Benchmarking Big Data Systems

Abstract

In every aspect of data storage, there is a constant need to effectively and efficiently retrieve data. Whether this is a social media platform or an online database warehouse, each have their own way of querying their own data. As large-scale data analysis becomes a growing factor in today's industry, different data systems are implemented based on their technical features. In this paper, we will look at two well-known and distinct data systems, and develop and test benchmarks on a large dataset. This dataset will consist of hundreds of thousands of records separated amongst multiple tables, loaded on each cluster of machines for the specific data system. Each cluster will consist of four nodes each with a dual core ratio. The results of the tests on the data systems, show very reasonable and apparent differences of the two as well as ample evidence of the output.

General Terms

Data System, Dataset, Benchmark, Cluster Use Cases, Cloud Computing, SQL

1. Introduction

Before getting into the nature of this experimental benchmark, we should take a look at the fundamentals of data systems and using them on such large datasets. Sort of the ground basis for or representation of proper data systems is Apache Hadoop. Developed in 2005 and designed to move up from single server uses to thousands of machines. This allows for local configuration and storage of data. Hadoop is a cohesive software library that can handle processing large datasets across clusters of machines using similar methods. This data system offers high availability services on top of the cluster of machines, by utilizing the library itself which detects and handles failures at the application level. No longer does it primarily rely on the hardware for availability issues. Apache Hadoop is widely used as the cornerstone to many other Apache projects like Cassandra, Pig, Spark, Hive, and HBase. Some well-known companies that utilize the services of Hadoop include: Amazon, Spotify, Hulu and EBay. What makes Hadoop so a reliable and popular data system, is its use MapReduce. The framework splits the entire input dataset into their own independent groups, which are each processed using the specified map function (A single function that is sent to all nodes set for processing). Once the jobs have completed their output, the framework will then sort them and input them to the reduce tasks. This process allows for faster and more efficient throughput of data processing. Hadoop also introduces other modules like Common, HDFS and YARN. Common which is short for common utilities, supports all other modules. Hadoop Distributed File System

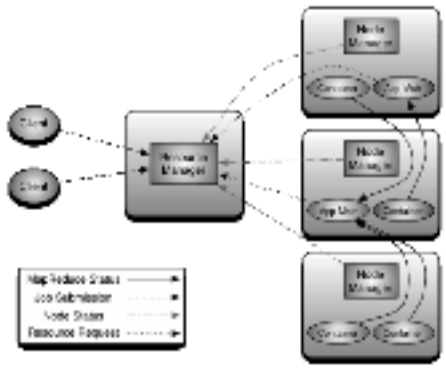

Figure 1: YARN diagram of Apache Hadoop

(HDFS) works alongside MapReduce to provide high-throughput access to application data. Finally, YARN acts in handling job scheduling and cluster resource management. YARN is a keystone in the optimization of MapReduce and overall Hadoop efficiency. In terms of general composition, YARN consists of separate daemons known as the ResourceManager (RM) and ApplicationMaster (AM). Now the processes that come into the system are sent to the ResourceManager that delegates and has full authority over resources, and once resources are allocated, the NodeManager takes over supervision of its specific container (Figure 1). All containers within the nodes are then subject to MapReduce processes at the request of the ApplicationManager. The AM allows for a robust system to prevail by issuing service restarts in the event of failures from container jobs. The complexity and sustainability that Apache Hadoop brings to other projects, causes for non-stop upward mobility and the need for even more efficiency in large data processing. In the next section we will discuss our first data system that is to be benchmarked on a large-scale dataset.

2.1 Apache Hive

When deciding which data systems would be utilized in our dataset, the main thought was seeing how something similar and physically built upon Apache Hadoop would fair against different systems. Apache Hive put out a stable release in 2016 imprinting most of its framework on top of Hadoop. This new data system provided tools that enabled easy access to stored data via HiveQL, which is very similar to normal SQL. Hive enables retrieval of files that are either directly stored on HDFS or on another storage system like Apache HBase. Hive is very versatile in that not a single “Hive format” in which data must be stored, this helps with different types of extensions of data like CSV or TSV text files to be inputted seamlessly. On a downward note, Apache Hive is not suitable for online transaction processing (OLTP). But is typically and more efficiently designed for traditional data warehousing, where data is constantly being expanded and bloated. It looks to strive for maximum scalability and at the same time upholding performance, fault-tolerance, and loose-coupling (low component dependency) with input formats. Initially developed at Facebook, Hive is used by other large companies like Netflix and Amazon Web Services. Knowing much more about Apache Hive, we can jump into the overall setup on the Google Cloud cluster platform.

