EECS4314 Assignment 1: Apache Hadoop Conceptual Architecture

Randy Agyapong
David Iliaguiev
Hashim Al-Helli
Zhongran (Julian) Deng
Sied Hoa (Heny) Tjin

October 19, 2016

Contents

List of Figures

List of Tables

1 Abstraction

This report provides an overview of the conceptual architecture of Apache Hadoop. We will discuss the problems that led to the development of Hadoop as well as the different components of Hadoop and how they work together to solve the problems. We will analyze and explain the major components in the Hadoop system. These compnents are YARN, MapReduce, and HDFS. Several Use Cases will be described to show a typical interaction between the client and the sytem. Hadoops architecture allows for an ease of implementation between developers, to collaborate and easier create user applications for production. Finally, we we discuss lessons learned in creating this report.

2 Introduction and Overview

Data can be classified as Big Data when it becomes difficult to store, search, or even manage data using traditional management tools. Such Big Data may reach sizes in the petabytes, or 2^{50} bytes. Even the Internet produces petabytes of data daily, whether it'd be info generated by the millions of users browsing, or automatically through systems and programs. Managing these large scales of data in a reliant and efficient manner ultimately led to the creation and development of Apache Hadoop.

HDFS, whose full name is Hadoop Distributed File System, is an open-source, distributed, scalable and portable file system written in Java for the Hadoop framework [6]. HDFS is designed for commodity hardware to store and process big data across multiple machines to achieve parallel computing. It provides an ideal pattern to store large amounts of data overcoming the limit of traditional data storage pattern. It provides a foundational file system structure for MapReduce algorithm. Basic concepts of HDFS including HDFS cluster, name node, data node, data block, data block replication, secondary name node, backup node will be explained in detail in below sections.

MapReduce is the heart of Hadoop. It is a processing technique and a programming model for distributed computing, which allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. Its parallel processing makes job execution extremely fast and efficient. The program contains two important tasks, namely Map and Reduce. In the earlier version, MapReduce took care of scheduling tasks, monitoring them and re-executing any failed tasks.

YARN is a resource manager that was created in Hadoop 2.0 to overcome the problems with MapReduce 1.0. YARN takes over the job of resource management from MapReduce, while letting MapReduce manage data processing. YARN is a major improvement in hadoop 2.0 because it allows for scalability, overcoming the single point of failure, and provides better resource utilizations [16].

Table 1: Major differences between Hadoop 1 and Hadoop 2

Point	Hadoop 1	Hadoop 2
1	Supports MapReduce (MR) processing model only. Does not support non MR tools.	Supports MapReduce as well as non-MapReduce tools
2	MapReduce does both the cluster management and the processing of data	YARN (Yet Another Resource Manager) does cluster management, while MapReduce or other tools does processing of data
3	Limited scalability of nodes up to ∼4000 nodes	Has higher scalability up to ∼10000 nodes
4	Has a single point of failure because of single NameNode	Overcame the single point of failure with a standby NameNode. In case of Namenode failure, an automatic takeover will be initiated by the standby node
5	No support for Microsoft Windows	Added support for Windows platform

3 Architecture

Apache Hadoop is an open-source software framework, designed to process and store Big Data. Its unique design architecture allows it to transcend beyond the limitations of traditional management tools and systems. It was designed to effectively process large volumes of information by connecting many commodity computers together to work in parallel, i.e. it utilizes the underlying parallelism of the CPU core [29].

- This solves the problem of botttleneck of a single disk drive (by applying the same tasks to many disks, each holding a different portion or slice of the overall data)
- By tying many commodity computers together with a single CPU, the machines can read dataset in parallel, leading to higher throughput at a reduced cost compared to using one single high-end server
- Servers can be added or removed from the cluster dynamically without interrupting the operation of Hadoop. This makes Hadoop highly scalable

Apache Hadoop is designed with the fundamental assumption that hardware failure is common, and it is the job of the system to handle hardware failure automatically by the platform. Hence, self-healing is crucial to have tasks continue working.

- Hadoop is designed to use large number of commodity servers for computation and storage. As number of machines increases, so does the probability of server failure.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability, rather Hadoop library itself has been designed to detect and handle failures at the application layer

3.1 HDFS

Prior to the emergence of HDFS, a traditional relational database pattern was broadly used in industry and academics. It nicely solves the problem of storing structured data of small size. However, it is not ideal enough to process large amount of data and it does not provide sufficient support to store structured data. Each block of data for traditional relational database model is small of approximately 51 bytes [18]. Another problem is there is a huge amount of I/O operations while large data is being read, which consumes processing power greatly. The birth of HDFS solved this problem. Each block of data for a HDFS file system is large. By default, it is 64MB and could be increased. Also, it reads large data sequentially after a single seek operation, which saves processing power greatly.

