Evaluation of Storage Systems for Big Data Analytics

by

Shilpa Nagendra

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2017 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Ming Zhao
Ross Maciejewski
Chun-Jen (James) Chung

ARIZONA STATE UNIVERSITY

December 2017

ABSTRACT

Recent trends in big data storage systems show a shift from disk centric models to memory centric models. The primary challenges faced by these systems are speed, scalability, and fault tolerance. It is interesting to investigate the performance of these two models with respect to some big data applications. This thesis studies the performance of Ceph (a disk centric model) and Alluxio (a memory centric model) and evaluates whether a hybrid model provides any performance benefits with respect to big data applications. To this end, an application TechTalk is created that uses Ceph to store data and Alluxio to perform data analytics. The functionalities of the application include offline lecture storage, live recording of classes, content analysis and reference generation. The knowledge base of videos is constructed by analyzing the offline data using machine learning techniques. This training dataset provides knowledge to construct the index of an online stream. The indexed metadata enables the students to search, view and access the relevant content. The performance of the application is benchmarked in different use cases to demonstrate the benefits of the hybrid model.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Professor Dijiang Huang for giving me an oppurtunity to work on this project. His supervision, extensive knowledge of the subject, patience and practical advice were invaluable to me during the work.

I would like to thank Dr.Chun-Jen(James) Chung for his patience, constant encouragement and advice in stimulating discussions during various stages of the work. I would like to thank Yuli for his support and timely help in configuring the infrastructure and tuning the experiments. I want to thank my colleagues at Computer science department, for allowing me to access the cloud resources.

I would like to thank Professor Ming Zhao and Professor Ross Maciejewski for the timely responses and guidance on the thesis work.

Also I extend my gratitude to my family and friends for all the moral support at tough times. Last but not least, I am indebted to my husband Aravindhan Krishnan for his constant encouragement, timely advice and moral support in helping me finish the thesis with ease.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1 INTRODUCTION 1

1.1 Problem Statement 2

1.2 Contributions 3

1.3 Outline 4

2 RELATED WORK 6

2.1 Big Data Storage 6

2.1.1 Disk Centric Storage 7

2.1.2 In-Memory Storage 8

2.1.3 Hybrid Storage 8

2.1.4 Summary 9

2.2 Big Data Analytics 10

2.2.1 Video Analytics 11

2.2.2 Online Learning with Video Knowledge Base 12

2.2.3 Summary 13

3 STORAGE SYSTEMS 14

3.1 Storage Architectures 14

3.1.1 File Storage 14

3.1.2 Block Storage 15

3.1.3 Object Storage 16

3.2 Storage Models 17

3.2.1 Disk Centric 17

3.2.2 Memory Centric ... 18
3.3 Software-Defined Storage ... 19
4 DISTRIBUTED DISK CENTRIC STORAGE SYSTEMS ... 20
4.1 Hadoop Distributed File System ... 20
4.1.1 Architecture ... 21
4.1.2 Metadata Management ... 22
4.1.3 Writing Data to Disk ... 23
4.1.4 Reading Data from Disk ... 24
4.1.5 Applications ... 25
4.2 Gluster File System ... 25
4.2.1 Architecture ... 25
4.2.2 Working of GlusterFS ... 26
4.2.3 Types of Volumes ... 27
4.2.4 Hashing Algorithm ... 28
4.2.5 Applications ... 29
4.3 Ceph ... 29
4.3.1 Ceph Philosophy ... 29
4.3.2 Ceph Architecture ... 30
4.3.3 Ceph Storage Cluster ... 31
4.3.4 Ceph Supported Storage Types ... 40
4.3.5 OpenStack and Ceph ... 41
5 CLOUD INFRASTRUCTURE ... 45
5.0.1 Features ... 45
5.0.2 Comparative Study of Distributed Storage Systems ... 46

CHAPTER Page

5.0.3 Inference on the Back End Storage 49

5.0.4 Ceph Cluster Recommendation 49

5.1 MobiCloud 51

5.1.1 Ceph Configuration 52

5.1.2 Ceph Benchmarks 56

5.2 ThoTh Lab 61

5.2.1 Motivation 61

5.2.2 Infrastructure in ThoTh Lab 63

5.2.3 Ceph Configuration 64

6 DISTRIBUTED MEMORY CENTRIC STORAGE SYSTEM 68

6.1 Alluxio 68

6.2 Alluxio Architecture 69

6.2.1 Alluxio Components 69

6.3 Storage in Alluxio 70

6.3.1 Tiered Storage in Alluxio 71

6.3.2 Lineage 73

6.3.3 User Location Policy 74

6.4 Spark 74

6.4.1 Hadoop and Spark 74

6.4.2 Spark Features 75

6.4.3 Spark Ecosystem 76

6.4.4 Spark Architecture 77

6.4.5 Resilient Distributed Datasets 78

6.5 Alluxio and Spark 79

CHAPTER Page

6.5.1 Problems with Spark . . . 79

6.5.2 Benefits with Alluxio . . . 79

6.5.3 Applications with Alluxio . . . 80

7 BIG DATA ANALYTICS USING SPARK . . . 81

7.1 Comparitive Study of Ceph and Alluxio Storage Systems . . . 81

7.2 Benefits of Combining the Systems . . . 82

7.3 Big Data Analytics using Spark . . . 83

7.3.1 Configuration . . . 84

7.3.2 Spark Line Count Experiment . . . 85

7.3.3 Results . . . 86

7.3.4 Inference . . . 88

8 TECHTALK . . . 90

8.1 Introduction . . . 90

8.2 Architecture . . . 92

8.2.1 Functionalities . . . 92

8.2.2 Components . . . 94

8.2.3 Interaction with Ceph and Alluxio . . . 94

8.2.4 RAKE Algorithm . . . 95

8.2.5 External Tools . . . 97

8.3 Web Server . . . 98

8.3.1 Offline Store . . . 98

8.3.2 Live Feed . . . 99

8.3.3 Cache . . . 99

8.4 Client . . . 99

8.5 Index Server ... 102
8.5.1 Index Generation ... 102
8.5.2 Update Index ... 103
8.5.3 Live Lecture Feed ... 103
8.5.4 TechTalk Application Settings ... 106
8.5.5 Streaming Server ... 107
8.5.6 Offline Data Analytics ... 107
8.6 Performance Evaluation ... 108
8.6.1 Index Server ... 108
8.6.2 Streaming Server ... 111
8.6.3 Data Sharing between Applications ... 112
8.6.4 Data Analytics on Logs ... 113
9 LIMITING STORAGE IN ALLUXIO ... 114
9.1 Alluxio Sessions ... 114
9.2 Design ... 115
9.2.1 Alluxio Client ... 115
9.2.2 MyLocationPolicy ... 116
9.2.3 Alluxio Session Manager ... 118
9.2.4 MyAllocator ... 118
9.2.5 MyEvictor ... 119
9.3 Implementation ... 120
9.3.1 Enabling and Configuring Parameter to Limit Storage ... 120
9.3.2 Alluxio Client ... 120
9.3.3 MyLocationPolicy ... 121

9.3.4 Alluxio Session Manager 121
9.3.5 MyAllocator 122
9.3.6 MyEvictor 123
9.4 Results 124
9.4.1 Client and Worker 124
9.4.2 Session Manager 125
9.4.3 Comparison 126
9.4.4 Observation 128
9.4.5 Inference 128
9.4.6 Future Enhancements 129
10 CONCLUSION 130
10.1 Conclusion 130
10.2 Future Enhancements in the Application 132
10.3 Other Use Cases 133
REFERENCES 135

LIST OF TABLES

Table Page

5.1 Comparison of Distributed Storage Systems 47

5.2 Minimum Hardware recommendations 50

5.3 Ceph Configuration 52

5.4 Ceph-Mon Configuration 53

5.5 Ceph-Admin Configuration 53

5.6 Ceph-OSD Configuration 54

5.7 Ceph Status 54

5.8 Available Hardware 65

5.9 Ceph Cluster Configuration 65

5.10 Ceph-Mon Configuration 66

5.11 Ceph-Admin Configuration 66

5.12 Ceph-OSD Configuration 66

5.13 Ceph Status 67

7.1 Comparison of Ceph and Alluxio Storage Systems 81

9.1 Defining Max Bytes per Session 120

9.2 Defining Location Policy 121

9.3 Defining Allocator Class 123

9.4 Defining Evictor Class 123

9.5 Worker Settings 124

9.6 Block Name to ID Mapping 125

9.7 Session Size 126

9.8 Session Block Association 126

LIST OF FIGURES

Figure | Page
4.1 HDFS Architecture 22
4.2 Working of GlusterFS 28
4.3 Ceph Storage Cluster 32
4.4 Ceph Object 34
4.5 Ceph Replication 36
4.6 Ceph as Storage for OpenStack 42
5.1 Ceph Cluster Read Write Performance 55
5.2 Ceph Performance w.r.t Different Factors 62
5.3 Thothlab Architecture 64
6.1 Alluxio Architecture 73
6.2 Spark Architecture 77
6.3 Spark Ecosystem 78
7.1 Hybrid Architecture 83
7.2 Big Data Analytics with Alluxio on Compute Node 85
7.3 Execution Time for Line Count in Scala 87
7.4 Execution Time for Line Count in PySpark 89
8.1 TechTalk Architecture 92
8.2 TechTalk Client Application on Laptops 100
8.3 Client Confirmation on Success 100
8.4 Live Lecture Streaming 102
8.5 TechTalk Application Settings 106
8.6 Growth of Index w.r.t Video Count 109
8.7 Index Performance with Different Data Structures 111
8.8 Performance Benefit using Alluxio with 1G Index 112

Figure Page

8.9 Performance Benefit using Alluxio with 10G Index 113

9.1 Alluxio Client 117

9.2 Alluxio Worker 118

9.3 Eviction 119

9.4 Tiered Storage 127

Chapter 1

INTRODUCTION

With the growth in technology, big data has been an important paradigm in many areas such as science, engineering, social network and other data-driven tasks. The key requirements of big data storage are that it can handle very large amounts of data and keep scaling. These days the term big data is often associated with the analytics that extract value from raw data and is not limited to a particular size.Datasets grow very rapidly with the vast increase in the information sensing devices like mobiles, laptops, watches,and security cameras.

The traditional storage systems do not provide high performance with large scale data. We need systems that can store huge data and provides access to the data instantly. To keep up with growth and meet efficient data processing needs, there are several distributed storage systems like Ceph [1], GlusterFS [2]and HDFS [3] available in the market.

There are different architectures that support different storage types. Data can be stored in the form of file, blocks or objects. Files are stored in directories in a file system with its metadata like the creation date, modified time, size, owner of the file and many other attributes. But with the growth in data it is tedious to maintain the tree structure and its metadata for billions of files. That brought the block storage where application is responsible for storing blocks of data. The application decided what to store, where to store and with little or no metadata. It brought granularity but increased latency for applications which required metadata. Lately objects are the most preferred method of storage. Objects are stored in a flat structure. Each object contains the data and the metadata associated with it. There is no limit on

how much metadata or type of metadata that is associated with it. Cloud storage systems like Amazon S3, EMC Atmos, Rackspace store data in the form of objects.

The SDS storage systems can be classified as disk centric and memory centric models. Disk centric models store data on the disks and are distributed, scalable, reliable and fault tolerant. Ceph and Gluster are some of the storage systems with its support for object storage. These object storage systems enable storing unstructured data and tons of metadata associated with it. Hence its more suitable for storing big data. In most of the distributed storage systems, the data nodes are decoupled from compute nodes and data needs to be fetched from storage nodes for any kind of analytics. These systems do not provide high performance with data analytics due to the reduced disk speed and repeated data reads.

To meet the data processing speed, there is a shift from disk centric to memory centric model. In memory centric model the data is processed in DRAM and is shared at memory speed. These systems can act as a memory cache to the disk storage systems. Data residing in memory results in significantly reduced query response times and reduces the need for indexing making it suitable for data analytics applications. Apache Spark [4] framework is one such which uses in-memory rdds for computation. Alluxio[5] is one of the memory centric storage system that uses DRAM to increase the data processing speed. Memory centric models provide high performance, but memory does not fit in the whole data and DRAM is also not cheap. It would be interesting to investigate the performance of a hybrid model, which can use the under storage for offline data and in memory for the data analytics and ad-hoc queries.

1.1 Problem Statement

Big data analytics require speed, scalability and performance. To be able to meet with all these demands, we need a software defined storage system that can scale

with dynamic workloads and be able to access data instantly. In addition to that, the system should be agnostic to the type of data and handle both structured and unstructured data. Hence a hybrid model combining both the benefits of the disk storage model and the memory centric model is needed to address these challenges.

1.2 Contributions

The goals of this thesis is to address the challenges in big data analytics using the hybrid model and demonstrate the use case of the model using a big data application. This includes setting up infrastructure for a cloud ThoTh Lab [6] for online learning and develop an application to assist users in online learning using video knowledge base.

The contributions of this thesis are listed below :

- First, the thesis compares the available distributed storage systems - HDFS, GlusterFS and Ceph. We choose Ceph as the backend storage and deploy a cloud with Ceph as the back end storage and tune the parameters to improve its performance.

- Second, the thesis compares Ceph (a disk centric model) and Alluxio (a memory centric model). A hybrid model - Alluxio (memory-centric) is configured with Ceph (disk-centric) as under storage. It compares and benchmarks the performance of text analytics on direct Ceph and on Alluxio with Ceph as under storage using Spark application. The hybrid model reduces the latency in ad-hoc queries due to reduced disk speeds and repeated reads from storage nodes.

- Third, We design an application - TechTalk with this hybrid model. It uses Ceph to store big data and Alluxio for data analytics and ad-hoc queries. The

application is a tool to assists users in online learning using video knowledge base. We intend to use this application in the cloud Thoth Lab (a virtualised hands on laboratory for Computer science education) for personalized learning.

To the best of author's knowledge, this is the first application created with the hybrid model - Alluxio as memory cache to Ceph as under storage for Video analytics. The functionalities of the application include live recording, offline storage, content indexing of videos and reference generation. The index in memory enables admins and end users search the video by content in near real time. All ad-hoc search queries or any kind of data analytics operation uses Alluxio to query the result. Live recording of lectures also uses Alluxio to generate live captions and references to related lectures in the store. The experiments and results demonstrate that the use of in-memory storage for ad-hoc queries and data analytics improves the performance of the system by $\approx$10x.

- Alluxio is hadoop compatible but does not allow limiting storage per Alluxio client. This leads to clogging of DRAM resources to only one session causing latency in other sessions. Additionally, We enhance the modules in Alluxio to be able to limit the storage of memory per Alluxio client session to reduce the pooling of resources by only one session.

1.3 Outline

Chapter 2 describes the related work in big data storage, big data analytics and hybrid models to improve performance of systems.

Chapter 3 describes the storage types, storage models and software defined storage systems.

Chapter 4 provides the overview of the existing distributed disk storage systems for big data analytics. It describes the architecture and applications of HDFS,GlusterFS and Ceph storage systems.

Chapter 5 compares the distributed disk storage systems, requirements of the cloud infrastructure and the deployment of the cloud using Ceph. It provides a brief description of the steps involved in creating and tuning the Ceph cluster.

Chapter 6 describes the need for in-memory storage systems and the architecture of Alluxio.

Chapter 7 provides the comparative study of Ceph and Alluxio, the benefits of combining the systems and simple analytics with ad-hoc queries on the hybrid model.

Chapter 8 describes the application TechTalk and its architecture designed with the hybrid model. It describes the functionalities of the application, the strategies used to analyze and content index videos, use cases of choosing appropriate storage system for storage and ad-hoc queries, and the benefits of using the hybrid model.

Chapter 9 is an additional work that describes how to be able to limit the usage of memory in Alluxio per session.

Chapter 10 finally provides concluding remarks about the hybrid model, future work and the other use cases of the TechTalk application.

Chapter 2

RELATED WORK

The related work is divided into big data storage describing the existing disk centric, memory centric and hybrid storage systems and big data analytics describing video analytics and online learning using video knowledge base.

2.1 Big Data Storage

Big data [7] is a term that describes large volumes of high velocity, complex and variable data that require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information. With the increase in the tremendous growth of data and the demand for instant access, there has been research in industry and academics on the distributed storage systems that can be trusted, secure, fault tolerant, runs on low-cost, scale to big data storage, handle dynamic workloads, reliable and are suitable for big data analytics.

Different architecture stores data in different format. Data can be stored as files blocks or objects. File storage cannot scale due to huge metadata and block storage induces latency which requires meta data. Objects can store data and tons of meta data associated with it. Lately objects [8] are the most preferred method of storage and the future building block for storage systems [9]. Objects simplify the storage of big data that can be structured or unstructured text, audio or video or any large binary data.

Q.He et al. in their work [10] discuss the limitation of traditional NAS and SAN storage systems, the importance of distributed cloud storage and the existing cloud storage systems in the industry. Some of the distributed disk based storage systems

in the industry that handle big data are HDFS [3], GlusterFS [2],Cassandra [11],and Ceph [1].

2.1.1 Disk Centric Storage

K.N.Aye et al. [12] discuss that distributed systems like HDFS and GlusterFS with huge number of nodes and disk space meet the scalability needs of big data. Hadoop has become the popular platform for large-scale data analysis. Hadoop Distributed File System(HDFS) [3] is used as the default file system that provides scalability and fault tolerance. However there is a limitation to store metadata with a single namenode system's memory causing bottle neck to scale out. GlusterFS does not have this limitation as it uses elastic hashing algorithms to place data in the system. Similar to GlusterFS, Ceph [1] is a distributed storage system with unified storage and uses CRUSH [13] to place data, removing the need for centralised metadata server. Ceph with its distributed metadata service and ability to expand with cluster size change can be used as a scalable alternate to the HDFS as indicated in [14].

Ceph[1] is open-source, distributed, scalable, reliable and provides high performance. Ceph uses CRUSH [13] hashing algorithm to find the location of the data, avoiding the need for a centralised server and latency in the performance. The clients can find the location and contact data nodes or OSDs directly to fetch and store data. Ceph has tight integration with OpenStack. The benchmarks of the performance of Ceph with OpenStack cloud is presented in [15] and the results indicate that it has good performance and scalability.

Cassandra is a distributed storage system for managing large amounts of structured data, resembling a database, but does not support full relational model. It provides high write throughput enabling clients to have dynamic control over data layout and format.

On the other hand, giants like Facebook and Google have in-built storage systems that suit their needs. Facebook uses its own storage system Haystack and now f4 [16] to store the corpus of photos, images and other Binary Large OBjects(BLOBs). Google uses GFS/Big Table [17] as the distributed storage system to store structured data and handle all the real time queries for the google products.

2.1.2 In-Memory Storage

Recently, there has been a shift from disk centric models to memory centric models, which uses DRAM for all operations to accelerate the performance of systems dealing with large scale data. RAMCloud [18] is one of the storage system which uses the main memories of thousands of commodity servers to process large scale data to improve the performance of the data intensive applications. With this approach to have all data in memory is very costly and replication may not be available for those application which require copies. We may not need DRAM to store all the big data, but is required only for processing and analytics.

2.1.3 Hybrid Storage

On the other hand, there are hybrid models where both in-memory storage and the disk storages are deployed. Memcached [19] is a general purpose key value store in DRAM used in database systems as a cache. The Bigtable [17] storage system used by google can load entire column into memory, reducing the disk reads. The schema enables client to dictate whether to read the data from memory or from disk.

Alluxio[5] is one such memory centric distributed storage system and is primarily integrated with the understorage systems to accelerate data analytics. It is Hadoop compatible and can work with hadoop frameworks like Spark without any new changes. It uses lineage for fault-tolerance.

A Gupta et al. [20] use the fast primary storage as a caching tier to the secondary storage tier. This is achieved by caching subset of data in the primary storage using tiers in VDFS, a scale out file system. The work also compares their approach with Alluxio. Alluxio as a caching layer benefits applications performing data analytics and Baidu has obtained 30x speed up on workloads by eliminating the latency gap between storage cluster and compute cluster. But this is not that very useful for non-big data applications or applications that does check pointing in file system.

This thesis show cases a hybrid model for big data storage and its analytics. TechTalk, a big data application is developed on the hybrid model, a combination of disk centric (Ceph) and memory centric system (Alluxio) to show case the benefits of the hybrid model. Our hybrid model storage system target both the structured and unstructured data that can scale to petabytes or exabytes using Ceph, RADOS object storage system. The memory cache of Alluxio with Ceph as under storage accelerates the data analytics and ad-hoc query response times. The data in Alluxio is loaded from under storage and is used only for analytics. There is no check pointing for intermediate data to the under storage. All the analytics are done with Spark on Alluxio, enabling data sharing at memory speed and requiring no new changes to the storage framework.

2.1.4 Summary

Big data storage demands scale and instant access. To be able to store all data and metadata associated wth big data objects are the most preferred [8] way of storage. There are various distributed systems like HDFS,Ceph,GlusterFS to store big data. HDFS [3] is the default file system used with hadoop to process large datasets, but with its namenode limitation [14] cannot store huge metadata. Ceph and GlusterFS shine better in this aspect of storage [14]. Ceph [1] with its unified storage and tight

integration with OpenStack [15] is suitable for cloud storage for ThoTh Lab. There is a shift to in-memory storage from disk centric systems. RAMCloud [18] works entirely in DRAM, but is very costly to deploy. Memcached [19] is key value store used in databases. Also frameworks like Spark improves the performance, but is limited in sharing data between applications. The hybrid model, with Alluxio [5] acting as a memory cache to under storage and able to share data at memory speed acts as a viable option for big data storage and excellent performance.

2.2 Big Data Analytics

Big Data Analytics require speed, scalability and performance. It demands increase in data processing speeds to access data instantly. As mentioned in [7], size is one dimension of the big data, other dimensions include velocity and variety.

FAST [21] is one of the technique for real time data analytics on the cloud by exploiting the semantic correlation between datasets to reduce processing. This method identifies the required data and narrows the scope of data to be processed. There are other frameworks which uses in-memory storage to accelerate the data analytics in the under storage. Spark [22] hadoop framework uses in-memory storage of data in the form of Resilient Distributed Datasets(rdds) [4]. Spark can share data between jobs and the nodes in the cluster and not applications.

However, Alluxio can share data between applications at memory speed. Applications that use Alluxio for accelerating the performance has been increasing. A cloud computing platform set up for autonomous driving [23] connects Spark and Robot Operating System(ROS). Alluxio is used as in-memory storage with Spark and OpenCL in their infrastructure to speed up analytics. The results show that they have managed to achieve 30X speed over HDFS. Similar techniques were proposed in [24] to use Alluxio as cache in robotic clouds with multiple persistent storages like HDFS,Ceph

and S3 to extract value information from raw data. ArcGIS [25] is now using Alluxio to accelerate the mapping and Spatial Analytics.

YinMem [26] uses Alluxio as the in memory storage system for iterative calculations and storing intermediate results for large scale data analytics. The results show that YinMem has achieved 3x speedup to Spark for computing eigenvalues and eigenvectors of a 16-million scale spark matrix.

An experiment [27] conducted by Alluxio team shows there is a 20x performance benefit in data analytics with the use of Alluxio with Ceph as under storage. The case study [28] shows that Alluxio is used by Baidu to perform do ad-hoc queries and it provides 30x better performance over the traditional model of querying data from direct storage. There are other enterprises like Barclay and Qunar that are adopting Alluxio with under storage to improve the performance.

Similar to the above scenarios, Our thesis uses Alluxio to accelerate the data analytics and for ad-hoc queries with Ceph as under storage. Alluxio is a distributed cluster which can easily expand dynamically. Alluxio can be setup with memory ranging from a few GB to TB. The infrastructure setup for the hybrid model creates avenues to create any new applications that perform analytics with machine learning on the cloud. The infrastructure can be extended to use SparkSQL [29] queries to accelerate the search and information retrieval from the store.

2.2.1 Video Analytics

The big data analytics can be text analytics, audio analytics, video analytics, social media analytics or predictive analytics. With the increase in online and offline videos there is a need to index video content for easy search and retrieval.

Video analytics is becoming popular with video sharing websites and cameras used for surveillance. The key challenge [7] with video content analysis is with the

size of data. Automatic video indexing and retrieval is also another domain of video analytics. The indexing of videos can be done using the metadata obtained from sound track, the transcripts and the visual content of the video. Audio analytics and text analytics techniques can be applied to index a video based on the associated soundtracks and transcripts, respectively.

There are two system architectures defined in [7] for video analytics. First the Server-based architecture where video is captured and streamed to a centralised server for analytics. Second in the Edge-based architecture the analytics is performed on done on the client when the raw data is captured by the system. Server-based architecture are easy to maintain as they are centralised, but with that approach, load balancing and fault-tolerance becomes a problem unless the architecture is distributed and has techniques for replication. With Edge-based approach the onus is on the client side to perform the analytics, requiring resources and software to be present on each client. Similar to the former, our application TechTalk streams the data to the Alluxio cluster and data is analysed on the server to provide references to the online stream.

