Project Report

Performance Evaluation of Hadoop and Spark Platform for Data Intensive Applications

CS546 – Parallel and Distributed Processing

Instructor: Dr. Sun, Xian-He

2nd December 2016

Submitted By:

Aditya Jadhav

A20377887

Sumedha Gupta

A20377983

Abstract

The main objective of this project is to compare the performance of Hadoop and Spark on data intensive applications, such as for iterative and non-iterative computations. To demonstrate the differences between the two, the runtime architectures of both Spark and Hadoop will be compared. Also, the performance evaluation will be done based on the different parameters which are running time, memory usage, CPU usage and fault tolerance capability. In this project, we will highlight the performance comparison between Spark and Hadoop as the growth of data size. At the end, there will be conclusions based on our result to get an idea about the performance capability of both for different kind of operations.

Table of Contents

1. Introduction 5

   Overview of Hadoop and Spark 5

2. Objective and Motivation 7
3. Problem Statement 8
4. Important Existing Solutions 8
5. Description of our solution 9

   Failed Approach 9

   Our Approach 9

   Algorithm Description and why we have chosen these two algorithms 10

6. Performance Evaluation 10

   Cluster Architecture 11

   Hardware and Software Configuration on each node 11

   Datasets for Algorithms 11

7. Evaluation of Results 12

   Results & Analysis 13

8. Conclusion & Future Work 17
9. Split Up of Work among Team Members 18
10. References 18

List of Figures

Figure 1Hadoop Components 6
Figure 2 Spark Architecture 7
Figure 3 Cluster Architecture 11
Figure 4 Word Count Running Time 13
Figure 5 Page Rank Running Time 14
Figure 6 CPU & Memory Usages 15
Figure 7 Verification of Fault Injection 16
Figure 8 Fault Tolerance Running Time 16

1. Introduction

As the data is increasing faster than the processing speeds, there is an increasing need of data processing and analysis in the Information industry. The process of exploration and analysis of large amount of data fulfill this need by automatic and semi-automatic means. Earlier, we used to use single system with few processors and there was a restriction on the speed of the processing as well as on the size of the data that can be processed at a time. The only solution is to parallelize on large clusters. To implement this kind of processing variety of cluster computing frameworks have been proposed to support large-scale data-intensive applications.

Two of the major frameworks that are widely used in both enterprises and web industries are Hadoop and Spark. Apache Hadoop provides an open source implementation of MapReduce [1]. MapReduce, introduced by Google is one such successful framework for processing large data sets in a scalable, reliable and fault-tolerant manner. Spark is a MapReduce-like cluster-computing framework [4]. Both are open source framework, which can be used to process the huge amount of data in parallel. In this project, we attempt to do a performance evaluation between Hadoop and spark.

Overview of Hadoop and Spark

Apache Hadoop is a framework for the distributed processing of big data across clusters of computers using MapReduce programming data model. In the past few years, Hadoop has earned a reputation as one of the widely used big data analytics engine. It has proven to be very helpful in storing and managing vast amounts of data economically and efficiently.

It has two main parts – a data processing framework (MapReduce) and a distributed file system (HDFS) for data storage in a reliable and fault tolerant manner [7].

Hadoop runs applications using the MapReduce where the data is processed in parallel on different CPU nodes.

MapReduce **(Processing Engine)**	**Hadoop Utilities**
Cluster Resource Management (YARN)	
Hadoop Distributed File System (HDFS) **(Data Storage Engine)**	

Figure 1 Hadoop Components

MapReduce is the processing engine of Hadoop, which provides support for parallel computing.

It consists of two stages: Map Stage and Reduce Stage. The MapReduce framework operates exclusively on <key, value> pairs, that is, 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.

HDFS is the data storage engine of Hadoop. The data in a Hadoop cluster is broken down into smaller pieces (called blocks) and distributed throughout the cluster. In this way, the map and reduce functions can be executed on smaller subsets of your larger data sets, and this provides the scalability that is needed for big data processing.

YARN is a cluster resource management framework in Hadoop [2]. YARN includes two key daemons: a resource manager to schedule jobs and tasks or allocate resources across the cluster, and node managers to start and monitor containers.

Hadoop features: reliability, scalability, and fault tolerance.