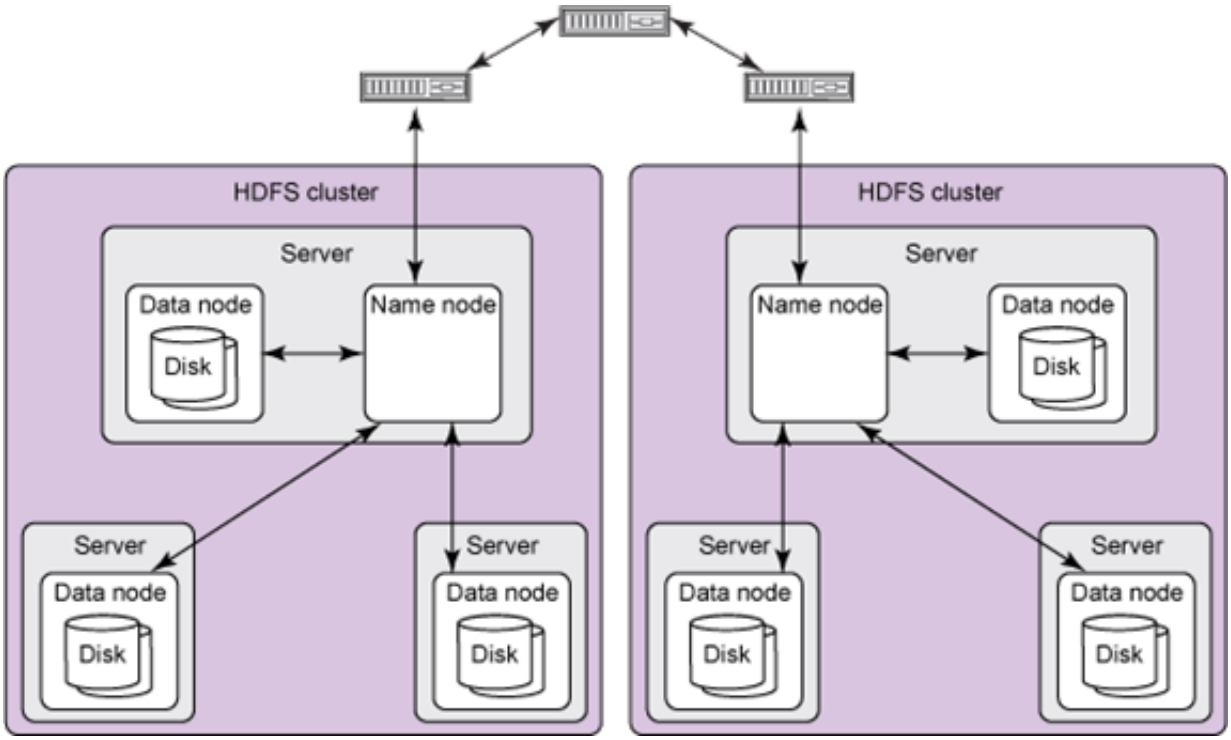

Figure 1: Typical architecture of HDFS [19]

The diagram shown above represents a typical HDFS architecture. Despite version updates from time to time, the basic architecture is maintained. An HDFS consists of

multiple servers. There is one server holding the NameNode which is very crucial as there is only one NameNode existing in the whole cluster. Several DataNodes belong to the NameNode. They can reside in the server with the NameNode and they can also be stored in different server. DataNodes hold data blocks, which are where the real files are stored. This architecture demonstrates an implementation of parallel computing.

NameNode

A NameNode is unique within a cluster in Hadoop 1.0. It is used to store metadata such as directory structure of file systems as well as basic information of the file system. A NameNode is aware of what certain file data is stored within DataNode. In version 1.0, it was a single point of failure, which means, if NameNode becomes unavailable, the whole system will go down. A NameNode holds two persistent files: transaction log namely edit logs and namespace image namely fsimage. Edit logs keeps recording changes of metadata (e.g. creating new file) while fsimage maintains the information of the entire file system namespace (e.g. mapping of blocks and file system properties) [20].

Secondary NameNode

Secondary NameNode is not a backup daemon for the NameNode [21]. We cannot treat it as a back-up NameNode and we cannot completely depend on secondary NameNode for the recovery process of a NameNode. It is configured more like a check-pointer [22]. Secondary NameNode stores the latest checkpoint in a directory by maintaining two persistent file edit logs and fsimage which is structured the same way as the primary NameNodes directory. Such checkpointed image is always ready to be read by the primary NameNode [23] and thus reduce the restart time of NameNode [21].

DataNode

DataNode stores and maintains data blocks which hold the actual file data. It is responsible for storing and retrieving data blocks upon request from NameNode or the client. There can be multiple numbers of DataNodes depending on the system requirement. Operations such as read, write, block creation and replication can be performed on DataNode.

Data Block

Data block holds actual file data. Each file is split into one or more blocks that are stored and replicated in DataNodes. By default, each block is 64MB in size, and the size could be increased to 128MB. The data blocks are replicated for fault tolerance. Figure 2 shows an example of data block replication, where data are replicated in multiple data nodes. The data blocks are distributed in DataNode system within the cluster and thus ensures the replica of data is maintained. The administrator of HDFS can designate which rack a node belongs. This is called rack awareness.

Interactions of Each Components

All HDFS communication protocols build on TCP/IP protocol. Client communicate with the NameNode using a Remote Procedure Call (RPC)-based protocol. Each data node serves up blocks of data over the networking using block protocol specific to HDFS [19].

