Indian Institute of Technology Delhi

Dept. of Computer Science

B.Tech Thesis

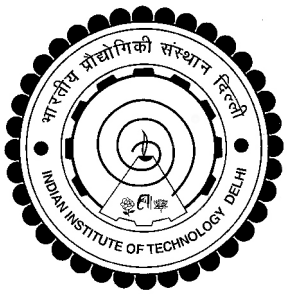
HStore 2.0

Integrated Storage on Hadoop

Authors:
Ch.Y.N.S.Avinash Karthik
N.N.Chaitanya
K.Goutham

Supervisor:
Dr. Suresh C Gupta

November 26, 2016



Acknowledgements

We would like to thank Prof. Suresh C Gupta for his guidance and motivation throughout the project. He constantly urged us to explore cutting edge cloud technologies which inspired us to work relentlessly. His experience paved the way to overcome major obstacles. We are very grateful to work under his guidance.

We also thank Prof. Vinay Ribeiro for his involvement in the committee and his encouragement.

Ch.Y.N.S.Avinash Karthik
N.N.Chaitanya
K.Goutham

Contents

Chapter 1

Introduction

1.1 Software Defined Storage

Storage infrastructure that is managed and automated by intelligent software as opposed to by the storage hardware itself. In this way, the pooled storage infrastructure resources in a software-defined storage (SDS) environment can be automatically and efficiently allocated to match the application needs of an enterprise.

1.1.1 Separating the Storage Hardware from the Software

By separating the storage hardware from the software that manages the storage infrastructure, software-defined storage enables enterprises to purchase heterogeneous storage hardware without having to worry as much about issues such as interoperability, under- or over-utilization of specific storage resources, and manual oversight of storage resources.
The software that enables a software-defined storage environment can provide functionality such as deduplication, replication, thin provisioning, snapshots and other backup and restore capabilities across a wide range of server hardware components. The key benefits of software-defined storage over traditional storage are increased flexibility, automated management and cost efficiency.

1.1.2 Key Characteristics

Software Defined Storage differs from traditional storage in several architectural elements.

- **Commodity Hardware**

  All the intelligence in SDS is placed in the software layer so that we can use simple commodity hardware for the physical storage.

- **Scale-out Architecture**

  SDS uses a building-block approach to storage that allows users to dynamically add and remove resources.

- **Resource Pooling**

  All the available storage resources are pooled into a single logical entity that is managed centrally.

- **Automation**

  Extensive automation is provided through which users can request storage resources in terms of capacity and performance rather than physical location of drives.

- **Programmability**

  In addition to the in-built automation, rich APIs are provided using which the administrators and third-party applications can integrate the control plane across storage layer and other layers to deliver work flow automation as per their requirements.

1.2 Problem Statement

The aim of this project is to build a software defined storage solution which provides all three software systems - File Storage, Block Storage and Object Storage in software similiar to Ceph which is currently deployed in IIT Delhi's academic cloud - Baadal. A higher goal of this project is to provide Baadal with HStore 2.0 for its storage needs.

Hadoop is a well established, robust SDS which provides File Storage. We intend to build Block Storage and Object Storage on top of Hadoop, thus providing a complete software system. This is the target of our project.

Chapter 2

Literature Review

We use existing technologies in Software Defined Storage like HDFS and HBase as building blocks and build on top it. We shall briefly review them before we discuss our object and block storage.

2.1 Hadoop

"Apache Hadoop is a framework that allows distributed processing of large data sets across clusters of computers using simple programming models".

It is designed to run on a cluster of commodity computers and rely on soft- ware for scalability, reliability and performance. It software layer converts a set of fault prone hardware into a single fault-tolerant system with graceful fault detection and handling. It is composed of two sub-components, Hadoop Distributed File System(HDFS) and Hadoop MapReduce. HDFS is a distributed fault-tolerant file system which is designed to run on commodity or cheap hardware.It is suitable for large data sets and streaming access to files with high throughput.

