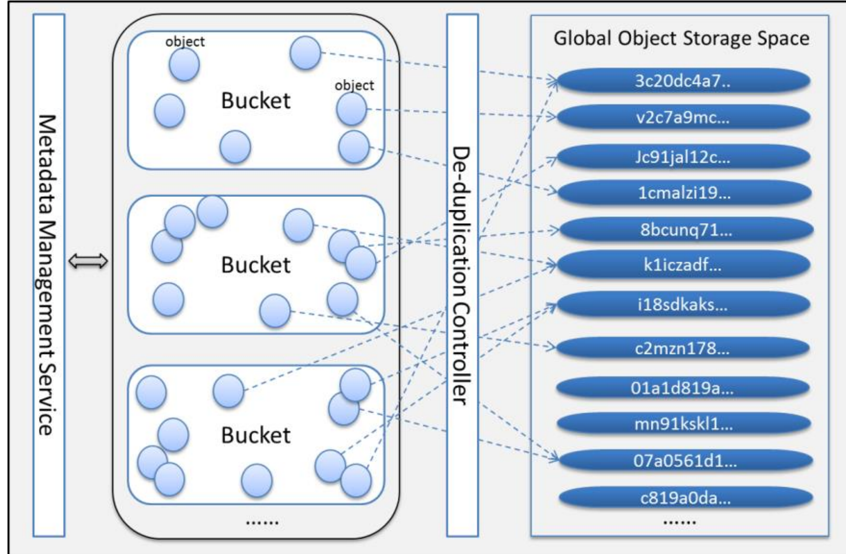


Fig. 2. Bucket-Object to Global Object Storage Space mapping

Object Data Caching Facility.
The ODCF contains the Object Caching Controller (OCC) and Object Caching Space (OCS). With CACSS's OCDF, it is possible to cache certain frequently-accessed objects into the OCS. Such space provides multi-level cache capability, accelerating data access. It can be implemented by a mix of RAM, SSD and other high speed storage devices. The OCC keeps records of all object data accesses, and manages a global object access ranking table. Depending on the total spaces available on the OCS, the OCC intelligently decides which top accessed object data should be cached and where it should be cached. For example, the most accessed object data will be copied into RAM, while medium accessed object data or frequently accessed large files that cannot fit in to the RAM will reside in the SSD. When an object access request is received, the OCC first checks to determine if the latest object data are located in the OCS; if not, data will be returned from the ODSS or the GOSS. We have also enabled the "Caching on Demand" service feature, which allows users to specify exactly which object data should be cached.

4 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 the ODSS. Hadoop also supports MapReduce framework [39] 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 [40]. 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 (**Fig. 4**). Using Linux Ram Disk technique, we employ server RAMs in Tomcat Clusters to serve as the Object Caching Space.

4.1 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 (**Fig. 3**).

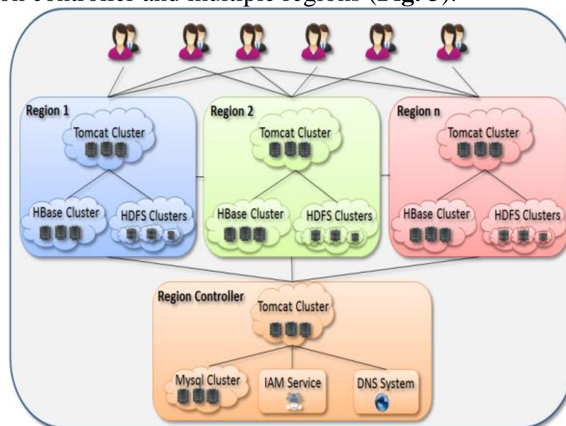


Fig. 3. Implementation of CACSS

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 [41, 42]. 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, and at the same time, a DNS A record is 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 [43]) 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.

