Yelp Restaurant Recommendation System

Team 18
Deepak Patil dpatil@ncsu.edu
Harika Malapaka hsmalapa@ncsu.edu
Uddhav Bhosle ubhosle@ncsu.edu

Data Intensive Computing
Project Period (dates) : September - December (2018)

Abstract

Businesses like Yelp which are public facing need their applications and operations to be done in real-time. The project's aim was to develop a distributed computing system for the ever growing business data and to deliver results to the users on a real-time basis. For this problem, the data was distributed on a Hadoop cluster in the HDFS store. On top of this cluster a Spark cluster was set up using YARN as a resource manager. This allowed to compute data operations like aggregation, machine learning in a faster way by parallelizing and distributing the operations. The data was also stored reliably to prevent any losses due to failure and availability at all times. One of the decisions in creating this system was to scale allow the Spark cluster to scale along with the Hadoop cluster. To do this the Spark cluster was set up on the Hadoop cluster and every slave machine had a Hadoop Datanode and a Spark worker. This system enables users to get real time results for their queries along with providing the business with a fast way of learning patterns from the data.

1 Introduction

There has been an exponential growth in data in the recent years. Storing this big data and processing it is a challenge for this generation computer and software engineers. Distributed storage and computing provides a solution for this challenge. Here, software system are shared among multiple computers to improve efficiency and performance.

In this project, a Hadoop cluster was set up. As in Google File System's initial paper, there was one Master node that served the other slaves. In Hadoop, terminology the master node is called the 'Namenode' and the slaves are called 'Datanodes'. There was a YARN resource manager running on top of the Hadoop cluster which enabled to run a Spark cluster alongside the Hadoop nodes on the same machine.

In this paper, a system for real-time aggregation and machine learning is proposed. This system is based on the data from Yelp. It shows how big data can be stored efficiently and accessed in a fast manner. Along with that fast and reliable computations on such data using Spark are proposed.

The data storage in done in the Hadoop File System(HDFS). For replication, the idea of the magic number '3' was used in this system too. There were 3 replicas of all the data in the HDFS. This data was distributed by the Hadoop cluster in the datanodes hence allowing a much larger storage space along with redundancy, availability and disaster recovery.

1.1 Why Spark?

The Spark and Hadoop master reside on the same machine. Spark controls the workers by distributing the computing load on them and aggregating their results. The data objects in Spark are called Resilient Distributed Datasets or simply RDDs. These RDDs are in-memory and hence faster than other disk based computing frameworks.

2 Background and Previous Works

Doug Cutting, who was working in Google labs, realized the need for parallel computations and came up with an idea/algorithm called MapReduce. Then, he created an open source project by name 'Hadoop' on which he can run the MapReduce algorithms. This paper especially emphazises on the use of commodity hardware to get the expected results on a large scale. This developed the idea of parallel computing for Big Data at an industrial scale.

There are many works before which worked on the similar idea of computing the data for predicting using a distributed framework. One of them had used both MapReduce and Spark for comparison. In this work [1], a good comparative analysis was done. Reduced CPU, disk overheads due to RDD caching and faster computations were observed in Spark.

In another similar work that talks about Spark computations [2], it concentrated on 3 types of Spark deployment. They included : Spark standalone, on Hadoop-Yarn, and Spark - in MapReduce. In this work as well, HDFS was used as it's a stable storage. But it even did a comparative analysis of how it would be if data is stored in local memory where data sharing is easy.

3 Proposed Method / Claim

Make use of existing data intensive computing techniques to predict and analyze important artifacts that can be used in business. A prediction made now with existing data would be a good decision to analyze future trends and take further steps. This project is concentrated on a small domain - distributed data and operations in recommendation systems.

Whether to use a Spark Cluster directly or to deploy it on top of Hadoop cluster is interesting point of argument. Since the data is stored in the HDFS, it made sense to run the Spark cluster on top of the Hadoop cluster. HDFS provided redundancy and distribution along with the data being available to the Spark worker processes that were running on the same machines where the data was stored.