HDFS is a master/slave architecture. HDFS system consists of two type of nodes - a namenode and several datanodes. Namenode contains the information regarding which parts of a file is present in which datanodes. This information is maintained for every file in the namenode in its memory. Datanodes contain the data corresponding to files. Namenode allocates datanodes as to which blocks of a given file each datanode has to store. Clients then directly talk with the datanode to read data from the datanodes. Simply put, namenode handles all the namespace operations like opening and closing files, renaming files and storing mappings from blocks to datanodes. Datanodes handle data requests like read and write requests for

a file from the file system clients, block creation, deletion requests etc. Data is replicated on different datanodes for fault tolerance and higher bandwidth depending upon the replication factor.

Mapreduce is a framework for scheduling jobs and managing cluster resources in hadoop. Hadoop bases itself one important assumption that if often true that "Moving Computation is Cheaper than Moving Data". This becomes perticularly relevant when the dataset is huge. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. When the compute nodes and the storage nodes are the same, which is the case typically, it is computation that has actually moved to the data. This breakup of tasks into map and reduce gives us a great flexibility in running jobs parallely as all map tasks are independent of each other.

2.2 Hadoop ACL's : FINE-GRAINED PERMISSIONS FOR HDFS FILES IN HADOOP

Securing any system requires implemention of layers of protection. Access Control Lists (ACLs) are typically applied to data to restrict access to data to approved entities. Application of ACLs at every layer of access for data is critical to secure a system.

For several years, HDFS has supported a permission model equivalent to traditional Unix permission bits. For each file or directory, permissions are managed for a set of 3 distinct user classes: owner, group, and others. There are 3 different permissions controlled for each user class: read, write, and execute. When a user attempts to access a file system object, HDFS enforces permissions according to the most specific user class applicable to that user.

However, this permission model is not expressive enough to allow for finer permissions. It forces complexity on to cluster administrators to manage additional users and groups. It also forces complexity on to end users, because it requires them to use different accounts for different actions.

In general, plain Unix permissions aren't sufficient when you have permission requirements that don't map cleanly to an enterprise's natural hierarchy of users and groups.

HDFS ACLs have become available since Apache Hadoop 2.4.0.

HDFS ACLs give the ability to specify fine-grained file permissions for specific named users or named groups, not just the file's owner and group. HDFS ACLs are modeled after POSIX ACLs. Best practice is to rely on traditional permission bits to implement most permission requirements, and define a smaller number of ACLs to augment the permission bits with a few exceptional rules.

It's important to keep in mind the order of evaluation for ACL entries when a user attempts to access a file system object:

If the user is the file owner, then the owner permission bits are enforced Else if the user has a named user ACL entry, then those permissions are enforced Else if the user is a member of the file's group or any named group in an ACL entry, then the union of permissions for all matching entries are enforced(The user may be a member of multiple groups) If none of the above were applicable, then the other permission bits are enforced

2.3 HBase

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

The important features of HBase are as follows -

- Random real-time read/write access to bigdata.
- Strictly consistent reads and writes
- Automatic failover support
- Linear and modular scalability.

2.4 Existing Solutions

2.4.1 Ceph

Ceph, a free-software storage platform, implements object storage on a single distributed computer cluster, and provides interfaces for object-, block- and file-level storage. Ceph aims primarily for completely distributed operation without a single point of failure, scalable to the exabyte level, and freely available.

Replication of data makes it fault-tolerant and usage of commodity hardware without requiring any specific hardware support are its salient features. As a result of its design, the system is both self-healing and self-managing, aiming to minimize administration time and other costs.

It also provides seamless access to objects using native language bindings or RADOS (Reliable Autonomic Distributed Object Store) gateway, a RESTful interface that's compatible with applications written for Amazon S3 and Open-Stack Swift.

Ceph's RADOS Block Device (RBD) provides access to block device images that are striped and replicated across the entire storage cluster. Ceph's RBD also integrates with Kernel Virtual Machines (KVMs). Several IaaS cloud platforms (i.e.: OpenStack, CloudStack) officially support Ceph to provide a block storage solution.