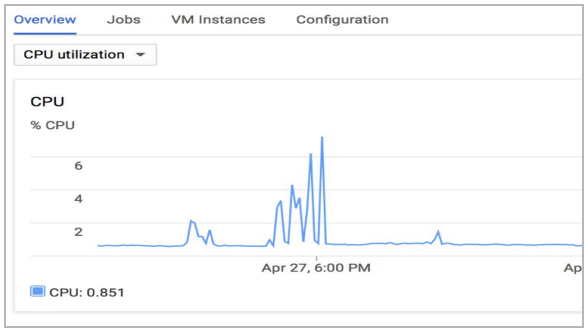

Figure 2: CPU utilization after running jobs using Apache Hive on Google Cloud Platform.

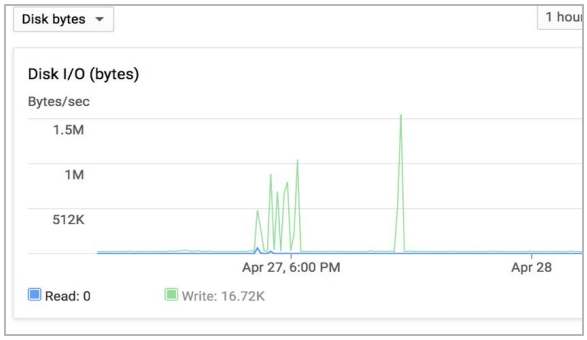

Figure 3: Disk usage after running jobs using Apache Hive.

2.2 Hive Setup

The first issue to note is that if you are utilizing a simple personal account, that you will only be subjected to an 8-core max cluster. This being said, if you wish to set up a cluster of 8 nodes, then have 1 core for each. Otherwise in this experiment, we will be choosing four nodes with 2-cores each. **1)** Create the system to hold one master node and n-number slaves. Set the slave nodes to three total with each holding 2-cores as well as the master. **2)** Make sure the APIs for the 'Dataproc' and Google Cloud Engine are properly turned on (Menu - API Manager - Dashboard - Enable API). **3)** A bucket needs to be made in the storage section of the platform. Once created, make a data directory that will hold any number of distinct folders that contain your uploaded datasets. **4)** Once the cluster has successfully been established, the next thing to do is setup Apache Hive. We will need to connect to the master node using Secure Shell (ssh). At the cluster details menu, click on the ssh tab to the left of the master node. A command prompt should load up and enter the corresponding commands on page six of the Hive Setup website in resources portion of the paper. **5)** The next step entails creating the tables within Beeline. For this experiment we use external tables for each .csv that was loaded into storage. Once the tables are created, try testing simple select statements on the dataset to make sure everything is setup properly. **6)** Finally, the data system is setup to test fully on the dataset, assuming detailed queries were created for abstract benchmarking. To create a job for Apache Hive, hover over the menu and travers to 'Dataproc' and choosing 'Jobs'. Choose 'Submit Job' at the top, and make the type 'Hive'. Now that we have the entire framework set up with jobs ready to be submitted, we can move into the our other system Apache Spark.

3.1 Apache Spark

In terms of reliability and efficiency, some data systems that we looked at, Apache Hadoop and Hive, have utilized the process of MapReduce to handle jobs quickly and cleanly. Apache Spark on the other hand, is a revolutionary design that can work as its own standalone mode or run Hadoop cluster through YARN. What makes it stand out over Hive, is that Spark can perform both

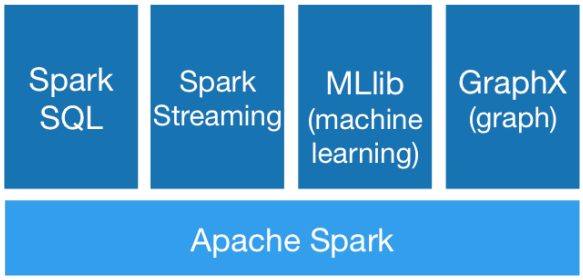

Figure 4: Apache Spark has access to many libraries that can be combined seamlessly into one application.