Since the project is concentrated on finding predictions, it would require machine learning algorithms. All the machine learning algorithms work in an iterative way and Spark would be a better option when compared to MapReduce considering that data is static. Secondly, various user queries would be fired interactively and continuously to this system. So Spark would again get a better mark when compared to MapReduce. Also the Spark cluster will be set up on top of the YARN resource manager. The Spark workers which run on the same nodes as the Hadoop datanodes can use the HDFS resources efficiently.

The project claims

- to solve the issue of limited computation resources(memory,storage) on stand alone systems and also in improving the latencies that are observed in computations when the size of the dataset grows to large quantities.
- To answer to user queries about a restaurant like, what are it's reviews or which restaurant has a better rating in a specific state/city.
- To predict the success of a restaurant which might open in future using the existing data.

4 Plan and Experiment

4.1 Data

The dataset used for this project was Yelp Business Data from kaggle.

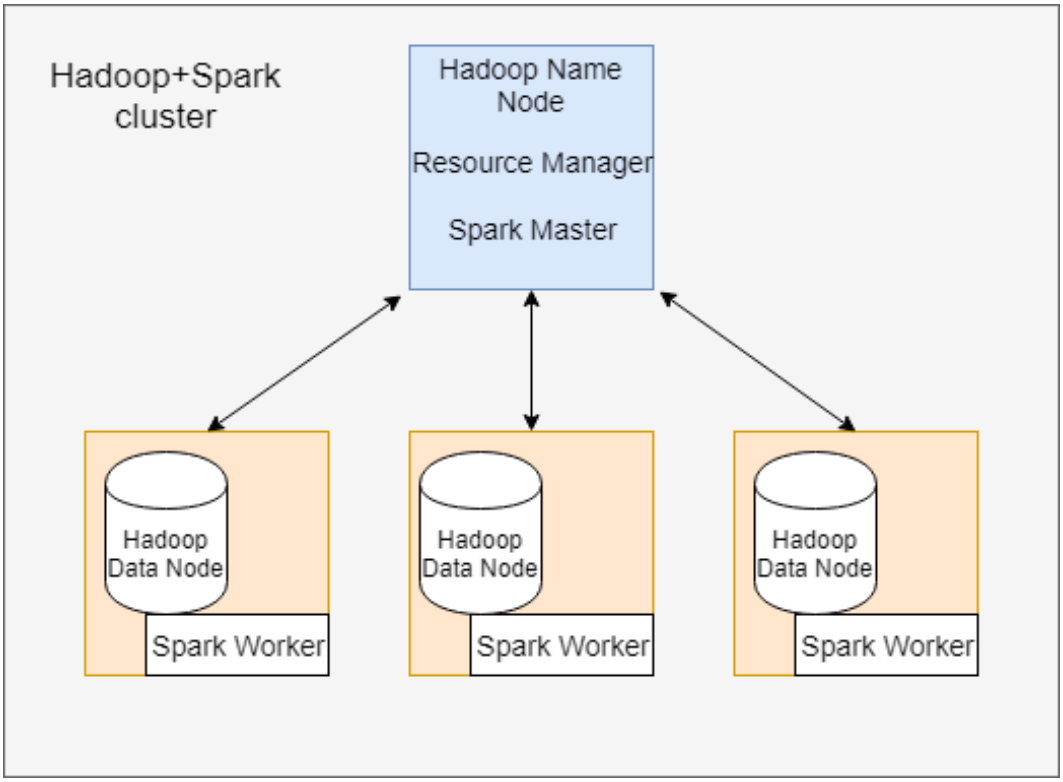

Figure 1: System Overview

Kaggle Yelp Dataset Link

Dataset size (combining) all csv files was approximately 2.5 GB (compressed version, uncompressed size approx is 4.8 GB). In order to match Big Data size, the data is replicated in Python using randomized data duplication. Doing this, the data size is taken upto around 100GB. The system was initially setup and tested using the original data of size of 4.8 GB.

This dataset has 7 files. They are all related to different kinds of attributes including restaurants in 11 metropolitan cities in the USA, user reviews etc. A restaurant would have many attributes like it's rating, stars, cuisine, address and so on.