The Ceph File System (Ceph FS) is a POSIX-compliant filesystem that uses a Ceph Storage Cluster to store its data. Ceph FS uses the same Ceph Storage Cluster system that provides object storage and block storage interfaces. Ceph has 4 different kinds of daemons.

- Cluster Daemons - Keeps track of active and failed cluster nodes
- MetaData Server - Stores metadata of inodes and directories.
- Object Storage Devices - Stores the content of the files
- REST gateways - Exposes the object state layer as an interface compatible with Amazon S3 and OpenStack Swift APIs.

Chapter 3

Storage Services at IIT Delhi

3.1 Introduction

A storage service is a service that provides computer storage space and related management services. They offer periodic backup and archiving.

Advantages of managed storage are that more space can be ordered as required. Depending upon your Storage Service Provider (SSP), backups may also be managed. Faster data access can be ordered as required. Also, maintenance costs may be reduced, particularly for larger organizations who store a large or increasing volumes of data. Another advantage is that best practices are likely to be followed.

Disadvantages are that the cost may be prohibitive, for small organizations or individuals who deal with smaller amounts or static volumes of data and that there's less control of data systems. Managed Storage generally falls into one of the following categories:

3.1.1 Locally managed storage

Advantages of this type of storage include a high-speed access to data and greater control over data availability. A disadvantage is that additional space is required at local site to store the data.

3.1.2 Remotely managed storage

Advantages of this type of storage are that it may be used an off site backup, it offers global access (depending upon configuration) and adding storage will not require additional space at the local site. However, if the network providing connectivity to the remote data is interrupted, there will be data

availability issues, unless distributed file systems are in use. In cloud computing, Storage as a Service (SaaS) involves the provision of off-site storage for data and information.

3.2 Storage Services at IIT Delhi

3.2.1 Important Terms

SAN

Storage area network (SAN) is a network which provides access to consolidated, block level data storage. SANs are primarily used to enhance storage devices. A SAN does not provide file abstraction, only block-level operations. However, file systems built on top of SANs to provide file-level access, and are known as shared-disk file systems.

NAS

Network-attached storage (NAS) is a file-level computer data storage server connected to a computer network providing data access to a heterogeneous group of clients. NAS is specialized for serving files either by its hardware, software, or configuration.

IIT Delhi provides remote storage to students via following ways -

3.2.2 Storage at GCL and CSC

These are lab storages. Both have similar storage implementations provisioned by a NetApp based single disk which uses NAS since SAN cannot be used with Network File System.

3.2.3 Storage at Baadal

This is a cloud storage. The service is provided using Ceph implementation based on NAS over Netapp and mounted using NFS. The link to the source: https://github.com/iitd-plos/baadal2.0

Chapter 4

Study of Existing Cloud Technologies

4.1 Introduction

As a part of the initial background understanding of how to approach the project, we extensively studied the existing cloud technologies like GCP, Microsoft Azure, Amazon Web Services, Ceph and OpenStack, and compared the unique features and services each one had to offer.

Cloud storage comes in all shapes and forms. Direct comparison between providers is often difficult because they focus on different aspects of the service. Often people will base their decision on the amount of free storage available. However, this is only one element that you might want to consider. Most solutions will include a degree of free storage, though if you are backing up photos and media this can run out quickly. It's therefore worth looking ahead to see how much you will have to pay, and how much space it will get you. Aside from that, you should look at what operating systems are supported, and whether you will be able to back up and use files from more than one machine – if you are working on documents at home as well as in the office, for example, this may be important.

Sharing and syncing files may or may not be a priority for you, depending on whether you want one-off or live backups. Cloud security is also an important factor. There have been a number of high-profile hacks of cloud storage providers, and although all of them claim to have a good security policy, the reality is that only a handful give you real control over your data – look for a 'zero knowledge' policy, which means that they cannot view or hand over your files, even if they wanted to. There are very few companies

that have managed to combine a good user interface with strong security and a competitive price, so you are likely to have to make a trade-off, depending on what criteria are most important to you.