Weakness of HDFS in Hadoop 1.0

There was a single point of failure problem existing in version 1.0. Once NameNode

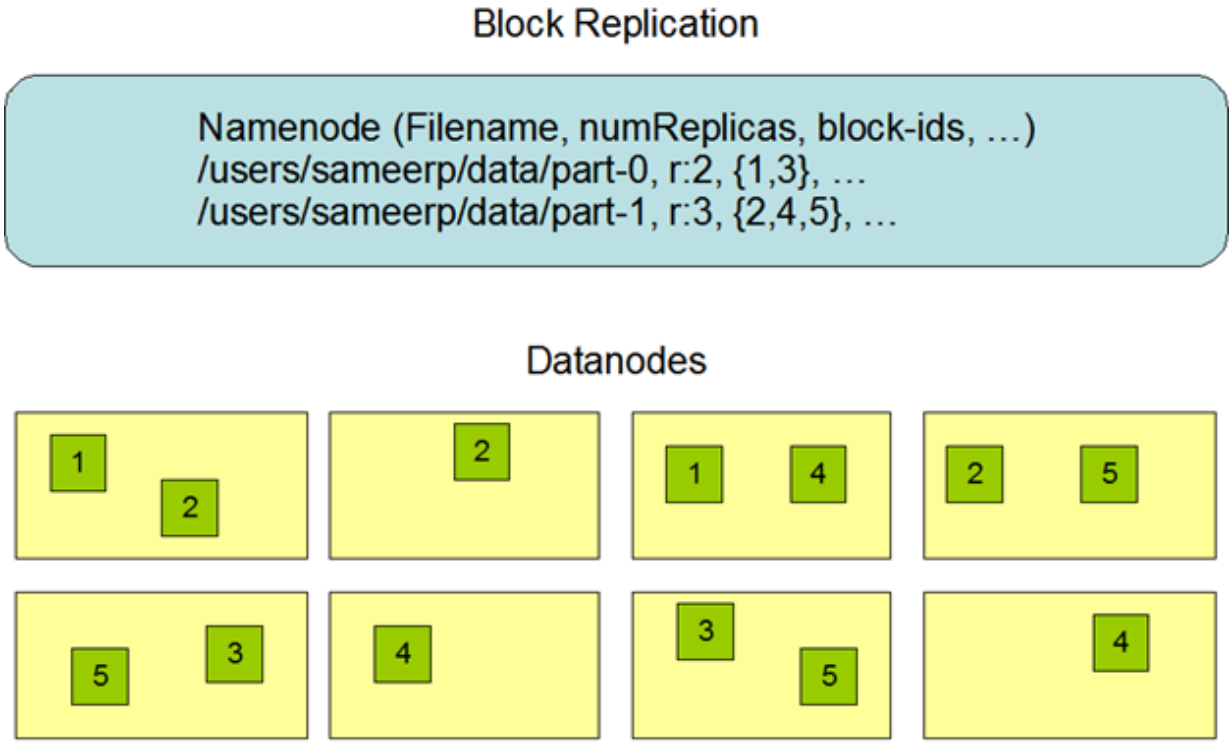

Figure 2: Data Block Replication[20]

server goes down, the whole cluster becomes unavailable. Measures need to be taken to ensure High Availability.

Support of Standby NameNode in later version

In later version, standby NameNode is supported to ensure the High Availability feature of Hadoop. The standby NameNode is in the same cluster with the primary NameNode. These nodes in the cluster are in Active/Passive state and only one NameNode is allowed to be in Active state at a time.

3.2 Yet Another Resource Negotiator

YARN (Yet Another Resource Negotiator) is an application framework introduced in Hadoop 2.0 to overcome the problems of Hadoop 1.0 MapReduce which were scalability, availability, resource utilization, and support of alternative paradigms and programming environments. YARN components are Resource manager, Application Master, and Node Manager [16].

Resource manager allocates the resources to the applications based on the applications needs. YARN uses a separate Resource Manager to schedule and manage all their jobs in the cluster. The Application Master component request resources from the resource manager and run/monitor the tasks. The node manager component monitors the resources

usage and report back to the resource manager. All nodes are managed by a node manager working together with the Resource Manager. The job resources are assigned to different containers (computing resource that contains processing node and a memory). Each application must start an Application Master to manage the actual task for the job. The Application Master is run by the container, scheduled by the resource manager, and managed by a node manager. To run an actual application, the Application Master request additional containers from the resource manager. These containers are where the actual work will be done for the cluster. The relationship between the Resource Manager and the Application Master can be dynamic meaning that containers can be allocated or deallocated dynamically during run time. Figure 3 is an example off the flow of a MapReduce job in Hadoop 2.0. [16]

