EECS 4314: Hadoop Conceptual Architecture Report

Authors
Mina Zaki
Alexander Aolaritei
Dillon Bondarenko
Camillo John (CJ) D'Alimonte
Jeremi Boston
Daniel Nowak

October 19, 2016

Contents

1 Hadoop and Its Interacting Parts

1.1 What is Hadoop?

Hadoop is open source software that is designed to store and process data that is too large for one device or server. Hadoop delegates tasks across servers, allowing many devices to run concurrently. Massive amounts of data can be analysed since the tasks are split across many devices allowing jobs to be completed faster. Hadoop is broken into HDFS and MapReduce. However, in Hadoop 2.0, YARN was introduced as a way of improving parts and functionality of MapReduce.

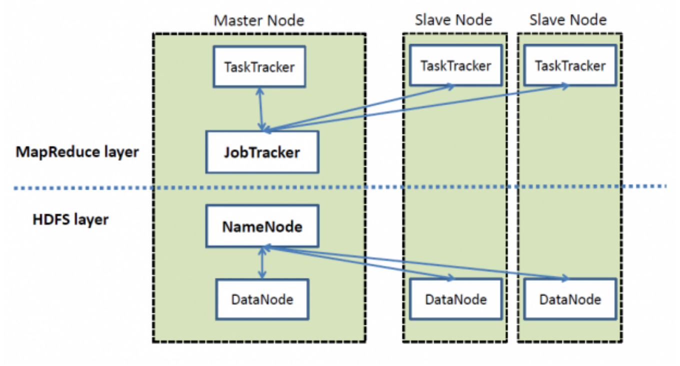

Figure 1: High Level Architecture

1.2 What are the Parts of Hadoop?

- *Hadoop Distributed File System* - HDFS is where Hadoop stores the data. It follows a master/slave architecture where the NameNode is the master and the remaining nodes are called DataNodes. The NameNode tracks the storage in the cluster. This allows Hadoop to record and organize where each piece of data is stored among the many devices in the system. The DataNodes are on the remaining devices in the system and they house the data stored in Hadoop. Each data file is replicated three times and stored on machines across the cluster, ensuring that if one machine is unavailable then the entire system will not go down. The final element is the client machine which has Hadoop installed. This is the main control panel where data is loaded into the cluster by first passing to the NameNode which distributes the data to the DataNodes. The client machine is also responsible for submitting MapReduce jobs and viewing the results of the MapReduce jobs.

- *MapReduce* - MapReduce is how Hadoop is able to process the large amounts of data stored in HDFS. The master in MapReduce is called the JobTracker and the remaining nodes are called TaskTrackers. The main strength of the Hadoop system is that it is able to divide one job into many small tasks that can each be executed independently on a machine. In this case, the machines with DataNodes also have the TaskTracker. Once a client requests a job to be completed, the JobTracker divides this job into many small tasks and then delegates the tasks among the cluster to the slave nodes. The slave nodes (or TaskTrackers) report back on their current progress to the JobTracker. One main principle of Hadoop is data locality. This means that whenever possible, Hadoop, instead of moving data which involves slow transfers on a network, will choose to run the program on the node that stores the data in HDFS. This allows processes to be run much quicker and it prevents you from having to move large amounts of data around aimlessly. This is one of the reasons that Hadoop replicates the data three times as to increase the chance of data locality, because it is less likely that all three machines will be unavailable at the same time.

- *Yet Another Resource Negotiator* - YARN was a very important feature in Hadoop 2.0 that greatly improved performance. YARN replaced the JobTracker and TaskTracker in MapReduce from Hadoop 1.0.

1.3 How do the Parts Interact?

When a job is run from a client machine it is sent to the JobTracker in MapReduce. The JobTracker then divides this job into many small tasks and assigns machines to perform the small tasks. It uses the NameNode in HDFS to decide which machines have the relevant data stored on them. The TaskTracker in the machines that are assigned a task receive the task from the JobTracker. Now the task is run on the data stored in that machines DataNode. Each TaskTracker continually updates its progress to the JobTracker. Finally when all of the tasks are completed, the result is displayed on the client machine.