4.2 Various Comparisons of Cloud Storage Services

4.2.1 Comparison of APIs of Cloud Platforms

Amazon	Open Stack	Google Cloud Platform
Create Bucket	Create Container	Insert Bucket
List Buckets	List Containers	List Buckets
Delete Bucket	Delete Container	Delete Bucket
		Get Bucket
		Update Bucket
	List Objects	List Objects
Put Object	Store Object	Insert Object
Get Object	WriteObjectToFile	Get Object
Delete Object	Delete Object	Delete Object
		Update Object
		Compose Objects
		Copy Object
		List Objects Filter
		Patch Object
		WatchAll
Delete Bucket ACL		Delete Bucket ACL
Get Bucket ACL		Get Bucket ACL
Insert Bucket ACL		Insert Bucket ACL
List Bucket ACL		List Bucket ACL
Update Bucket ACL		Update Bucket ACL
Delete Object ACL		Delete Object ACL
Get Object ACL		Get Object ACL
Insert Object ACL		Insert Object ACL
List Object ACL		List Object ACL
Update Object ACL		Update Object ACL

4.2.2 Comparison of leading Persistent Disk Services

Features	Amazon EBS	Compute Engine
Volume types	EBS Provisioned IOPS SSD, EBS General Purpose SSD, Throughput Optimized HDD, Cold HDD	Standard persistent disk (HDD), SSD persistent disk
Volume attachment	Can be attached to only one instance at a time	Read-write volumes: Can be attached to only one instance at a time Read-only volumes: Can be attached to multiple instances
Attached volumes per instance	Up to 40	Up to 128
Maximum volume size	16TiB	64TB
Redundancy	Yes	Yes
Snapshotting	Yes	Yes
Snapshot locality	Regional	Global

4.2.3 Comparison of Limitations of VM interactions with Locally Attached Storage

Feature	Amazon EC2	Compute Engine
Service name	Instance store (also known as ephemeral store)	Local SSD
Volume attachment	Tied to instance type	Can be attached to any non-shared-core instance
Device type	Varies by instance type	SSD
Attached volumes per instance	Varies by instance type	Up to 8
Storage capacity	Varies by instance type	375GB per volume
Live migration	No	Yes
Redundancy	None	None

4.2.4 Different storage options provided by GCP - Google Compute Engine

	Persistent disks	Local SSDs	Cloud Storage buckets	RAM disks
Storage type	Efficient and reliable file storage	High-performance file storage	Affordable object storage	In-memory file storage
Price per GB/month	0.04 - 0.17 $	0.218 $	0.01 - 0.026 $	3.37 - 3.71 $
Maximum space per instance	64 TB	3 TB	Almost infinite	208 GB
Scope of access	Zone	Instance	Global	Instance
Data redundancy	Yes	No	Yes	No
Encryption at rest	Yes	Yes	Yes	N/A
Custom encryption keys	Yes	No	Yes	No
Machine type support	All machine types	Most machine types	All machine types	All machine types
Zone availability	All zones	All zones	All zones	All zones

Chapter 5

Object Storage

5.1 Definition

Object Storage is a storage architecture that manages data as objects. A simple example for an application that requires object storage could be any software system that has large user-generated content, like Facebook, twitter and a lot of social networking sites. They would need to store unstructured data like images, music, videos, and documents. Microsoft's Azure, Amazon S3 are some examples of object stores in public clouds.

Objects in object storage system can be of any size ranging from small objects like text files to very large objects like HD videos. Each object also has some metadata associated with it. The metadata and data are often physically separated to achieve better management and indexing purposes. Object storage system could be compared to a valet parking system at a restaurant. In valet parking, the customer gives the car to a valet and gets a receipt in return. The customer does not know where the car is parked or if the car was moved from one parking spot to another when he was in the restaurant. When he gives the receipt back, the car is returned by the valet. Similarly in the object storage system, the user gives the object to the object storage system and saves it. He does not know how or where it is stored, and when he supplies the object key, the object's data is returned to the user.