2.2.2 Online Learning with Video Knowledge Base

Education professionals use video knowledge base for building interactive and smart learning as described in [30]. Students use videos from streaming sites like (e.g., YouTube, Coursera, Khan Academy, EdX, Udacity, Iversity) on a diverse number of terminals (desktop, smart phone, tablets). There are prototypes available for content indexing of videos by image and speech analysis [31] of the videos. Our application is meant to be used as a tool in one such online learning system, ThoTh Lab [6], a cloud-based hands-on virtual laboratory for Computer Science (CS) education for personalised learning. It is a remote web-accessing virtual laboratory

originally designed to reduce lab management overhead for instructors and improve learning experience for CS students. It uses machine learning approaches to model student characteristics during the learning process.

TechTalk is created to assist users in online learning using video knowledge base. In our thesis, we use a simple strategy to content index videos by converting speech to text and extracting keywords from this transcribed text using RAKE [32]. The metadata generated is used for search and data analytics. This technique can extended to any other type of data available as lab content from the online learning system in the form of text files, videos, and audio. The index is simple, easy to parse, store and run analytics.

2.2.3 Summary

Hybrid storage system is widely used in big data analytics. Alluxio is one such system used by - Arcgis for spatial analytics [25], YinMem [26] for iterative calculations, Baidu [28] for search and retrieval, for analytics in robotic clouds [23] [24]. Text, audio, video and social media analytics [7] are the prevalent in big data analytics. The demand for video analytics and the use of video knowledge base is increasing in education sector. Smart learning environments with personalised learning based on user behavior as in ThoTh Lab [6] improves the students learning experience. We design an application TechTalk, to perform video analytics with Alluxio as memory cache to Ceph under storage and use it as a tool in ThoTh Lab to assist users in online learning using video knowledge base.

Chapter 3

STORAGE SYSTEMS

3.1 Storage Architectures

Data can be stored as files, blocks or objects. Each architecture has its own advantages and use cases. The storage types, technologies, applications and how object based storage is an emerging standard in the storage systems is described in [8] and [9].

3.1.1 File Storage

In a file based storage, a file is the smallest unit structured in a file system. All files are organized in a tree like hierarchy and each file can be accessed by its path. A file can be individual or present in a folder called directory. Every file has its contents and metadata associated with it. Metadata is a set of attributes like the size of the file, who created the file, when it was created or modified time,who has access privileges and so on. Users can read, write or execute these files. The file system is responsible for the placement of data, maintenance of metadata,security and as well as implementing file sharing by locking and unlocking files as needed. NAS (Network Attached Storage) is an approach to share files among users on a network. File storage [8] is great for sharing files locally and if the number of files and the associated metadata are limited. They are suitable in the range of thousands to a few millions, but with billions the performance decreases. NAS delivers great performance on a LAN(Local Area Network) but decreases with users on WAN(Wide Area Network).

3.1.2 Block Storage

Block storage is a storage architecture that manages data in the form of blocks. A block is a chunk of data, and when appropriate blocks are combined, creates a file. Each block has an address, and the application retrieves a block by making a SCSI call to that address. Unlike the NAS or file based storage where file system controls the placement of data, here the application decides where to place the data, how to organize the storage and how to combine the blocks. There is no metadata associated with the block like the files. In other words, the block is simply a chunk of data that has no description, no association and no owner. This level of granularity control in the application allows it to extract the best performance from a given storage array. This is the reason why block storage has been widely used in the performance-centric data based and transaction oriented applications. The latency is greatly reduced with the reduce in distance between the application and access of data.

The main advantages with block storage is the granularity [8], no or little storage of metadata and great performance. However, when the distance between application and storage is increased the performance reduces due to latency. To overcome this, block storage is used locally. The examples of block-based storage in use today are direct-attached storage (DAS) and storage-area networks (SAN). DAS connects block-based storage devices directly to the I/O bus of host machines, via SCSI or SATA/SAS. This provides high performance and security, but connectivity can be a problem. SCSI is limited by the width of the I/O bus. This concern for connectivity, and the desire to improve sharing of storage devices led to the development of SAN systems. These provide a switched fabric for enabling fast interconnection between a large number of hosts and storage devices.

3.1.3 Object Storage

Most of the distributed storage systems are based on object storage, in which conventional hard disks are replaced by intelligent object storage devices (OSDs), which combine a CPU, network interface and a local cache with an underlying disk. An Object storage is a storage architecture that manages data as objects. An object is defined as data along with all its metadata, and unique global identifier. This global identifier usually called ID is typically calculated from the content of that object (both file and metadata) itself. Objects are usually put together in a pool or bucket on OSDs. An object is always retrieved by an application by presenting the object ID to object storage. Objects are stored in a flat structure. They may be local or geographically separated, but because they are in a flat address space, they can be retrieved by an application using its id. Unlike file storage, object storage does not allow writing/modifying only a part of the object. The entire object must be updated as a whole.

Object storage provides users flexibility to customise metadata. Each Object has a unique set of metadata, such as the type of application, capabilities, association to other entities, importance of an application and so on. An Object is not limited to any type or amount of metadata.Since an object storage can have tons of metadata, it opens up vast opportunities for analytics. It is widely used for diverse purposes such as to store backups, archives, web content - photos (facebook) and songs (spotify) and provide online collaboration services (Dropbox). Object storage is ideal for storing big data - vast growing content across geographical locations with tons of metadata.

In an object storage system, space for the objects is allocated by the object store itself, and not by higher level software such as a filesystem. All operations on an object, such as reading/writing at a logical location in the object and deleting the

object carry a credential. The object store should verify that the users request carries a valid credential. This allows the storage system to enforce different access rights for different parts of a volume, thus increasing the granularity of access. Unlike a file or block, an object can be accessed using an HTTP-based REST application programming interface. These are simple calls such as Get, Put, Delete and a few others. Their simplicity is an advantage, but they do require changes to the application that were probably written to use SCSI, CIFS or NFS calls. An object store is easy to manage, can scale almost infinitely, transcend geographic boundaries in a single namespace and can carry a ton of metadata, but it is generally lower-performance and may require changes to the application code.Some of the commercial object storage systems are Amazon S3, EMC Atmos and Rackspaces Cloud Files.

3.2 Storage Models

The goal of large distributed systems is to provide users reliability and availability. This is provided by using a cluster of commodity servers with inexpensive disks to store data. But big data analytics is driving new requirements for distributed memory across clusters for real-time streaming, interactive queries, analytics and graph processing.

3.2.1 Disk Centric

As the distributed systems scale out to store large data, new nodes are added to the cluster. Data is replicated over several nodes to ensure availability even if one of the nodes goes down. Frameworks added to the system provide data processing even in case of hardware failures. Hadoop and HDFS [33], which uses the map-reduce programming model, is an example of such a framework. Ceph [1] is another disk centric storage platform designed to present object, block, and file storage from a sin-

gle distributed computer cluster. Ceph's main goals are to be completely distributed without a single point of failure, scalable to the exabyte level, and freely-available. The data is replicated, making it fault tolerant. Even with these in place, with the emerge of big data and real time analytics, there has been a constant demand to increase the processing speed.

3.2.2 Memory Centric

As the importance of speed in data-processing has increased, there is greater focus on storing the data in memory (i.e. RAM) rather than on the disk. Having data in memory results in significantly reduced query response times suitable for data analytics applications. Data stored in memory can also reduce the need for data indexing. With the reduction in RAM costs, and the increased RAM space in 64-bit operating systems, in memory data processing is becoming an increasingly feasible form of data processing.

RAMCloud [18] is one of the storage system which uses the main memories of thousands of commodity servers to process large scale data to improve the performance of the data intensive applications.

The shift towards in memory data processing was highlighted by the development of Apache Spark [4] by UC Berkeleys AMPLab. By use of in memory data structures, it can increase the speed of data processing many fold, compared to a disk-based framework like Hadoop. Spark [5] is a frameowrk can only share in memory data with in the application. But this is over come with Alluxio[5], a memory centric fault-tolerant distributed file system, which enables reliable file sharing at memory-speed across cluster frameworks, such as Spark and MapReduce. Instead of replicating multiple copies of the data, it uses a technique called lineage to maintain a log of the transformations needed to regenerate that data. In the event of data loss, these

transformations are replayed to regenerate the data. Alluxio caches working set files in memory, and enables different jobs/queries and frameworks to access cached files at memory speed. Thus, avoids going to disk to load datasets that are frequently read.

Apache Ignite[34] is another framework which uses in-memory model. Unlike Spark, which uses RAM only for processing, Ignite provides an in memory file system, as well as support for computing paradigms like MapReduce and MPI.

3.3 Software-Defined Storage

There are various systems coming up with the advancement in software like Software-Defined Storage, Software-Defined Networks, and Software-Defined Datacenters as described in [35]. All these systems are managed by an intelligent software rather than the hardware itself. The main goal of these systems is to provide enterprises the abstraction to the heterogeneous hardware used underneath, without having to worry about the interoperability. These systems provide more flexibility, automated management and cost efficiency.

A storage infrastructure that's managed by the software is called Software-Defined storage (SDS). The resources are managed and allocated on demand by the software there by managing proper utilization of resources. It also provides an abstraction of the hardware used underneath. Enterprises can buy hardware from different vendors, but with these systems they don't have to worry about the interoperability of the hardware or under/over utilization of the resources. Since its software based, new modules can be added to the system to provide functionalities like deduplication, replication, snap shots, backup and recovery.

Chapter 4

DISTRIBUTED DISK CENTRIC STORAGE SYSTEMS

This chapter describes the distributed disk storage systems - HDFS [33],GlusterFS [2] and Ceph [1], its architecture and the applications.

4.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) [3] is an open source distributed file system designed to run on commodity hardware and has been written in Java for the Hadoop framework. HDFS was originally built as infrastructure for the Apache Nutch web search engine project and is now an Apache Hadoop subproject. It provides high throughput and is most suited for applications with large data sets. Also being fault tolerant, it is designed to run on low cost hardware. HDFS was built with following norms and assumptions -

- With huge number of machines and support on low cost hardware, hardware failures are a norm rather than exception.
- HDFS is designed for applications that need streaming access to the data sets. They are for batch processing and designed for high throughtput of data access, rather than the general purpose file systems
- HDFS is designed for applications that deal with large data sets but require write once and read many access model. A Map reduce application or a web crawler fits perfectly with this model.
- HDFS is designed for computations on large data sets.
- HDFS is portable across heterogeneous hardware and software platforms.

4.1.1 Architecture

HDFS has a master/slave architecture [33]. The components are described below.

- An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on as shown in Figure 4.1

- The data in HDFS is scattered across the DataNodes as blocks. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.The default size of each block is 128 MB.

- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file systems clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. Apart from these two daemons, there is a third daemon or a process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

- HDFS has file system namespace. Users can create files and group them into directories. The users can create files, delete files and manage user permissions. However the links are not permitted.

- HDFS replicates blocks for fault tolerance.The namenode manages the replication, it keeps monitoring the data nodes via heart beat and keeps track of

the replication. HDFS is rack aware, thus avoiding the replication on the same rack.

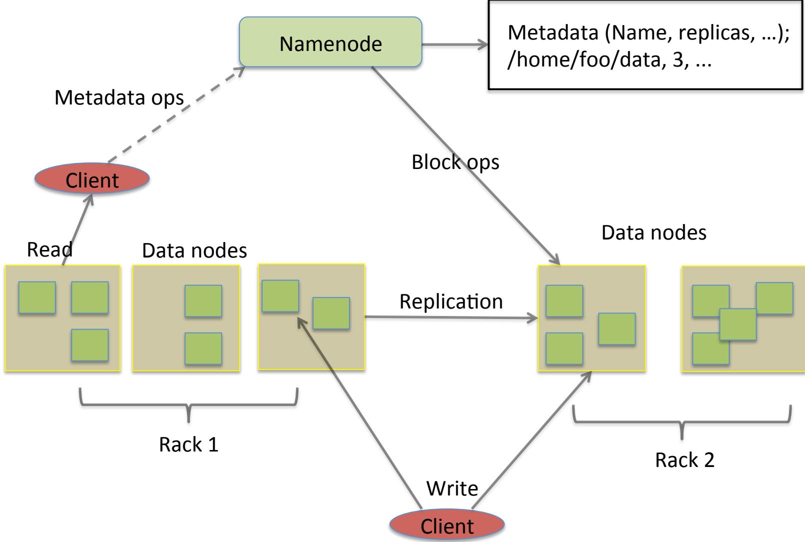


Figure 4.1: HDFS Architecture

4.1.2 Metadata Management

HDFS namespace is stored on the NameNode and it uses a transaction log called the EditLog to persistently record every change that occurs to file system metadata. For example, creation of a new file inserts a record into the EditLog. Similarly, changing the replication factor of a file causes a new record to be inserted into the EditLog. The NameNode stores the edit log in a file in the local file system. The entire file system namespace including the mapping of blocks to files and file system properties is stored in a file called the FsImage. The FsImage is also stored as a file in the NameNodes local file system. The NameNode keeps an image of the entire

file system namespace and file blockmap in memory. When the NameNode starts up, it reads the FsImage and EditLog from disk, applies all the transactions from the EditLog to the in memory representation of the FsImage, and flushes out this new version into a new FsImage on disk trucating the old EditLog because its transactions have been applied to the persistent FsImage. This process of truncation after applying the new version is called as checkpoint. On the other hand, the DataNode stores HDFS data in files in its local file system with no knowledge about HDFS files. It stores each block of HDFS data in a separate file in its local file system. The DataNode does not create all files in the same directory. Instead, it uses a heuristic to determine the optimal number of files per directory and creates subdirectories appropriately. It is not optimal to create all local files in the same directory because the local file system might not be able to efficiently support a huge number of files in a single directory. When a DataNode starts up, it scans through its local file system, generates a list of all HDFS data blocks that correspond to each of these local files and sends this report called Blockreport to the NameNode.

4.1.3 Writing Data to Disk

- When an HDFS client wants to write data to disk.The client divides the file into n blocks each of size that's set in the configuration file. Then it sends the write request to the name node, specifying the number of blocks.The NameNode will then grant the client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually. The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness. For each block, the name node will provide the client with a list of IP addresses of data nodes. The number of data nodes returned will be equal to the replication factor set on the client.

- Before writing the blocks, the client confirms whether the DataNodes, present in each of the list of IPs, are ready to receive the data or not. In doing so, the client creates a pipeline with the first one in the list.

- As the pipeline has been created, the client will push the data into the pipeline.The data is replicated based on replication factor from the first node that receives the data to the other nodes.

- Once the block has been copied into all the DataNodes, a series of acknowledgements will take place to ensure the client and NameNode that the data has been written successfully. Then, the client will finally close the pipeline to end the TCP session.

4.1.4 Reading Data from Disk

While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.

- The client will reach out to NameNode asking for the block metadata.

- The NameNode will return the list of DataNodes where the blocks are stored.

- After that client will connect to the DataNodes.

- The client starts reading data from the DataNodes in parallel.

- Once the client gets all the required file blocks, it will combine these blocks to form a file.

4.1.5 Applications

Primarily HDFS is used as default file system for batch processing of data with MapReduce framework. Spark is another application that by default uses HDFS for processing big data. The HDFS file system is not restricted to MapReduce jobs, it can be used for other applications, many of which are under development at Apache. The list includes the HBase database, the Apache Mahout machine learning system, and the Apache Hive Data Warehouse system. Some of the commercial applications include log and clickstream analysis of various kinds, marketing analytics,data mining,image processing,processing of XML messages,web crawling and/or text processing.

4.2 Gluster File System

GlusterFS is a scale-out network-attached storage file system, developed originally by Gluster Inc and then by Red Hat, which has GlusterFS integrated with Red Hat Enterprise Linux called the Red Hat Gluster Storage

4.2.1 Architecture

GlusterFS has following components.

- Servers or Bricks : Servers are typically deployed as storage bricks, with each server running a glusterfsd daemon to export a local file system as a volume.

- Translators : Translators are setup to handle the user requests and convert them to requests for storage. Both Clients and Server have a set of translators. Various functionalities of the file system are implemented as translators, including file-based mirroring and replication, volume failover, scheduling and disk caching, storage quotas, and volume snapshots with user serviceability.

- Clients : The GlusterFS client process connects to servers with a custom protocol over TCP/IP, InfiniBand or Sockets Direct Protocol, creates composite virtual volumes from multiple remote servers using stackable translators. By default, files are stored whole, but striping of files across multiple remote volumes is also supported. The final volume may then be mounted by the client host using its own native protocol via the FUSE mechanism, using NFS v3 protocol with a built-in server translator, or accessed via gfapi client library. Native-protocol mounts may then be reexported via the kernel NFSv4 server, SAMBA, or the object-based OpenStack Storage (Swift) protocol using the "UFO" (Unified File and Object) translator.

- FUSE : GlusterFS is a user space file system that interacts with kernel VFS and non-privileged user application. The file system provides APIs that can be accessed from user space.

4.2.2 Working of GlusterFS

GlusterFS installed in server mode, has a daemon called glusterd (gluster management daemon) running on each node. After the service start a trusted server pool (TSP) is created with all server nodes. Bricks export the directories in these servers, any number of bricks from this TSP can be clubbed to form a Volume, leaving it up to client-side translators to structure the store. The clients being stateless do not communicate with each other and have translator configurations consistent with each other. After the creation of volume, glusterfsd process runs in each brick belonging to the volume. A set of configuration files are setup, after which the file system is ready to be mounted on the client machines for the local storage.When the volume is mounted on the client, the client glusterfs process interacts with server glusterd

process to exchange information about the bricks in the volume and the list of client translators. With this information, the client directly communicates with the glusterfsd process and the volume is ready to be used by client. Any system call like the file operations when issued by the client in the mounted filesystem, the VFS (Virtual File System) will send the request to the FUSE kernel module. The FUSE kernel module will in turn send it to the GlusterFS in the user space of the client node. The client translators in the GlusterFS will translate these user requests to storage operations. The client has the knowledge of the bricks and the request is propagated to the server. The request will finally reach storage server node that contains the brick in need, the request again goes through a series of translators known as server translators, reach VFS on server and then will communicate with the underlying native file system. The response will retrace the same path. The entire process is described in Figure 4.2.

4.2.3 Types of Volumes

A collection of bricks is called the volume. GlusterFS supports different types of volumes.

- Distributed : Files are distributed across the bricks in a volume. By default, the volumes are created as the distributed volumes. The data is not replicated so any brick failure can cause loss of data.

- Replicated : Files can be replicated across bricks using a replication factor by the client. This provides access to duplicate data on the other brick in case of any failures.

- Distributed and Replicated : Data is distributed across brick as well as replicated. Data is distributed across brick as well as replicated.

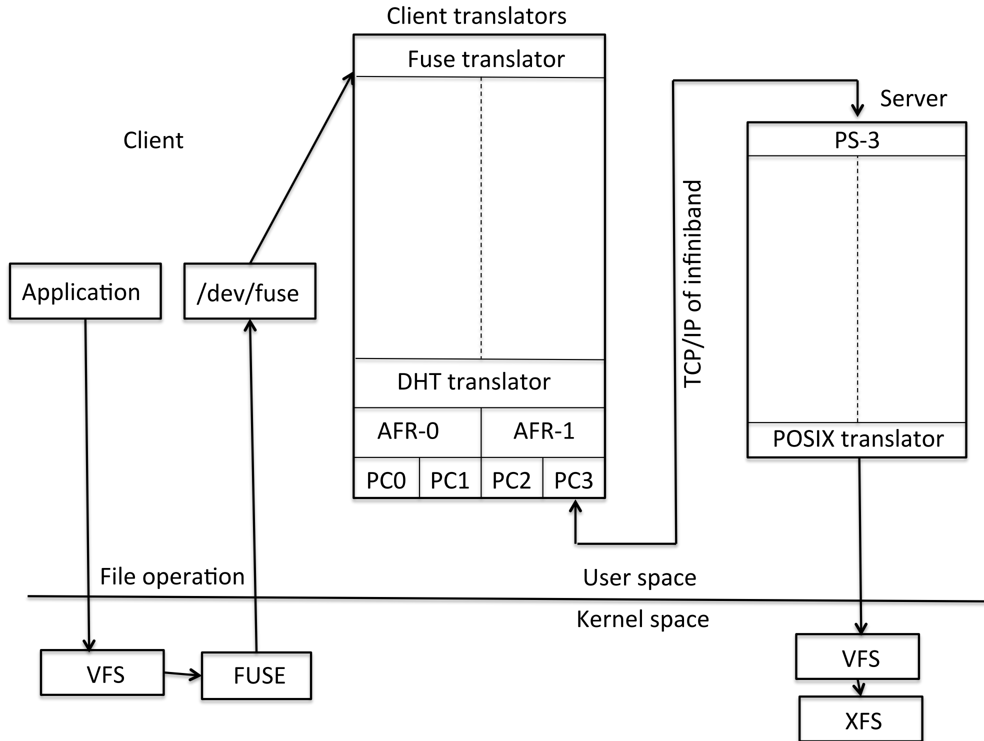


Figure 4.2: Working of GlusterFS

- Striped Glusterfs : A large file, which is accessed frequently reduces performance when placed on a single brick. This method strips the large files into many smaller chunks and places it on each brick there by distributing the load.

- Distributed Striped : This is same as Striped but the stripes are now distributed to more bricks, with the number of bricks a multiple of number of stripes.

4.2.4 Hashing Algorithm

GlusterFS relies on an elastic hashing algorithm, rather than using either a centralized or distributed metadata model. The system places and locates files algorithmically. All storage node servers in the trusted storage pool have the intelligence to locate any piece of data without looking it up in an index or querying another server.

Red Hat Storage uses an elastic hashing algorithm to locate data in the storage pool removing the common source of I/O bottlenecks and single point of failure. Data access is fully parallelized and performance scales linearly.

4.2.5 Applications

GlusterFS is suitable for large scale static files which do not change often, like the images, video, audio files. So its widely used in media industry. GlusterFS applications are used in cloud computing, streaming media services like pandora, and content delivery networks.

4.3 Ceph

Ceph [1] is a free software storage platform designed to present object, block, and file storage from a single distributed computer cluster. Ceph‘s main goals are to be completely distributed without a single point of failure, scalable to the exabyte level, and fault tolerant.

4.3.1 Ceph Philosophy

Ceph is an open source software that is highly community based where anyone in the community can decide on the features and works with all standard storage hardwares. Ceph is now owned by Inktank. The architecture team aims to have Ceph as a self sustaining model that is software oriented rather than a hardware based architecture. If its a hardware based architecture, all fixes to any problem in hardware in a large cluster would involve an immediate manual labor of replacing the defective hardware.Hence Ceph software is self managing and is infinitely scalable, with no single point of failure.

4.3.2 Ceph Architecture

The primary challenge with big data has been to provide a reliable, high performance and scalable storage system. The traditional centralised storage systems with client server storage model for databases, file systems does not provide high performance resulting in I/O bottlenecks as the system scales to thousands of nodes. All the requirements with respect to data placement, failure detection and recovery placed huge burden on the clients, controllers for metadata management there by limiting scalability. To overcome these issues, the distributed cluster storage systems came into light. These systems replaced hard disk drives with Object Storage Devices (OSD) that came with a CPU, network and local cache with an underlying disk. It replaced the file or block level interfaces and provided opportunities to save more data. Clients could directly communicate with the OSDs to do direct I/O reads and writes improving the scalability.

Ceph architecture is based on the assumptions that all large scale systems with peta/exa byte storage is inherently dynamic, these systems are built incrementally and the loads keep shifting; the node failures or replacement of nodes are bound to happen and cannot be treated as an exception. Hence the architecture has to be ready to handle all such scenarios.

RADOS

Ceph storage cluster depends on RADOS (Reliable, Autonomic Distributed Object Store) [36] that scales to thousands of devices by leveraging the intelligence present in individual storage nodes.RADOS preserves consistent data access and strong safety semantics while allowing nodes to act semi-autonomously to self manage replication, failure detection, and failure recovery through the use of a small cluster map.

4.3.3 Ceph Storage Cluster

A Ceph Storage Cluster [1] consists of the below components:

- Ceph monitors (ceph-mon) keep track of active and failed cluster nodes
- Object storage devices (ceph-osd) stores the content data.
- Metadata servers (ceph-mds) store the metadata of inodes and directories
- RESTFUL apis(ceph-rgw) expose the object storage layer as an interface compatible with Amazon S3 or OpenStack Swift APIs.
- CRUSH Algorithm

Ceph monitor and Ceph OSD daemon track the states of the entire system. The master copy of the cluster map is placed on the Ceph monitor. This map is replicated in a cluster to ensure high availability in case of monitor failure. All clients communicate with the Ceph cluster to retrieve the cluster map. Ceph OSD Daemon keeps track of the states of OSD (self and others) and updates the monitors with the information. All the storage cluster clients and each Ceph OSD can compute the data location information using CRUSH algorithm. Ceph decouples metadata operations and data access by using generated hash functions without the need to look up the file allocation tables.