But after the release of Apache Spark, there has been rapid adoption by enterprises across wide range of industry. Spark has designed to run on top of Hadoop and it is an alternative to the traditional batch map/reduce model that can be used for real-time stream data processing and fast interactive queries.

Spark is a cluster computing framework and an engine for large-scale data processing. It constructs distributed collections of objects, resilient distributed datasets (RDDs) in memory, and then performs a variety of operations in parallel on these datasets.

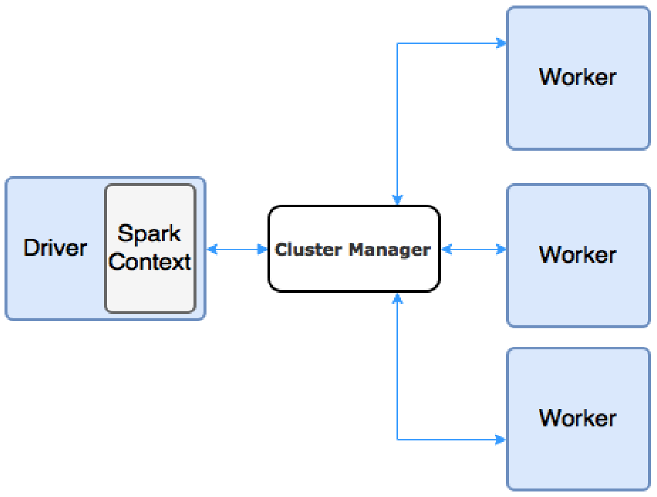


Figure 2 Spark Architecture

Spark is an alternative to the MapReduce framework. It is mainly used for real-time data stream processing or applied to iterative algorithms. RDDs is a distributed memory abstraction, and each RDD is a read-only, partitioned collection of elements across the cluster that can be operated on in parallel.

2. Objective and Motivation

The objective of this project is to discuss the comparison of - Hadoop Map Reduce and the later introduced Apache Spark – both of which provide a processing model for analyzing big data. Although both of these options are based on the concept of Big Data, their performance varies significantly based on the use case under implementation. This is what makes these two options worthy of analysis with respect to their variability and variety in the dynamic field of Big Data.

The motivation behind the project is the big 'Big data' question: *Spark or Hadoop -- Which Is The Best Big Data Framework?*

In this project, we compare these two frameworks along with providing the performance analysis by implementing parallel applications on both and analyzing their running times, memory & CPU usage, fault tolerance and scalability.

3. Problem Statement

Hadoop is a very popular general-purpose framework for many different classes of data-intensive applications. However, it is not good for iterative operations because of the cost paid for the data reloading from disk for each iteration. Spark, which is designed to have a global cache mechanism, can achieve better performance in response time for iterative process. Therefore, performance in terms of time has been evaluated for Spark over Hadoop, but the other factors such as fault tolerance, elasticity, memory consumption and other system performance criteria were not deeply analyzed in the past.

4. Important Existing Solutions

There are many performance evaluations that have been done between Hadoop and Spark in terms of their running time. The existing solutions clearly imply that the running time of both the frameworks Hadoop and Spark increases, as there is an increase in the size of input or datasets.

We also found some solutions where the memory and CPU usage have been evaluated between the two. There has been a tremendous performance advantage shown by Spark when it comes to CPU usage. On the other hand, Hadoop found to be less efficient when it comes to efficiently utilization of the resources provided by the processing unit. But when it comes to the memory usage, Spark has been proved to be less efficient as it is in memory computing framework and all the computations that it performs are stored in the memory of the computer rather than storing them on disk. Hadoop on the other hand utilizes less memory as compared to Spark due its coordination with the Hadoop Distributed File System. However, there has not been a deep analysis of their performance based on the fault tolerance capability and the impact on their performance whenever a fault occurs.

5. Description of our solution

To fulfill our objective and to have better results than existing solutions and for the performance evaluation to be more efficient we have chosen two algorithms. We researched a lot about which algorithm should we go for, as there are varieties of algorithms we can use to compare the performance. So, we decided to go for one iterative and one non-iterative algorithm. The reason behind that is to evaluate the results for two different types of algorithm. Spark is good for iterative algorithms as it has cache mechanism so we tried to compare it with Hadoop and to find out the difference between their performances and to depict some results on the speed up that Spark gains over Hadoop. On the other hand, we have chosen a non-iterative algorithm and check, which one performs better in that case for all the parameters that we have set for the evaluation criteria.