5.2 Advantages of Object Storage

- Unlike files in a file storage system which have fixed metadata like file size and date of creation, objects can have rich user-defined metadata.

Metadata in object storage system can have any number of custom defined metadata attributes.

- Protocol Support - Traditional file system protocols (CIFS and NFS) communicate on TCP ports that are available on internal networks, but are not usually exposed to the Internet. But, object storage system is usually accessed through a REST API over HTTP.

- Scalability - Object storage works well as a scalable data store for unstructured data which are not frequently updated. In cloud storage systems, it is well suited for file content like images and videos.

5.3 Object Storage Features

5.3.1 ACLs

Amazon S3 and Google Cloud Platform provide security features by supporting Access Control Lists for both objects as well as buckets.

One significant improvement of HSTORE2.0 is ACL provisioning. ACLs with the same features as in S3 and GCP have been provided in HSTORE2.0. For a given bucket or an object each user can either be an owner, writer, reader or may not have access at all.

As described in the literature, HDFS has in built ACL for individual files and directories. Using these in-built ACL features, the ACLs of HSTORE2.0 have been designed.

The three roles and their implementations are described below:

- **Reader**

  The reader of a bucket/object can only read the data and metadata of bucket/object without being able to change it.

  When a user is given reader access to a bucket/object, the HDFS ACL for the corresponding directory/file will be set as read only for that user. If the user tries to write to it, he will naturally be prevented by the HDFS from doing so. The API for changing the ACL will not allow the user to change the ACLs for the bucket/object by checking whether or not the user is the owner of the bucket/object.

- **Writer**

  The writer of a bucket/object can read and write to the data and the metadata of the bucket/object without being able to read or change the ACL's.

  Granting a user writer access to bucket/object sets the HDFS ACL for the corresponding directory/file to read and write access for the user.

- **Owner**

  The owner of a bucket/object can read and write to the data and the metadata of the bucket/object. The owner can also the change the ACL's of the bucket/object and grant read, write, owner permissions to other users of object storage.

  Owner access for a user to directory/file means read, write and execute permissions are set in HDFS ACLs for corresponding directory/file.

ACL manipulation APIs

An owner of bucket/object can perform the following operations of ACL's:

Bucket and Object ACL manipulation APIs

- Delete - Delete the ACL entry for a user
- Get - Retrieve the ACL entry for a user
- Insert - Add the ACL entry for a user
- List - Get the list of all ACL entries for a file/directory
- Update - Change the ACL entry for a bucket/object corresponding to a user.

5.3.2 A General Note in relation to Bucket and Object APIs

:

In the bucket and object API's, the access level of the user for the object/bucket is checked and correspondingly the operation is either allowed or disallowed for that user.

5.3.3 Buckets

Currently, HSTORE already supports the following API calls for buckets:

- Put Bucket
- List Buckets
- Delete Bucket

After the study of Amazon S3, GCP and Microsoft Azure, the some additional API calls have been added to HSTORE 2.0. Following is a description of the API calls along with their design:

- **Get Bucket**

  Returns the metadata of the bucket. This can be implemented by storing the metadata in the corresponding HBase table and retrieving the metadata for appropriate entry.

- **Update Bucket**

  Updates the information within a bucket This can be implemented by updating the metadata of appropriate entry in HBase table for buckets

5.3.4 Objects

Currently, the project supports the following API calls for objects:

- **List Objects**

  Lists all the objects present in a given bucket. Every object will have atleast one owner and a user can be an owner of any number of objects.

- **Put Object**

  Creates an object for a user. User should mention a bucket on which he has owner rights so that he can put this new object in that bucket.

- **Get Object**

  Returns the required object. The user must have read/owner rights to get that object.

- **Delete Object**

  Deletes an object if the user is one of it's owners. Once an object is deleted, all the users who have read/owner rights over this object can no longer access it.