Features

Ceph‘s RADOS [36] provides with object, block, and file system storage in a single unified storage cluster making Ceph flexible, highly reliable and easy to manage. The figure 4.3 depicts the unified cluster.

- Ceph FileSystem (CephFS) is a POSIX-compliant distributed file system.

- Cephs RADOS Block Device (RBD) provides access to block device images that are striped and replicated across the entire storage cluster.
- Ceph Object storage provides(radosgw) seamless access to objects using native language bindings, a REST interface thats compatible with applications written for S3 and Swift.

LIBRADOS is a library (supports C/C++/Java/PHP) that allows applications to access RADOS.

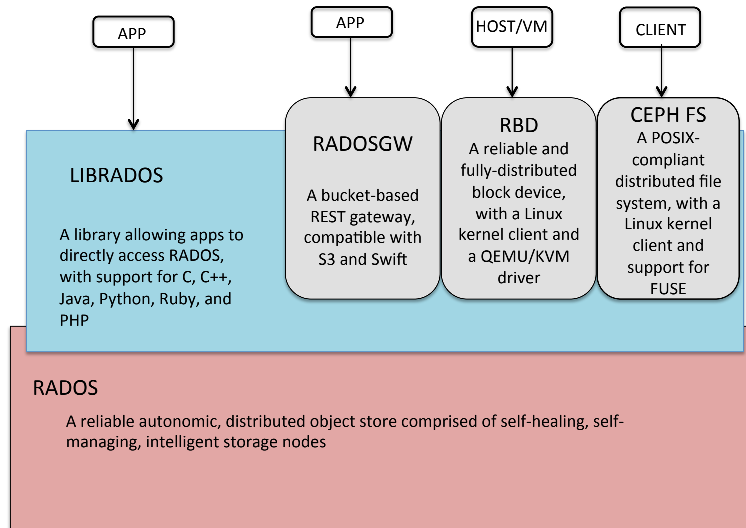


Figure 4.3: Ceph Storage Cluster

Crush Algorithm

Both clients and OSD daemons use CRUSH (Controlled Replication Under Scalable Hashing) algorithm to determine the object location instead of looking at the central

file table or inode list table. CRUSH is a pseudo-random data distribution algorithm that efficiently distributes objects across a storage cluster. It maps inputs such as the identifier of a placement group (PG - a group of objects) to a list of OSDs in which the object and its replicas are to be stored. In addition to the PG id it also takes cluster map, a set of placement rules and replication factor as input. The data distribution is defined by the placement rules that specify how many storage devices are chosen from the cluster for the replicas and the restrictions for replica placement. One such rule could be that three replicas have to be placed in devices on different server cabinets so that they do not share the same electrical circuit.

CRUSH provides a better data management mechanism compared to older approaches, and enables massive scale by evenly distributing the work to all the clients and OSD daemons in the cluster. It uses intelligent data replication to ensure resiliency, which is better suited to hyper-scale storage. The algorithm is consistent with the changes in nodes,that results in minimal object migration to reestablish uniform object distribution. Hence, Ceph uses this algorithm to determine the location of data storage, instead of large look-up tables, which can grow very large for a lot of data.

Cluster Map

Ceph depends upon Ceph Clients and Ceph OSD Daemons having knowledge of the cluster topology, which is inclusive of 5 maps collectively referred to as the Cluster Map as described below.

- The Monitor map contains the cluster fsid, the position, name address and port of each monitor. It also indicates the current epoch, when the map was created, and the last time it changed.

- The OSD map contains the cluster fsid, when the map was created and last modified, a list of pools, replica sizes, PG numbers, a list of OSDs and their status (e.g., up, in).

- The PG (Placement Groups) map contains the PG version, its time stamp, the last OSD map epoch, the full ratios, and details on each placement group such as the PG Id, the Up Set, the Acting Set, the state of the PG (e.g., active + clean), and data usage statistics for each pool.

- The CRUSH Map contains a list of storage devices, the failure domain hierarchy (e.g., device, host, rack, row, room, etc.), and rules for traversing the hierarchy when storing data.

- The MDS (Meta Data Server) map contains the current MDS map epoch, when the map was created, and the last time it changed. It also contains the pool for storing metadata, a list of metadata servers, and which metadata servers are up and in.

Data Management

RADOS manages the distribution, replication and migration of objects. All the data that's received by the Ceph storage cluster is stored as objects on OSDs in a flat

ID	1234
Binary data	01010101010101010 01110001110110001 10010100111111000
Meta data	name1 value1 name2 value2 nameN valueN

Figure 4.4: Ceph Object

namespace. Ceph OSDs handle read and write operations to the disks. Each object has an object id, binary data and metadata associated with it as shown in Figure 4.4. The object identifier id is unique across the cluster and the metadata is a set of name value pair. Along with storage,data replication is also managed by OSDs. OSDs actively collaborate with each other for data replication and recovery in the case of failures. For any direct requests, RADOS sends cluster maps to the client and the MDS. Client uses the CRUSH algorithm to determine the location to perform reads and writes. By computing the object locations, the client can communicate directly with the OSDs, which contributes to scalability of Ceph.

The CRUSH algorithm determines the placement group in which an object will be placed. Objects and their replicas are placed in placement groups. A placement group consists of several objects, while an OSD stores objects belonging to different placement groups. One of the OSD's is called primary. It serializes all requests to the placement group. When it receives a write request, the primary OSD forwards it to all other OSDs in the group. The primary OSD identifies the secondary and tertiary OSDs from the cluster map and replicates the object to the appropriate placement groups in the secondary and tertiary OSDs, and responds to the client that object is stored successfully. If it has to write locally as well, it does so only after the other OSDs have written the replicas to their memories. The client then receives an acknowledgement from the primary OSD. At this point, the data have been replicated in memory in all the replica OSDs. The client still has the data in its buffer cache. Only after the data have been committed to the disks of all the replica OSDs does the primary OSD send a commit notification to the client. The client can then delete the data from its buffer cache.

Data is asynchronously written using the copy-on-write method. This means that once-written data are no longer directly modified. Instead, the data to be changed are

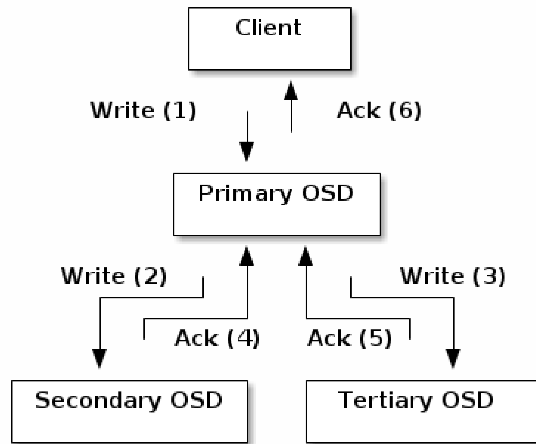


Figure 4.5: Ceph Replication

written in newly allocated blocks, and the relevant metadata are accordingly updated to point to the new blocks, rather than the old ones. This ensures unsuccessful write operations can be fully rolled back. Each OSD maintains a log of object versions for each placement group in which it participates. If an OSD fails, the other OSDs of the placement group can identify missing objects by comparing the logs, and thus recover the object. Each OSD monitors the state of other OSDs in its placement group using heartbeat messages, which are sent along with replication traffic. When an OSD detects an unresponsive OSD, it notifies the monitor of the situation and, in response, receives a new cluster map from the monitor marking the unresponsive OSD as down. Heartbeat messages are still sent to this unresponsive OSD and, after some time, if it still does not respond, the monitor is accordingly alerted and it issues a new cluster map marking that OSD as out.

Monitors

Ceph monitors are light weight processes that manages the cluster map and maintains a master copy of the cluster map. It also provides authentication and logging services. It uses a variation of Paxos protocol[37] to establish consensus about maps and other information across the cluster. When object storage devices fail or new devices are added, monitors detect and create a new cluster map.

A client can determine the location of all monitors (there can be multiple monitors in a cluster), OSD daemons and metadata servers by connecting to one monitor and getting a cluster map. With the current copy of the cluster map and the CRUSH algorithm, the client can compute the location for any object. The nature of Paxos requires majority of monitors to be running to establish a quorum. The minimum requirement of a Ceph cluster is to have one monitor but that introduces single point of failure. If the monitor is down, then Ceph clients cannot retrieve the cluster map. Hence its advised to have a cluster of monitors. Ceph website [37] advises to have an odd number of monitors. An odd number has higher resiliency than even numbers. That is with in a cluster, its possible that some of the monitors can fall behind the current state of the cluster. Example, on a 2 monitor deployment, there can be no failures in order to maintain a quorum. With 3 monitors, one failure can be tolerated; in a 4 monitor deployment, one failure can be tolerated; with 5 monitors, two failures can be tolerated. Hence an odd-number is advisable and Ceph uses a majority of monitors (e.g., 1, 2:3, 3:5, 4:6, etc.) and the Paxos algorithm to establish a consensus among the monitors about the current state of the cluster.

Cephx Authentication

Ceph uses cephx authentication system that uses shared keys for authentication. The client and monitor should have the client's secret key for communication. The protocol is such that the client can be authenticated without revealing the secret key. The Ceph client contacts the monitor. The monitor returns an authentication data structure similar to a Kerberos ticket that contains a session key for use in obtaining Ceph services. This session key is itself encrypted with the users permanent secret key, so that only the user can request services from the Ceph monitor(s). The client then uses the session key to request its desired services from the monitor, and the monitor provides the client with a ticket that will authenticate the client to the OSDs that actually handle data. Ceph monitors and OSDs share a secret, so the client can use the ticket provided by the monitor with any OSD or metadata server in the cluster. The protection offered by this authentication is between the Ceph client and the Ceph server hosts. The authentication is not extended beyond the Ceph client. If the user accesses the Ceph client from a remote host, Ceph authentication is not applied to the connection between the users host and the client host. To avoid central point of contact, each monitor has an ability to authenticate users and distribute keys. Cephx session keys also expire, preventing the attacker from using another users identity or altering the previous messages.

Scalability and High Availability

In traditional client server model, clients talk to server for all operations. This hinders performance and scalability introducing a single point of failure, if the central server is down then the whole system is down. However Ceph address this using a cluster, where there is no centralised gateway. The clients talk to the OSDs directly to read

or write data. This is achieved by the use of CRUSH algorithm. Ceph data is also replicated on other nodes to achieve high availability and for data safety. Ceph OSDs and clients knows about the cluster. Like Ceph clients, each Ceph OSD Daemon knows about other Ceph OSD Daemons in the cluster. This enables Ceph OSD Daemons to interact directly with other Ceph OSD Daemons and Ceph monitors. Ceph has some smart features listed below which makes it better than the centralised server.

- This ability of Ceph clients, monitors and OSD daemons to interact with each other means that Ceph OSD Daemons can utilize the CPU and RAM of the Ceph nodes to easily perform tasks that would bog down a centralized server.
- Ceph OSD Daemons use the CRUSH algorithm to determine the object location and computes the replicas of the objects. There is no lookup or maintenance of huge amount of metadata like the filesystem inodes to determine the location.
- Ceph clients connect to OSD directly and thus maintains the session to the OSD. This removes the bottle neck of a centralised component having a limited number of connections it can support.
- Ceph OSD joins a cluster and has two states - up and down. The state up denotes that the Ceph OSD is running, and down denotes either the OSD daemon is not responding to requests. When Ceph osd is down it cannot update its status to the monitor. To handle this, the Ceph monitor has a light weight process to ping the OSD periodically and update its status. In addition, the OSDs can determine if the neighbouring OSDs are down and update the monitor, thus reducing the burden on the monitor.
- Ceph OSD daemons compares object metadata or object data in the replicas to

clean up or scrub objects with placement groups. This process of data scrubbing detects file system errors, bad sectors in a drive and other device related bugs.

4.3.4 Ceph Supported Storage Types

Ceph Object Storage

Ceph software libraries provide direct access to the object based storage system RADOS to the client applications. These libraries also provide a foundation for some of Ceph's advanced features, including RADOS Block Device (RBD), RADOS Gateway, and the Ceph File System.

- The Ceph librados software libraries enable applications written in C, C++, Java, Python and PHP to access Cephs object storage system using native APIs.

- RADOS Gateway provides Amazon S3 and OpenStack Swift compatible interfaces to the RADOS object store.

Ceph Block Storage

Ceph can be mounted as a thinly provisioned block device. Ceph RBD interfaces with the same Ceph object storage system that provides the librados interface, and it stores block device images as objects. Since RBD is built on top of librados, RBD inherits librados capabilities, including read-only snapshots and revert to snapshot. Ceph improves read access performance for large block device images by striping images across the cluster. Ceph's RADOS Block Device (RBD) also integrates with Kernel Virtual Machines (KVMs), enabling Ceph's virtually unlimited storage to KVMs running on Ceph clients.

Ceph File System

Ceph provides a traditional file system interface with POSIX semantics. Ceph file system runs on top of the same object storage system that provides object storage and block device interfaces. The Ceph metadata server cluster provides a service that maps the directories and file names of the file system to objects stored within RADOS clusters. The metadata server cluster can expand or contract, and it can rebalance the file system dynamically to distribute data evenly among cluster hosts. This ensures high performance and prevents heavy loads on specific hosts within the cluster.

4.3.5 OpenStack and Ceph

OpenStack is free and open source software platform for cloud computing which provides infrastructure as a service (IaaS). It provides set of tools to build and manage private and public clouds. It started as a joint program by Rackspace and NASA and is now managed by OpenStack Foundation.

Ceph is open source, scalable and distributed and is becoming the best storage solution for cloud computing technologies. There are several reasons for Ceph to be an under storage for OpenStack. Figure 4.6 provides an overview of how the data from OpenStack is configured to interact with the back end Ceph storage.

- Ceph is scale-out unified storage platform and satisfies the requirements of OpenStack. OpenStack requires scaling in all types of storage - file (Manila), block (Cinder) or Object (swift). All these are integrated in one storage system unlike the traditional ones, which provides different interface for each storage type.
- Ceph is cost effective and leverages Linux as the Operating system and not

any proprietary operating system thus letting the users buy hardware from a single vendor. Ceph is open source project like OpenStack, this helps in tighter integration and cross project development.

OpenStack as defined by the company is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack has several components each providing different functionality. The benchmarks presented in [15] also shows that Ceph provides both reliability and scalability with OpenStack cloud.

Glance

OpenStack Image (Glance) is the Image Service that can store disk and server images in a variety of back-ends including Swift. Glance can add, delete, share, or duplicate images.

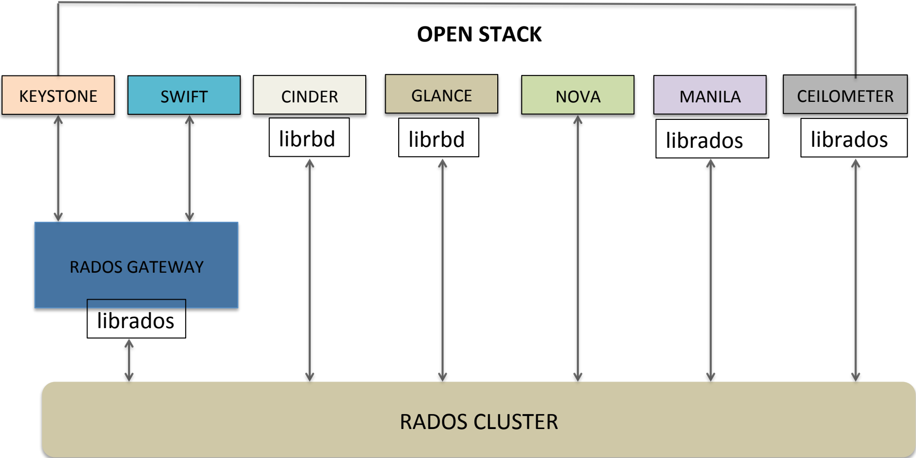


Figure 4.6: Ceph as Storage for OpenStack

Keystone

OpenStack Identity (Keystone) is the authentication service and provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins.

Nova

OpenStack Compute (Nova), the compute service is a cloud computing fabric controller that manages and automates pools of computer resources. It can blend well with virtualization technologies like KVM, VMWare, and Xen as well as with high-performance computing (HPC) configurations.

Neutron

OpenStack Networking (Neutron) is the networking service for managing networks and IP addresses. It provides networking models ranging from flat networks to VLANS for different applications or user groups and manages static and DHCP IP address. Users can create their own networks, control traffic, and connect servers and devices to one or more networks. Administrators can use Software-defined Networking (SDN) technologies like OpenFlow to support high levels of multi-tenancy and massive scale. OpenStack networking provides an extension framework that can deploy and manage additional network servicessuch as Intrusion Detection Systems (IDS), load balancing, firewalls, and Virtual Private Networks (VPN).

Cinder

OpenStack Block Storage (Cinder) is the block storage service and provides persistent block-level storage devices for use with OpenStack compute instances. It creates an abstraction around block storage by providing a driver. Vendors can use this driver to integrate with the block storage.

Horizon

OpenStack Dashboard (Horizon) provides administrators and users a graphical interface to access, provision, and automate cloud-based resources.

Swift

OpenStack Object Storage (Swift) is the object storage service widely used in cloud computing. It's designed to scale enormously, always available and highly fault tolerant.

Ceilometer

OpenStack Telemetry (Ceilometer) provides a single point of contact for billing systems across all current and future OpenStack components. The delivery of counters is traceable and audited. The counters must be easily extensible to support new projects and agents doing data collections and should be independent of the overall system.

Manila

OpenStack Shared File System (Manila) is a file system service used in creating and managing shares in a vendor agnostic framework in a standalone or in a network environment.

Chapter 5

CLOUD INFRASTRUCTURE

This chapter describes how to set up the back end infrastructure for a cloud that provides a highly-customizable virtual laboratory platform for users in research and education. Users can design, construct, and deploy a virtualized computing system for computer related labs. This lab environment ranges from a wide range of applications and network topologies that mimic a real world computing lab.

5.0.1 Features

We need to consider the requirements and features before choosing the back end storage for the cloud. Primarily like any other clouds, the infrastructure requires to be Software Defined Storage [35] that can scale to support large scale storage. The platform should be able to expand on demand with increase in users. The size of data for each lab can vary from giga bytes to peta bytes. The cloud storage system [10] should be distributed, reliable and always available. There should be no single point of failure and on any failure, it should be able to recover and restore to the last check point to be able to use the labs again. The system should be data agnostic, that is should be able to store files, blocks or objects. The labs will have an ability to create and manage virtual machines with various applications and have different network topologies. Since the platform is designed for research and education, it is better to choose an open source storage platform that can be deployed at low cost and further customized as required. In addition to this, the storage platform must be suitable for big data analytics and provide high performance with reduced latency.

5.0.2 *Comparative Study of Distributed Storage Systems*

We compare three of the Software Defined Storage systems properties by studying the architecture, technologies and methods applied as described in Hadoop [33], Ceph [1] and GlusterFS [2]. Table 5.1 shows a comparative study of the three open source distributed storage systems with the key features, their similarities and the differences with respect to the features available. We also study the experiments conducted in [38] to understand the features available in each of the distributed file system.

Apache Hadoop, an open-source framework, is the primary and default file system for most of the big data analytics applications. Ceph and GlusterFS, are independent community projects, and are available under the GNU Public License version 3 (GPLv3) for maximum openness. Both of these are open source, scale-out, Software Defined Storage products running on commodity hardware. Red Hat is a heavy investor in both projects, allowing it to deliver enterprise grade versions of both software applications, known as Red Hat Ceph Storage and Red Hat Gluster Storage, respectively. Both of these appear to solve same problems,but each have different capabilities and perform at best in different applications. Red Hat suggests [39] Ceph is better for OpenStack and Gluster for big data analytics, but both could do either job.

Table 5.1: Comparison of Distributed Storage Systems

Property	Hadoop	Ceph	Gluster
Definition	Open source file system used for distributed storage and processing large data sets using MapReduce programming model.	Open source scalable Object storage on a single distributed cluster without a single point of failure and scalable to exabyte level.	Scale out network-attached distributed storage file system mostly used in cloud computing.
Type of storage	Files split into blocks	An object-store system, called RADOS, with a set of gateway APIs that present the data in block, file, and object modes.	GlusterFS is scale out NAS supporting files and in addition supports block-based storage and object storage with swift by creating an additional abstraction layer.
Architecture	NameNode, DataNodes, BackupNode and Secondary NameNode (Check point Node).	Admin node, OSD (Object Storage Devices), MON (Monitors), CephFS, RBD, RADOS Gateway	Bricks, Translators, Volumes and Sub volumes.
Data Placement	HDFS splits the data onto the data nodes in the cluster.	Ceph uses CRUSH algorithm to place data within the placement groups. The client uses the algorithm to determine where the data has to be placed.	Gluster uses a hashing algorithm to place data within the storage pool.

Features	Scalable, fault tolerant, faster data processing by parallel and batch processing	Fault tolerant providing replication, Information distribution, Scalability	Scalable, Cost effective data management platform for streaming file and object in cloud environments,
Client Communication	Client talks to data nodes with the information received from name nodes.	Client communicate with the monitors for the cluster map and performs read writes directly on to OSDs.	Clients directly communicate with Glusterfsd.
Fault tolerance and Replication	Data is replicated on to the data nodes, but HDFS is rack aware. So the data is not replicated on to he same rack.	The redundancy factor is maintained at pool level and the distribution is managed by CRUSH. The data is replicated on to OSDS.	The redundancy is achieved by replicating data in bricks.
Default Block size	64-128 MB	64 KB	128 KB
Journals	N/A	All data is first written to the journals and then replicated to OSDS.	N/A
OpenStack Suited	Not Applicable	The Ceph Block Device (RBD) has extensive integration with Nova, Cinder and KVM.	File-based architecture which, while capable of running block services, makes it less optimised for backing Nova and Cinder.
Applications	Various big data applications - HBase, spark, Hive to run map reduce jobs.	Yahoo/Flick to use as the storage cluster	Steaming services like Pandora for fats lookups and data retrieval

5.0.3 Inference on the Back End Storage

From the above comparisons, the most suitable and cost effective solution would be choose Ceph as the back end storage.

- Ceph provides file,block and object storage from a single distributed computer cluster.
- Ceph is completed distributed, scalable and enables replication there by making it fault tolerant.
- Ceph is open-source and provides a unified storage solution, which saves more man power to maintain another different setup for each type of storage.
- Easily integrate applications like OpenStack. Hence, Ceph is the viable storage platform for VM images and application data in OpenStack.

5.0.4 Ceph Cluster Recommendation

Ceph website [40] recommends a few requirements and suggestions to setup a cluster that is scalable, reliable and no single point of failure with maximum read and write performance.

Minimum Hardware Requirements

As per the Ceph website [40],table 5.2 gives the minimum hardware configurations to set up the Ceph storage.

Table 5.2: Minimum Hardware recommendations

Process	Criteria	Minimum Recommended
ceph-osd	Processor	1x 64-bit AMD-64
		1x 32-bit ARM dual-core or better
		1x i386 dual-core
		RAM 1GB for 1TB of storage per daemon
		Volume Storage 1x storage drive per daemon
		Journal 1x SSD partition per daemon (optional)
		Network 2x 1GB Ethernet NICs
ceph-mon	Processor	1x 64-bit AMD-64/i386
		1x 32-bit ARM dual-core or better
		1x i386 dual-core
		RAM 1 GB per daemon
		Disk Space 10 GB per daemon
		Network 2x 1GB Ethernet NICs
ceph-mds	Processor	1x 64-bit AMD-64 quad-core
		1x 32-bit ARM quad-core
		1x i386 quad-core
		RAM 1 GB minimum per daemon
		Disk Space 1 MB per daemon
		Network 2x 1GB Ethernet NICs

Monitors

Ceph monitors are light weight processes which maintain the master copy of the cluster map. It is recommended [41] to use odd number of monitors, as odd number of monitors has more resiliency to failures than even number of monitors. Ceph monitors uses a variation of Paxos [37] protocol to sync maps and other information across the cluster. This protocol requires most of the monitors to be up and running to establish consensus or be in quorum.

Pools and Placement Groups

As described in Ceph website [42], Pools are the logical groups for storing data on the OSD. If there is no pool defined, it goes to the default pool. Ceph maps objects to placement groups (PGs), which are shards or fragments of a logical object pool that place objects as a group into OSDs. Placement groups reduce the amount of per-object metadata when Ceph stores the data in OSDs.

Journals

Ceph uses journal for speed and consistency. All the writes are first done to the journal configured per OSD and then queues the writes to journals to create replicas. Hence journal is configured for each OSD, but on a different hard disk or SSD. If journals are not configured, then Ceph stores the journal on the same disk as the Ceph OSD data. But that reduces the performance. Using a separate disk for journals other than the OSD is recommended by the ceph website.