Failed Approach

First, we tried to perform the evaluation using Oracle VM Virtual Box, we created a virtual machine and configured it with the required parameters and settings to act as a cluster node. This referenced virtual machine was then cloned, as many times as there will be nodes in the Hadoop and Spark cluster. Only a limited set of changes were then needed to finalize the node to be operational (only the hostname and IP address need to be defined). But we were not getting the expected results for the running time and also it would not be that accurate so we decided to deploy on some other platform.

Our Approach

We started with the configuration of Hadoop and Spark on Amazon EC2 platform. Then we ran different parallel algorithms with same data set with dynamic allocation and removal of nodes to do the p*erformance evaluation* in terms of running time, memory & CPU usage and fault tolerance on both the above architecture.

We have chosen two different algorithms for evaluation in this project. The datasets of each algorithm running on Hadoop and Spark are based on the same algorithm. We have run algorithms with the same configurations of hardware and the default settings of Hadoop and Spark; a resulting comparison on performance should be feasible and reasonable.

To evaluate the running time and memory & CPU usage was not that challenging as compared to evaluating the fault tolerance capability as the data processing and analytics applications are typically composed of multiple stages each of which requires different resource allocation. So, efficient utilization of resources would require the ability to dynamically scale the deployment while the job is in progress. To check the extent of this ability we will perform dynamic allocation and de-allocation of nodes to both the architecture and acknowledge their response.

Algorithm Description and why we have chosen these two algorithms

First, we are running a *non-iterative algorithm* “Word Count” on both Hadoop and Spark with same datasets and same size of input and analyzing the performance. The algorithm takes a file as an input and splits one line at a time into words and outputs key-value pairs and then pass these to the reduce function which sums up the counts for each unique word and writes the outputs on HDFS.

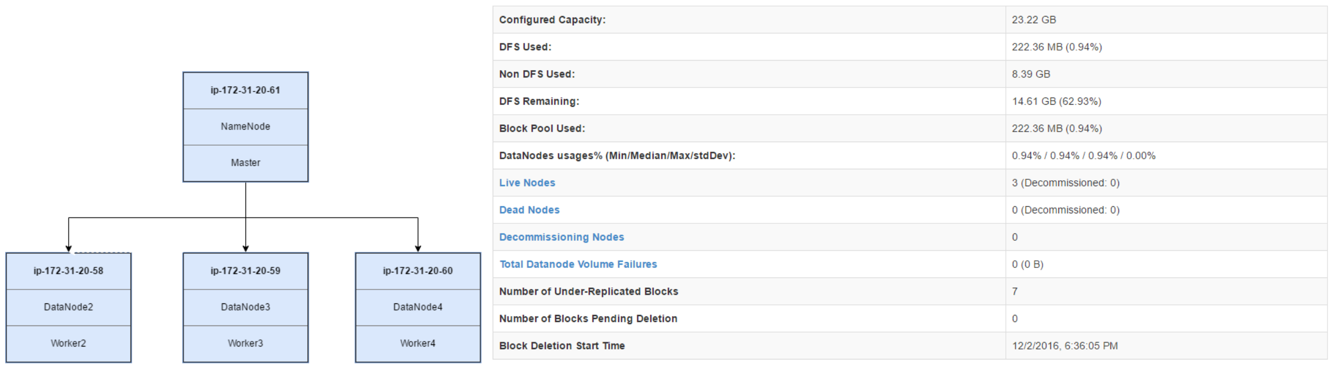
Second, we are running an *iterative algorithm* “Page Rank” on both Hadoop and Spark and analyzing the performance. We have chosen this algorithm as the implementation of this involves multiple iterations. We have taken a common Page Rank algorithm as the main focus of this project to evaluate the performance. This algorithm basically provides the quality ranking of each page to facilitate a better searching process and this is the way Google search engine works.

6. Performance Evaluation

To compare the performance of Hadoop & Spark a cluster with four instances is used where Hadoop & Spark are installed and deployed. Three experiments have been conducted using the same algorithm and programming language on this cluster. The running times and memory & CPU usage are plotted using graphs to manifest the performance difference.