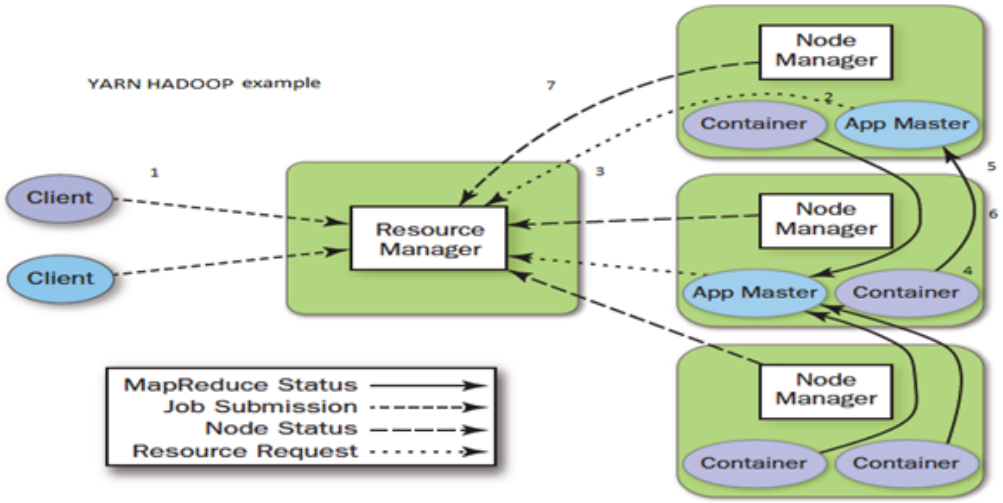

Figure 3: Control flow of MapReduce using YARN

3.3 MapReduce

MapReduce is the second main subsystem/component of Apache Hadoop version one. It ultimately provides the processing portion of the system. The MapReduce engine in version one included both cluster resource management and MapReduce processing. MapReduce is assigned with the following tasks of the system:

- Framework for performing parallel processing on data in distributed file system
- Check the code was executed successfully
- Performing sort which takes place between map and reduce stages

- Sending sorted data to a certain computer
- In Version1: perform task of Resource Manager and Scheduler

Mechanism of MapReduce Program [25]

The goal of MapReduce programs is to compute large volumes of data in a parallel fashion. This requires dividing the workload across a large number of machines. The model enable linear scalability, provided there is no data or resource sharing between every node.

Conceptually, MapReduce programs transform lists of input data elements into lists of output data elements. A MapReduce program will do this twice, using two different list processing: map, and reduce. Thats mean MapReduce program/framework consists of two main transformations, map transformation and reduce transformation, and they run in parallel with each other.

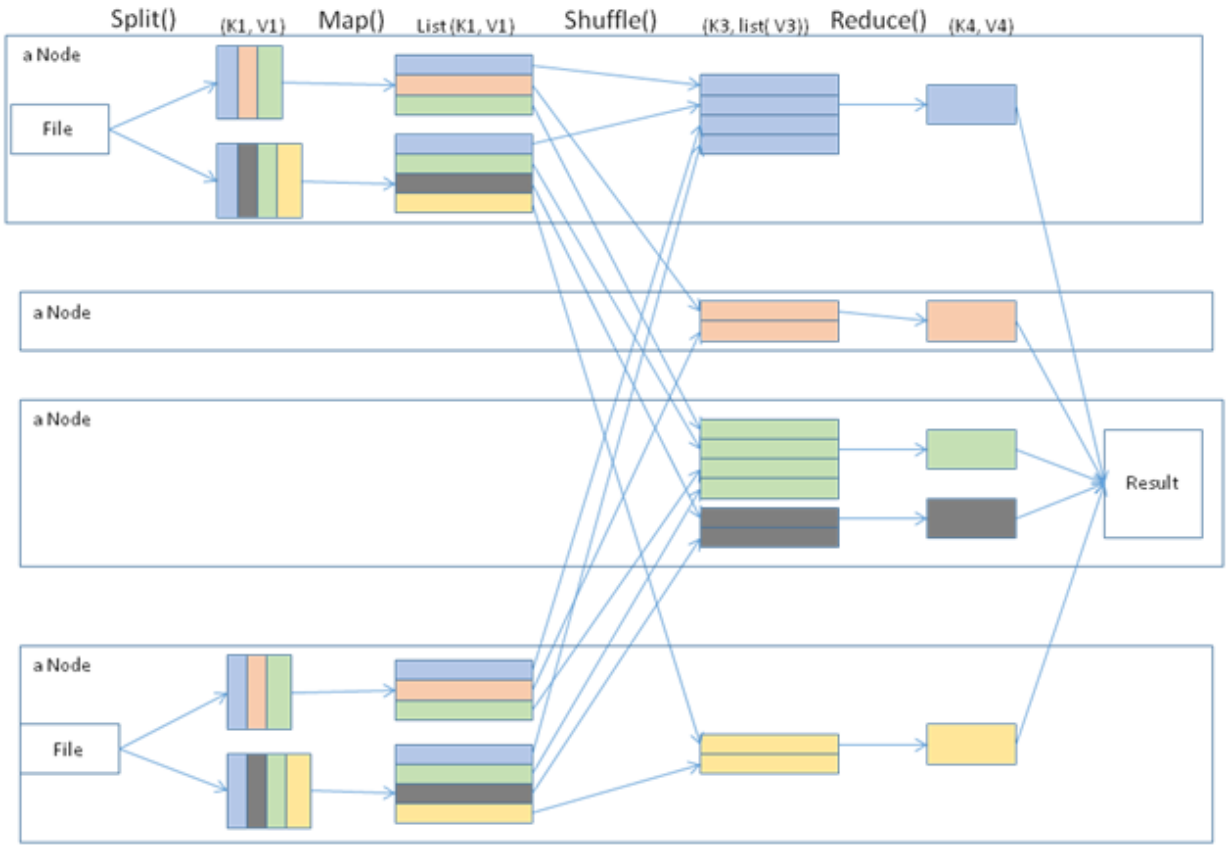

Figure 4: Control flow of MapReduce using YARN [26]

Input Splits: Prior to executing the MapReduce program, local input data is pre-loaded and split. The split used is logical boundaries based on the input data. For example, split

size can be based on the number of records in a file or actual size in bytes. The number of splits corresponds to the number of map process used in the map stage.