4.2 Setup

Infrastructure

For this project experiments were carried out with 3 cloud environment and decision was made to work on 2 of them which are AWS and Digital Ocean. The following sections show the details of the entire setup.

Operating System:

As all the enterprise servers run Linux, hence ubuntu-16.04 LTS (a linux variant) is used to install and setup entire infrastructure. All the EC2 instances as well as the Digital Ocean droplets were running same ubuntu-16.04 version. All the basic necessary packages like ssh, iptables were either available or installed on all the nodes.

Programming Language

Python-2.7 was used as the major programming language for the project with the Spark library PySpark-2.4.0. PySpark allows user to interface with Resilient Distributed data in Apache Spark and Python. The prepossessing on the data was done using the R programming language.

Cluster Details

The ecosystem was setup on two different cloud providers. One on AWS and other on Digital Ocean.

On AWS, 4 EC2 instances were created. As one can see in the diagram of AWS console (Figure 2), the top two nodes are Name/master and Secondary Master nodes. Whereas later two are Data/Slaver nodes. Figure 3 shows all the services that are running on EC2 after hadoop, spark services were started on every node.

On Digital Ocean, 4 droplets (VMs) were created, 1 as Master node and 3 are data/slave node.

Both the clusters have similar configurations have similar configurations with small change. EC2 has secondory Namenode while Digital ocean does not.