Cluster Architecture

In this project, both Hadoop and Spark are deployed on four machines as shown in the figure below. In this architecture, the master acts as namenode and driver process for Hadoop and Spark respectively and the slaves acts as datanode and workers for Hadoop and Spark respectively.



Configured Capacity:	23.22 GB
DFS Used:	222.36 MB (0.94%)
Non DFS Used:	8.39 GB
DFS Remaining:	14.61 GB (62.93%)
Block Pool Used:	222.36 MB (0.94%)
DataNodes usages% (Min/Median/Max/stdDev):	0.94% / 0.94% / 0.94% / 0.00%
Live Nodes	3 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	7
Number of Blocks Pending Deletion	0
Block Deletion Start Time	12/2/2016, 6:36:05 PM

Figure 3 Cluster Architecture

Hardware and Software Configuration on each node

Hardware Configuration		
Host IP Address	CPU	Storage
172.31.20.61 (Master)	2 vCPUs, 2.4 GHz, Intel Xeon Family 8 GB RAM	8 GB SSD
172.31.20.58 (Slave 1)		
172.31.20.59 (Slave 2)		
172.31.20.60 (Slave 3)		
Software Configuration		
Operating System		Ubuntu 14.04 LTE, 64-bit
Apache Hadoop		2.7.3
Apache Spark		2.0.2

Datasets for Algorithms

We have chosen the following datasets, which contain random words and the output will be the frequency of each word. The following table shows the data size used for the evaluation for the first algorithm, which is word count.

Dataset Name	Dataset Size
Wordcount_dataset1.txt	35MB

Wordcount_dataset2.txt	50MB
Wordcount_dataset3.txt	100MB
Wordcount_dataset4.txt	210MB
Wordcount_dataset5.txt	500MB
Wordcount_dataset6.txt	1000MB

Fig. Datasets for Word Count Algorithm

The datasets for the Page Rank are taken from http://snap.stanford.edu/data/ to evaluate the performance of Hadoop and Spark for the Page Rank algorithm. The following table gives the datasets description:

Dataset Name	Dataset Size	Nodes	Edges
wiki-Vote	1 MB	7,115	103,689
web-Google	73.6 MB	875,713	5,105,039
cit-Patents	267 MB	3,774,768	16,518,948

Fig. Datasets for Page Rank Algorithm

7. Evaluation of Results

The following comprises the evaluation of this project:

1. Comparison of utilization of resources and performance in terms of execution time in seconds, CPU & memory usage and fault tolerance between Hadoop and Spark.
2. To ensure a fair comparison, we have taken care of the following things:
    - ❖ Hadoop and Spark run on the same cluster and uses HDFS as the file storage system;
    - ❖ The implementation in Hadoop and Spark are based on the same programing language and algorithm;
3. The results for both the algorithms are taken as an average of the results that we got after performing the process several times.