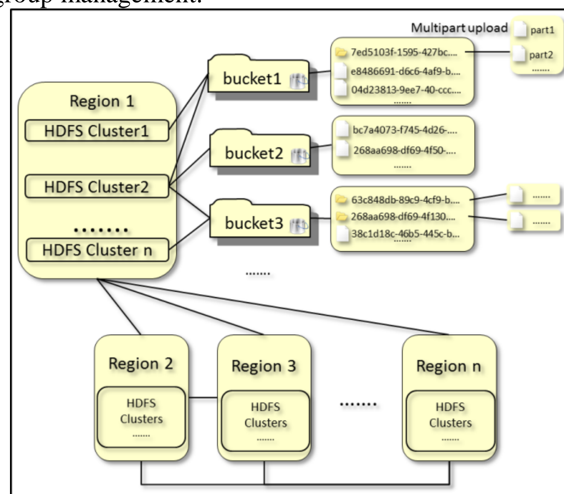


Fig. 4. Implementation multi-region HDFS clusters for storing buckets and contents of objects

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 complete separation of file metadata 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 the IC-Cloud platform, and is expected to work with a variety of distributed file systems through POSIX or their APIs without much effort.

5 Experiments

We have done two sets of experiments so far. The first set was performed on top of Amazon EC2 instances, to enable the comparison of CACSS and Amazon S3 under similar hardware and network environments. We used JetS3t [44], 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 [10]. A similar method was adopted here to evaluate the overall throughput of CACSS.

Fig. 5 and **Fig. 6** 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 1KB, 1MB, 100MB and 1GB. Both Amazon S3 and CACSS perform better with larger transaction sizes, because smaller size files would require more transaction overhead. For files larger than 1MB, 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 evaluate CACSS's metadata management. 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 utilised using an empty file, because for this experiment we were only interested in the performance of the metadata access. **Fig. 8** 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. **Fig. 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 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.

The second set of experiments was performed on top of the IC-Cloud in order to compare the effectiveness of the object caching mechanisms we implemented. We used four virtual machines (VMs) to create a CACSS cluster. One VM with 8GB of

memory and 4 CPU cores was used to run MySQL, HDFS NameNode, HBase Hmaster and Tomcat with the CACSS application and allocated RAM disk space. The other three were each configured with 4GB memory and 2 CPU cores to run HDFS DataNodes and HBase Regionservers. The two graphs in **Fig. 9** illustrate respectively the total object downloading time with and without object caching enabled from CACSS of various sizes. When object caching was enabled, we saw an improvement in average download speed for all object sizes, especially for objects with sizes of 1MB and 50MB.

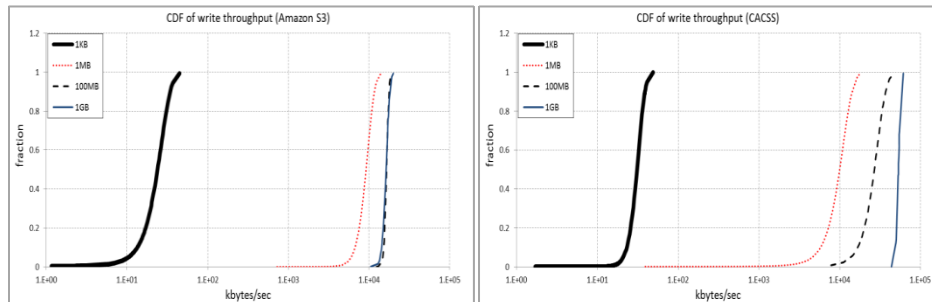


Fig. 5. Cumulative Distribution Function (CDF) plots for writing transactions from EC2 to Amazon S3 and CACSS of various sizes.

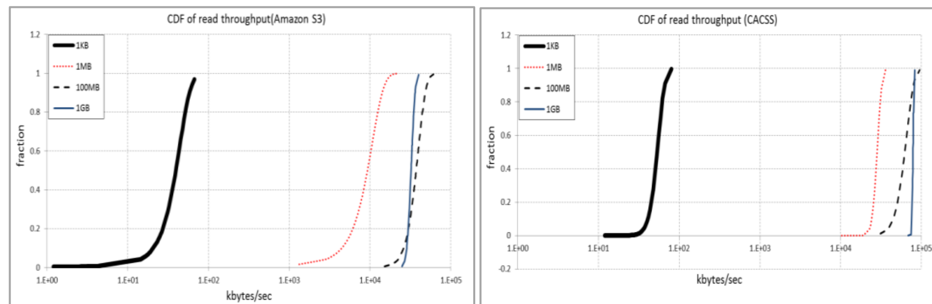


Fig. 6. CDF plots for reading transactions from EC2 to Amazon S3 and CACSS of various sizes.