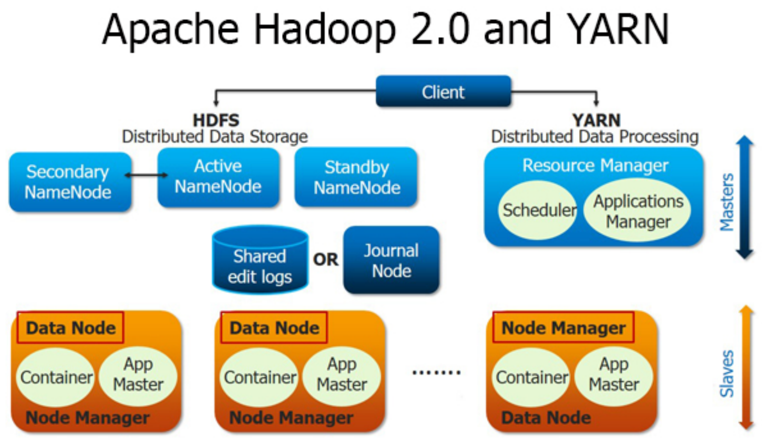

Figure 2: Component Interaction

2 Evolution of the System

Almost every project conceived goes through evolution in one way or another. Very rarely will there be a release of a particular project without some changes being made to its initial requirement specifications. Hadoop is no exception to this. The most obvious example of evolution that has shaped Hadoop can be observed through the update list and change logs on their website. Although the logs provide a clear indication of the evolution process Hadoop undertakes, it is not to say that evolution can not occur anywhere else as modifications can also occur during any stage of the product's development. Once Hadoop was released to the public, it underwent a vigorous testing and maintenance phase where users can submit bugs or general feedback to help improve the system to their changing requirements. These requests and modifications can be seen in their update list where they address bugs and add new features for their customers.

2.1 Hadoop Version 1.X

The most important feature of this line of release would be file security. In earlier release version 0.X, no authentication was performed on machines accessing Hadoop. All map tasks were run under a single account, which would allow access to all the other cluster's data. This is a much needed addition to the Hadoop feature list since it deals with Big Data while interacting with the internet. A disk-failure procedure was implemented as well. This works well on a multi-node system since it allows for the ability of one of the nodes to malfunction, while allowing for the nodes to compensate for the resource loss. The entire node is blacklisted from the TaskTracker meaning no new tasks will be given to it. Until the node is successfully repaired, it will stay on the blacklist. HBase provides real time read/write access to large data sets. This is imperative to the storing of multi-structured or sparse data. It serves as a very large lookup table that easily spots what the user's query was. Hadoop-client and hadoop-minicluster were introduced for ease of client installation and testing, respectively. It also allowed a "run-as-user" and "non-secure' mode with this."

2.2 Hadoop Version 2.X

This version of Hadoop is where YARN officially makes it into the roster of features. This is the core of Hadoop that allows multiple data processing engines. SQL, real-time streaming, and batch processing functionalities were now compatible given this modification. HDFS Federation makes its debut here as well; it improves on the HDFS structure. It separates namespace and storage, enables generic block storage layer, and multiple namespaces in clusters. The multiple name nodes used do not require coordination with each other, meaning data nodes can be linked to more than one name (multiple names point to the same data as well). The block pools here belong to their

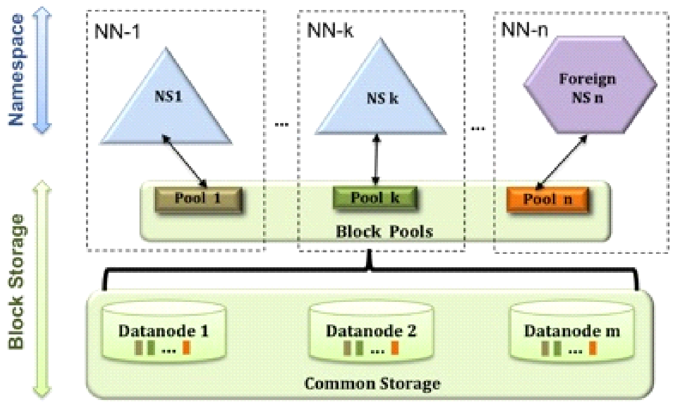

Figure 3: HDFS Federation

own namespace and the datanodes store these blocks for access by the pools. This allows all the namespaces to generate an ID for new blocks created which function independently of other block pools. Some of the other features introduced are as follows: multi-resource scheduling of both CPU and memory, HDFS snapshots which are useful for freezing a moment in time before one of the nodes malfunctions, and OfflineImageViewer which, as the name suggests, allows clients to view the data clusters without being connected via the internet.