Results & Analysis

Result for Running Time of Word Count for Hadoop and Spark

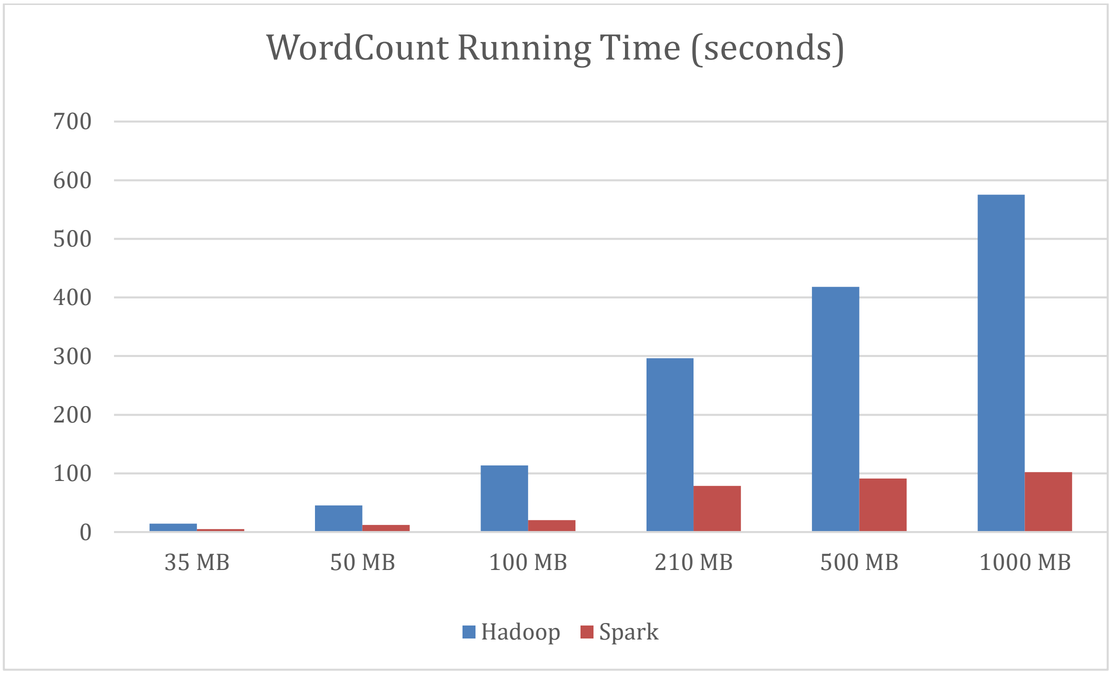


Figure 4 Word Count Running Time

Analysis

In the case of Word Count, we have taken the datasets that we have listed earlier and observed that for a small data set there is a not much difference between Spark and Hadoop because the data is processed in the local node no matter whether it is in Hadoop or in Spark. But as the size of the dataset increases there is a better performance of Spark than Hadoop. This is because Spark supports in- memory computation whereas Hadoop MapReduce writes back to the HDFS after a map or reduce action.