5.1 MobiCloud

MobiCloud is a smaller cluster (similar to ThoTh Lab) providing a platform for students to create a lab, be able to build and manage virtual machines for learning.

Admin	1
Monitors	3
OSD nodes	9
MDS	N/A

Table 5.3: Ceph Configuration

We intend to install Ceph cluster and tune the parameters to obtain maximum read and write performance. The next step is to integrate OpenStack with Ceph as the back end storage.

5.1.1 Ceph Configuration

Ceph cluster consists of three nodes built on virtual machines ceph-left,ceph-middle and ceph-right.

- The Ceph cluster has debian-hammer version installed with one admin node, three monitors and 9 OSDs and the configuration is listed in table 5.3.
- Ceph monitors : 3 monitors are setup with 10G network, 40GB RAM and with ubuntu 14.04 installed, listed in table 5.4. Each node containing 3 OSDs with a total 9 OSDs as in table 5.6. The Ceph admin node configuration is same as the monitor node, is listed in table 5.5.
- Journals : Each OSD is configured with a journal of 20 GB on hard disk.
- Replication size: The number of copies of the object to be created. The minimum replica is 2 and extended to 3 copies.
- Ceph placement groups: Ceph website [42] provides guidelines on the number of placement groups to be configured. We benchmark with different pgs in each

Table 5.4: Ceph-Mon Configuration

nodes	ceph-middle,ceph-left,ceph-right
Network Speed	10G
Ethernet	p2p5(10G), p5p2(10G)
RAM	40GB
CPU	Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz,6 cores
OS	Ubuntu 14.04 trusty 64 bit

Table 5.5: Ceph-Admin Configuration

nodes	ceph-middle
Network Speed	10G
Journal SSD size	127 GB
Ethernet	p2p5(10G), p5p2(10G)
RAM	40GB
CPU	Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz,6 cores
OS	Ubuntu 14.04 trusty 64 bit

pool and tune the cluster to get maximum benefit.

Ceph Status

The status and health of the cluster can be determined using Ceph health command as indicated in table 5.7. The command returns the cluster configuration and status of the pools, placement groups, size of the cluster and data that's in them.

Table 5.6: Ceph-OSD Configuration

nodes	ceph-left	ceph-middle	ceph-right
Number of OSD	3	3	3

Table 5.7: Ceph Status

cluster	5d0ce9fd-d3ae-4a97-8072-b3755d8df84f
health	HEALTH_OK
monmap	e1: 3 mons at Ceph-left=192.168.2.218:6789/0,Ceph-middle=192.168.2.220:6789/0,Ceph-right=192.168.2.222:6789/0, election epoch 86, quorum 0,1,2 Ceph-left,Ceph-middle,Ceph-right
osdmap	e441107: 9 osds: 9 up, 9 in
pgmap	v763245: 512 pgs, 4 pools, 39021 MB data, 9758 objects 164 GB used, 22110 GB / 22274 GB avail 512 active+clean

Initial Benchmarks

The read and write speeds in the Ceph cluster is listed in the figure 5.1.The initial bench marks had a very low performance of 20Mb/s. The diagnostics of a few of the parameters influencing the cluster performance is listed below.

- **Hard Disk Speed** : OSDs are setup to store the data and the speed of the reads and writes to the hard disk is determined using the 'dd' command. The disk speed as listed in Listing: 5.3 is 227 MB/s.

- **Network Speed** : Data transfer from the client to the OSD is done via the network and the speed of the network is determined using the iperf command.

Pool	Pool Id	Pg num	Replication Size	Crush Rule Set	Number of Writes	Number of Reads (Seq/Ran)	Write (MB/s)	Read (MB/s)
data	0	64	2	1	60	60 S	18	11
data	0	128	2	1	60	60 S	20	30

Figure 5.1: Ceph Cluster Read Write Performance

The network speed listed in Listing: 5.1 is ≈100Mb/s.

- **Monitors** Recommended to have an odd number of monitors and the cluster has three monitors, which is still good wrt to resiliency against failures. Even if one monitor fails, we have two more to be able to communicate with each other.

Listing 5.1: Ceph Cluster Network Speed

```
ceph@Ceph-right:~$ iperf -c 192.168.2.220 -i1 -t 10
------------------------------------------------------------
Client connecting to 192.168.2.220, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.2.222 port 43683 connected
with 192.168.2.220 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  12.4 MBytes   104 Mbits/sec
[  3]  1.0- 2.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  2.0- 3.0 sec  11.1 MBytes  93.3 Mbits/sec
[  3]  3.0- 4.0 sec  11.1 MBytes  93.3 Mbits/sec
------------------------------------------------------------
```

5.1.2 *Ceph Benchmarks*

We inspect a few parameters and tune those settings to test the performance of the Ceph cluster. Following a few other recommendations as indicated below.

- Journal: Use SSD as Journal. The speed of the SSD is almost twice that of the HDD, as indicated in Listing: 5.2 is 447MB/s. Since the journal write speed is higher, the queue to write is cleared faster and thereby increases the speed of writes in the cluster. OSDs may see a significant performance improvement by using journal on an SSD and the OSDs object data on a separate hard disk drive.

- Network speed: Increasing the network speed from 100Mb/s to 1Gb/s, further to 10Gb/s to see the impact on the Ceph cluster.

- OSD rules : Following a few recommendations for the OSD rules:
  - 1GB of RAM for 1TB of storage space.
  - Distribute number of OSD same on all nodes for load balance and fault tolerance.

- Crush Rule Set: By default the rule set is set to 0. We can define different rules based on how the data placement is required.

Listing 5.2: Journal SSD Read Write Speed

```
ceph@Ceph-middle:~/ceph-cluster$ dd if=/dev/zero of=here bs=1G
count=1 oflag=direct
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 2.40412 s, 447 MB/s
```

Listing 5.3: Ceph OSD HDD Read Write Speed

```
ceph@Ceph-middle:~$ sudo dd if=/dev/sdb1 of=here
bs=1G count=1 oflag=direct
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 7.52179 s, 143 MB/s
```

Ceph benchmarks were computed using the Ceph command. The pools, size of each pool and the RADOS bench mark are listed below.

- Various pools are created with the placement groups and figure 5.4 gives the list of pools configured. The pg num was adjusted based on the recommendations given in Ceph website [42].
- There are 6 pools and each pool contains data of variable size. The data size of the largest pool images is of size 1.6 TB. It contains the images of all operating system needed in ThoTh Lab, is listed in Figure 5.5
- Rados benchmark shows the read and write speed of the cluster and the data placement in MB/s.The average read speed is ≈1450MB/s and average write speed is ≈160MB/s as given in Figure 5.6.

Listing 5.4: Ceph Pools

```
ubuntu@ceph1:~$ ceph osd dump | grep size
pool 0 'rbd' replicated size 2 min_size 2
        crush_ruleset 0 object_hash rjenkins
        pg_num 256 pgp_num 256 last_change 64
        flags hashpspool stripe_width 0
pool 1 'test' replicated size 2 min_size 2
        crush_ruleset 0 object_hash rjenkins
        pg_num 512 pgp_num 512 last_change 60
        flags hashpspool stripe_width 0
pool 2 'images' replicated size 3 min_size 2
        crush_ruleset 0 object_hash rjenkins
        pg_num 128 pgp_num 128 last_change 365
        flags hashpspool stripe_width 0
pool 3 'volumes' replicated size 3 min_size 2
        crush_ruleset 0 object_hash rjenkins
        pg_num 128 pgp_num 128 last_change 84
        flags hashpspool stripe_width 0
pool 4 'backups' replicated size 2 min_size 2
        crush_ruleset 0 object_hash rjenkins
        pg_num 128 pgp_num 128 last_change 362
        flags hashpspool stripe_width 0
pool 5 'vms' replicated size 3 min_size 2
        crush_ruleset 0 object_hash rjenkins
        pg_num 128 pgp_num 128 last_change 88
        flags hashpspool stripe_width 0
```

Listing 5.5: Ceph Pools Size

```
ubuntu@ceph1:~$ rados df
pool name                 KB      objects
backups              8323073         2033
images            1733508069       211787
rbd                  3923969          959
test                97935361        23911
vms               2163680685       267868
volumes                    0            1

total used       11909957972       506559
total avail      34944790828
total space      46854748800
```

Listing 5.6: Ceph Rados Benchmark

```
ubuntu@ceph1:~$ rados bench -p test 10 write --no-cleanup
                && rados bench -p test 10 rand
Maintaining 16 concurrent writes of 4194304 bytes
        for up to 10 seconds
Object prefix: benchmark_data_ceph1_40886
  sec Cur ops   started  finished  avg MB/s  cur MB/s
    0       0         0         0         0         0
    1      16        68        52   207.953       208
                    ...
    9      16       401       385   171.089       216
   10      16       423       407   162.779        88
Total time run:         10.604052
Total writes made:      424
Write size:             4194304
Bandwidth (MB/sec):     160
 sec Cur ops   started  finished  avg MB/s  cur MB/s
    0       0         0         0         0         0
    1      15       363       348   1391.65      1392
                     ...
    9      15      3272      3257   1447.32      1488
   10      16      3640      3624   1449.37      1468
Total time run:        10.056857
Total reads made:      3640
Read size:             4194304
Bandwidth (MB/sec):    1.45e+03
```

With the tuning of settings in the Ceph Cluster, the performance benchmarks are computed. The performance with the change in replication factor, network speed and use of journals are indicated in Figure 5.2(b),Figure 5.2(c) and Figure 5.2(a) respectively.

- With the use of SSD as Journal, the read speed further increases by 10x and write speed by 5x as indicated in the Figure 5.2(a).
- The read and write speed decreases with the increase in replication size. The benchmark shows that the read and write speed with replication factor 2 is 180MB/s and 120MB/s respectively. But with replication size 3, the write goes down to 50MB/s and read to 80MB/s. This is plot in Figure 5.2(b).
- The read and write increases with the increase in network speed and the performance obtained with the use of SSD and replication size 2 is indicated in Figure 5.2(c). Overall, after these fixes, read speed increases from 20MB/s to 1450MB/s and write speed increases from 18MB/s to 160 MB/s, which is about 72x and 10x speed respectively.

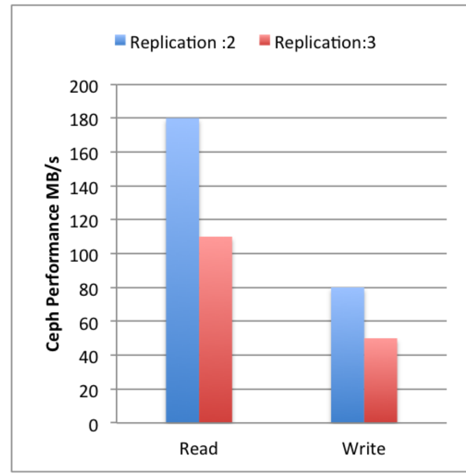
5.2 ThoTh Lab

ThoTh lab [6] is a platform, where users can design, construct, and deploy a virtualized computing system for computer-related labs.

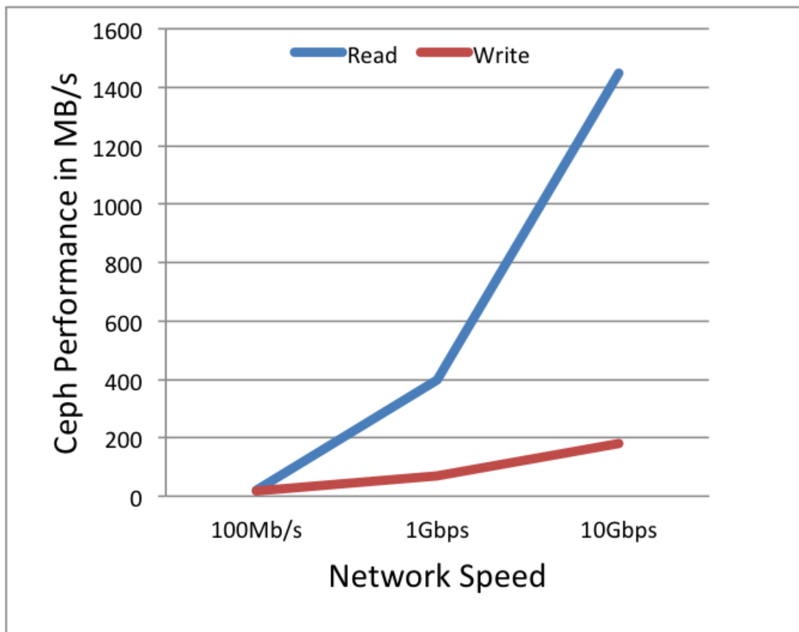
5.2.1 Motivation

The traditional lab model has serious challenges. The learning system is common to all students. Individual learner should be treated differently base on their needs and preference. Adaptive web technology gives learners opportunities for taking ownership of their learning. Personalized Lab can be paced to student needs, tailored to

(a) Type of journals



(b) Replication factor



(c) Network speed for a replica size of 2

Figure 5.2: Ceph Performance w.r.t Different Factors

learning preferences, and customized to the knowledge level of different learners. Personalized learning is often referred to a new learning approach by taking individual parameters such as learning preferences, abilities, skills and knowledge into account. The personalized learning models are established in ThoTh Lab [6] - a cloud-based hands-on virtual laboratory for Computer Science (CS) education.

ThoTh Lab is a remote web-accessing virtual laboratory and it was originally designed to reduce lab management overhead for instructors and improve learning experience for CS students. By introducing new personalized learning capabilities, we can transfer ThoTh Lab from a traditional hands-on lab resource provisioning system to an active personalized e-learning platform for CS education. The system can track and assess students hands-on projects or activities to monitor students lab performance, and then provide intelligent suggestions or resources to improve students learning experience and outcomes.

The challenge in this model is to take user learning preferences into account while constructing the personalized learning environment. This requires collection of large scale of user behavior data. So the infrastructure must be setup to store this data and be able to perform online analytics on this data at memory speed.

5.2.2 Infrastructure in ThoTh Lab

The architecture of the lab created using Ceph as back end storage and the applications running on it is described in Figure 5.3. All the applications are configured to interact with the Ceph under storage and store the lab content in Ceph. OpenStack is one such application which provides a set of tools for Cloud Computing. Since Ceph can store file, blocks and objects, OpenStack is configured to the Ceph storage to create virtual machines, volumes for storage, manage different networks and create other lab content.

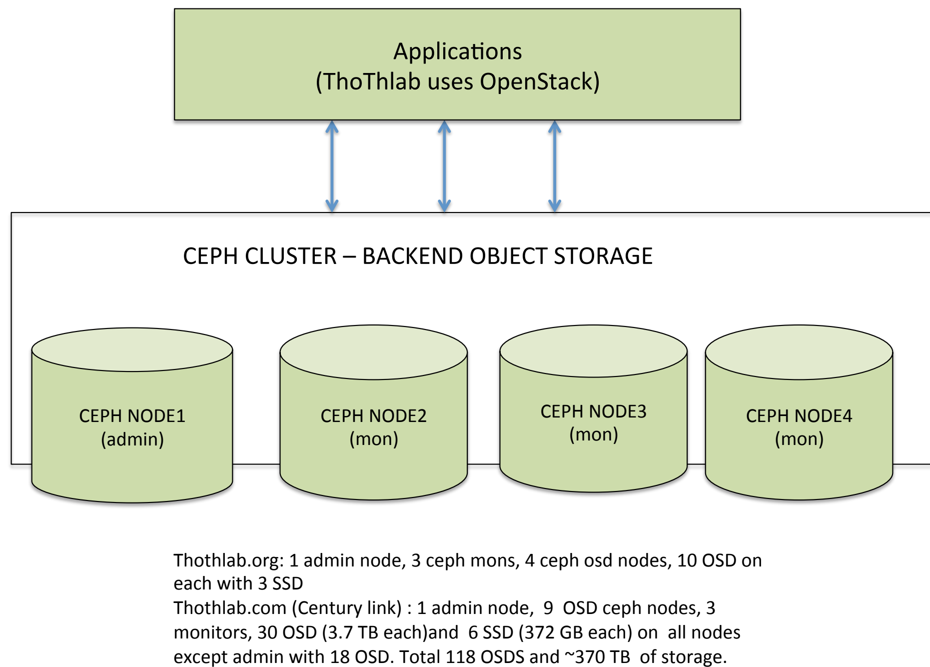


Figure 5.3: Thothlab Architecture

5.2.3 Ceph Configuration

The hardware configuration of the back end storage is given in the table 5.8. The back end storage can support upto ≈300TB of data upto 9TB of SSD for the journal space.

Ceph Cluster

Ceph cluster is configured with one admin node, three monitors and seventy OSD nodes with a single meta data server with RADOS gateway to cater to the needs of the clients via REST APIs. The cluster configuration is shown in table 5.9. Ceph cluster has 3 monitors, 72 OSDs and is capable of supporting 270TB of data. The three monitors with a network speed of 10G and 125GB RAM is configured as listed

Table 5.8: Available Hardware

Node	OSD Count	size(TB)	Total(TB)	SSD Count	size(GB)	Total(TB)
ceph1	18	3.7TB	66.6	6	372	2.2
ceph2	30	3.7TB	111	6	372	2.2
ceph3	30	3.7TB	111	6	372	2.2
ceph5	30	3.7TB	111	6	372	2.2

Table 5.9: Ceph Cluster Configuration

Admin	1
Monitors	3
OSD nodes	72
MDS	1

in table 5.10. The admin node has similar configurations with that of the monitor. There are 72 Ceph OSD nodes, for storing data. There are four machines which has 18 OSD each and a SSD journal for each HDD as listed in table 5.12

Ceph Status

The status of the Ceph cluster gives the health of the cluster.The status command output is shown in table 5.13. The status lists the cluster id, status code - usually HEALTH_OK refers to all good, HEALTH_WARN for a warning and other status codes as defined in Ceph Website [42], number of monitors with their configuration, the number of OSD nodes, the placement group map with the number of pools, data stored and free space available.

Table 5.10: Ceph-Mon Configuration

nodes	ceph2,ceph3,ceph5
Network Speed	10G
ethernet	p2p5(10G), p5p2(10G)
RAM	125GB
CPU	Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz,6 cores
OS	Ubuntu 14.04 trusty 64 bit

Table 5.11: Ceph-Admin Configuration

nodes	ceph1
Network Speed	10G
ethernet	p2p5(10G), p5p2(10G)
RAM	125GB
CPU	Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz,6 cores
OS	Ubuntu 14.04 trusty 64 bit

Table 5.12: Ceph-OSD Configuration

nodes	ceph1	ceph2	ceph3	ceph5
OSD (TB)	18*3.7	18*3.7	18*3.7	18*3.7
Journal Size(GB)	18*20	18*20	18*20	18*20

Table 5.13: Ceph Status

cluster	447d8cc0-2e89-4e5e-9923-40a50dd3e3c7
health	HEALTH_OK
monmap	e1: 3 mons at ceph2=10.8.3.12:6789/0, ceph3=10.8.3.13:6789/0, ceph5=10.8.3.15:6789/0 election epoch 6, quorum 0,1,2 ceph2,ceph3,ceph5
osdmap	e626: 72 osds: 72 up, 72 in flags sortbitwise
pgmap	v2361: 2048 pgs, 1 pools, 0 bytes data, 0 objects 4334 MB used, 261 TB / 261 TB avail 2048 active+clean

Chapter 6

DISTRIBUTED MEMORY CENTRIC STORAGE SYSTEM

6.1 Alluxio

Alluxio [5] is a memory-centric, fault-tolerant, distributed storage system, which enables reliable data sharing at memory-speed across a data center. Alluxio achieves high read and write throughput without compromising the fault tolerance by reducing data replication. Alluxio unifies data access and bridges computation frameworks and underlying storage systems.

Applications only need to connect with Alluxio to access data stored in any underlying storage systems. In big data ecosystem, Alluxio lies between computation frameworks or jobs, such as Apache Spark, Apache MapReduce, or Apache Flink, and various kinds of storage systems, such as Amazon S3, OpenStack Swift, GlusterFS, HDFS or Ceph. Alluxio brings significant performance improvement to the stack; for example, Baidu [28] uses Alluxio to improve their data analytics performance by 30 times. Beyond performance, Alluxio bridges new workloads with data stored in traditional storage systems. Users can run Alluxio using its standalone cluster mode, for example on Amazon EC2, or launch Alluxio with Apache Mesos or Apache Yarn.

Alluxio is Hadoop compatible. This means that existing Spark and MapReduce programs can run on top of Alluxio without any code changes. Alluxio uses the concept of lineage, where the lost output is recovered by re-executing the tasks that created the output. Initially, Alluxio stored all data only on memory. But in many cases, memory is not sufficient to fit in all those data and memory capacity is still comparatively expensive to its alternates. Hence now besides utilizing the DRAM,

Alluxio also uses other secondary storage devices. Hierarchical storage management distributes the data across the storage tiers available starting from the fastest to the slowest speed. The latest data will be put into the top tier(the fastest), when top tier's storage space is full, it will swap the old data to the succession tier. When the data on succession tier is accessed again, it can be chosen to promote the data back to the top tier. Therefore, Alluxio, as a storage engine between computing framework and distributed file system, enabling memory-speed data sharing and larger data capacity by using tiered storage strategy.

6.2 Alluxio Architecture

Alluxio architecture [43] has a single master and multiple workers. At a very high level, Alluxio can be divided into three components, the master, workers, and clients as shown in Figure 6.1. The master and workers together make up the Alluxio servers, which are the components a system admin would maintain and manage. The clients are generally the applications, such as Spark or MapReduce jobs, or Alluxio command-line users. Users of Alluxio will usually only need to interact with the client portion of Alluxio.

6.2.1 Alluxio Components

Alluxio Master

Alluxio may be deployed in one of two master modes, single master or fault tolerant mode. The master is primarily responsible for managing the global metadata of the system, for example, the file system tree. Clients may interact with the master to read or modify the metadata. In addition, all workers periodically heartbeat to the master to maintain their participation in the cluster. The master does not initiate communication with other components; it only interacts with other components by

responding to requests. In addition to managing meta data, master also has a work flow manager [5] to track lineage information, create check points asynchronously and interact with the cluster resource manager to allocate resources for recomputation.

Alluxio Worker

Alluxio workers are responsible for managing local resources allocated to Alluxio. These resources could be local memory, SSD, or hard disk and are user configurable. Alluxio workers store data as blocks and serve requests from clients to read or write data by reading or creating new blocks. However, the worker is only responsible for the data in these blocks; the actual mapping from file to blocks is only stored in the master.

Alluxio Client

The Alluxio client provides users a gateway to interact with the Alluxio servers. It exposes a file system API. It initiates communication with master to carry out metadata operations and with workers to read and write data that exist in Alluxio. Data that exists in the under storage but is not available in Alluxio is accessed directly through an under storage client.

6.3 Storage in Alluxio

Alluxio has two layers : lineage and persistence. Lineage saves the information of the tasks on the disks, which can further be used to recompute the lost data during any failures, there by providing the fault tolerance. Persistence saves the data without the lineage data to disks as defined by the storage types below.

Alluxio uses two different storage types: Alluxio managed storage and under storage.

- Alluxio managed storage is the memory, SSD, and/or HDD allocated to Alluxio workers.
- Under storage is the storage resource managed by the underlying storage system, such as S3, Ceph, Swift or HDFS.

6.3.1 Tiered Storage in Alluxio

Alluxio supports tiered storage, which allows Alluxio to manage other storage types in addition to memory. Currently, Alluxio Tiered Storage supports these storage types or tiers:

- MEM (Memory)
- SSD (Solid State Drives)
- HDD (Hard Disk Drives)

Using Alluxio with tiered storage allows Alluxio to store more data in the system at once, since memory capacity may be limited in some deployments. With tiered storage, Alluxio automatically manages blocks between all the configured tiers, so users and administrators do not have to manually manage the locations of the data. To manage the placement and movement of the blocks, Alluxio uses allocators and evictors to place and re-arrange blocks between the tiers. Alluxio assumes that tiers are ordered from top to bottom based on I/O performance. A tier is made up of at least one storage directory. This directory is a file path where the Alluxio blocks should be stored. Alluxio supports configuring multiple directories for a single tier, allowing multiple mount points or storage devices for a particular tier.

- Writing Data: When a user writes a new block, it is written to the top tier by default (a custom allocator can be used if the default behavior is not desired).

If there is not enough space for the block in the top tier, then the evictor is triggered in order to free space for the new block.

- Reading Data : Reading a data block with tiered storage is similar to standard Alluxio. Alluxio will simply read the block from where it is already stored. If Alluxio is configured with multiple tiers, then the block will not be necessarily read from the top tier, since it could have been moved to a lower tier transparently.

Allocators

Alluxio uses allocators for choosing locations for writing new blocks.Here are the existing allocators in Alluxio.