Mapping Step Map(): Map(K1,V1) → list(K2,V2): The mapping processes run concurrently. For large amounts of data, many map processes can be operated at the same time. The specified mapping process is determined the user. MapReduce program will try to execute the mapper on the machines where the block resides and (if there is data replication) the least busy node. Last choice being any node in the cluster that has access to HDFS. Each map process will have (key, value) pair input, and it will generate a list of (key,value) pairs.

Shuffle Step: Before the parallel reduction stage can complete, all similar keys must be combined and counted by the same reducer process. Results from map processes are shuffled to the same reducer process. Hence, if there is only a single reducer process, shuffle step is not necessary.

Reducing Step Reduce(): (K3, list(V3) → (K4, V4): Reducing lets you aggregate values together. A reducer function receives an iterator of input values from an input list. It then combines these values together, returning a single output value. The results are written to HDFS. Each reducer will write an output file.

Share Nothing Architect [27]: The framework of data processing in Hadoop implements the Share Nothing Architecture

- Each node is independent of other nodes in the system
- No share resources can become bottlenecks
- Lack of shared data: each node is processing distinct subset of data, hence no need to manage access to shared data

Pipe and Filter Architect: MapReduce program uses the pipe and filter Style.

- Map and Reduce stages are the filters.
- The node of map and reduce are independent of each other.
- Data stream routed from nodes of Map stages, to nodes of Reduce stages.

Advantages:

- It is easy to understand the flow of the data
- It is reusable as any two nodes can be connected together
- It is scalable since new nodes can be added to the cluster
- It support concurrent execution, hence the map tasks and reduce tasks can be run concurrently

Disadvantages: Loss of performance due to message passage, marshalling and unmarshalling between any two nodes (especially in the shuffle stage)

Advantage of this framework:

1. Easily manage workflow (associated with transparent process feature)
2. Scalable: No shared resources, hence addition of nodes adss resources to the system and does not add further contention. As input data increases, just need to apply more nodes (linear scalability)
3. Fault tolerant: Each node is independent, hence no single points of failure. A failed process in one node can be restarted on another node. The system can support multiple failures, depending on the number of data replication. Hence hardware failure will only slow down the process, but not entirely crash the whole job. Inputs are immutable, hence the result can be recalculated.

All data elements in MapReduce are immutable, meaning that they cannot be updated. If in a mapping task you change an input (key, value) pair, it does not get reflected back in the input files; communication occurs only by generating new output (key, value) pairs which are then forwarded by the Hadoop system into the next phase of execution.

Performance Critical: The number of slots needs to be carefully configured based on available memory, disk, and CPU resources on each slave node. Memory is the most critical of these three resources from a performance perspective. As such, the total number of task slots needs to be balanced with the maximum amount of memory allocated to the Java heap size.

The ratio of map slots to reduce slots is also an important consideration. If you have too many map slots and not enough reduce slots for your workloads, map slots will tend to sit idle, while your jobs are waiting for reduce slots to become available.

Basically, the pushing of the job (scheduling) is processed with the objective of Reducing network traffic on the main backbone network. This is crucial, since if one of the job tracks slow down, the entire MapReduce Job slows down as well, especially if it is the last piece of information. Distinct sets of slots are defined for map tasks and reduce tasks because they use computing resources quite differently. Map tasks are assigned based on data locality, and they depend heavily on disk Input, Output (I/O), and CPU. Reduce tasks are assigned based on availability, not on locality, and they depend heavily on network bandwidth because they need to receive output from map tasks.

In Hadoop version 1, the JobTracker was the global control flow. It needs to run on a master node in the Hadoop cluster as it coordinates the execution of all MapReduce applications in the cluster. Its a mission-critical service and in fact is the single point of failure: should the JobTracker crashed, all the currently running MapReduce tasks are killed. This is remedied with the introduction of YARN.

To support immutability of the input, there is a lot of reading and writing happening to the disk after each MapReduce task which slow down the process and hence make it less suitable for iterative processing.

The shuffle step requires Remote Procedure Call to proceed to reduce stage. The data movement from one node to another can be expensive in terms of processing time, and might clutter the backbone network.

Concurrency: The MapReduce job, managed by every node, execute concurrently: all map and reduce process run concurrently.

Limitation of MapReduce in Hadoop Version 1: MapReduce 1.0 tracks the availability of slots without considering the system load of the allocated machine. Measurement by system load will be a better reflection of the actual availability of the machine [28].

If one of the track slow downs, the entire MR job gets delayed (especially when it is waiting for its last bit of the data).

Availability: If Job Tracker fails, all queued and running jobs are killed.

Other than MapReduce program, there is no support for alternate programming paradigms and services.

Scalability: due to single JobTasker, the maximum number of nodes and tasks are fixed according to the capacity of the Job[27].