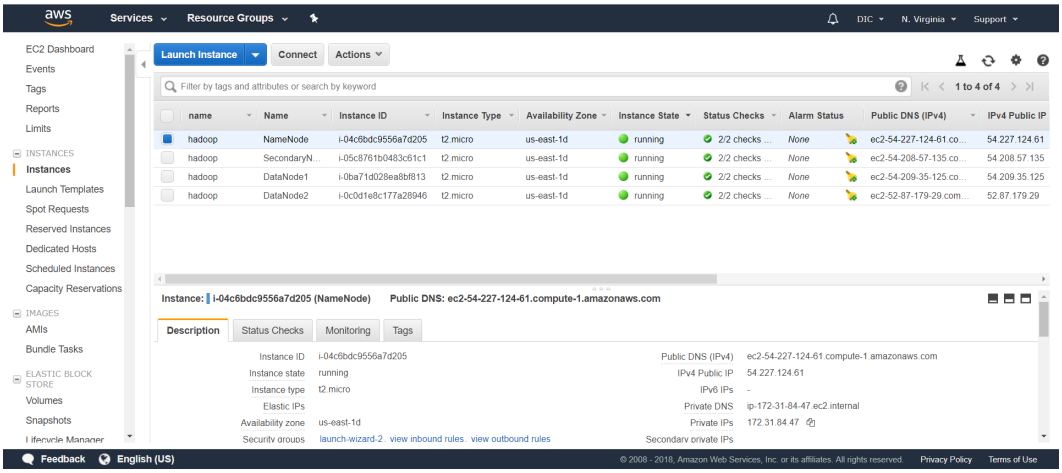

Figure 2: AWS Console

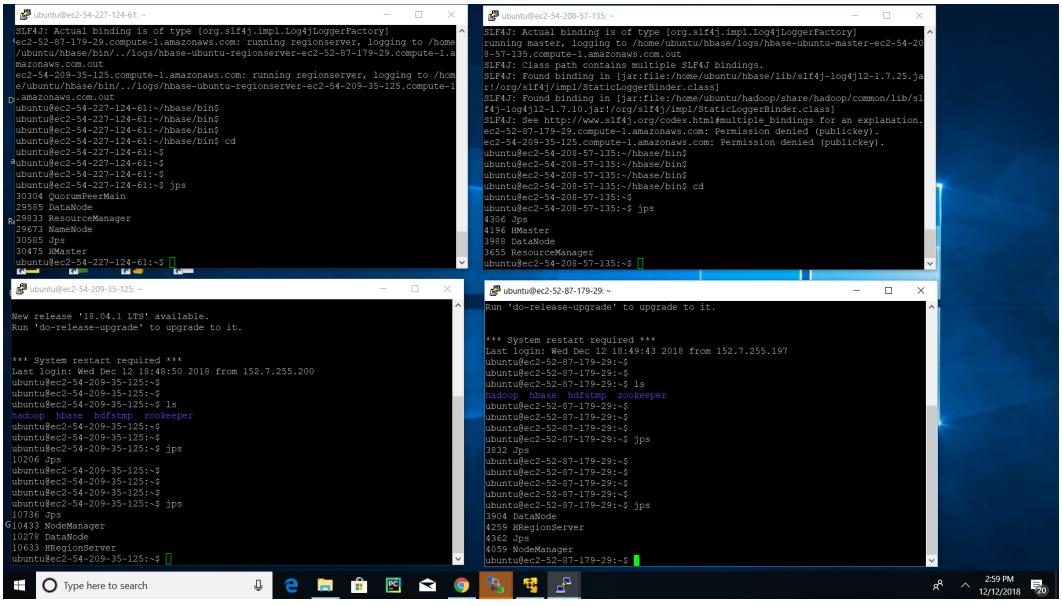

Figure 3: Services running on each node on EC2

4.3 Details of Experiment

Before beginning with storage of data in HDFS environment there were few prepossessing and modification tasks performed on data.

Many files in dataset had either missing values or unrelated values which were not useful. With the help R and python codes prepossessing and cleaning was performed. For example, data column involving True or False values had irrelevant string values. These were replaced with N/A values so that they will not badly affect the results.

After cleaning is done, next step was replication of cleaned data using python. Replication of dataset generated data of size around 100 GB. This new set of data then loaded into HDFS system. Figure 4 shows the data availability in datanode.

Before cluster setting there was 1 important configuration needed. Configuring IP-table rules to allow communication between all the nodes. Without adding these rules nodes were not able to communicate with each other.

Architecture / Data-Flow

A hadoop cluster was set up for the distributed data computing. In the architecture, two types were tried: On DigitalOcean, there was a Namenode process on one of the machines and there were 3 other machines which were the Datanodes. These Datanodes stored data in the Hadoop File System(HDFS) with a replication of 3. On another AWS cluster, a similar architecture was set up. But in this case there was another Secondary Namenode process running on a separate machine. This helped with the failure of the Master node.

Coming back to the Digital Ocean setup, there was a YARN Resource Manager running on top of this Hadoop cluster. This manager was used to run Spark on top of the Hadoop cluster. This reduced the latency of the computations in Spark as compared to the running a separate Spark cluster and accessing the HDFS data remotely. Thus having a kind of coupling in the data store and computaion engine helps to improve the latencies involved in various computations. The master node of the Spark cluster was on the same machine as the Hadoop Namenode. To prevent the Master node from being overloaded by tasks and to reduce any bottlenecks, there was no Worker process running on top of this machine.

Computation

All of the logs and tracking of the Spark's data flow and computations is done by YARN. The tasks assigned to Spark were using PySpark which is the Python support for Spark programming. Also for Machine Learning applications a Spark library MLlib was used. This library is useful in distributing and parallelizing the Machine Learning operations in the cluster. Spark assigns the operations to various worker nodes, which can be done explicitly or through libraries like MLlib which do it implicitly.

5 Results

The results can be divided into two major sections. They are 1) data storage, replication and distribution and 2) the parallel and distributed operations carried out on this data.

5.1 Distributed data storage

Data was stored in HDFS with 3 replicas of every element in the dataset. This redundancy ensured that there is availability of data at almost all times. The failure of any nodes in the cluster did not mean that data is lost and or unavailable until the node comes back to life. This ensured that any operations carried out were not impacted by failures of any machine.

Also redundancy did not cause any inconsistency in the data. The data consistency was taken care by the HDFS. This provided a strong combination of redundancy + consistency.

It was also easy to read/write data to the HDFS given the simple and intuitive user commands.