- GreedyAllocator : Allocates the new block to the first storage directory that has sufficient space.
- MaxFreeAllocator : Allocates the block in the storage directory with most free space.
- RoundRobinAllocator : Allocates the block in the highest tier with space, the storage directory is chosen through round robin.

Evictors

Alluxio uses evictors for deciding which blocks to move to a lower tier, when space needs to be freed. Existing ones include -

- GreedyEvictor : Evicts arbitrary blocks until the required size is freed.
- LRUEvictor : Evicts the least-recently-used blocks until the required size is freed.

- LRFUEvictor : Evicts blocks based on least-recently-used and least-frequently-used with a configurable weight. If the weight is completely biased toward least-recently-used, the behavior will be the same as the LRUEvictor.

Asynchronous Eviction

Space reserver makes tiered storage try to reserve certain portion of space on each storage layer before all space on any given layer is consumed. It will improve the performance of bursty write, and may also provide marginal performance gain for continuous writes when eviction is continually running.

6.3.2 Lineage

Alluxio can achieve high throughput writes and reads, without compromising fault-tolerance by using Lineage, where lost output is recovered by re-executing the jobs that created the output. With lineage, applications write output into memory,

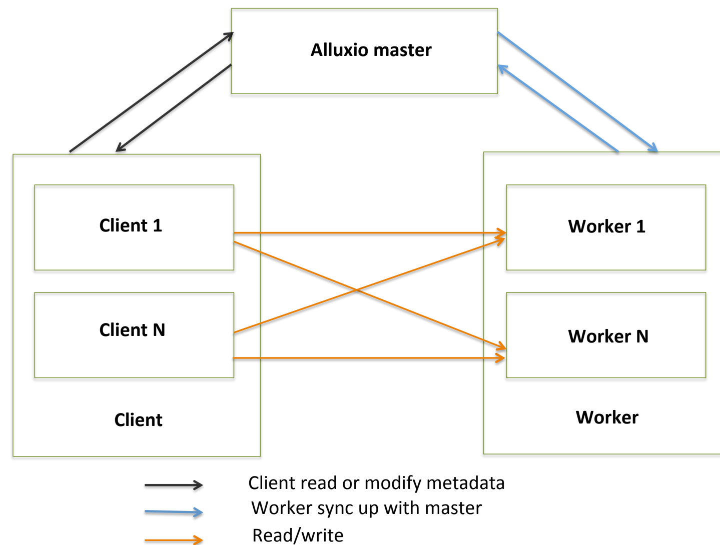


Figure 6.1: Alluxio Architecture

and Alluxio periodically checkpoints the output into the under file system in an asynchronous fashion. In case of failures, Alluxio launches job recomputation to restore the lost files. Lineage assumes that jobs are deterministic so that the recomputed outputs are identical. If this assumption is not met, it is up to the application to handle divergent outputs.

6.3.3 User Location Policy

Alluxio provides location policy to choose which workers to store the blocks of a file. A few built in policies include

- LocalFirstPolicy : Returns the local host first, and if the local worker doesn't have enough capacity of a block, it randomly picks a worker from the active workers list.
- MostAvailableFirstPolicy : Returns the worker with the most available bytes.
- RoundRobinPolicy : Chooses the worker for the next block in a round-robin manner and skips workers that do not have enough capacity.
- SpecificHostPolicy : Returns a worker with the specified host name.

6.4 Spark

Apache Spark [22] is an open source technology used in big data processing.The framework was developed in 2009 in UCB AMPLab and open sourced as Apache project.

6.4.1 Hadoop and Spark

Apache Hadoop is an open-source software framework used for distributed storage and processing of very large data sets in a cluster framework. This framework has

two layers : storage layer - HDFS (Hadoop Distributed File System) and the data processing layer - MapReduce. MapReduce consists of two phases - map and reduce. Hadoop first splits the files into large blocks and distributes them across the cluster. Once the data is distributed, the nodes process the data in parallel. After the computation, data of key value pair is fed to the reduce system to put together all the results. The output data of each job, between each step has to be stored in the hadoop distributed file system before the next step can begin. This approach is slow due to replication on disk storage. For complex computations, it required sequence of the jobs. In the sequence the later job input required output from previous and introduced latency with these dependencies. In addition the maintenance of a cluster with all the tool set for hadoop made it even harder.

However, Spark removes this problem by storing data in-memory and sharing data across the nodes in the cluster. This lets programmers develop complex, multi-step pipelines using using directed acyclic graph (DAG) pattern and share data in memory between various jobs. Spark runs on top of existing Hadoop Distributed File System (HDFS) infrastructure to provide enhanced and additional functionality. Spark is as an alternative to Hadoop MapReduce rather than a replacement to Hadoop.

6.4.2 Spark Features

The features of Spark [44] are listed as below:

- With in-memory data storage, Spark is several times faster reducing the expensive shuffles during data processing. Spark supports lazy evaluation of big data queries, which helps with optimization of the steps in data processing workflows.
- Spark holds intermediate results in memory rather than writing them to disk, this reduces loading of the same data sets multiple times.

- Spark stores as much as data in memory and then will spill to disk. It stores part of a data set in memory and the remaining data on the disk. Hence it can be used for processing datasets larger than the aggregate memory in a cluster.
- Spark provides concise and consistent APIs in Scala, Java and Python. Also offers interactive shell for Scala and Python.
- Spark is written in Scala Programming Language and runs on Java Virtual Machine (JVM) environment. It currently supports the following languages for developing applications using Spark -Scala, Java, Python, and R.

6.4.3 Spark Ecosystem

Spark comes with various other libraries [45] other than the core library that's used in data mining, machine learning and big data analytics. Figure 6.3 displays the various libraries available in the spark ecosystem.

- Spark Streaming: Spark Streaming can be used for processing the real-time streaming data. This is based on micro batch style of computing and processing. It uses the DStream which is basically a series of RDDs, to process the real-time data.
- Spark SQL: Spark SQL provides the capability to expose the Spark datasets over JDBC API and allow running the SQL like queries on Spark data using traditional BI and visualization tools. Spark SQL allows the users to ETL their data from different formats its currently in (like JSON, Parquet, a Database), transform it, and expose it for ad-hoc querying.
- Spark MLlib: MLlib is Sparks scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression,

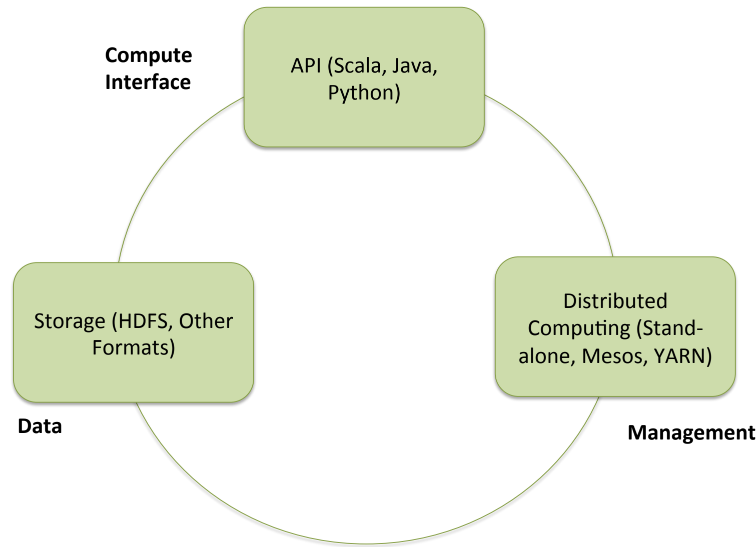


Figure 6.2: Spark Architecture

clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.

- Spark GraphX: GraphX is the new (alpha) Spark API for graphs and graph-parallel computation.

6.4.4 Spark Architecture

Spark Architecture includes following three main components and indicated in Figure 6.2.

- Management Framework Spark can be deployed as a Stand-alone server or it can be on a distributed computing framework like Mesos or YARN.

- Data Storage: Spark uses HDFS file system for data storage purposes. It works

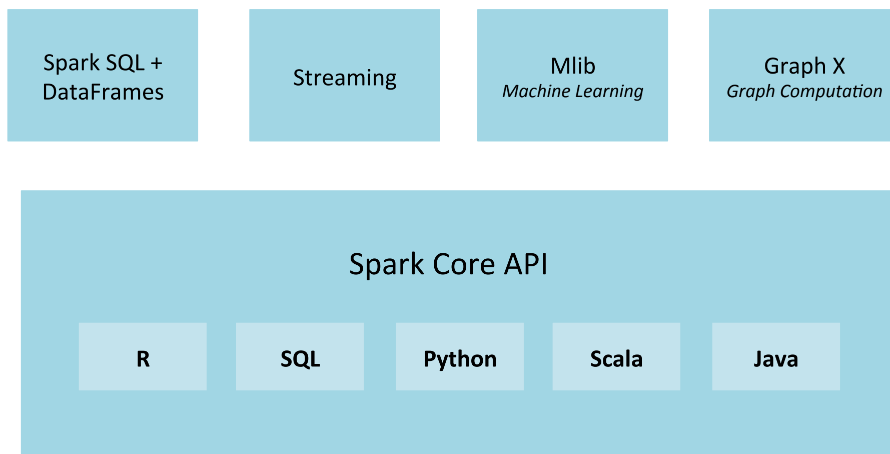


Figure 6.3: Spark Ecosystem

with any Hadoop compatible data source including HDFS, HBase and Cassandra.

- API: The API provides the application developers to create Spark based applications using a standard API interface.

6.4.5 Resilient Distributed Datasets

Resilient Distributed Dataset or RDD [4] is a new abstraction that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators. They help with rearranging the computations and optimizing the data processing. They are also fault tolerance because an RDD knows how to recreate and recompute the datasets.RDDs are immutable. RDD supports two types of operations - Transformation and Action. Transformations don't return a single value, they return a new RDD. Nothing gets evaluated when you call a Transformation function, it just takes an RDD and return a new RDD. Some of the Transformation functions are

map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, and coalesce. Action operation evaluates and returns a new value. When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned. Some of the Action operations are reduce, collect, count, first, take, countByKey, and foreach.

6.5 Alluxio and Spark

Spark already stores the data in memory and the rdds persisted in memory are shared between jobs. However there are a few problems that can be solved with Alluxio.

6.5.1 Problems with Spark

- Spark can only share rdds with the other spark jobs. Different frameworks may require the same dataset. Usually these data sets are stored on disks and every time loaded into these frameworks causing delays.
- Spark execution engine and the storage engine reside in memory. When the process crashes the entire data in memory is lost and needs to be recomputed from disk. This requires reload from disk.
- If multiple Spark tasks are run, each process requires separate memory manager, there by duplicating the memory and the garbage collection in each jvm.

6.5.2 Benefits with Alluxio

- Different frameworks can share the same data at memory speed. The file persists in Alluxio memory rather than on the disk or HDFS, there by enabling data sharing at memory speed.

- During the crash of the spark manager, only the data in the spark memory is lost. Spark does not have to recompute as the data still resides in the Alluxio memory.
- Even when multiple Spark process is run, alluxio is the memory manager, there by having only one heap and no garbage collection.

6.5.3 Applications with Alluxio

A cloud computing platform set up for autonomous driving [23] connects Spark and Robot Operating System(ROS). Alluxio is used as in-memory storage with Spark and OpenCL in their infrastructure to speed up analytics. Similar techniques were proposed in [24] to use Alluxio as cache in robotic clouds. ArcGIS [25] is now using Alluxio to accelerate the mapping and Spatial Analytics. YinMem [26] uses Alluxio as the in-memory storage system for iterative calculations and storing intermediate results for large scale data analytics. The case study [28] shows that Alluxio is used by Baidu to perform do ad-hoc queries and it provides 30x better performance over the traditional model of querying data from direct storage. Other enterprises like Barclay and Qunar are adopting Alluxio with under storage to improve the performance.

Chapter 7

BIG DATA ANALYTICS USING SPARK

7.1 Comparitive Study of Ceph and Alluxio Storage Systems

Below is the comparison of two storage systems obtainined by studying the architecture of Ceph [1] and Alluxio [5]. The properties, key features, their similarities and the differences and interaction with the application is studied and used for comparison in Table 7.1.

Table 7.1: Comparison of Ceph and Alluxio Storage Systems

Property	Ceph	Alluxio
Storage	Disk Centric Storage System	In Memory Storage System
Definition	Ceph is unified and distributed object storage system designed for high performance, scalability and reliability.	Alluxio is distributed file system which enables data sharing at memory speed across cluster computing frameworks.
Architecture	Storage layer	Memory layer sandwiched between applications and the storage
Fault Tolerance	Replication	Deterministic jobs use lineage and non-deterministic use replication.
Components	Ceph OSDs, Monitor, Meta data server and Gateway	Master, Workers and Proxy(Web server)
Storage	Allows file,block and data storage	Big data applications can store data on tiered Storage (RAM,SSD and HDD) or use under storage on HDFS or Ceph.

Data knowledge	Data is stored as file, blocks or objects	Agnostic to the semantics of the data stored
Security	Cephx authentication	Authentication and Authorization - Thrift RPC.Every file has a meta-data of the user using the system
Best fit	Offline storage excellent for backup and restore	Applications which run deterministic jobs, data is immutable, frameworks that care of data locality can benefit the data sharing in memory speed.
Application Interaction	Spark application directly uses the under storage Ceph.	Alluxio is used as the memory cache between Ceph under storage and the application Spark on compute node.

7.2 Benefits of Combining the Systems

Alluxio is configured to interact with the under storage using Amazon S3 API [46]. Alluxio can save the data in memory and persist the same data or load the data from the under storage system. As shown in Figure 7.1, applications talk to Alluxio, if the data is not loaded in memory, it can fetch the data for the first time and continue to save it in memory. Once loaded the data can be shared across any application.

- Alluxio acts as a memory cache to Ceph.
- Alluxio‘s in-memory storage residing between Ceph storage and the applications such as Spark, enables tremendous increase in the data access speed.
- Alluxio uses lineage, so the replication is reduced. But with the non-deterministic

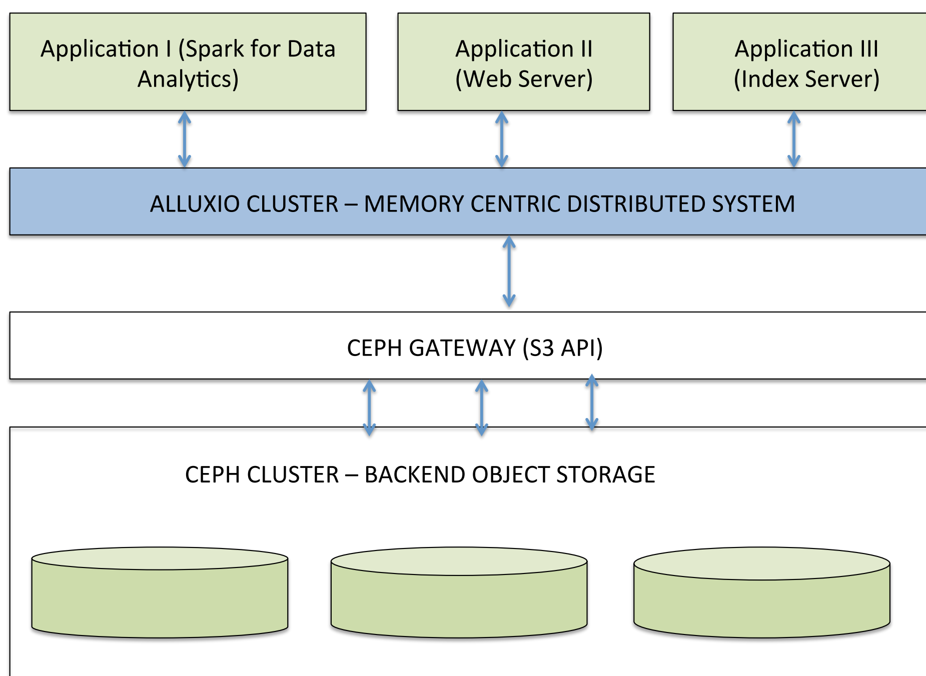


Figure 7.1: Hybrid Architecture

jobs, the data can be stored in Ceph and replication is managed by Ceph without having to use any other third party layer for replication.

7.3 Big Data Analytics using Spark

Alluxio is configured to interact with the under storage system using S3 API. Alluxio can save the data in memory and persist the same data to the under storage or load the data from the under storage system. If the data is not loaded in memory, Alluxio can fetch the data for the first time and continue to save it in memory. Once loaded the data can be shared across any applications.

Below are the list of experiments conducted and the results were used to derive at specific conclusion.

- Evaluate metrics for text analysis with direct Ceph and Alluxio with Ceph.

- Build TechTalk application to perform content indexing and obtain metrics with the use of Alluxio. The application will perform both online data analysis and offline data analysis.

- Hadoop is the widely chosen system for batch jobs or any data mining. Before we start developing the application, we compare the performance evaluation of Alluxio with Ceph and the HDFS as under storage. If HDFS provides a better performance with files than Ceph, then HDFS can be used only for data analysis in the application.

7.3.1 Configuration

Storage Cluster - Ceph

The storage cluster is the backend infrastructure of the cloud ThoTh Lab, an online learning system with the following components.

- **Ceph Cluster**: The Storage manager consists of one admin node and 3 monitors forming the cluster,72 OSD with each 3.7 TB.

- **Gateway** : The Ceph gateway is setup to allow the clients to interact with the cluster using S3 Apis.

Compute Cluster - Alluxio

- **Master**: Runs the Alluxio master and Spark services.

- **Worker** : There are 3 workers including the master configured as a worker with 30G of RAM each.

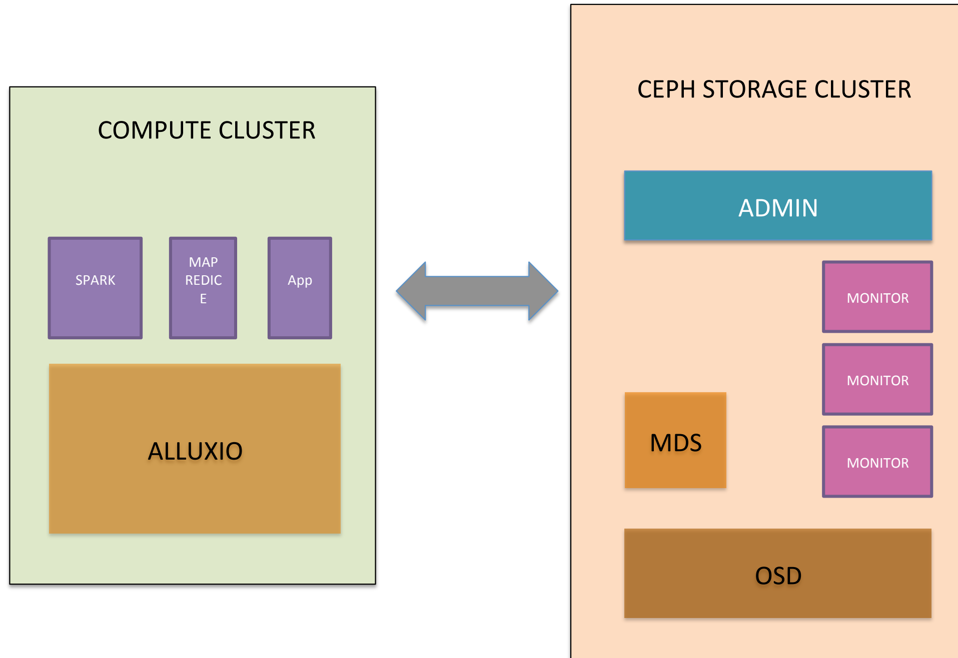


Figure 7.2: Big Data Analytics with Alluxio on Compute Node

7.3.2 Spark Line Count Experiment

An experiment is conducted to determine the line counts of a file in Spark. The files are stored on the under storage system. Spark scripts accessed the files on the storage system to determine the line count. The algorithm is presented in algorithm 1. Spark loads the dataset in the under storage to the memory and provides it to the map reduce framework to count the lines in the input files. We record the time taken to perform line count. We repeat the same experiments for 10 times and record the average time for repeated line count. Then the Spark application is restarted and then line counts are calculated again. In the next set, Alluxio is configured to use one of these stores and the first time when the command is run, Alluxio loads the

file on to the memory and subsequent line counts are performed on the file that is in the memory. The Spark application is restarted and then line counts are calculated again.

Algorithm 1: Spark line count

```
  Input  : textfile,storage
  Output: linecount,ttfirst,ttrepeat
1 start Spark
2 ttfirst = Time taken to count the lines in the file in the under storage for the first time
3 for k = 1to10 do
4 |   ttrepeat += Time taken to count the lines in the file in the under storage for the first time
5 end
6 ttrepeat = ttrepeat/10
7 Restart Spark
8 ttfirst = Time taken to count the lines in the file in the under storage for the first time
```

The experiment is run four times run on HDFS, Alluxio with HDFS as under storage, Ceph and Alluxio with Ceph as under storage. The time taken to perform the operation on each is recorded and a graph is plotted with these values for file sizes - 17KB,1G,10G and 25G files. The experiment is run with two programming languages - Scala and Python with Spark. Scala, also the Scalable Language is a general purpose programming language which supports functional and object oriented programming. The source code is intended to be compiled to Java byte code and the code runs on JVM(Java virtual Machine). PySpark is the Python API for Spark. Since the language is python, the source code is run in the interpreter.

7.3.3 Results

The time taken to count the lines for different file size is tabulated. For a 10G file, with direct Ceph it takes 730s,70s and 730s for initial count, repeated count in same session and after restart in a different session. For the same file with Alluxio on Ceph, it takes 428s,30s and 30s for initial count, repeated count in same session and after

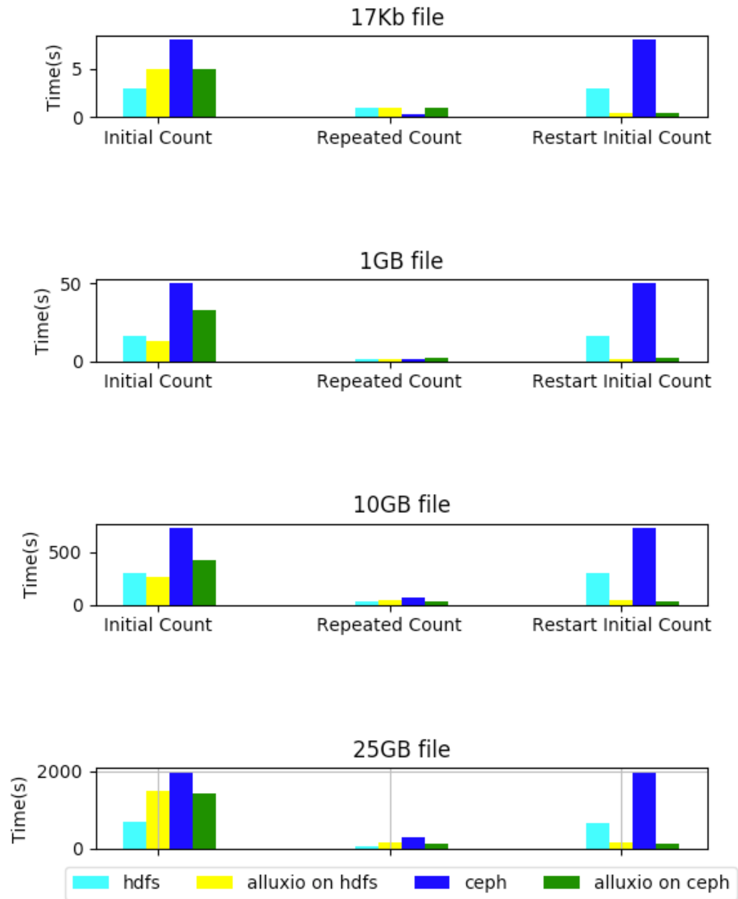


Figure 7.3: Execution Time for Line Count in Scala

restart in a different session. Repeated count in Spark increases the speed by 100x, but once restarted or with a new session for the same data set the time increases to 730s. When Alluxio is used, the time taken for initial count reduces, but even after restart it takes only 30s. Hence, Alluxio provides 24x better performance when tried with the direct storage in a new session and 14x better performance when compared to the initial count on Alluxio. The experiment is repeated for different file size and an average of ≈10x better performance is obtained.

7.3.4 Inference

The first time line count takes almost same time whether done from disk file or using Alluxio on both HDFS or Ceph. The loading takes time as Alluxio has to load the file from the disk to the memory. But repeated counts to the same data takes very less time as the data is in memory. Partial results can be saved in the memory to improve the performance even better. Scala scripts on repeated counts with Alluxio provide $\approx$10x times faster as the data is already in the memory as indicated in Figure 7.3. But the same repeated counts with PySpark is only $\approx$2x faster as shown in Figure 7.4.