4 Use Cases

Figure 3 describes the control flow of an input file using YARN and MapReduce. The type of input is related to the mapper and reducer class that MapReduce will use. Although YARN and MapReduce are completely decoupled from HDFS, the location of the input and the output can be from HDFS. Here are the steps:

1. The client submits the job to the Resource Manager.
2. The Resource Manager takes responsibility in distributing the job to its slaves/nodes. It launches the Map Reduce Application Master in one of these nodes.
3. The Application Master registers with the Resource Manager so that theres a direct two-way communication between the two entities.
4. The Application Master and the Resource Manager negotiate for resources. For example, the Application Master may ask for two nodes of a certain size, and the Resource Manager either permits this request or denies it, depending on the resources available.
5. Once negotiations are complete, the Application Master sends the container launch specification to the other Node Managers.
6. The Node Managers in their own individual nodes launch their containers.
7. Each node runs their Map task.

8. After the completion of the Map task of the first node, the nodes run their Reduce task

9. The outputs are aggregated and returned to the client

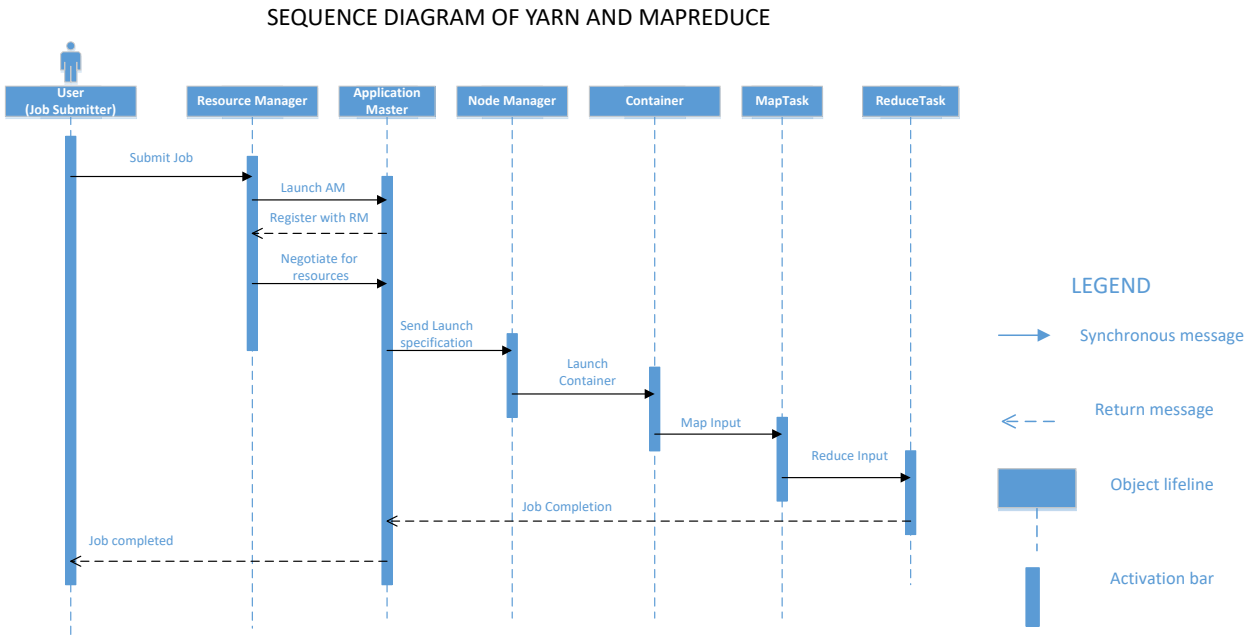

Figure 5: Sequence Diagram of YARN and MapReduce

Figure 4 diagram describes the control flow of a client reading a file stored in HDFS. The client issues a read command, and HDFS successfully returns the read file. Here are the steps:

1. The HDFS client opens the Distributed File System

2. The Distributed File System gets the block locations of the nodes along with any other notable information from the NameNode