2.3 Hadoop Version 3.X

Currently, there is an alpha release of version 3.0 which was released on the 3 September, 2016. Alpha phases come prior to the beta phase as the alpha phase guarantees no quality or reliability and are not intended for production use; they are intended as a prototype style used to gather as much information as possible to address bugs that need to be fixed for the stable release of this product. The intention is to always release early and release often to quickly iterate on feedback collected from downstream users. It is vital to note that even while these versions have already been released, work is still continued on versions 0.X, 1.X, and 2.X.

3 The Future of Hadoop

As Big Data continues to evolve in the coming future, developers will face unprecedented challenges on how to effectively manage larger quantities of data. Flexible data storage is the way of the future as the push to make everything convenient for the user continues to be emphasized. Competitors will continue to challenge Hadoop's future development. Google's own file system is a great example of this as they were the initial spark with the release of the Google File System and MapReduce, ultimately driving the creation of Hadoop. Now other companies have implemented their own file system, like Microsoft's own Bing. Hadoop had its grass roots as a search engine called Nutch. In the future, it might evolve to become a brand new way to index, store, sort, and even search through large data sets in a faster manner.

4 Data Flow & Control

The flow of data and control processes in Hadoop can be described by the following notion: **Moving Computation is Cheaper than Moving Data**. A computation requested by some application is more efficient if it is executed near the data it itself operates on. This notion is particularly true when the size of the data set is large. Network congestion is minimized and the throughput of the overall system increases. The migration of the computation closer in proximity to where the data is located rather than moving the data to where the application is running has advantageous consequences for the entire system. Hadoop consists of the following three components that support this notion:

- Yet Another Resource Negotiator - YARN
- Hadoop Distributed File System - HDFS
- MapReduce

4.1 YARN

As part of the architectural core of Hadoop, YARN allows multiple data processing and accessing engines (open-source or proprietary) such as interactive SQL, real-time streaming, data science and batch processing to simultaneously handle data sets stored in a single platform. The YARN infrastructure is the framework responsible for providing the computational resources (i.e. CPUs & Memory) needed for the execution of various applications. YARN's architecture follows the traditional *Master/Slave* model as the Resource Manager *(Master)* is responsible for scheduling resources appropriately for different tasks, checking the liveness of the node managers, and checking the status of the specific tasks while the Node Manager *(Slaves* - multiple per cluster) is responsible for monitoring resource usage and for launching the containers of each application. The capacity of such resources is determined by the amount of memory available for allocation. The Application Master has the responsibility of negotiating the appropriate allocation of resource containers from the scheduler and tracking and monitoring their progress. A Container is a fraction of the Node Manager capacity and it is used by the client for running a program or application. The dynamic allocation of the cluster resources improves utilization over traditional static MapReduce operations of previous Hadoop versions while the YARN Resource Manager focuses exclusively on scheduling as data center processing capabilities expand exponentially (clusters consisting of thousands of nodes managing petabytes of data).