Result for Running Time of Page Rank for Hadoop and Spark

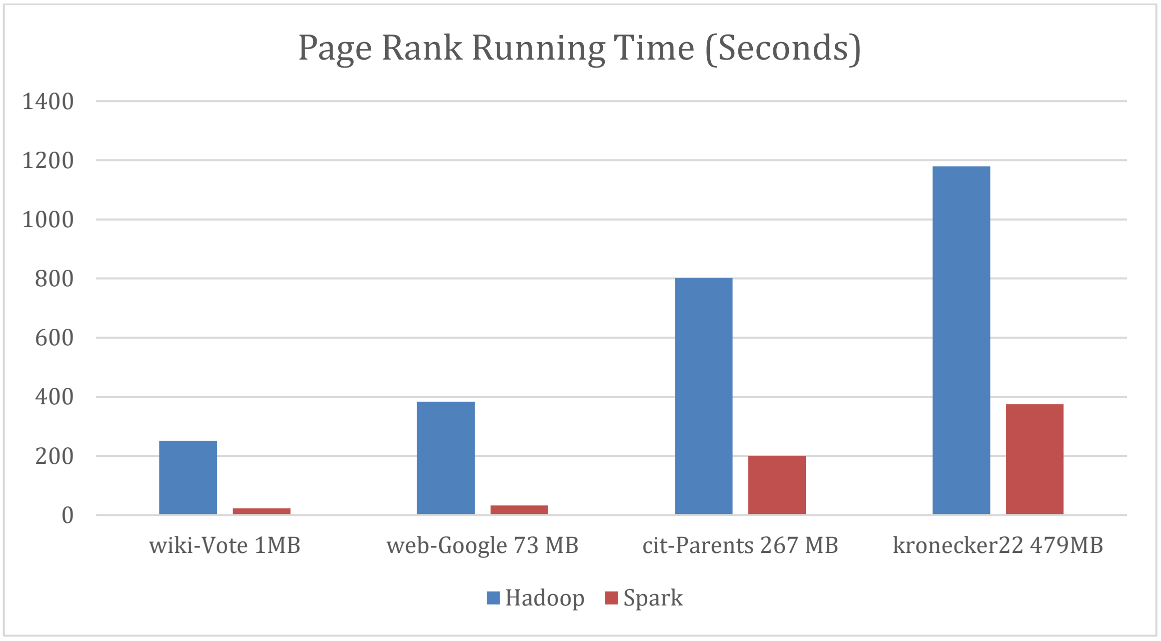


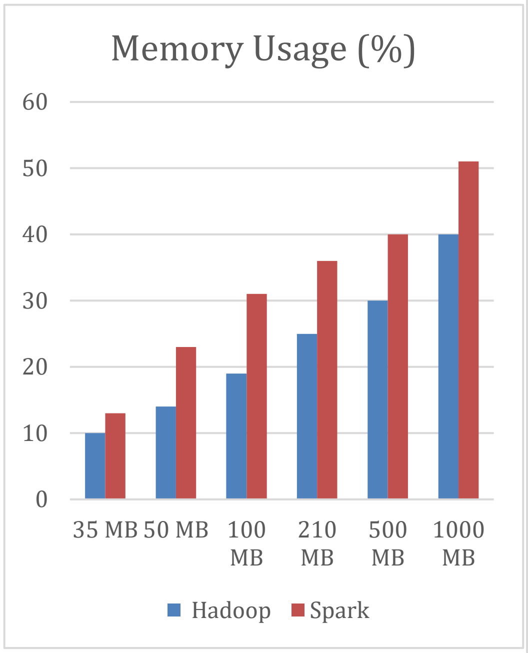
Figure 5 Page Rank Running Time

Analysis

Spark outperforms Hadoop in this case as Spark utilizes memory-based storage but MapReduce in Hadoop processes disk-based operations. For fewer iterations, there is not much performance difference but as the growth of data size and number of iterations are executed, there is a significant performance improvement of Spark over Hadoop.

Memory & CPU usage

The results for the memory and CPU usage are captured using a distributed monitoring tool called *Ganglia.*



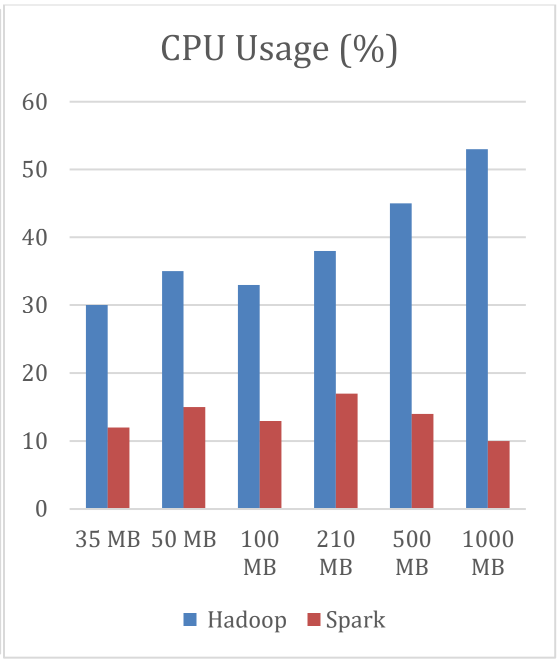


Figure 6 CPU & Memory Usages

Analysis

For running the algorithms both Hadoop and Spark requires high memory consumption as shown but for Spark it is large as it performs in-memory computation.

The CPU usage on the other hand is better in case of Spark and it has an outstanding performance on saving CPU resource.