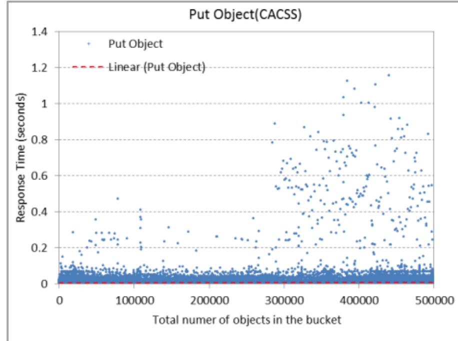


Fig. 8. Put Object requests

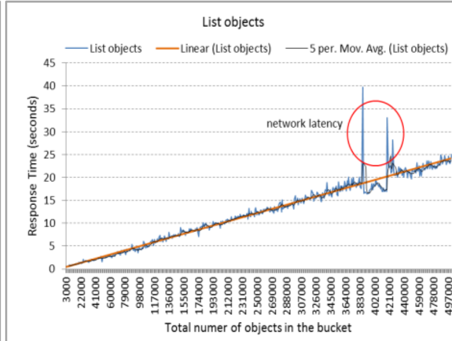
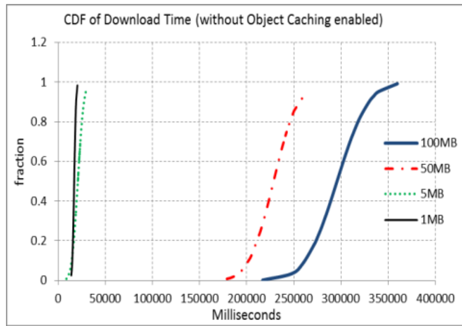


Fig. 7. List all objects requests



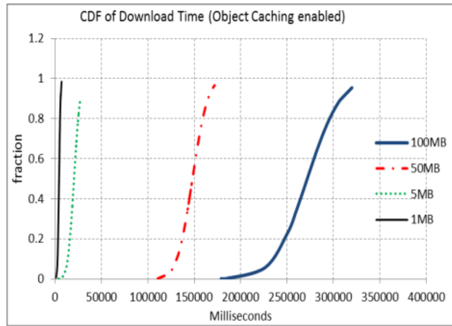


Fig. 9. CDF plots for object downloading with and without Object Caching enabled from CACSS of various sizes.

6 Conclusions

In this paper, we described the design and implementation of CACSS, a big data storage system, taking into account the generic principles of data storage efficiency and durability, scalability, performance and reliability. CACSS has been deployed on top of IC-Cloud infrastructure since 2012 and has served as the main storage space for several internal and external collaborative projects. CACSS delivers comprehensive features such as data access through the S3 interface (the de facto industry standard), native and user defined object metadata searching, global data de-duplication and object data caching services. The storage model we propose offers service providers a considerable advantage by combining existing technologies into a single customized big data storage system. Furthermore, CACSS performance was found to be comparable to Amazon S3 in formal tests, with similar read/write capabilities. We have seen improvement in performance with object caching enabled through preliminary experiments. However, there is still much improvement and evaluation work to be done on the newly added features such as object data de-duplication and object data caching services. These features will be addressed and their effectiveness validated in our future work.