3. The NameNode returns a FSDataInputStream object

4. The HDFS client issues a read() command to FSDataInputStream.

5. FSDataInputStream issues read() to the various DataNodes that contain parts of the file

6. The files contents are returned to the Data Stream
7. The data stream returns the information to the client

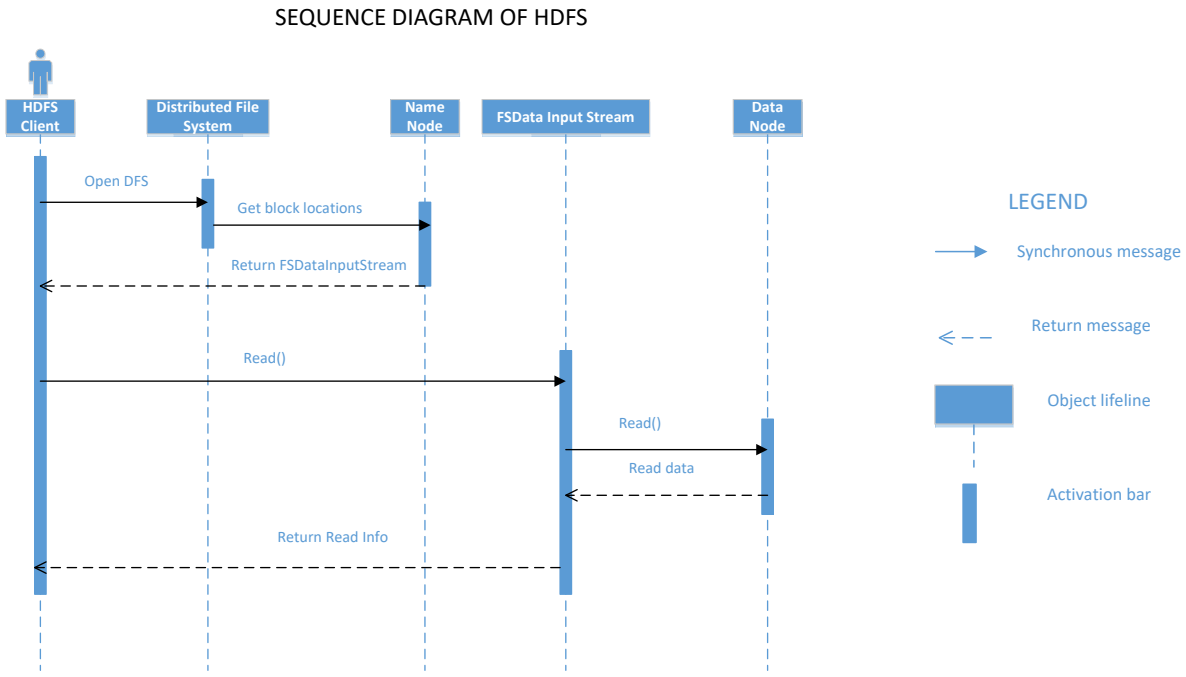

Figure 6: Sequence Diagram of read() in HDFS

5 Applications and Implications for Developers

Apache Hadoop has many ways for developers to create applications for users. It can be a knowledge base for AI applications such as a heuristic weather predictor or make use of financial forecasting using mathematical models. It can be a storehouse of data used for production facilities or a framework for retail and commercial software, the applications are endless. The major key to Hadoops success is the wide variety of ways to create applications for variously skilled developers to access, manipulate and maintain the system. We will look at these to see how developers can implement applications as well as more abstractly designing software that uses the Hadoop framework and talk about the implications for developers and interested parties.

5.1 Access

Apache Hadoop is built on the TCP/IP protocols which allows widespread use of the system where parallel computations can be done close to where the data exists on the system. Users do not have to worry about performance of these computations as NameNode provides multithreading which avoids the bottleneck of saving to disk so that the application can run parallel to each other and the developers only need to check that the transactions being done have been saved.[4] The developers can continuously maintain the software without having to worry about users having access simultaneously as HDFS implements a single-writer, multiple-reader model such that the users will continue to access and the updates occur only when the users are inactive.[4]

Developers can create many ways for software administrators to maintain their data on the system. They can create a web-platform so that the administrators could browse HDFS instances through an HTTP browser.[8]The developers can also create a way to access the HDFS instances easier by using an NFS gateway so that the HDFS can be mounted as part of the clients local file system.[9] Hadoop also provides the FS shell command line interface for skilled administrators to access the HDFS directly through command line.[14] This also allows developers to create batch programs for their applications using scripting languages such as Python.

5.2 Application Development

The Hadoop architecture makes it easy for developers to create software using various programming languages through detailed libraries and APIs. The HDFS provides a FileSystem Java API so that developers can use the Java programming languages to implement programs to access and manipulate the HDFS file system.[6,13] MapReduce maintains the jobs and tasks that run on the system and Hadoop gives the application Master REST API to allow developers to check the status of the running MapReduce application master, this includes the Jobs API and Tasks API for the Java programming language.[10] With Hadoop 2 providing YARN on top of MapReduce developers can use the YARN web services REST APIs which are a set of URI resources that give access to the cluster nodes, applications and application historical information.[12] With these developers can create software to maintain all applications that are running and finished running on the system to optimize the software for their users.

For security developers can use the Hadoop Auth API for Java, HTTP and SPNEGO, a library consisting of client and server components to enable Kerberos SPNEGO authentication for HTTP browsers securing the web services created for administrators.[6, 15] Also the CredentialProvider API can be used to maintain passwords securely as it is a SPI framework for plugging in extensible credential providers, as it abstracts the use of sensitive tokens, secrets and passwords from the details of their storage and management.[5] The Hadoop system also gives developers the option of using other languages such as C with the language wrapper for the Java API and REST API, as well as SQL querying.[7]

With Hadoop using Java as the basis for their framework and APIs provided, developers streamline the software development lifecycle. As Java is an object-oriented language developers have all the advantages of abstraction of implementation that comes with such a language such as modularity, maintainability, the use of OODP (Object-Oriented Design Patterns), and able of creating new frameworks and systems for the software that uses Apache Hadoop. The Layers of Hadoop also highlights this with the separate APIs provided for each of the core systems, HDFS, MapReduce, YARN and others in the Hadoop ecosystem allowing for the distribution of work amongst developers making developing and maintaining software easier.

5.3 Project Bylaws and Committership - Contributing to Apache Hadoop

Apache Hadoop defines bylaws under which the project operates and has criteria for committers, developers and users that want to contribute to the project, responsible for reviewing and integrating code changes.