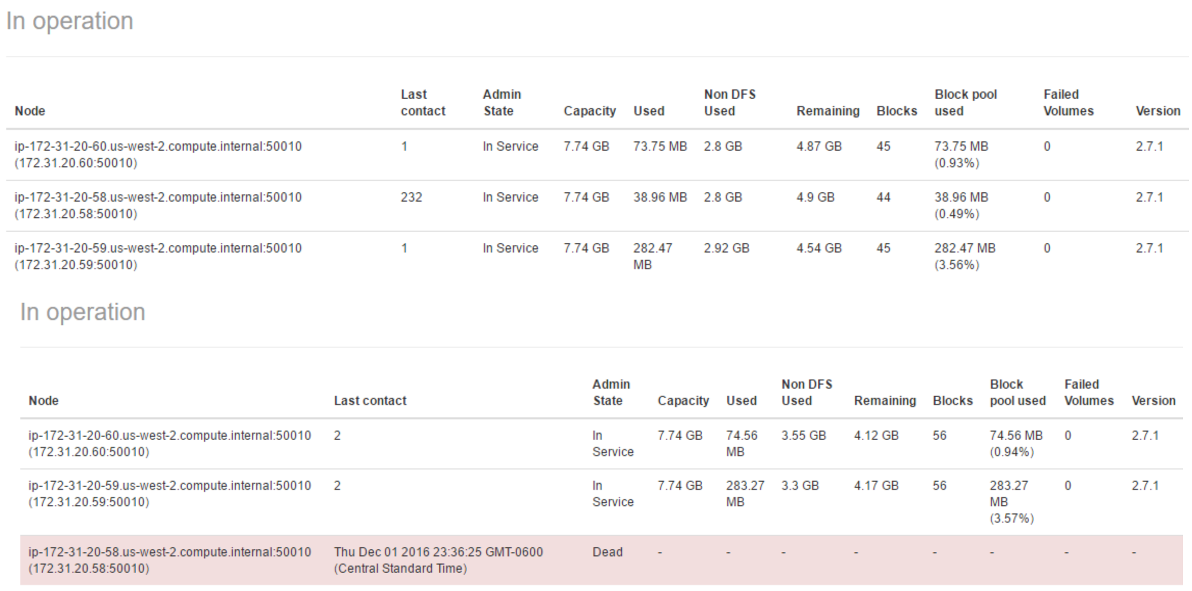
Fault Tolerance Capability

We got the feedback to include more on fault tolerance and how we injected the fault. We have performed the evaluation again, rechecked our results and validated our fault injection script. So, we have tested this parameter of performance by injecting a fault deliberately. Also, we have taken replication factor of 3 for HDFS.

Fault Injection is done using the following shell script:

```
for i in `cat nodes`; do
        echo $i;
        ssh  $i 'jps | grep DataNode | cut -d" " -f1 | xargs -rn1 kill'
done
```

This script takes an input file "nodes" which contains the IP- address where you want to inject the fault and kill the datanode process. We have injected a fault while the algorithm is running and captured the following results by killing one datanode process. We have ensured that the fault has been injected by checking the "last contact" of the node with the master using the monitoring tool as shown in the below figure. The last contact of the node process keep increasing and eventually it becomes dead.



In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-172-31-20-60.us-west-2.compute.internal:50010 (172.31.20.60:50010)	1	In Service	7.74 GB	73.75 MB	2.8 GB	4.87 GB	45	73.75 MB (0.93%)	0	2.7.1
ip-172-31-20-58.us-west-2.compute.internal:50010 (172.31.20.58:50010)	232	In Service	7.74 GB	38.96 MB	2.8 GB	4.9 GB	44	38.96 MB (0.49%)	0	2.7.1
ip-172-31-20-59.us-west-2.compute.internal:50010 (172.31.20.59:50010)	1	In Service	7.74 GB	282.47 MB	2.92 GB	4.54 GB	45	282.47 MB (3.56%)	0	2.7.1

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-172-31-20-60.us-west-2.compute.internal:50010 (172.31.20.60:50010)	2	In Service	7.74 GB	74.56 MB	3.55 GB	4.12 GB	56	74.56 MB (0.94%)	0	2.7.1
ip-172-31-20-59.us-west-2.compute.internal:50010 (172.31.20.59:50010)	2	In Service	7.74 GB	283.27 MB	3.3 GB	4.17 GB	56	283.27 MB (3.57%)	0	2.7.1
ip-172-31-20-58.us-west-2.compute.internal:50010 (172.31.20.58:50010)	Thu Dec 01 2016 23:36:25 GMT-0600 (Central Standard Time)	Dead	-	-	-	-	-	-	-	-