On the other hand both HDFS and Ceph provide almost same performance with Alluxio and Spark. Had HDFS given better results, we could deploy another node with Alluxio to perform any file analytics for batch jobs. Since that's not the case, Alluxio with Ceph is good enough for processing.

Based on the inferences above, we design the application properties as described below -

- Though Spark has caching and in-memory rdds, two different jobs or applications cannot share data like Alluxio. All offline data can be stored on the Ceph. Alluxio will act as a memory cache to Ceph and can load the data from the under storage for the offline data and retain the frequently used data in memory.

- The results are different with PySpark and Scala. PySpark is run with python interpreter and Scala runs with JVM. Hence Scala is faster than PySpark. So better to use Scala to compute the results and save in Alluxio.

- Spark and Alluxio seem to provide more or less performance results with text analysis on HDFS and Ceph. Since already Ceph provides all types of storage

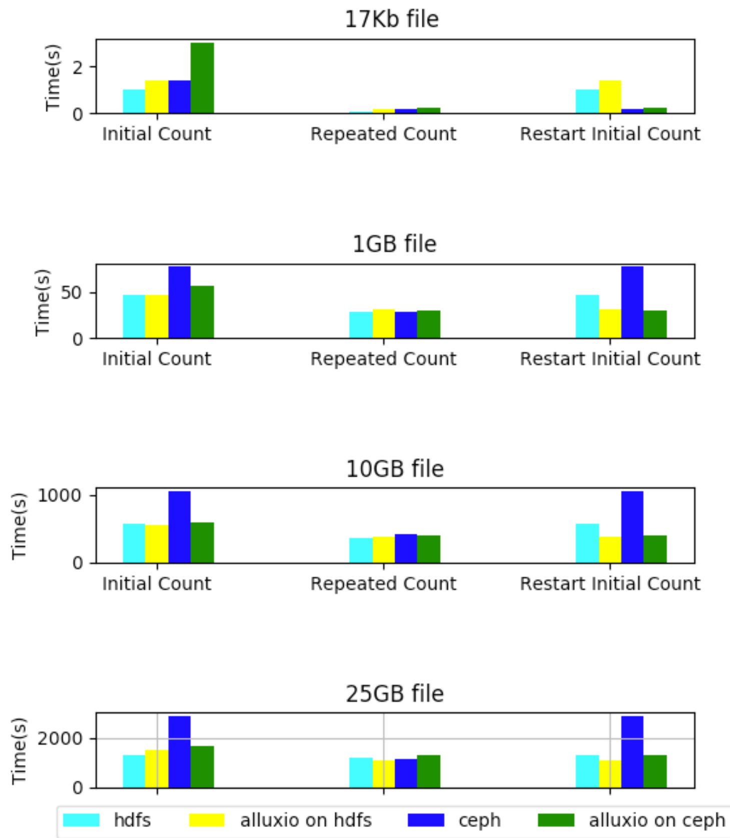


Figure 7.4: Execution Time for Line Count in PySpark

file/block and object. We can use Ceph as it is configured now, rather than configuring another node with HDFS for batch processing.

The conclusion from these experiments is that Alluxio when combined with Ceph provides about ≈10x times faster with Scala on repeated operations on the files that's already loaded in memory than doing it on the under storage directly. Using Scala over PySpark reduces the time delay in computing the results. Also use of HDFS or Ceph Object storage in the ThoTh Lab does not make any difference w.r.t to the Spark jobs run.

Chapter 8

TECHTALK

8.1 Introduction

TechTalk is an application designed for teachers and students to assist in online learning using knowledge base of video lectures. The functionalities of the application include offline storage, live recording, content analysis and reference generation. TechTalk enables teachers to record live class lectures and upload lectures from youttube or any other data source to the offline store. Using machine learning techniques the knowledge base is constructed by analyzing the offline data, this training dataset provides knowledge to search, view, construct the index of an online stream and provide references to related lectures.

There are two objectives for designing this application. First, it is constructed to showcase the importance of a hybrid system utilizing the best features of disk centric and memory centric distributed storage systems. Second, most of the available online learning system provides access to contents but users have a hard time in searching the lectures and their references to the related ones. We intend to build a system which assist users in online learning based on the user behavior and analysis. TechTalk application is a tool that can be integrated with such a system to assist learners with a video knowledge base. The infrastructure of ThoTh Lab, a cloud based hands on virtual laboratory for Computer Science education is built with Ceph as the under storage that can store data in the form of files,blocks and objects. TechTalk uses Alluxio, the memory centric distributed system with Ceph as the under storage for online and offline data analytics. The application consumes and processes video

lectures and notes uploaded from any data source to construct an index with all metadata associated to the lectures. When a lecture is recorded live, the contents of the video stream is analyzed and references related to current lecture topics are searched in the offline store. Depending on the user search query and other criteria like frequently accessed, most visited the data is filtered and displayed in the web console for the students to consume data. The offline data is stored in the Ceph storage, data analytics and index construction is done in Alluxio to serve all the services and browsers.

The TechTalk application includes a client for live streaming, a web server to view, search and manage content store, an index server to generate references,and a streaming server to play the contents in the store. The servers can be hosted on any of the clouds to index the data in the offline store and create an on demand data lookup for the live stream. Index server analyzes the video contents by converting audio to text and applies keyword extraction algorithm to extract the keywords in the lecture along with its time frame. The indexed metadata enables the students to search, view and access the relevant contents available in the store.

The back end storage Ceph is capable of storing all types of data and supporting applications like OpenStack which can be used for creating labs. Alluxio enables data sharing between applications at memory speed. Data analytics and index processing is faster when data is already available in memory rather than on the disk. TechTalk demonstrates the data sharing between index server, web server and the Spark scripts which is used for data analytics. We conduct experiments and demonstrate use cases to determine how Alluxio and Ceph together benefits such big data applications.

8.2 Architecture

The architecture of the TechTalk application is illustrated in Figure 8.1. The application consists of a client, web server, streaming server, index server and the web portal.

8.2.1 Functionalities

There are two views to the application - admin view and the end user view.

- Admin Features : The teachers form the admin group. They can create, build and manage the content in the store. Below list describes the functionalities available to admins.

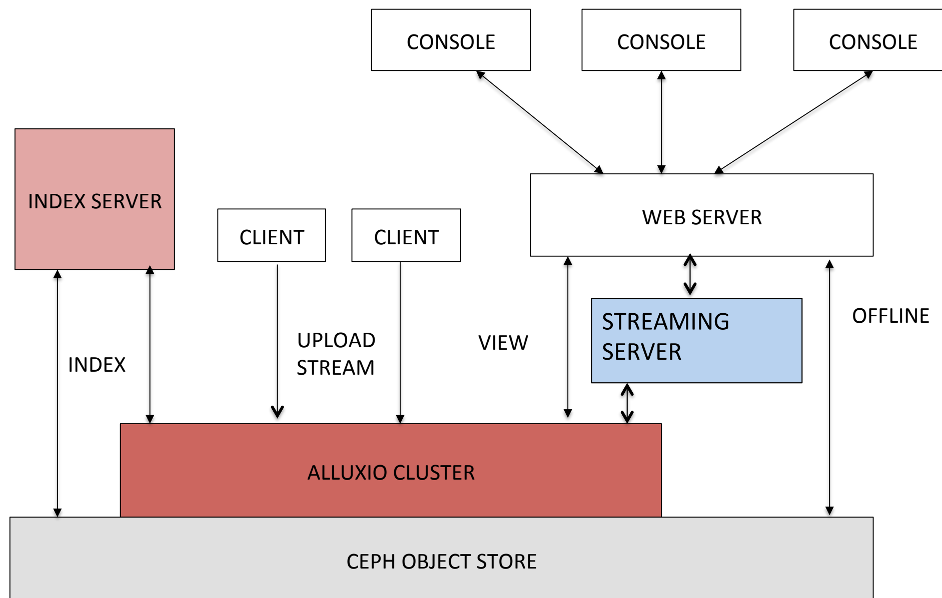


Figure 8.1: TechTalk Architecture

  - **Build Offline Store Content** : Admins can upload videos using youtube url or any other data source to build contents in the offline store.
  - **Record Lectures**: Admins can use the client software to record lectures during the class. The lecture is analyzed and uploaded to the offline store. All notes related to the class can also be uploaded to the offline store.
  - **Manage Content** : TechTalk provides a portal for admin to interact with Ceph to upload, download and delete contents in the store. Admins can also manage contents in Alluxio and monitor the statistics of the system.
  - **Index Management** : The web portal provides a view of the index that's constructed in the Alluxio. Also admins can update or regenerate the index on demand if needed from the portal.

- End User Features: The students are the end users who consume the refined data in the store.
  - **View** : Students can view all the lectures available in the offline store. They can access and stream the videos. Every lecture has associated notes and transcribed data from the audio.
  - **Search**: Students can search videos using keywords delivered in the lecture. The portal lists the videos along with the time frames the keywords are mentioned in the lecture.
  - **Live feed** : During the live recording, students can see the live feed of the lecture with the transcribed text and the related videos to the lecture in the same live feed page. They can navigate to the related lecture to find the notes and other relevant information.

8.2.2 Components

- Console :It is the web user interface for any user to interact with the server. It provides two views : admin view and end user view.
- Web Server : Its the entry point to console to interact with the application via REST APIs. It serves all the console requests and aids the admin users to upload content to the offline store. All uploaded data is analyzed and metadata associated to each lecture is updated in the store.
- Index Server : It has two roles index generation and live lecture reference generation. All videos in the offline store is analyzed in steps. The audio is transcribed to text and keywords are extracted using machine learning technique. With new data uploaded, it regenerates and updates the index with the new data at frequent intervals. During live recording, when the client uploads the audio and video content, server combines the audio and video stream, generates references and updates the index in Alluxio processing the incoming stream. When the streaming is complete, it accumulates all the data into one file and saves it to offline store and updates all the meta data in the offline store.
- Client : It is a light weight tool that runs on the laptop to record lectures. It requires access to the microphone and the camera on the laptop and streams the video and audio content to the Alluxio cluster.

8.2.3 Interaction with Ceph and Alluxio

The web server, index server and client communicates the under storage Ceph and Alluxio to upload and retrieve data using REST apis.

- **Communication with direct Ceph** : Ceph RADOS gateway supports REST

Apis [46] thats are compatible with Amazon S3 Api. **Amazon S3 APIs**: Amazon Simple Storage Service Application Programming Interface, widely known as Amazon S3 API are the REST APIs that enables to interact different end points. The S3 REST APIs that are available currently support list of features like uploading and downloading objects,listing and viewing the contents of a bucket,creating and deleting the bucket.

- **Communicating with Alluxio** : The application interacts with Alluxio Proxy configured in the cluster via REST apis. These apis provide access to interact with the data in memory. Alluxio can be configured to interact with the Ceph store using S3 API [47] and there are configurable options like CACHETHROUGH, which saves all data that's in memory also on the disk in the under storage. Once the data is saved in under storage, the data can be accessed either via Alluxio or directly using the Ceph storage.

8.2.4 RAKE Algorithm

RAKE [32] : Rapid Automatic Keyword Extraction is a widely known technique for keyword extraction in Natural Language processing. This algorithm is used to extract keywords from documents, where keyword extraction is a process of extracting the words that provides the most relevant information or main topics discussed in a document. We use this algorithm to extract keywords from the subtitles of the lecture. This process involves -

- Candidate Selection: Extract all possible words and phrases that can potentially be keywords. Remove stop words based on the smart word list provided as input and the punctuations in the document.

- Properties Selection: Calculate the properties that determine its a keyword, for

example calculate the frequency, position in the document or similarity to word words, part of speech pattern, popularity in the language.

- Scoring and Selecting words: Supervised machine learning that learns the importance of properties from the trained model.

Algorithm 2: RAKE

Input : Sentence, Stopwordlist

Output: Keywords

1. sentencelist = Split if the input has more than a Sentence into a list
2. stopwordpattern = Recognise and build stop word pattern from Stopwordlist
3. keywords = Generate candidate key words from sentencelist and stopwordspattern
4. wordscores = calculate word scores from the previously obtained references from the offline store
5. keywordcandidates = generate keywords scores for $< keywords, wordscores >$
6. keywords = select the keywords with highest scores

RAKE algorithm takes the input text and a set of stop words list. First, it splits the input text into multiple sentences. All the stop words are analyzed and the pattern is built for all the provided words to create a filter that can remove these words from the input text. It generates the important keywords as candidate words filtering the stop words from the pattern generated in the previous step. It also generates word scores based on the references in the offline store. With the extracted keywords and the word scores, the scores are generated for these keyword candidates. The keywords with highest scores is returned as the output. The steps involved in RAKE algorithm is listed in algorithm 2.

The algorithm extracts keywords from SRT files, which is generated using autosub, an automatic speech recognition system. However, even a short fragment contains a variety of words, [48] and can be related to several topics. Also using an automatic speech recognition (ASR) system introduces errors among them, making it difficult to extract the required keywords. The other factors that affect the algorithm:

- Language : English language with the English dictionary
- Stop words : We need to be able to provide stop words correctly otherwise the important keywords will get filtered. For ex: adding ”time” to the stop list can drop words like ’time complexity’ or ’run time’. Since the application is subject only to technical keywords, we increase the list of keywords so that we generate a smaller index and the search is faster.
- Training set to evaluate the scores : Most frequently occurring technical words are used for training and evaluating the scores.

8.2.5 External Tools

AutoSub

Autosub [49] is a command line utility for automatic speech recognition (asr) and subtitle generation. Given a video or an audio file as an input, the tool performs voice activity detection to find the speech regions and generates transcription for those regions by invoking parallel requests to Google Web speech API. It is built on top of ffmpeg, a complete cross platform solution to record, convert and stream audio and video.

Youtube-dl

Youtube-dl [50] is a command-line program to download videos from YouTube.com. We use this tool to build offline videos in the store. This tool is platform independent and is used by admins to upload to Ceph store using Amazon S3 Apis.

8.3 Web Server

Web server is a python flask server which anchors to the http requests from the consoles. Its a portal to serve all requests from the users. Web server has direct access to Ceph store and the Alluxio cluster. It uses Ceph for all storage and Alluxio for live updates.

8.3.1 Offline Store

Console provides an entry point in the admin view to upload new videos to the store. Logged in admins can provide the youtube url in the portal. Web server downloads video from the data source, processes the stream and uploads the downloaded video and other meta data generated along with the video to the store.The steps to process youtube urls is provided in the algorithm 3. It first downloads the data from youtube using youtube-dl. Then generates subtitle file using autosub [49]. The subtitles file is parsed to extract keywords using RAKE [32] and a json file containing the extracted keywords and the time is saved as index. All the metadata that gets generated in the process is uploaded to the Ceph store along with the video file to the Ceph store. This is the process to build content in the store for variety of topics.

Algorithm 3: Offline Storage

Input : Youtube url

Output: Upload video and meta data to Ceph Storage

1. videofile = Use youtube-dl to download video from youtube repository
2. srtfile = Use autosub to generate srt file containing $< subtitles, fromtime, totime >$ from videofile
3. keywordstext = Parse srtfile and extract keywords using RAKE algorithm 2
4. json = Generate index containing $< keywords, time >$ from keywordtext and srtfile
5. notes = Generate notes for the file using autosub from videofile
6. Upload videofile,metadata$< srtfile, json, notes >$ to Ceph storage using S3 REST api

8.3.2 Live Feed

As the lecture is recorded live, the portal displays the list of live sessions, videos generated from current client stream, transcription of the audio of the lectures and its references to the related lectures in the offline repository.

8.3.3 Cache

Cache is a light weight storage on the web server to cater to the needs of the web consoles connected to store any static files, temp files and other metadata. The default size of the cache is set to 2 GB. It can be configured from the settings section in the application. There is a clean up thread in the server which prunes the old data depending upon the configuration and frequency interval defined.

8.4 Client

The client is a light weight application that can be run on laptops or machines which records the lectures. It requires access to the camera and the microphone of the device. It can record the audio and video during the lecture and stream the data to the Alluxio cluster. The graphical user interface of the tool is in Figure 8.2. It has a start and stop button and allows user to save the lecture with a relevant name to it. On start, the data is streamed to Alluxio and on stop there is a message box as in Figure 8.3 indicating the success.The client application creates a GUID for every session with the host name of the machine and creates a folder with that name on Alluxio. It uploads all the audio and video streams to that folder in Alluxio at configured interval. The client is expected to have an internet connection when the video is recorded. Each session of the client is recognized by the GUID. The audio and video saved in the folder is in the form of .wav and .mp4 files respectively.

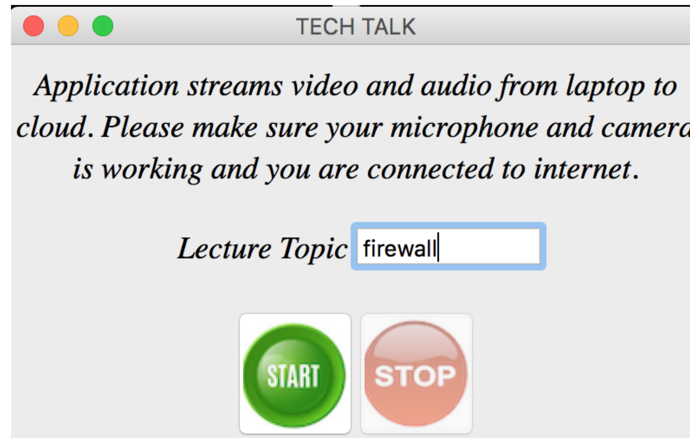


Figure 8.2: TechTalk Client Application on Laptops

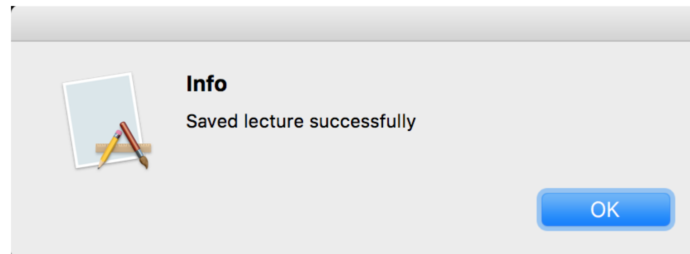


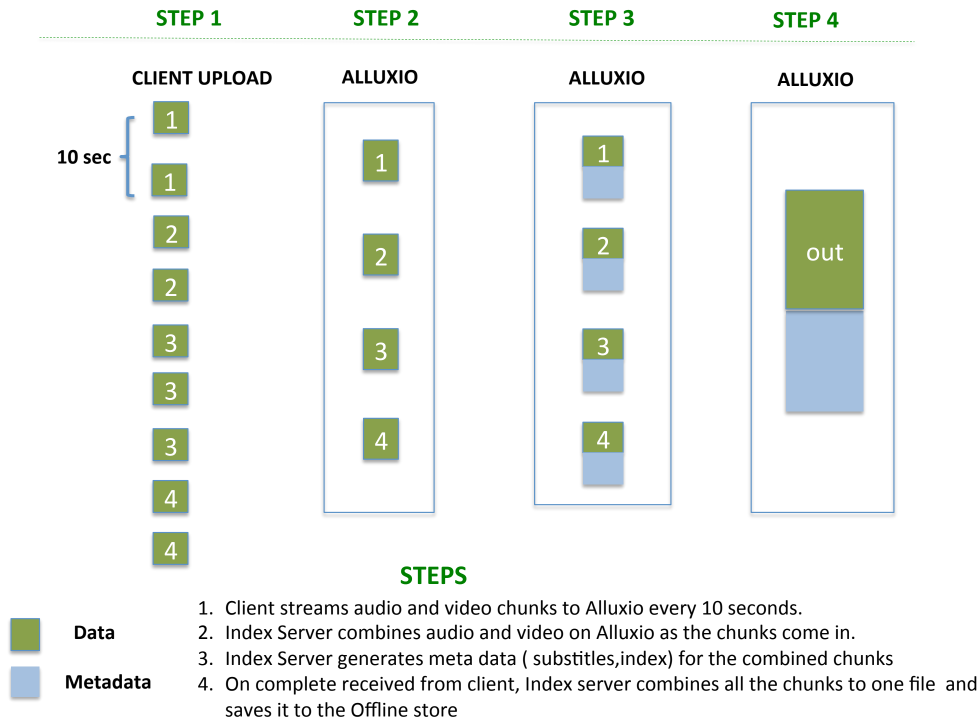
Figure 8.3: Client Confirmation on Success

On the start button click from the user, the client application launches different threads to do the tasks in parallel. The audio thread records the audio using PyAdudio library. The video thread records the video using OpenCv library. The timer thread keeps track of the time, and at frequent intervals, here set to every 10 seconds signals the upload thread to stream the buffer saved by the audio and video thread to Alluxio. Once the application signals stop, the audio and video threads stops recording. A complete signal is sent to Alluxio indicating the session is completed. The steps involved in the streaming on the client is listed in algorithm 4. The algorithm is illustrated with the Figure 8.4. The data in Alluxio is then processed by the index server to combine all the partial files into one single lecture and generate associated metadata.

Algorithm 4: Live Recording Client

```
    Input  : Audio and Video Stream
    Output: Upload video and meta data to Alluxio
 1  Thread1 : Record Audio
 2  while stop is not true do
 3      Use pyaudio to record microphone data
 4      Saves the frames in the memory
 5      if timer event is set then
 6          switch to next available buffer
 7      end
 8  end
 9  Thread2 : Record Video
10  while stop is not true do
11      Use OpenCv To record Video Frame
12      Saves the frames in the memory
13      if timer event is set then
14          switch to next available buffer
15      end
16  end
17  Thread3 : Timer
18  while stop is not true do
19      Raise at event at set interval to
20  end
21  Thread4: Uploader
22  while stop is not true do
23      Read the circular buffer in Memory
24      Upload to Alluxio
25      Clear the buffer
26  end
```

Figure 8.4: Live Lecture Streaming



8.5 Index Server

Index server processes several tasks including index generation, processing online lecture stream and updating index at frequent intervals.

8.5.1 Index Generation

Index is the collection of metadata of all lectures in the video knowledge base residing in the Ceph store. The metadata is generated for each video in the form of json files. All json files are combined together into one large index and uploaded to Alluxio. This file is named as main.json. A copy of the main index is check pointed in the Ceph store for fault tolerance. It consists of the video file name, extracted

keywords using RAKE and the time interval of occurrence of the series of text. This index is consulted to retrieve results for the search operation. In addition to that, several PySpark scripts are integrated to the system to analyze the index. Analysing index is important to find the occurrences of technical words and its frequency, the unique words that appear only once, non technical words, nouns that are not relevant and many other statistics. Depending on these statistics, admin can train the stop words list so that the index is tuned to contain only the technical words resulting in a smaller, cleaner and relevant index.

8.5.2 Update Index

At frequent intervals, the main index in Alluxio is updated with the new references generated from the new uploads in the offline store. The entire index is not generated, but the latest one from Alluxio is obtained and updated the new references. Then the index in Alluxio is cleaned up to upload the new one to the memory. A dedicated thread runs at configured time interval usually once a day or more depending on the frequency configured in the settings to update the index. In addition to this, admins can train the new stop words and regenerate index for all lectures in the offline store. The portal provides an option to regenerate the index as required.

8.5.3 Live Lecture Feed

Client uploads video and audio streams in the form of .wav and .mp4 files respectively with a GUID to Alluxio. When the index server detects that there is a live lecture, it combines streams to generate a mp4 file synchronizing audio and video files. Sequential mp4 files are concatenated using ffmpeg to generate a single mp4 file. A srt file (voice to text) with the text and the time frame is generated for each partial file using the tool auto-sub. An index containing the main keywords is generated in

json file using the RAKE algorithm for each partial mp4 file. These partial results are stored in Alluxio for the web server to stream the references to the consoles. When a complete message in the form of a file, indicating the end of stream is received, all the partial or intermediate files are combined to one single video file and then the srt files and index json file is regenerated. The main index file is updated with the new references for this lecture. This lecture is now treated as an offline lecture and can be used for future references.

The algorithm to process live lectures is listed in algorithm: 5. The index server live lecture thread keeps polling for a livelectures folder in Alluxio. If it finds a folder and a GUID under that folder, it invokes one thread per folder to process the content uploaded by the client. The thread launched corresponding to each GUID, looks for the audio and video files in that folder. It sorts all the audio and video files under that GUID folder. It invokes tools to combine the video and audio files till a time frame (indicated by the number in the audio and video file). As the data is uploaded, the partial mp4 files are created (pcout files) and they are combined to form cout files. For every pcout mp4 file, the audio is transcribed to text and index is generated using the RAKE algorithm. The live lecture thread also launches a complete thread to poll the complete sessions. Once a complete is received for a session, the thread starts combining the cout files for each session and generates an out file. The out file is the final file which gets saved to the offline store. This file is marked as a recorded lecture. The out file and its metadata is then uploaded to the offline store using Amazon S3 REST apis. The partial files and the metadata generated on Alluxio is cleaned up after the combining is done.