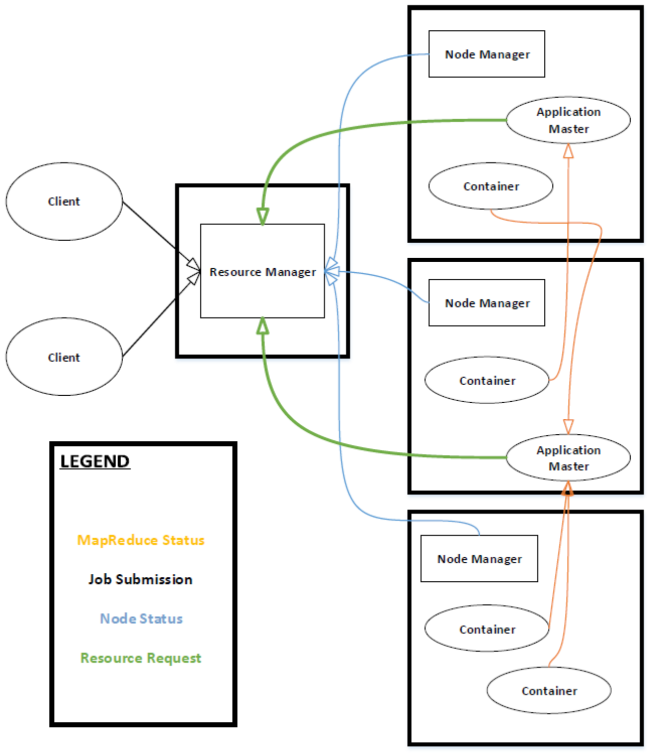

Figure 4: Event Sequence in YARN

4.2 HDFS

HDFS is the framework responsible for providing permanent and reliable distributed storage. Designed to host large volumes of data the HDFS consists of a single NameNode which maintains all the meta-data (permissions, modification and access times, namespace, and disk space quotas) of the filesystem while also storing the mapping arrangement of data blocks to nodes. The NameNode is the file-system's only point of failure and the system is failure tolerant as a result of the implementation of data block replication as the practice of representing particular data blocks in one or more nodes. Users access the filesystem using the Client and as in most conventional filesystems, the client supports the creation, writing, and deletion of files and directories. When a specific application reads a file, the Client will first ask the NameNode for a list of DataNodes that host replicas of the blocks of the file. The DataNode is contacted directly by the Client and the transfer of the desired block is requested. When the Client receives a writing request, it asks the NameNode to choose a DataNode to host replicas of the first block of the file. A pipeline is then created from node-to-node and the data is sent. Once the first block is full, the process continues through to the next block.

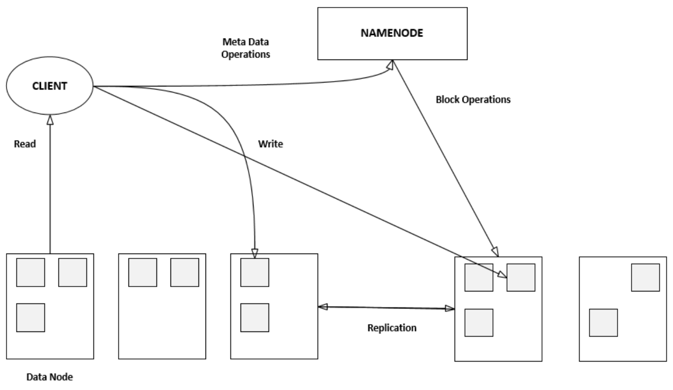

Figure 5: HDFS Architecture

4.3 MapReduce

Writing applications that process extensive amounts of data in-parallel can be achieved through the MapReduce software framework. The MapReduce framework operates on ⟨key, value⟩ pairs. The framework views the input to the job as a set of ⟨key, value⟩ pairs and produces a set of ⟨key, value⟩ pairs as the output of the job. A job in MapReduce splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The output of the maps is sorted and used as input for the reduce task. The MapReduce framework and the HDFS are running on the same set of compute and storage nodes. This allows for the effective scheduling of tasks on the nodes in which the data is already present thus resulting in the reduction of the IO bandwidth.