There are a set of roles defined by Apache projects that have certain rights and responsibilities. There are the Users of the Hadoop system which provide feedback to developers via bug reports and suggestions, these users sometime get so involved that they themselves become contributing developers. The Contributors are volunteers that contribute their time, code and resources to Hadoop and continue to help until they may be invited to become committers to the project. Committers are responsible for the technical management of the project and have access to subproject subversion repositories so that they can maintain and build on top of the current development of the project. As noted above they are given access through invitation approved by active PMC (Project Management Committee) members and also may cast binding votes on technical discussion regarding subprojects. These committers can eventually become a member of the PMC themselves after contributing to the project after a long time. There are also committers that are Release Managers they volunteer to produce a Release Candidate according to certain restrictions and are responsible for building consensus around it to achieve a successful Product Release vote. Finally the PMC is responsible for the management and oversight of the Hadoop codebase, they decide what is distributed as products, maintain the projects shared resources, speak on behalf of the project, resolve license disputes, nominate new members and committers and maintain the bylaws.[1,2]

The decision making concerning the project is done through a voting system. As noted above the PMC members and committers have a vote. The votes are placed as so:

- a +1 for a Yes and available to help
- a +0 for agreement but will not be able to help
- a -0 for someone who does not agree but is not concerned if the action goes ahead, and

- a -1 for a No and is counts as a veto.

Approvals are done differently for different actions:

- A consensus approval requires 3 binding +1 votes
- a lazy consensus requires no -1 votes
- a lazy majority requires 3 +1 votes and more +1 votes than -1 votes, and
- a lazy two thirds majority which requires at least 3 votes and twice as many +1 votes and -1 votes.

The vetoes that are cast must be followed with a valid reasonable explanation otherwise it is not considered. The various actions that can be voted on are:

- code changes
- a product release
- adoption of a new codebase
- a new branch committer
- new committers
- new PMC members
- branch committer removals
- committer removals
- PMC member removals
- modifying bylaws

Voting remains open for seven days for all actions with the exception of product releases which run for a period of five days.[1] The guidelines of which additions are made to the Apache Hadoop project allows for general developers using the system to contribute and become a part of the project to better their software as well as the system it runs off.

6 Conclusion

Apache Hadoop is a very powerful, yet efficient system for data processing and management. It's versitle and readily available to run on commodity-hardware. Each major component of Hadoop plays a special role in holding the system together. YARN provides the resource management and scheduling in order to accomplish tasks. HDFS acts as the file hub, maining huge amounts of data in various nodes and clusters. Finally, MapReduce provides the processing technique and programming modelling for distributive computing. Users, developers, and managers can interact and contribute freely on this open-source platform in order to bring forth ideas to the system. With size of data increasing exponentially over the years, Apache Hadoop provides a reliable base to manage Big Data.

7 Lessons Learned

One of the more difficult aspects in gathering information is trying to synchronize our understanding with the wide variety of sources available on the internet. Finding a good source that suits our current level of understanding of the system took some time. We should be more proactive communicating with the instructor for suggestions on appropriate resources. This will reduce the amount of time we may need to look for appropriate resources. At the initial stage, when we were all new to the subject, understanding the interactions between different components was challenging. We learned the material from different resources on the internet, and every resource has its own technical acronyms and its unique explanations, as such, trying to synchronize the information took some work.

8 References

[1] https://hadoop.apache.org/bylaws.html
[2] https://hadoop.apache.org/committer_criteria.html
[3] http://www.aosabook.org/en/hdfs.html
[4] http://hadoop.apache.org/docs/current/
[5] http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/CredentialProviderAPI.html
[6] http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
[7] http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/LibHdfs.html
[8] http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html
[9] http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsNfsGateway.html

[10]http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredAppMasterRest.html
[11]http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/HistoryServerRest.html
[12]http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/WebServicesIntro.html
[13]http://hadoop.apache.org/docs/current/api/index.html
[14]http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/UnixShellAPI.html
[15]http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/HttpAuthentication.html
[16]D.Eadline,Hadoop 2 Quick Start Guide,Crawfordsville,Indiana:Addison-Wesley,2016
[17]http://blog.qburst.com/2014/08/hadoop-big-data-analytics-tools/- by Prapha Rajan ,2014
[18] Hadoop Architecture Tutorial — Hadoop Tutorial For Beginners. (2016). Retrieved October 18, 2016, from https://www.youtube.com/watch?v=iIPItenMIOw
[19] An introduction to the Hadoop Distributed File System. (n.d.). Retrieved October 18, 2016, from https://www.ibm.com/developerworks/library/wa-introhdfs/#communications_protocols_sidebar
[20] HDFS Architecture Guide. (n.d.). Retrieved October 18, 2016, from http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[21] Multi-host SecondaryNameNode Configuration - Cloudera Engineering Blog. (2009). Retrieved October 18, 2016, from http://blog.cloudera.com/blog/2009/02/multi-host-secondarynamenode-configuration/
[22] [HADOOP-4539] Streaming Edits to a Backup Node. - ASF JIRA. (n.d.). Retrieved October 18, 2016, from https://issues.apache.org/jira/browse/HADOOP-4539
[23] HDFS Users Guide. (n.d.). Retrieved October 18, 2016, from https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[24] B. (n.d.). Apache Hadoop HDFS - Hortonworks. Retrieved October 18, 2016, from http://hortonworks.com/apache/hdfs/