These are similar to those provided by Swift. GCP has additional options like :

- **Compose Objects into a single Object**

  Archive set of small files into a single object file and store appropriate user defined metadata like timestamps

- **Copy Object**

  Create an extra copy on demand

- **List Objects based on filter**

  Search in the Objects table based on the filters in the HBase table

- **Update Object**

  In case of large files, you have to delete the old file and recreate the modified file, although this is not that efficient, we are looking at optimizations.

  In case of small files, you have to invalidate the old file and modify the HAR files table

- **Update metadata**

  Update the corresponding Metadata entries in the HBase table

Chapter 6

Block Storage

6.1 Definition

Block Storage is a storage abstraction where data is stored in volumes. Each volume is further divided into blocks. Each block is a sequence of bytes of fixed size. Each image will therefore contain a total of imageSize/blockSize blocks. Data is accessed whole blocks at a time rather than individual bytes.

Physical block storage devices are called hard disks and are connected to a computer with a SCSI or a SATA connector. In a cluster environment custom made block storage solutions are used which are connected to the servers with FC, FCoE, or iSCSI protocols. Virtualised block storage solutions provide this block and volumes abstraction in software using commodity network and existing storage resources. In block storage systems, raw storage volumes are created and these volumes can be treated as individual hard drives. This makes block storage usable for any kind of application including file storage, Virtual Machine File Systems etc. This abstraction of storage devices is then used by file systems or DBMS for use by applications and end users.

6.2 Additional Features

These are the additional Features to be provided as part of Block Storage for integration with BAADAL.

One major change of design we plan on performing is distributing the HBlock Server over multiple nodes. In the current design, the HBlock Server is run only on the namenode. This leads to unused main memory on the datanodes. This unused memory can be better utilized by running HBlock

Servers on the datanode machines as well. This also helps to increase the amount of addressable space using the Directory index.

Moreover, all the virtual disks are handled by these HBlock servers only without spawning any new processes. This means that there would be multiple directory indexes and multiple caches within the HBlock servers (one for each virtual disk).

6.3 New Features

6.3.1 Import and Export VM Image

Import

The VM from Amazon Elastic Cloud or Microsoft Azure can be exported in the .vhd file format. This file can be imported into HStore 2.0 using the API call **putImage()** which will then read the vhd file using the servlet input stream reader and store it as contiguous blocks.

Export

The VM image stored in the block storage of HStore 2.0 can be exported using the API call **getImage()** which gives the servlet input reader as output which is then utilized by the user to extract data from the file as contiguous blocks.

6.3.2 Namenode Load Reducer

Design

The HBlock server needs to store the Directory Index and Cache at all times. So, the memory of the HBlock server is a bottleneck. It is observed that the datanode process running on a different node is underutilizing its resources. We can transfer the additional load on the HBlock server to the underutilized datanodes and we must ensure that the fault domains are disjoint i.e. the metadata in HBlock server should have all associated files within the datanodes corresponding HBlock server. Now we must create a new process which would redirect the data request to appropriate HBlock server depending on where the datanode is.

Chapter 7

Future Work and Conclusion

7.1 Future Work

7.1.1 Encryption

We have initially thought of implementing the encryption part but later realized that it is the job of the Hypervisor to encrypt because we are encrypting so as to secure the data from other nodes on the same network. As the data enters and exits the network through the Hypervisor, it is the point where we have to encrypt and decrypt data.

7.1.2 Integration with Baadal

Kernel drivers and RPC is a prerequisite for this task. After they are complete, this software can be integrated with baadal for this software needs.

7.2 Conclusion

The ultimate goal of this project is to successfully provide our academic cloud(Baadal) with a software defined storage solution for all its storage needs. It should be able to provision disk images for VMs from block storage, provide object storage for required users, provide storage for performing Map reduce and other distributed computing operations. In this project we have designed and implemented ACLs and provisioned new APIs for Object Storage which makes it on par with cloud giants like AWS, Google Cloud Platform, Microsoft Azure. When it comes to Block Storage, we have optimized the design of HBlock server and provided the most crucial feature of import/export VM image, thus approaching the ultimate goal. But we

dearly acknowledge the fact that there is still some work left to do as mentioned in the future work section.

The End