Figure 7 Verification of Fault Injection

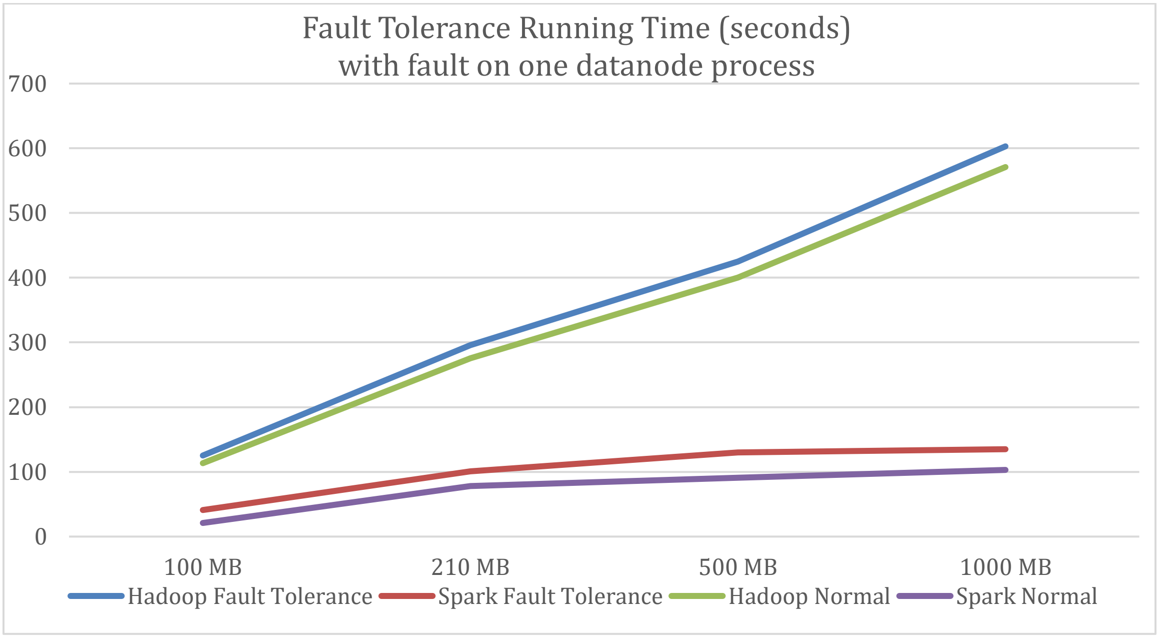


Figure 8 Fault Tolerance Running Time

Analysis

Both Hadoop and Spark are known to be fault tolerant. HDFS achieves fault tolerance mechanism by replication process while spark uses Resilient Distributed Datasets, which are fault tolerant collections. We found that when we are injecting a fault and killed one datanode process, the time to recover for Hadoop is slightly less than that of Spark. So, in this case Hadoop is better than Spark.

8. Conclusion & Future Work

In this term project, we have gained familiarity with Hadoop and Spark. Spark is not going to replace Hadoop as it does not have its own distributed file system and has to rely on HDFS or some other storage solution and also it is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries. We have drawn the following conclusions:

- Spark is much faster than Hadoop as it is an in-memory processing engine especially for iterative algorithms but if the data is too large to fit entirely in to the memory then there could be performance degradation for Spark.
- Spark has more memory utilization as it does the computations in synchronization with the memory whereas Hadoop does it with HDFS. On the contrary, Spark has much less CPU resource utilization than Hadoop.
- Hadoop is slightly more tolerant to failures than Spark. As MapReduce relies on HDFS and if a process fails in the middle of execution it could continue where it left off. On the other hand, Spark will have to process from the beginning so it takes a little more time to recover.

We have taken a small cluster but in future we could scale it up to more number of nodes. Also, we can evaluate the performance on other algorithms as well with more large datasets.

9. Split Up of Work among Team Members

Assignee	Task
Aditya	Configuration of Hadoop as a Stand-Alone Mode Configuration of Experimental Cluster on Amazon EC2. Configuration of Ganglia (Distributed Monitoring Tool) Installing Hadoop on the cluster. Running Algorithms and taking results on Hadoop Shell Script to inject Fault
Sumedha	Configuration of Spark as a Stand-Alone Mode. Installing Spark on the cluster. Running Algorithms and taking results on Spark Performance Comparison between Hadoop and Spark Project Documentation

10. References

[1] Jeffrey Dean, Sanjay Ghemawat "MapReduce: simplified data processing on large clusters" in Communications of the ACM.

[2] Hadoop Tutorial https://developer.yahoo.com/hadoop/tutorial/

[3] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments," in Proceedings of ICAC 2011.

[4] Apache Spark http://spark.apache.org

[5] Spark Configuration http://blog.insightdatalabs.com/spark-cluster-step-by-step/

[6] Shvachko K., Hairong Kuang, Radia S, Chansler, R The Hadoop Distributed File System Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium

[7] Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, 201

[8] Memory or Time: Performance Evaluation on Hadoop and Spark by Lei Gu. Huan Li

[9] Hadoop Configuration https://blog.insightdatascience.com/spinning-up-a-free-hadoop-cluster-step-by-step-c406d56bae42