Algorithm 5: Live lecture Processing

Input : GUID,audio.wav,video.mp4, completeFile

Output: video,metadata

```
1  foreach GUID in livelectures folder in Alluxio do
2  |   Start thread to Process GUID folder
3  |   Start thread to Process Complete files
4  |   MarkItProcessed
5  end
6  Thread - Process GUID Folder
7  while complete not exists do
8  |   while avlist is empty do
9  |   |   sleep
10 |   end
11 |   avlist = Fetch all audio.wav and video.mp4 under the GUID folder
12 |   Sort avlist per the stream order marked by the number from the client
13 |   foreach audio,video in avlist do
14 |   |   pcout = Synchronize each set of audio and video into one
15 |   |   Append the pcout file to partialy combined files list pcout-list
16 |   |   Delete audio,video files so that its not picked in next iteration
17 |   end
18 |   combine all videos in pcout-list to GUID-cout.mp4
19 |   Generate cout metadata - srt file and references
20 |   Start upload thread to upload < cout, references > to Alluxio
21 end
22 Thread - Process Complete File
23 while true do
24 |   if complete exists and no audio,videos files are in queue to process then
25 |   |   cout-list = Fetch all uploaded video files
26 |   |   GUID-out-file = combine all cout-list files to video file using ffmpeg
27 |   |   GUID-srt-file = use autosub to generate srt file
28 |   |   GUID-notes-file = use autosub to generate raw file
29 |   |   GUID-index-file = use RAKE algorithm and parse SRT files to generate index
30 |   |   Start thread to upload < video.metadata > to Alluxio and check point in Ceph
31 |   |   return
32 |   end
33 |   sleep
34 end
```

8.5.4 TechTalk Application Settings

The application allows the admin to configure and tune the parameters to get the performance benefit. Some of the parameters that can be configured are shown in the Figure 8.5.

- Use Web Server Cache : Admin can choose whether to use web server cache or not. If the setting is chosen, the downloaded data is retained in the cache for configured time else its cleaned up as soon as the operation is complete.
- Alluxio or Ceph : For any data retrieval, admin can choose to use Alluxio as a memory cache or directly get the data from offline storage.
- Web Server Cache size : The size of the cache on the server can be reconfigured. Its usually set with a default size of 2 GB.
- Web Server Cleanup Interval: The time configured determines how often to prune the items in the cache. The default is set to 2 hours.
- Index Server Refresh Interval: Index has to be regenerated and updated with

Figure 8.5: TechTalk Application Settings

the new items in the store. But the process of generation and updating the index takes time depending of the size of the index. This has to be done offline and can be scheduled to run every day once or once in two days.

8.5.5 Streaming Server

The streaming server is a Node.js server, an open source JavaScript runtime built on Chromes V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it suitable for I/O operations and streaming. The streaming server listens on port 35001. The web server redirects the queries to the streaming server to play video content.

8.5.6 Offline Data Analytics

Alluxio can share data between applications. PySpark and Scala scripts are setup to perform data analytics on the offline storage. The intermediate results and the final results are then saved on Alluxio to share the data between applications.

- Analytics on the index main.json to find the most occurring, latest updated, most searched, unique keywords and group lectures based on the keywords in the index.

- Analyze user behavior collected data to analyze the patterns.

- Analyze audit and time logs to do performance analytics to switch between the servers for load distribution.

- Analyze audit logs to track the most frequently used lectures.

8.6 Performance Evaluation

The application is hosted on a virtual machine with ubuntu 14.04 and 25GB of memory. All services such as the index server, web server and the streaming server is started on the machine. The performance of the application is measured by triggering parallel requests from web automation frameworks to check the response time of the system in various scenarios. Helium is a tool for testing websites and automating browsers. The software is setup to run the tool for multiple users to search contents and play videos simultaneously. The store currently has about 1000 videos with ≈123GB of videos of varying size with lectures covering 100 topics.

8.6.1 Index Server

Index server creates index from the metadata obtained by processing video lectures. The index file (main.json) resides in the memory cache Alluxio. The index server, web server and Spark scripts use this metadata to perform ad-hoc queries and data analytics.

- **Size of Index** : Index generation uses RAKE algorithm to extract keywords in each lecture using the inputs srt file and the stop word list. The index size is measured with the increase in number of videos. The initial index size is plot in Figure 8.6. The figure shows that 100 videos has 10 MB of index and the index size gradually increases to 200 MB for 1000 videos. The knowledge base of videos contains only lectures and is purely technical. Hence any non technical words can be filtered for search but that‘s not straight forward. Vowels, prepositions and other things like conjunction, interjection can be filtered. But there could be nouns which is used as example in the lectures. The choice of stop words to train the model has to be selectively chosen so that we don‘t drop those words

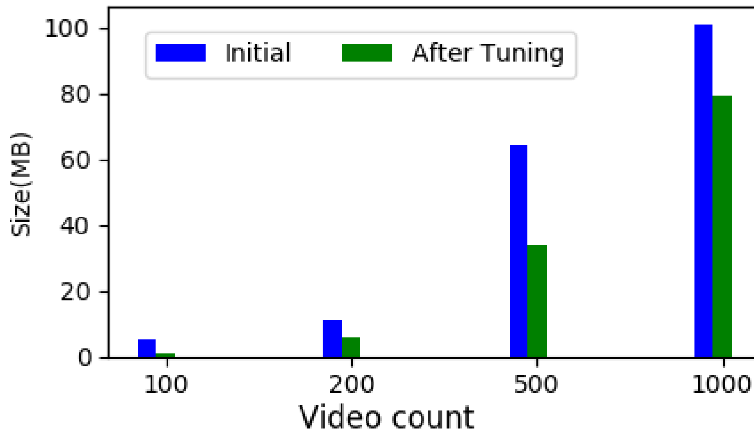


Figure 8.6: Growth of Index w.r.t Video Count

in index resulting in the failure of search. With the increase in the quantity of data, the list of stop words is trained to exclude the keywords that's not needed for e-learning. Then RAKE is tuned to include more stop words after analyzing index generated for ≈120GB of lectures. The index size after tuning is shown in Figure 8.6. We infer that for a store with 1000 videos and around ≈120GB video data there is ≈78MB of index.

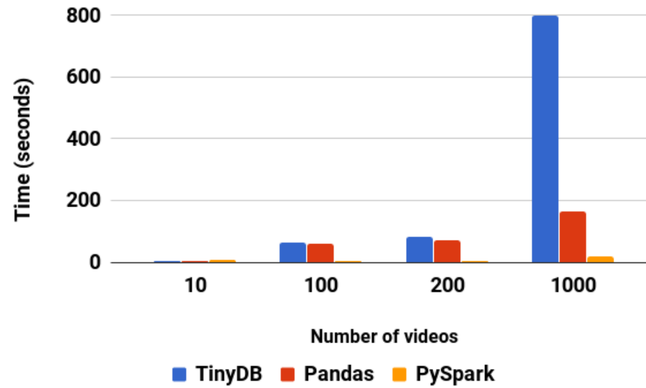
- **Storage** : Now that we generated the index, having index in memory is the fastest way to perform search. Index already resides in the memory on Alluxio cluster, so retrieval of index should be fast. The Alluxio cluster and web server are on the name network, but can be geographically separated, to further increase the search speed, we store a portion of the index in memory of the web server. It does not require any relational database but a db which can index data in a single table with single index and multiple data columns. We experiment with a few simple data structures to store in memory on web server.

  The tiny db in python is very simple, document oriented to store data and has

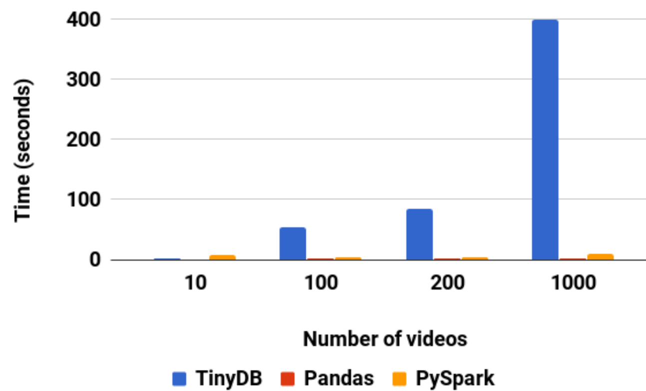
clean APIs to access them. But the load time of the index in memory and search is very slow. We switch to python panda data frames and PySpark. Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Its widely used in data analysis and to build index due to its simplicity and performance.Pandas provides data frames which is usually a set of rows and columns, they are not relational, but a grid to look up data. Its like a spread sheet with an index. This enables the search and retrieval faster than the normal spread sheets. We use panda dataframes to load the index in memory and experiment with PySpark dataframes for the same set of data.

We tabulate the time taken to load index for 10,100,200 and 1000 videos. For 1000 videos, TinyDB takes 800 seconds to load and 400 seconds to search from the index. With the same index, Pandas and Pyspark take 164 and 20 seonds to load the index to memory and 3 and 10 seconds for the search respectively. The load time and search time of each data structure is tabulated in Figure 8.7(a) and Figure 8.7(b) respectively. The evaluation shows that Pandas dataframes is the fastest.

With the growth in the size of the index, storing the entire index would be cumbersome. So the index is split in such a way that we generate 10 million rows every day based on recent search and the class lectures added, the data is first looked up in the web server cache, if there is a miss from that then we use PySpark to look up the index in Alluxio, which results in searching any data in less than a second. The search results for users from web automation framework for 100 parallel requests is an average of ≈0.4 seconds with the time frame of the occurrence of the text.



(a) Time Taken to Load the Index



(b) Time Taken to Search the Index

Figure 8.7: Index Performance with Different Data Structures

8.6.2 Streaming Server

The console is slow and web server is very slow in streaming data. It waits for the complete file to be loaded in memory to stream or play the data. We experiment to set up an asynchronous streaming server in NodeJs to serve all the video requests. The results after moving the streaming to Node.js server just includes the time taken to download the data from Alluxio or from Ceph storage. Then the server starts streaming instantly, while previously the data has to be stored in memory or the

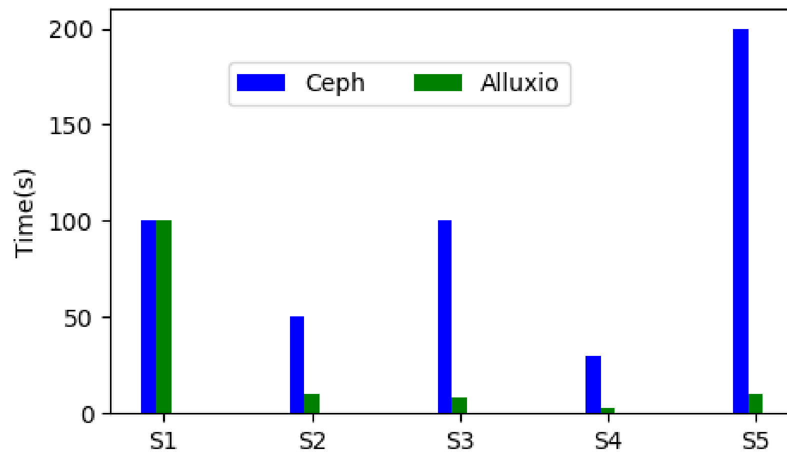


Figure 8.8: Performance Benefit using Alluxio with 1G Index

entire file data has to be streamed to be able to play the file. To stream 1GB video file, it took 5 mins and the server timed out. But with the shift to Node.js server, streaming starts in 100 ms and as the data streams the entire video can be viewed. Within 40s the entire video is available to be navigated.

8.6.3 Data Sharing between Applications

Alluxio benefits when the data is huge and different applications perform analytics on the same file. This can be illustrated with the file analytics on the index files in Alluxio. Index is created by generating more rows, to size 1GB and 10GB. The Scala Spark scripts to extract value from the index is run on the index. The experiment is repeated by placing the file on the Ceph storage and on Alluxio that is in the memory. There are 5 scripts run to perform different activity and as different jobs. Also the results are consumed by the web server and index server to feed the consoles and use the computed data to feed as input to other activities.

With a 1G Index, The first script S1 takes 100s to load the data into memory and compute the repeated words. Subsequent scripts S2,S3,S4,S5 to count the frequency,

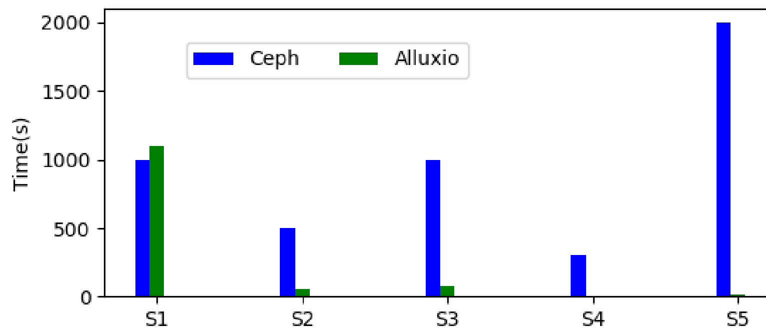


Figure 8.9: Performance Benefit using Alluxio with 10G Index

line count, last accessed, unique count takes one tenth of the time taken to perform the activity when compared to the time taken when loading the data from Ceph storage. The same experiment is repeated with 10G index. The performance of the activities is plot in the Figure 8.8 and Figure 8.9.

The inference is that using Alluxio, the first script takes same time as Ceph as it has to load the index file in memory. If the index file is already in the memory, the next scripts complete about 100 times faster than the ones which loads data from Ceph.

8.6.4 Data Analytics on Logs

All logs - webserver.log, indexserver.log, client.log and common.log are created under the application logs folder. As the name suggests, the client operations to record lectures is logged in client logs - this includes the REST calls to the Alluxio gateway to upload the streams. the index operations are logged in indexserver.log. Any requests sent to server are logged in webserver.log. A spark application is created to monitor these logs and run analytics on these huge text files to determine errors, warning and exceptions. This application can be used to create alert system for admins.

Chapter 9

LIMITING STORAGE IN ALLUXIO

9.1 Alluxio Sessions

Alluxio is a memory-centric, fault-tolerant, distributed storage system, which enables reliable data sharing at memory-speed across a data center. Hierarchical storage management in Alluxio distributes the application data across the storage tiers (RAM, SSD and HDD) starting from the fastest to the slowest. Applying Apache Yarn on applications like Spark or MapReduce allows to limit resources like cpu cores and memory cache size per job. However, Alluxio is unaware of the semantics of the client session. It stores the entire data set in the memory or distributes the data across the tiered storage. In this project, We limit the memory allocation of each Alluxio client session in the tiered storage based on a user defined policy. Limiting memory per session reduces the chances of pooling of DRAM resources to only one session. Hence reducing the latency while running Spark or map reduce jobs.

Applications like Spark cluster running on top of YARN, allows to dynamically share and centrally configure the same pool of cluster resources between all frameworks that run on Yarn. In addition, memory cache size can be limited per job [22]. If the storage exceeds the limit defined then data is stored in the next storage level, so it reserves the memory cache hit rate. However Alluxio, is unaware of the semantics of the application or the user running jobs. It stores the entire data set in the tiered storage in the order specified in the user defined policy.

Alluxio allows many applications to serve at memory speed. But at times, a session running big table application could occupy the entire memory there by reducing the

memory speed to other jobs running on Alluxio. Custom evictions to move data often between tiered storage could increase the latency to serve Spark or map reduce jobs. The significant benefit in limiting the memory storage is that we can control the usage of DRAM per session there by not clogging memory and reducing the latency that happens on every synchronous eviction.

9.2 Design

The main aspects of the design to limit memory per Alluxio client can be categorized as :

- Enable and configure parameters to limit storage per session.
- Alluxio clients (like Spark) propagate the session information to Alluxio worker.
- Location Policy returns the worker with maximum storage available per Alluxio client session.
- Alluxio Session Manager saves the session information and keeps track of the total storage used by the client session.
- Allocator and Evictor limits storage in memory for each session with the Session Manager APIs.

9.2.1 Alluxio Client

We are interested in limiting memory per Alluxio client. The steps involved in propogating session information is represented in Figure 9.1 and is detailed below.

Alluxio provides access to data through a file system interface. Files in Alluxio offer write-once semantics, files become immutable after they have been written in their entirety and cannot be read before being completed. Alluxio provides two

different Filesystem APIs, a native API and a Hadoop compatible API. Alluxio has a wrapper of the native client which provides the Hadoop compatible FileSystem interface. With this client, Hadoop file operations will be translated to FileSystem operations.

Every Alluxio client calls HDFSFileStream APIs when detects that the data has to be sent to Alluxio via the namespace used to read or write the file, and also computes the session id. Alluxio client creates a new session once its invoked. The unique session id is generated by hashing the process information of the client. The machine information and the process id of the client forms a unique hash. If there are multiple clients spawning requests from different machines, the pair of hash code generated makes the hash unique. The advantage of hash code is that the session id can be computed across nodes in a cluster with the knowledge of remote process id and the machine name even if the hash is not available and its irreversible. In other words, hash code consistently generates same information even if computed across the machines in a cluster. Since the two elements used in generating hash is unique to the session, the possibilities of hash collision is very low.

Alluxio client registers with the Alluxio master and passes the session id to the block master via Netty RPC protocol. Master stores the metadata information about the file and returns worker the information about where the client can write the data to. Client then registers with the worker and sends blocks of data to the worker to be stored in the tiered storage.

9.2.2 MyLocationPolicy

Each Alluxio client picks the worker which has max free space and that's the worker returned on every call. For every block requested MyLocationPolicy returns the same worker selected at the beginning of the session. With this policy, only one

worker node is selected per Alluxio client. With this, there is no need to store the metadata on the master which can avoid syncing of total size allocated on all the workers. The advantages are that the read is done with connection to just one worker and the blocks are not spread across various nodes. However the downside is that all blocks per Alluxio client is retained on a single worker.

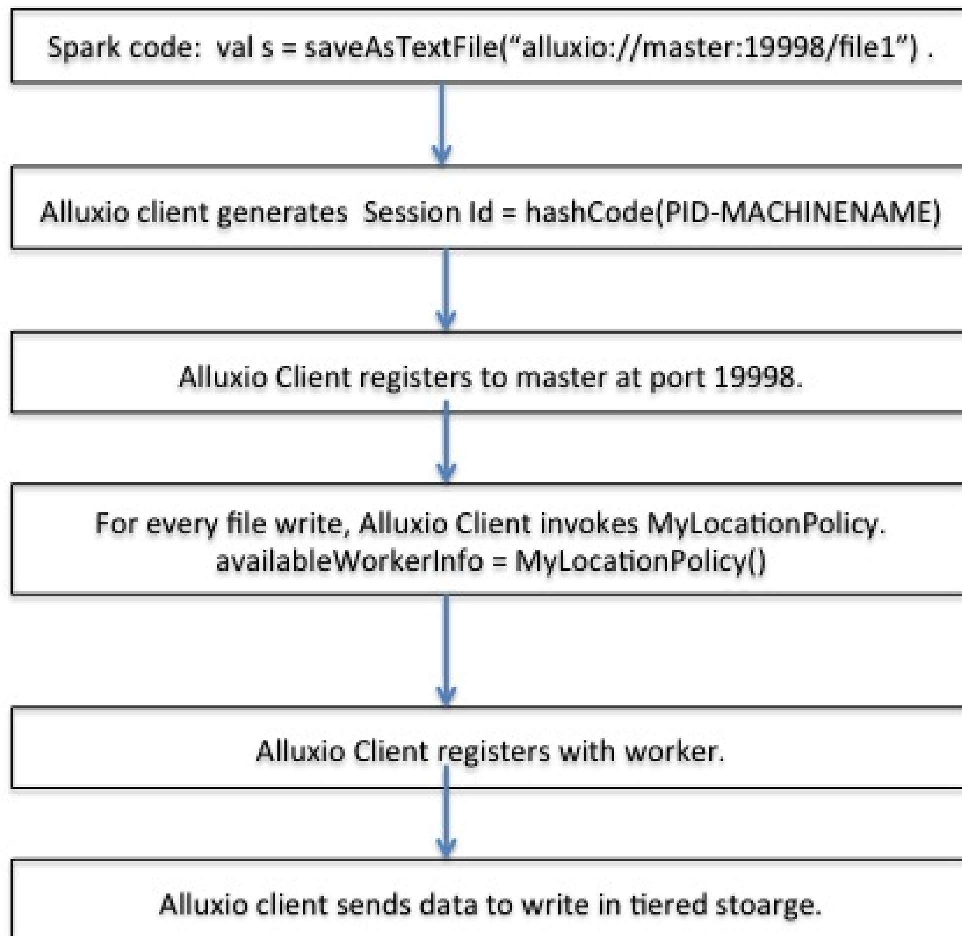


Figure 9.1: Alluxio Client

9.2.3 Alluxio Session Manager

Every Alluxio worker has a module to save the information of activities that occurs in a session. This module is called Session Manager. It is indicated in Figure 9.2. It is integrated with tiered storage to save storage information for all sessions connected to the worker. Session manager exposes APIs to verify the storage limit, save and retrieve the committed block information of each session. Session manager reads the user defined storage limit from the session file. Verify api compares the total memory storage of each session with the limit defined.

9.2.4 MyAllocator

MyAllocator is a customized allocator to allocate incoming blocks for a session in the top tier or the memory. By default all blocks are allocated in RAM. If the

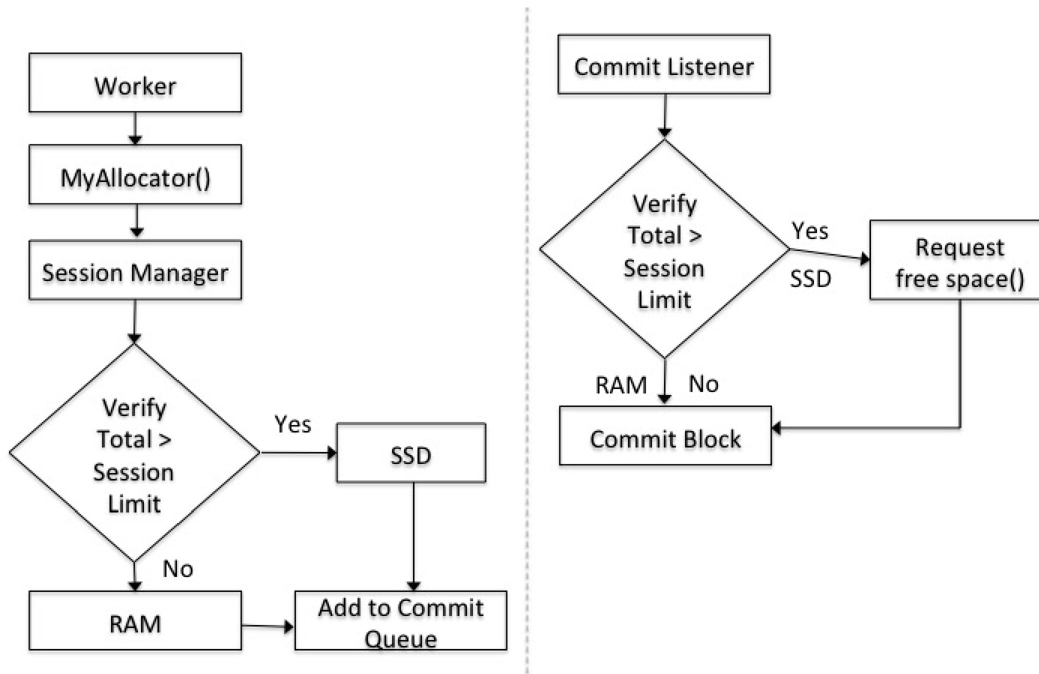


Figure 9.2: Alluxio Worker

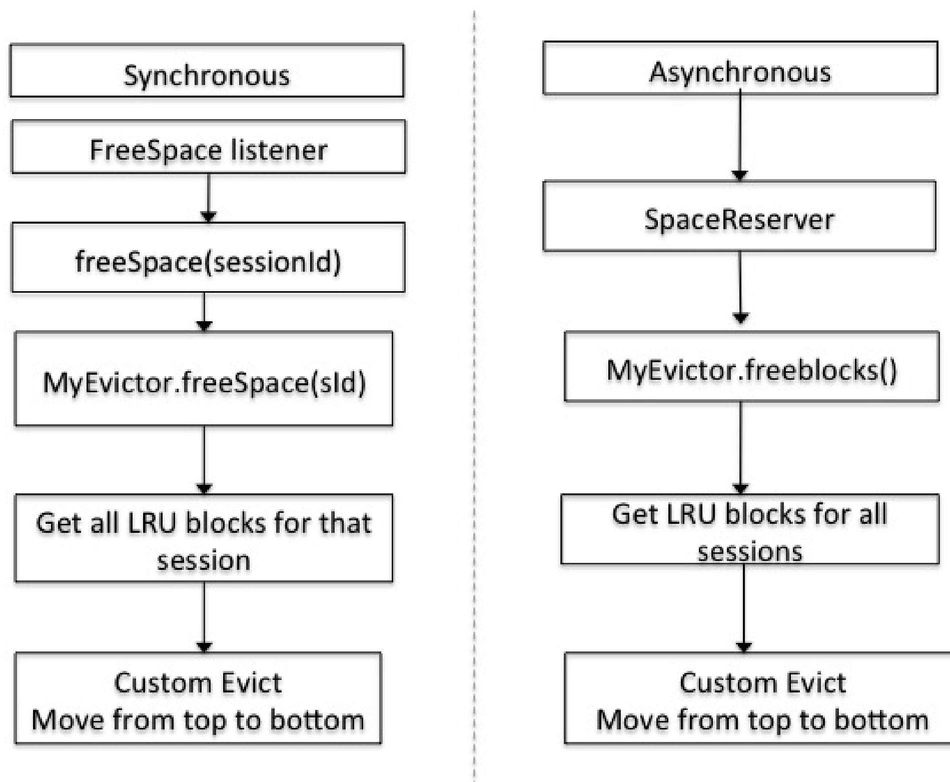


Figure 9.3: Eviction

session limit is reached, the blocks are allocated in the lower tiers with free space. Before committing blocks, session manager verify api is invoked to check if the limit is exceeded and commits it to appropriate tier. If the memory storage exceeds the limit defined, freespace is invoked to evict least recently used blocks of the session from the memory to the lower tiers. Though the freespace() is invoked synchronously, it is invoked in a different thread pool.