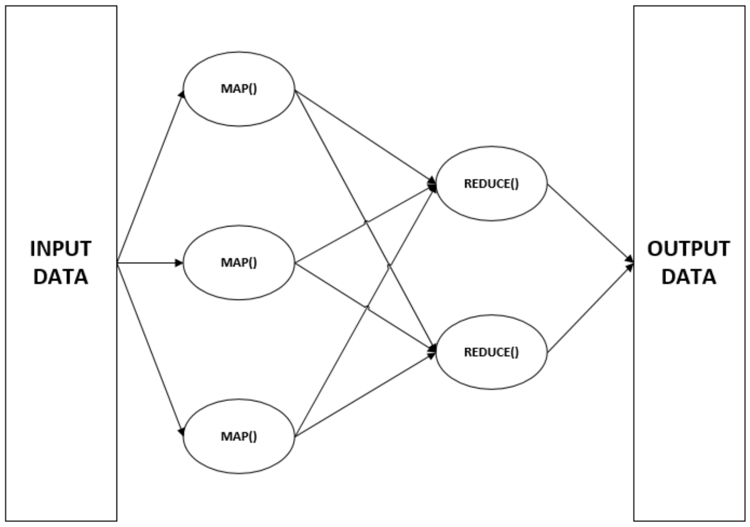

Figure 6: MapReduce Process

5 Concurrency

The strengths of Hadoop are linked to its efficiency dealing with large data sets and the ability to store large output data without needing a server room to process them. Attempting to do the following with commodity hardware makes efficiency and storage space a priority. This is where the YARN infrastructure comes into use, providing the master/slave architecture allocating the required resources. As previously mentioned, the Resource Manager allocates resources to the Node Managers, where each node manager with multiple containers having the ability to run different applications simultaneously. Using the ability to run multiple applications on separate clusters allows one to speed up the process of Map/Reduce tasks.

5.1 MapReduce Top Level

The MapReduce framework works in similar ways to YARN, in respects to the master and slave architecture. When a MapReduce application is commenced, there is an initialization of a single JobTracker which is the master that holds all the meta-data including the number of blocks or which rack the data is stored on, and the multiple TaskTracker slaves which hold the data to be processed. With this in mind, the JobTracker takes the submitted jobs from the client and assigns them to the TaskTrackers so that they may perform the map and reduce tasks. The timeline below, Figure 7, presents a simple sequence of a MapReduce execution. It is shown that the reduce phase in fact overlaps the Map phase by commencing prior to having mapped all the tasks. This is possible by having the NodeManager take an available container to start the reduction task and reducing the tasks which have already been mapped.

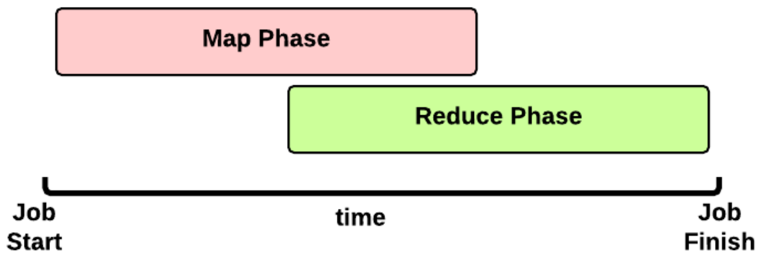

Figure 7: Simple MapReduce Timeline

5.2 Node Managers

Having mentioned the role of the NodeManagers, the focus now shifts to how it allocates the resources necessary to perform the required MapTasks. The example given in Figure 8 will be used to explain how the NodeManagers manage the MapTask jobs. Each NodeManager has a limited amount of resources to execute their specified amount of MapTasks which need to be completed. In this case we will say that each NodeManager has 2GB of RAM to allocate to the tasks with each MapTask requiring 1GB to execute. With these parameters, one NodeManager is able to run at most two MapTasks concurrently. Looking at the diagram it is shown that once a MapTask has finished, the NodeManager uses those freed resources to commence another MapTask. This process continues until all MapTasks have finished processing.