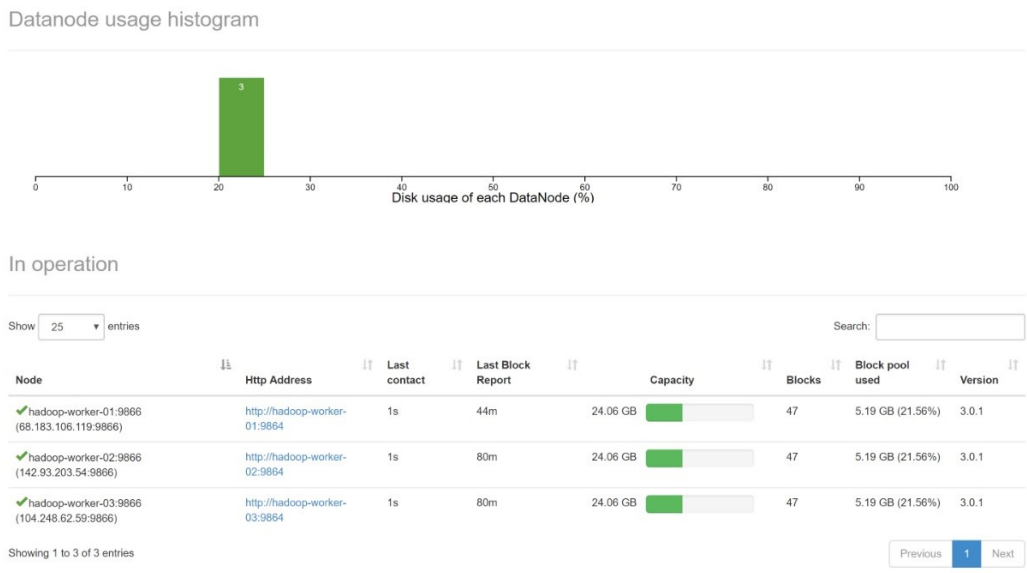

Figure 4: Hadoop GUI

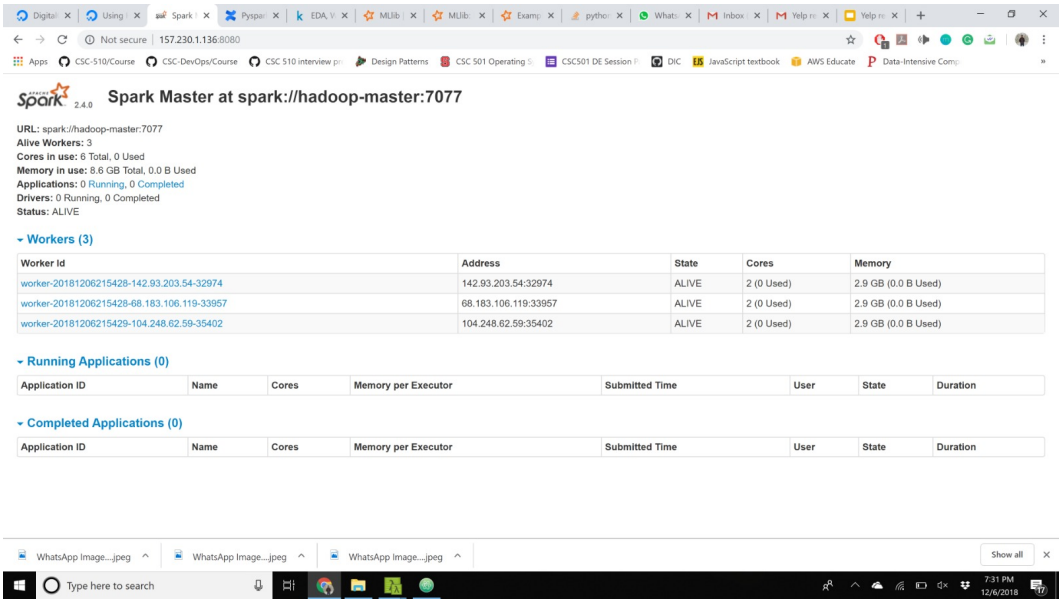

Figure 5: Spark GUI

5.2 Distributed operations

All operations carried out in Spark showed considerable amount of time to complete when the data was very small. It can be said that for small data standalone systems are a better option. But however as the data size goes on increasing the memory needs for the computation go on increasing. The memory of machines cannot be proportionally increased with the increasing data size.