9.2.5 MyEvictor

MyEvictor is a customized evictor to evict blocks of a session from one tier to another tier. When the top tier is full, evictor is invoked to free space in the top tier. MyEvictor first fetches the blocks belonging to the input session using Session

Table 9.1: Defining Max Bytes per Session

Parameter	alluxio.app.max.size.bytes
Default Value	1048576
Description	Storage limit in memory(bytes)

Manager apis. Once the block ids are fetched, custom eviction is invoked to move the least recently used blocks from memory tier to lower tiers thus freeing the space in the upper tier. Also MyEvictor clears up the blocks from the block session association in the session manager as shown in as shown in Figure 9.3

9.3 Implementation

9.3.1 Enabling and Configuring Parameter to Limit Storage

Alluxio accepts inputs according to the user defined policies in a settings file. Below is the new parameter that's set to limit the storage resource per Alluxio client. The input value for the parameter is defined in bytes. Default limitation is set to 128KB per client session

The configuration should contain the size limit per session. The parameter to define maximum size per session is provided in table 9.1

9.3.2 Alluxio Client

Alluxio client registers with the FileSystemMaster and BlockMasterClient of the master at port 19998. Client starts the job and splits it into various tasks. All tasks of a job belong to the same session and each task acquires the session id using the api : createNewsession() = getSID() = hashCode(processId-machineName)

Table 9.2: Defining Location Policy

Parameter	alluxio.user.file.write.location.policy.class
Value	alluxio.client.file.policy.MyLocationPolicy

9.3.3 MyLocationPolicy

Client uses FileWriteLocationPolicy to determine the worker node where the data has to be written. A customized policy - MyLocationPolicy overwrites the default policy that returns the worker with the most available bytes. Then, once a worker has been chosen, the same worker will be saved and returned for every task. The policy returns null if no worker is qualified and the job will fail. The configuration should define the location policy. The parameter to define the location policy is provided in table 9.2

9.3.4 Alluxio Session Manager

Session manager is an instance that resides in each worker and manages the storage limit of each session in memory. Worker invokes allocator with the session information and block meta data to allocate memory in any directory as defined by the policy.

- Global data structures : Session manager stores a session hash map with the total allocated bytes for each session and another map which provides association of all blocks with their meta data associated with a session. As a block is committed to the top tier, the total bytes consumed will be summed up and saved in the former map. When an evictor is invoked to move blocks from memory to lower tiers or remove them permanently, the consumed bytes will be reduced. The session information of one client in the global maps will be erased when this client closes or disconnects the session.

- Verification : Session manager exposes an api to verify if the session has reached the maximum storage limit configured on the worker node by the user in the settings file. Max limit read from the settings file is matched with the total bytes consumed so far in memory layer for the input session. The API logs an error reporting true or false as given by the API return value.

9.3.5 MyAllocator

MyAllocator looks for any directory in the top tier of the tiered storage. To begin with it does not allocate blocks to lower tiers but the incoming blocks are always put into any directory on RAM or memory, which is the top tier. Once the directory is found, the block will be put into a queue to cache to the memory. Then a new thread is spawned on an event to commit the block to the memory. OnCommitBlock() first verifies if the session has enough available memory to store the incoming bytes, if so it allows to save in the top tier else invokes freespace() api to do a cascade eviction of the existing blocks belonging to this session in memory to the lower tier. The new blocks will be created in the second tier. Once the freespace is invoked, available space will be created for this session and the required blocks can be moved to the memory as before.

However when asynchronous eviction with space reserver is enabled, by default the blocks are allocated to SSD once the session limit exceeds. CommitBlock does not invoke freeSpace rather it is invoked at every internal by the thread that Space reserver started. This lets the allocator to allocate incoming blocks in secondary tiers if the first layer is full and there is no synchronous eviction get invoked. Therefore, Space Reserver enabled does the asynchronous eviction as needed.

The configuration should contain the new evictor class that needs to be invoked. The parameter to define evictor class is provided in table 9.3.

Table 9.3: Defining Allocator Class

Parameter	alluxio.worker.allocator.class
Value	alluxio.worker.block.allocator.MyAllocator

Table 9.4: Defining Evictor Class

Parameter	alluxio.worker.evictor.class
Value	alluxio.worker.block.evictor.MyEvictor

9.3.6 MyEvictor

Eviction can happen in two ways : Synchronous or Asynchronous. Both evictions invoke freeSpace(), which invokes the registered evictor to evict least recently used blocks from top tier to bottom or vice versa. Custom evictor takes care of either removing the blocks permanently or moving them from one tier to another. During Synchronous eviction, freespace() api passes session information to MyEvictor. MyEvictor is designed to move blocks of the input session id to the lower tiers, there by increasing the storage space in memory. Asynchronous eviction works based on reserving space in each tier. The session information is of no meaning during this eviction. Hence when space reserver is enabled, evictor evicts required size of blocks in the memory to the lower tier from all the sessions. The session manager is informed of the block ids that were moved to the lower tiers from the memory. This operation enables the allocator to allocate the incoming blocks on to memory.

The configuration should contain the new evictor class that needs to be invoked. The parameter to define evictor class is provided in table 9.4.

Table 9.5: Worker Settings

Configuration Parameters	Worker1 / Worker2
alluxio.worker.tieredstore.level0.dirs.quota	1GB
alluxio.worker.tieredstore.level1.dirs.quota	1GB
alluxio.worker.tieredstore.level2.dirs.quota	5GB
alluxio.worker.allocator.class	MyAllocator
alluxio.worker.evictor.class	MyEvictor
alluxio.keyvalue.partition.size.bytes.max	1MB
alluxio.user.block.size.bytes.default	2MB
alluxio.app.max.size.bytes	33554432
alluxio.user.file.write.location.policy.class	MyLocationPolicy

9.4 Results

The additional of modules and experiments were conducted with Alluxio version 1.0.1 and Hadoop version 2.4.0. The below sections detail the configuration settings for these new parameters and the jobs run to show case the results.

9.4.1 Client and Worker

The configuration settings on each worker node is set as defined in table 9.5. As the client sends request to read or write the files from the worker, the block ids are generated. We have generated names of the blocks, for convenience as defined in table 9.6. As defined in the configuration below, the size of each block is 2 MB and each session can save up to 32MB of bytes.

Table 9.6: Block Name to ID Mapping

Block	Block Id
A1	151325418
B1	335544322
C1	134219314
C2	134219315
C3	134219316
C4	134219317
D1	167772160
D2	167772161
D3	167772162
D4	167772163
D5	167772164
D6	167772165
D7	167772166
D8	167772167

9.4.2 Session Manager

Allocate blocks invoke session manager APIs to save the session information, along with the blocks allocated. Each session saves the blocks allocated for that session. The table 9.7 gives the session id, total blocks allocated, the number of blocks in RAM,SSD and HDD. Further the table 9.8 gives the session block list allocated for each session with the names of the blocks allocated.

Table 9.7: Session Size

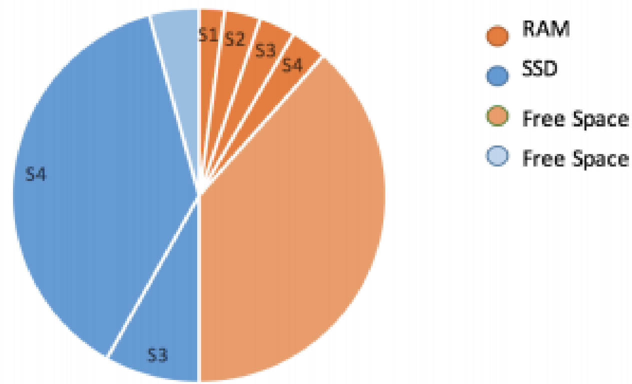
Session Id	Total (MB)	RAM (MB)	SSD (MB)	HDD (MB)
860820651	10	10	0	0
111808973	32	32	0	0
1174627532	200	32	168	0
2115169359	800	32	768	0

Table 9.8: Session Block Association

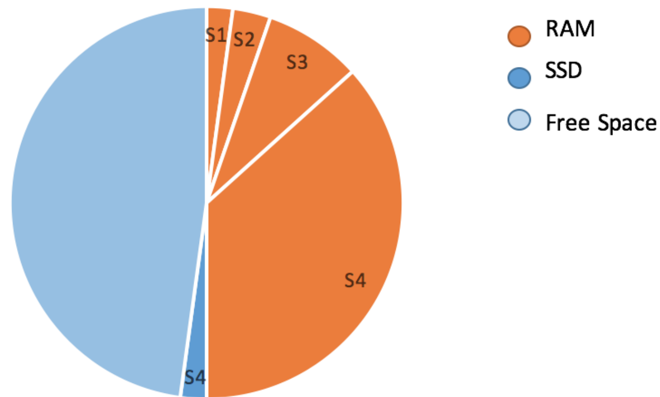
Symbol	Session Id	Blocks
S1	860820651	A1,A2,A3,A4,A5
S2	111808973	B1,B2...B16
S3	1174627532	C1,C2.....C100
S4	2115169359	D1,D2.....D400

9.4.3 Comparison

A comparison between the allocation of blocks in memory with and without limitation per session is drawn in Figure 9.4(a) and Figure 9.4(b) respectively. In the Figure 9.4(b) , the total written data size has filled up the first layer and it will cause continuous eviction with high latency if there are more incoming sessions running concurrently. However, as indicated in Figure 9.4(a), since we turn on the storage limitation, eviction can happen in advance and the memory can keep a appropriate capacity for future incoming sessions, which can support more sessions running and more balanced storage allocation.



(a) Limiting allocation in DRAM per session



(b) Occupancy in DRAM

Figure 9.4: Tiered Storage

9.4.4 Observation

The blocks are configured to be of size 2MB as indicated in table 9.5. From the results in table 9.7 and table 9.8, we observe the following :

- A file of size <2 MB, block of file size bytes resides in RAM.
- Any file of size <32 MB is saved to RAM.
- If a file of size >32 MB , 16 blocks each of 2 MB is saved in RAM, rest is stored in SSD or HDD depending on tier availability.
- Writing data in a session after the session has reached the limit, by default all blocks are allocated in SSD.
- Invoking freespace(), evictor picks up candidate blocks of the session id passed, but adds them back to RAM during custom eviction if RAM has availability of space. On further investigation its seen that candidate directory chosen to write blocks during free space does not choose the directory in other tiers if RAM has space even for a block to be written.
- Restarting worker, loses all client session information and the blocks associated to each session.

9.4.5 Inference

One of main benefits of Alluxio is that it enables data sharing among different applications at memory speed. Instead of using JVM memory like Spark, Alluxio uses DRAM as the first layer which brings higher throughput and also less garbage collection. But when we treat Alluxio as a pool of storage resource, we find that it cannot dynamically share and centrally configure this resource pool since Alluxio cannot be

aware of how the upper tier computation engine will use its storage. However, in our case, by limiting memory usage on different Alluxio clients, we can make a better use of the memory layer where you can throw entire first layer at a MapReduce job, then use some of it on a Spark application and the rest on big table query, without any changes in configuration, since the exceeded data will be evicted to the next layer automatically.

9.4.6 Future Enhancements

- With the current configuration of MyLocationPolicy, there is no need to sync the information between master and other workers. The policy is designed to return the same Alluxio worker during the entire session. It is possible that the worker could go down if a worker get restarted and the session information in the global map will be lost. To make this fault tolerant, we may have to reconstruct it from the lineage. Therefore, Worker Session information has to be fault tolerant.
- Limit the number of sessions interacting with Alluxio with an upper bound.
- Creation of data analysis report on usage per session id or client will help the admins to tune the system further.
- In addition, scale the setup and produce results with the input of variety of jobs and higher unit size of data to be written in Alluxio.

Chapter 10

CONCLUSION

10.1 Conclusion

The infrastructure setup for the hybrid model provides avenues to create new applications that perform new analytics with machine learning on the cloud. TechTalk designed to demonstrate the hybrid model is one such application.

After comparison of a few distributed systems like Hadoop,GlusterFS and Ceph, Ceph is chosen as the back end storage for the cloud. Ceph is open source, provides all types of storage - file,block and object storage from a single distributed cluster, scalable and highly fault tolerant. Ceph can easily integrate with OpenStack which is also open source and provides all tools for building and maintaining a cloud for creating labs.

Data analytics is slower with disk centric models. We experiment and compare the text analytics on direct Ceph and on Alluxio with Ceph as under storage with Spark scripts. The experiment to count the lines is same on both for the first time, but when the same experiment is repeated the line count with data in Alluxio memory there is a huge difference. The conclusion from these experiments is that Alluxio when combined with Ceph provides about $\approx$10x times faster with Scala on repeated operations on the files that's already loaded in memory than doing it on the under storage directly. However the same operations does provides only $\approx$2x benefits using PySpark. Using Scala over PySpark reduces the latency.

Spark has in-built memory. Though Spark has caching, two different jobs or applications cannot share data. Use multiple spark applications to perform offline

data mining and save results in the Alluxio. One of main benefits of Alluxio is that it enables data sharing among different applications at memory speed. These results can be saved in Alluxio and be shared with other scripts and applications to consume the data at memory speed. Also use of HDFS or Ceph Object storage in the cloud tuned in ThoTh Lab does not make any difference to the computation results of the batch Spark jobs run on the offline cloud.

The TechTalk application is tested with the infrastructure in the ThoTh Lab. It uses Ceph storage for the offline storage of lectures and Alluxio with Ceph as under storage for live recording, storing index and to do analytics. The use of Alluxio in the data analytics and ad-hoc queries brings about $\approx$10x performance benefit.

ThoTh Lab intends to provide users personalized learning based on the user behavior. The user behavior analysis will generate a lot of data that needs to be analyzed in order to derive conclusions on the patterns of the behavior. The current infrastructure that is set up provides avenues for all such applications.

Instead of using JVM memory like Spark, Alluxio uses DRAM as the first layer which brings higher throughput and also less garbage collection. But when we treat Alluxio as a pool of storage resource, we find that it cannot dynamically share and centrally configure this resource pool since Alluxio is not aware of how the upper tier computation engine will use its storage. However, by limiting memory usage on different Alluxio clients, we can make a better use of the memory layer where you can throw entire first layer at a MapReduce job, then use some of it on a Spark application and the rest on big table query, without any changes in configuration, since the exceeded data will be evicted to the next layer automatically.

10.2 Future Enhancements in the Application

The infrastructure for offline data and for online data analytics is now available for building various data analytics applications on the data that is in the store. The application can be plugged in the ThoTh lab to assist online learning based on user behavior and analysis using video knowledge base. The below enhancements can improve the efficiency of the system.

- Using Lucene with Alluxio : Since the index consists of only the keywords and the time they appear the meta data attributes now is less. But with the increase in data, and more applications the metadata will increase. Alluxio is integrating to run Solr in the cloud at memory speed. The index can be moved to Solr to support other data analytics operations and enable access refined data at memory speed.

- Session Management and Security : Right now, there is no session management and the permissions on each user to access the data. The token management and cache based on user defined behavior will improve the performance better.

- Scaling : The data in the store is now refined to 1K video with 120GB of data. Increase the content of the store to 1TB of data. The application is now running on virtual machine with access to Ceph back end storage and Alluxio via gateway and proxy respectively. Having the process run on a SSD and web cache on the web server for read and write of web server configurations will serve the consoles better.

- Live Feed: The transcription of the text from the video lectures is slow as the conversion from voice to text is done at the server end. This might provide notes and transcription of the audio in the lecture but does not speed up to

provide captioning of the lectures. There are a few factors which depends on the transcription speed - the speed at which the client streams video and audio to the Alluxio, the speed at which the web server translates the audio to text using google speech APIs. We need to find a better way to do this at the client end. But the down side is that the client might require more tools to be installed to convert the audio to text and more memory to do this processing. This has to be experimented to chart how the latency can be minimized to provide transcription for captioning of the live lectures.

- Context Based Search: The search now enables to search the contents of the video. But this can be enhanced to search based on context and synonyms. For example -
  - If the user is already browsing the algorithms related to bubble sort, the next search of time complexity should show the search w.r.t to the bubble sort and not in common to all searches.
  - If the user is searching firewall, the search should be able to identify any topics related to firewall like DMZ, how to configure firewalls etc.

10.3 Other Use Cases

- Translation plugin for video knowledge base : Provide captions to the live class lectures in the requested languages. Use the translation plugin to translate notes in requested languages to ease studying in required languages.
- Documentation of Products: This application will help in setting up documentation of any products. Users will be able to search all data related to the product and be able to view videos rather than reading texts.

- Meeting Manager : The products in the market allows to record meetings but they are not easy to search or content index the data or customize the search based on the needs of the organization. This application can be extended to use in meetings.

REFERENCES

[1] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.

[2] A. Davies and A. Orsaria, "Scale out with glusterfs," *Linux Journal*, vol. 2013, no. 235, p. 1, 2013.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE, 2010, pp. 1–10.

[4] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.

[5] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–15.

[6] Y. Deng, D. Huang, and C.-J. Chung, "Thoth lab: A personalized learning framework for cs hands-on projects," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 706–706.

[7] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[8] M. Mesnier, G. R. Ganger, and E. Riedel, "Object-based storage," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 84–90, 2003.

[9] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran, "Object storage: The future building block for storage systems," in *Local to Global Data Interoperability-Challenges and Technologies, 2005*. IEEE, 2005, pp. 119–123.

[10] Q. He, Z. Li, and X. Zhang, "Study on cloud storage system based on distributed storage systems," in *Computational and Information Sciences (ICCIS), 2010 International Conference on*. IEEE, 2010, pp. 1332–1335.

[11] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[12] K. N. Aye and T. Thein, "A platform for big data analytics on distributed scale-out storage system," *International Journal of Big Data Intelligence*, vol. 2, no. 2, pp. 127–141, 2015.

[13] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing.* ACM, 2006, p. 122.

[14] C. Maltzahn, E. Molina-Estolano, A. Khurana, A. J. Nelson, S. A. Brandt, and S. Weil, "Ceph as a scalable alternative to the hadoop distributed file system," *login: The USENIX Magazine*, vol. 35, pp. 38–49, 2010.

[15] X. Zhang, S. Gaddam, and A. Chronopoulos, "Ceph distributed file system benchmarks on an openstack cloud," in *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on.* IEEE, 2015, pp. 113–120.

[16] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, "f4: Facebooks warm blob storage system," in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation.* USENIX Association, 2014, pp. 383–398.

[17] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[18] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum *et al.*, "The case for ramclouds: scalable high-performance storage entirely in dram," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 92–105, 2010.

[19] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, vol. 2004, no. 124, p. 5, 2004.

[20] A. Gupta, R. Spillane, W. Wang, M. Austruy, V. Fereydouny, and C. Karamanolis, "Hybrid cloud storage: Bridging the gap between compute clusters and cloud storage," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 48–53, 2017.

[21] Y. Hua, H. Jiang, and D. Feng, "Fast: Near real-time searchable data analytics for the cloud," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE Press, 2014, pp. 754–765.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[23] S. Liu, J. Tang, C. Wang, Q. Wang, and J.-L. Gaudiot, "Implementing a cloud platform for autonomous driving," *arXiv preprint arXiv:1704.02696*, 2017.

[24] S. Liu, B. Ding, J. Tang, D. Sun, Z. Zhang, G. Tsai, and J.-L. Gaudiot, "Learn-memorize-recall-reduce a robotic cloud computing paradigm," *arXiv preprint arXiv:1704.04712*, 2017.

[25] "Arcgis and alluxio." [Online]. Available: https://github.com/mraad/arcgis-alluxio

[26] Y. Huang, Y. Yesha, M. Halem, Y. Yesha, and S. Zhou, "Yinmem: A distributed parallel indexed in-memory computation system for large scale data analytics," in *Big Data (Big Data), 2016 IEEE International Conference on.* IEEE, 2016, pp. 214–222.

[27] "Acclerating big data analytics on ceph storage with alluxio." [Online]. Available: https://www.alluxio.com/blog/accelerating-data-analytics-on-ceph-object-storage-with-alluxio

[28] "Baidu queries data 30 times faster with alluxio." [Online]. Available: https://www.alluxio.com/assets/uploads/2016/02/Baidu-Case-Study.pdf

[29] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data.* ACM, 2015, pp. 1383–1394.

[30] A. Kleftodimos and G. Evangelidis, "Using open source technologies and open internet resources for building an interactive video based learning environment that supports learning analytics," *Smart Learning Environments*, vol. 3, no. 1, pp. 1–23, 2016.

[31] Y.-L. Chang, W. Zeng, I. Kamel, and R. Alonso, "Integrated image and speech analysis for content-based video indexing," in *Multimedia Computing and Systems, 1996., Proceedings of the Third IEEE International Conference on.* IEEE, 1996, pp. 306–313.

[32] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," in *Text Mining: Applications and Theory*, 03 2010, pp. 1 – 20.

[33] D. Borthakur, "Hdfs architecture guide," *Hadoop Apache Project*, vol. 53, 2008.

[34] T. W. Dinsmore, "In-memory analytics," in *Disruptive Analytics.* Springer, 2016, pp. 97–116.

[35] F. Wu and G. Sun, "Software-defined storage," *Report. University of Minnesota, Minneapolis*, 2013.

[36] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, "Rados: a scalable, reliable storage service for petabyte-scale storage clusters," in *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07.* ACM, 2007, pp. 35–44.

[37] "Ceph recommends odd number of monitors." [Online]. Available: http://docs.ceph.com/docs/kraken/rados/operations/add-or-rm-mons/

[38] G. Donvito, G. Marzulli, and D. Diacono, "Testing of several distributed file-systems (hdfs, ceph and glusterfs) for supporting the hep experiments analysis," in *Journal of Physics: Conference Series*, vol. 513, no. 4. IOP Publishing, 2014, p. 042014.

[39] "Red hat - ceph and gluster storage." [Online]. Available: https://www.redhat.com/en/technologies/storage

[40] "Ceph features." [Online]. Available: http://docs.ceph.com/docs/master/architecture/

[41] "Adding or removing monitors in ceph." [Online]. Available: http://docs.ceph.com/docs/kraken/rados/operations/add-or-rm-mons/

[42] "How placement groups are used in ceph?" [Online]. Available: http://docs.ceph.com/docs/master/rados/operations/placement-groups/

[43] "Alluxio architecture." [Online]. Available: http://www.alluxio.org/docs/master/en/Architecture.html

[44] "Big data processing with apache spark." [Online]. Available: https://www.infoq.com/articles/apache-spark-introduction

[45] Z. Han and Y. Zhang, "Spark: A big data processing platform based on memory computing," in *Parallel Architectures, Algorithms and Programming (PAAP), 2015 Seventh International Symposium on*. IEEE, 2015, pp. 172–176.

[46] "Ceph s3 rest apis." [Online]. Available: http://docs.ceph.com/docs/master/radosgw/s3/

[47] "Alluxio support for s3 to interact with ceph." [Online]. Available: https://www.alluxio.com/docs/community/1.5/en/Configuring-Alluxio-with-Ceph.html

[48] M. Habibi and A. Popescu-Belis, "Keyword extraction and clustering for document recommendation in conversations," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 23, no. 4, pp. 746–759, 2015.

[49] "Autosub: Command-line utility for auto-generating subtitles for any video file." [Online]. Available: https://github.com/agermanidis/autosub

[50] D. Bolton, "Youtubedl: command-line program to download videos from youtube.com and a few more sites." [Online]. Available: https://github.com/rg3/youtube-dl