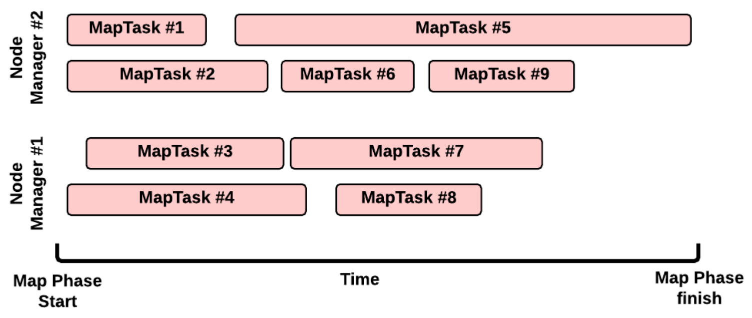

Figure 8: Map Phase MapTask Management

5.3 MapTask

Knowing how the NodeManagers allocate resources, the focus can shift to how MapTasks themselves execute. Here we have the typical initialization phase and the execution phase following consequently. The interesting part of the MapTask is how it manages the in-memory buffer. While the data is being processed, the output goes to the in-memory buffer. Depending on how much data is available, this output keeps filling the buffer and would normally come to a point where no more space is available. Referring to Figure 9, it shows that a spilling phase occurs while the execution phase is still in progress. When the in-memory buffer starts to run out of space, the spilling phase starts relieving the buffer by taking the data and moving it somewhere in the distributed file system. This allows the process to continue no matter how much data needs to be processed. The only problem to this method is that when execution finishes, the output is scattered around the distributed file system. Therefore, we use the shuffle phase to find all the processed data in the file system and bring it together to present the results.

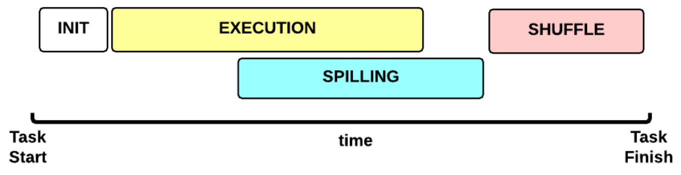

Figure 9: MapTask Sequence

6 Division of Responsibilities Among Participating Developers

Hadoop is a project of the larger corporation Apache Software Foundation. For which the company holds the trademark of the name Hadoop, and all of its code. Thus, Hadoop operates under principals set by Apache, called the "Apache Way". Each project of Apache has defined roles in which individuals may perform within the project.

To begin, we have users. These are participants who use the Hadoop software. This is where every developer starts their path towards becoming a contributor. As they use their user knowledge and perspective to further improve projects. Furthermore, users aid the project by giving feedback to developers, as well as report bugs and suggest future features.

Contributors are the participants in the Hadoop hierarchy. This group consists of all the volunteers who contribute documentation, code and time. Of these committers if one were to make a substantial contribution of any sort, it is possible for them to be invited to become a Committer. A contributor can only become a Committer, if the PMC passes a consensus approval. Committers are responsible for the technical management of the project. Furthermore, committers gain access to the subprojects subversion repositories. Another added responsibility is that Committers can cast votes with regards to any subprojects technical discussions. If a committer ceases to contribute to the project for a period of 6 months they are declared emeritus. To abolish this label, a Committer must ask the PMC for a consensus approval. Another role for the Committer, if they so choose, is the development of features that are a part of a speculative branch of the repository. The committers who contribute, are consistent contributors that are given access to contribute to the branch by the PMC. These committer's are labeled branch committers and are responsible for guiding their feature into a successful release. A final requirement is for Committers to sign a Contributor License Agreement with the Apache Software Foundation.

The second label a Committer can be given is a Release Manager. These are committers who volunteer with the goal of producing a Release candidate. One week before they intend to make a release candidate, they must state which branch they plan on releasing from. Their role in this situation is to construct a consensus surrounding the content of the particular Release Candidate. This is so a successful Product Release vote is obtained.

Finally we have the Project Management Committee (PMC). Members of this committee are committers who have made a substantial commitment to the project, and as a result have been invited to join the committee. For this invitation to be successful a consensus approval must be met. The committee responds to the ASF and their board on matters of the Hadoop codebase and management. The responsibilities for this committee are the following:

- Voting in new members of the PMC or committers
- Approving releases of the Hadoop project
- Maintaining the projects resources, these include the website, mailing list, and the codebase
- Maintaining the project guidelines and its bylaws

Finally they resolve the projects licensing disputes when they occur. The committee has a chair who is appointed by the ASF board. He reports to the board quarterly on matters of project management within the scope of Apache Hadoop. This position has a term of a year.