References

1. Amazon. *Amazon Simple Storage Service (S3)*. Available from: http://aws.amazon.com/s3/.
2. Google. *Google Cloud Storage Service*. Available from: http://code.google.com/apis/storage/.
3. *AWS Case Study: SmugMug*. 2013.
4. *http://aws.amazon.com/solutions/case-studies/elephantdrive/.*
5. *AWS Case Study: Jungle Disk.*
6. Amazon, *Amazon S3 - The First Trillion Objects.* 2012.
7. Gohring, N., *Amazon's S3 Down for Several Hours.*
8. Brodkin, J., *Outage hits Amazon S3 storage service.* 2008.
9. Li, Y., L. Guo, and Y. Guo, *CACSS: Towards a Generic Cloud Storage Service*, in *CLOSER*. 2012, SciTePress. p. 27-36.
10. Garfinkel, S.L. *An evaluation of amazon's grid computing services: EC2, S3, and SQS*. 2007. Citeseer.
11. Rackspace. *Cloud Files*. Available from: http://www.rackspace.co.uk.
12. Barr, J. 2011.
13. Wang, G. and T.E. Ng. *The impact of virtualization on network performance of amazon ec2 data center*. in *INFOCOM, 2010 Proceedings IEEE*. 2010. IEEE.
14. Garfinkel, S.L. *An evaluation of amazon's grid computing services: EC2, S3, and SQS*. in *Center for*. 2007. Citeseer.
15. Openstack. Available from: http://openstack.org.
16. Nurmi, D., et al. *The eucalyptus open-source cloud-computing system*. 2009. IEEE.
17. Abe, Y. and G. Gibson. *pWalrus: Towards better integration of parallel file systems into cloud storage*. 2010. IEEE.
18. Bresnahan, J., et al., *Cumulus: an open source storage cloud for science.* SC10 Poster, 2010.
19. Borthakur, D., *The hadoop distributed file system: Architecture and design.* Hadoop Project Website, 2007.
20. HBase, A.; Available from: http://hbase.apache.org/.
21. Carstoiu, D., A. Cernian, and A. Olteanu. *Hadoop Hbase-0.20.2 performance evaluation*. in *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*. 2010.
22. Khetrapal, A. and V. Ganesh, *HBase and Hypertable for large scale distributed storage systems.* Dept. of Computer Science, Purdue University, 2006.
23. Saab, P., *Scaling memcached at Facebook.* Facebook Engineering Note, 2008.
24. Barroso, L.A., J. Dean, and U. Holzle, *Web search for a planet: The Google cluster architecture.* Micro, IEEE, 2003. **23**(2): p. 22-28.
25. Chang, F., et al., *Bigtable: A distributed storage system for structured data.* ACM Transactions on Computer Systems (TOCS), 2008. **26**(2): p. 4.
26. Ongaro, D., et al. *Fast crash recovery in RAMCloud*. in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 2011. ACM.
27. Tianming, Y., et al. *DEBAR: A scalable high-performance de-duplication storage system for backup and archiving*. in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. 2010.

28. Yujuan, T., et al. *SAM: A Semantic-Aware Multi-tiered Source De-duplication Framework for Cloud Backup*. in *Parallel Processing (ICPP), 2010 39th International Conference on*. 2010.
29. Chuanyi, L., et al. *ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System*. in *Storage Network Architecture and Parallel I/Os, 2008. SNAPI '08. Fifth IEEE International Workshop on*. 2008.
30. Quinlan, S. and S. Dorward. *Venti: a new approach to archival storage*. in *Proceedings of the FAST 2002 Conference on File and Storage Technologies*. 2002.
31. You, L.L., K.T. Pollack, and D.D. Long. *Deep Store: An archival storage system architecture*. in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. 2005. IEEE.
32. Dubnicki, C., et al. *Hydrastor: A scalable secondary storage*. in *Proccedings of the 7th conference on File and storage technologies*. 2009. USENIX Association.
33. Jiansheng, W., et al. *MAD2: A scalable high-throughput exact deduplication approach for network backup services*. in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. 2010.
34. Guo, Y.-K. and L. Guo, *IC cloud: Enabling compositional cloud.* International Journal of Automation and Computing, 2011. **8**(3): p. 269-279.
35. Sandberg, R., et al. *Design and implementation of the Sun network filesystem*. 1985.
36. Carns, P.H., et al. *PVFS: A parallel file system for Linux clusters*. 2000. USENIX Association.
37. Schwan, P. *Lustre: Building a file system for 1000-node clusters*. 2003.
38. Gilbert, H. and H. Handschuh. *Security analysis of SHA-256 and sisters*. in *Selected Areas in Cryptography*. 2004. Springer.
39. Apache. *Hadoop MapReduce*. Available from: http://hadoop.apache.org/mapreduce/.
40. Borthakur, D., *Hadoop avatarnode high availability*. 2010.
41. Doclo, L., *Clustering Tomcat Servers with High Availability and Disaster Fallback*. 2011.
42. Mulesoft, *Tomcat Clustering - A Step By Step Guide*.
43. Amazon. *Route 53*. Available from: http://aws.amazon.com/route53/.
44. JetS3t. *JetS3t*. Available from: http://jets3t.s3.amazonaws.com.