Using Spark helped in computing various results ranging from simple aggregations to machine learning in an appreciably lower time. This is because Spark stores its objects(RDDs) in memory. Having all of the objects of data in memory along with an aggregated larger memory size of all of the worker machines helped improve performance greatly. Along with the memory considerations, Spark's MLlib library computed not only in distributed fashion but also in a parallel manner. This meant that the task was broken down into smaller parts and computed simultaneously amongt the

worker nodes. This too improved the time performance of the system. Spark is also fault tolerant just like the above HDFS data storage.

6 Observations

The performance however is not as fast as expected for the HDFS in comparison to other distributed databases that concentrate more on the performance than consistency of data.

Failure of worker nodes do not cause any failure in the operations. Operations are delegated to other nodes in the cluster according by the master.

7 Conclusion

The system designed above shows the need for distributed storage and computing and along with its effect on performance of the systems in the ever expanding big data world. Larger data sets mean that there is a need for storing data in more than one place and also enable reliability, speed of access and failure tolerance. HDFS provides a great reliable distributed data store platform that also enables speedy access of data from all of its cluster nodes. Such distributed big data also needs to be computed in a time that is comparable to that of the small data computations. Spark provides a great way of parallelizing such data computations that help to complete the operations with latencies that are comparable to same operations on small data on standalone machines. Along with this Spark is also failure tolerant. That is failure of any worker does not impact the computation in any adverse way than just increasing the latency a bit as the cluster will try to wait for the dead node if it dies.

8 Future Work

Being a semester long project, this project had 3 months time period. Spark computations on top of Hadoop cluster, HDFS and YARN was implemented. In future, it can be extended such that, an in-memory database can be used e.g Redis. This way, data would be available in memory and it can handle versatile data structures. Spark computes the above stated problem in say (3 minutes), but when Redis is used the time delays for accessing the data will reduce.

The number of nodes included in the cluster can also be increased so that even if replication factor is still kept as 3, there are other alternatives which are available to show replicas.

9 Acknowledgments

A special thanks to North Carolina State University and especially Computer Science department. We would also like to thank Dr. Vincent Freeh and Teaching Assistants Liang Dong and Fan Yang for constantly guiding us and providing feed back at every stage of the project. Your support was a driving energy for our accomplishments.

10 References

[1] "Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics": Juwei Shi, Yunjie Qiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold, Year:2015

[2] V Srinivas Jonnalagadda, P Srikanth, , Krishnamachari Thumati, "A Review Study of Apache Spark in Big Data Processing?, Year : 2016

[3] Shengti Pan, "The Performance Comparison of Hadoop and Spark", Year : 2016

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: "The Google File System"

[5] Jeffrey Dean and Sanjay Ghemawat: "MapReduce: Simplified Data Processing on Large Clusters"

11 Appendix

Few things were changed after we submitted the proposal because, as we moved along with the project, our approach towards it changed little.

Here are few changes :

1. As mentioned in the intial proposal, the data store that was supposed to be used was S3 (AWS product). However moving ahead it was decided to store the data in the Hadoop distributed file system. HDFS (although it's a file system) was also used in this architecture because it's provides high throughput even when large datasets are used. Also the use of S3 would have prevented the understanding of how data is distrbiuted, replicated and so on.
2. A Cassandra multi-node cluster was setup on VCL machines to be used as database for storage. But moving ahead it was decided to use HDFS as the size of the dataset was not that large and HDFS provided good results. Also since Spark was running on top of Hadoop, the overhead in utilizing the HDFS data by the Spark cluster was lower than that in a Cassandra cluster combined with the Spark cluster.
3. Scala, although mentioned is not used. Reason behind not using Scala was pretty straight forward. Python was used as all the team members were familiar with it. Also, Python has great support for Spark with PySpark tool and other programming tasks are simplified when using Python.

12 GitHub Link

The link to this Project repository is :

CSC591-DIC-Yelp-Recommendation-System