Within the scope of Hadoop, different decisions require different actions. They system in which decisions are made through the mailing list. Each committer is sent an email and their response determines their answer. There are four possible answers. A "+1" indicates a yes vote. A "-1" is a no vote. Furthermore, if a consensus is required, this vote is considered a veto which requires an explanation on why they disagree and which path they wish the project to follow. A "+0" is a vote that implies a willingness for the action to follow through, however the voters cannot contribute. Finally we have "-0" which indicates a disagreement with the action, however they do not care enough to impede it.

Different actions require different types of approvals. Each type of approval has a unique set of criteria's. There is Consensus approval, which requires there to be 3 Yes votes and no vetoes. A

Lazy consensus requires there to be no No votes. A lazy majority requires there to be 3 Yes votes and more yes votes than no votes. Finally a lazy 2/3 Majority requires there to me at minimum 3 yes votes and two times as many yes's as no's.

The following are actions taken in a Hadoop project and the corresponding required approvals: Product Release is when a product is ready for release the committee must vote on an official release date. Lazy majority is required among the members of the PMC. New Codebase is if the old codebase is to be replaced by a new one a Lazy 2/3 majority is required. However if the vote is not met, the old codebase is continued to be used. Change of code is when a committer makes a change to the codebase, which can be a change of the website, documentation and source code. For the code to initially be committed a single +1 is needed. There are also the actions of voting in new members. Voting in both a committee and a new PMC member requires a consensus approval. The voting in of a new branch committer requires a lazy 2/3 majority. Finally the committe can also vote on the removal of these member, and each require a lazy 2/3 majority.

7 Use Cases

7.1 Case #1

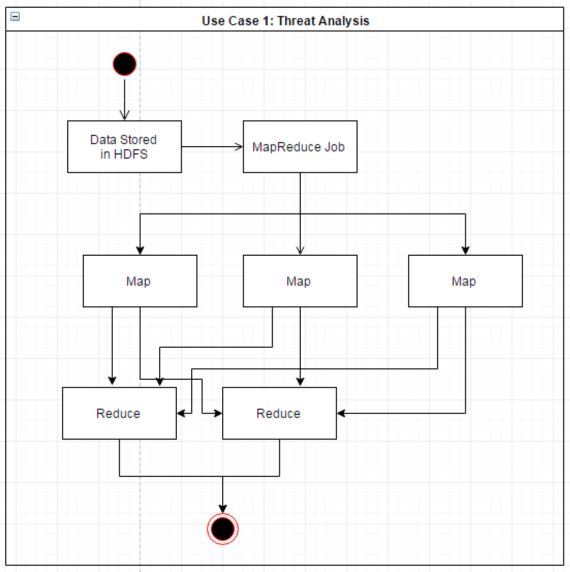

Figure 10: Case One

Online criminals write malicious viruses or malware. Computer users rely on software services to protect them from these viruses or malware. Thus, an antivirus software is created to protect users from such threats. The antivirus software computes a signature for each virus/malware it is aware of. As time passes, more and more signatures are added to the database. Over the past few decades the database grew enormously. The antivirus company must now compare possible threats to the huge database of malware signatures. This comparison check starts to take a considerable amount of time due to the large volume of signatures stored in the database.

This scenario can be solved with a solution implemented with Hadoop. The process can be observed in the state diagram shown in Figure 10. The first step towards the Hadoop solution requires the library of malware indexed by signatures to be stored within the Hadoop Distributed File System or otherwise known as HDFS. Afterwards, comparing a possible threat with the malware signatures stored in HDFS will be done by calling a MapReduce job. The first step in the MapReduce job is to begin the map stage. During this stage, the set of data stored in HDFS is converted to another set such that individual elements are broken down into key/value pairs or otherwise known as tuples. For this use case example, the library of malware types are mapped by being broken down to smaller components and searching for a malware signature match on that smaller set. After the map stage, the result is sent to the reduce stage which takes the output from the map job as input and combines those data tuples into a smaller set of tuples. So for this use case, the results from each mapper task come together and get reduced to a final result which would be a list of matching malware signatures.

7.2 Case #2

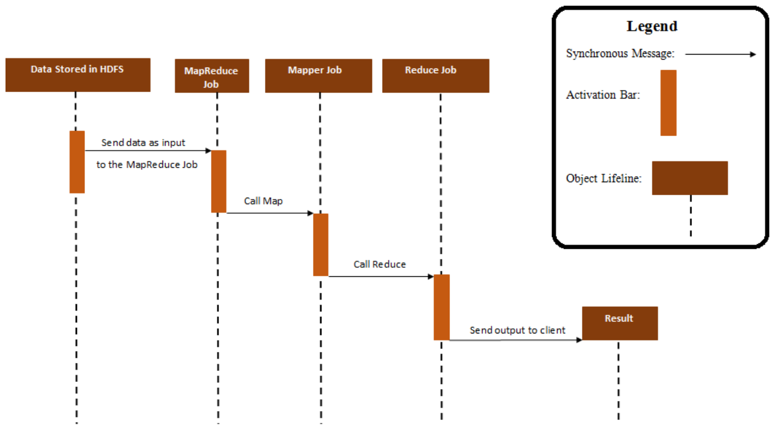

Figure 11: Case Two

An online commerce company wants to return relevant information to its users with its search engine. As time passes on, massive amounts of information is being added; making it much more difficult when processing the data. The search engine must be able to process files and extract keywords. Previous user activity and personal information must also be taken into account when delivering search results.

The problem in this scenario is that processing search queries is taking longer as time passes since more possible search results are being added to the database every day; making the database huge. This problem can be solved using a MapReduce Hadoop solution. The process can be observed in the sequence diagram shown in Figure 11. The first stage for the Hadoop solution is to store the massive amounts of data for search results and user information within the Hadoop Distributed File System (HDFS). The next stage requires a MapReduce job which will compare previous/current user information with the possible search results so that the most desirable results expected from the user are returned first. In order for this to take place, the MapReduce job must initiate its first task which is the mapping stage. During this stage, the groups of possible search results are mapped by being broken down into smaller components and searching for the best matched result by comparing previous user information to each mapped group. After this stage, the reduce stage begins which will take the output from each map job and reduce the results such that the most relevant results appear first.

8 Bibliography

"An Introduction to HDFS Federation - Hortonworks." Hortonworks An Introduction to HDFS Federation Comments. N.p., 23 Aug. 2011. Web. 11 Oct. 2016.

Apache Hadoop. "HDFS Architecture Guide." *HDFS Architecture Guide*. N.p., n.d. Web. 18 Oct. 2016.

"Apache Hadoop Project Bylaws." *Apache Hadoop Project Bylaws*. The Apache Software Foundation, n.d. Web. 18 Oct. 2016.

"Deploy an OpenStack Private Cloud to a Hadoop MapReduce Environment." *Hadoop MapReduce Environment*. IBM, 2014. Web. 18 Oct. 2016.

"CDH." *Introduction to Hadoop Security*. N.p., n.d. Web. 18 Oct. 2016.

Fvanvollenhoven Follow. "Hadoop, HDFS and MapReduce." *Hadoop, HDFS and MapReduce*. Dutch Java User Group (NLJUG) University Session, 27 June 2011. Web. 18 Oct. 2016.

Garment, Victoria. "Hadoop 101: The Most Important Terms, Explained." *Plotting Success*. N.p., 27 Mar. 2014. Web. 18 Oct. 2016.

"Hadoop Internals." *Anatomy of a MapReduce Job* ·. N.p., n.d. Web. 18 Oct. 2016.

Tutorialspoint.com. "Hadoop MapReduce." *Www.tutorialspoint.com*. N.p., n.d. Web. 18 Oct. 2016.

"Ten Common Hadoopable Problems." *Cloudera 10*. Cloudera 10 Hadoopable Problems., 2015. Web. 18 Oct. 2016.

"The Hadoop Distributed File System." *The Architecture of Open Source Applications:*. N.p., n.d. Web. 18 Oct. 2016.

"What Is MapReduce?" *IBM*. N.p., n.d. Web. 18 Oct. 2016.