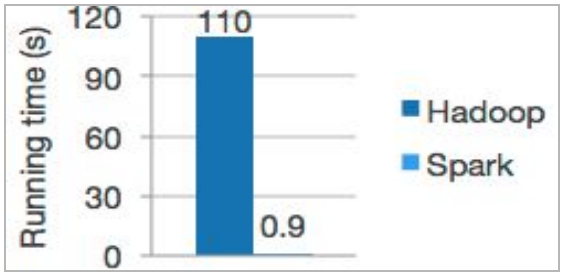

Figure 5: Apache Spark utilizing its DAG execution engine against Hadoop in a big data speed test.

batch processing like MapReduce, but also newer workloads like: streaming, interactive queries, and machine learnings (Figure 4). Typical data systems run queries on their data-sets directly from its storage on disk, but Spark utilizes memory space to quickly grab necessary output. If memory is too small to fit all the data, then Spark operations spill onto the disk, allowing the data system to run well on any sized data-set. One of the most standout features of Apache Spark is its speed. Spark utilizes its very own advance DAG execution engine that supports acyclic data flow and as explained previously, in-memory computing. When tested heads-up against Apache Hadoop, results were miles ahead, nearly 100 times faster runtime on large datasets (Figure 5). Apache Spark offers easy operation through languages such as Scala, Java, Python, and R. In this experiment we will be using Spark's own querying language very similar to SQL and HQL, known as Spark-SQL.

3.2 Spark Setup

Using the Spark setup reference and steps 1-4 from the previous Hive setup, we can create a fully operational Spark cluster on Google Cloud Platform. Note step 3 of the Hive set up for Spark should create a data directory as well as n number more within it. Depending on the dataset used in the cluster the n number inner directories will be determined by the amount of .csv files used. In this experiment we are using three different .csv files with the filename (ie. movies, tags, ratings) as the inner directory names. Once each file is distributed into its own directory that are all under the data folder, we are ready to create the tables under the master ssh. The difference with Spark's command for creating the tables is actually not that different, in that the only portion changed is the location of the data. When inputting the CREATE EXTERNAL TABLE command, end it with the following line: LOCATION 'gs://market-data-bucket/data/my_data.csv'; Obviously, make the appropriate changes to the directory names and .csv name in the above command. Once the tables have been properly created and data is uploaded, the cluster will only need jobs to execute for the benchmark. Issue the same step 6 format from the Hive setup; except use 'Spark-SQL' as the prefered type and the new query that is acclimated for this SQL type. Do note though, in this experiment we

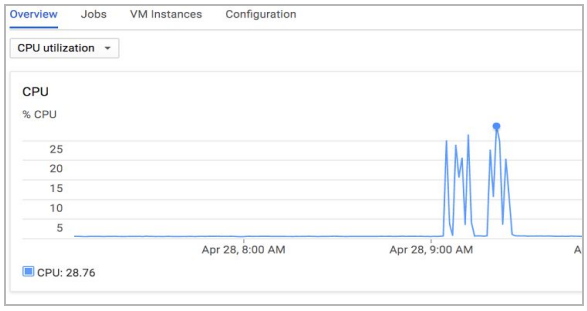

Figure 6: CPU utilization after running jobs on Apache Spark on Google Cloud Platform.

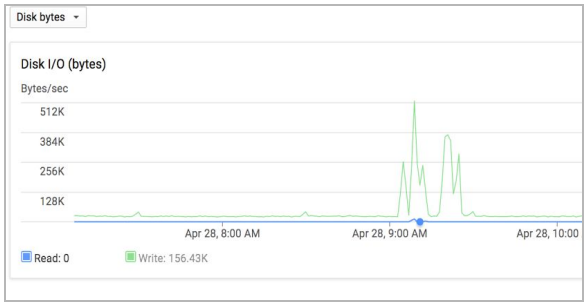

Figure 7: Disk usage after running jobs on Apache Spark.

were fortunate enough to not need to make changes to the four given queries for both Hive and Spark since they were so similar.

4. Benchmarks

The dataset in both Hive and Spark execution is on a collaboration of movie titles and ratings. There are around 50mb of data that include movie titles, ratings, and tags (ie. appealing, actors/actresses). Our data is split amongst the three as separate .csv files as well as directories (Spark only). The benchmarks that will be tested for both data systems are as follows: Overall speed test (sec), CPU utilization, and finally disk usage. The sql queries for the dataset exploit both simple and complex executions to test all ends of both data systems. Starting off with a simple select query:

Select all movie titles where Zach Galifianakis is a tag:

```
SELECT m.title
FROM movies m, tags t
WHERE t.tag = "Zach Galifianakis" AND
m.movieId = t.movieId
GROUP BY m.title;
```

The second query is the most mathematically induced by employing the SQL average function:

Select all movie titles and average rating with the tag "visually appealing":

```
SELECT m.title, avg(r.rating)
FROM movies m, tags t, rating r
WHERE m.movieId = t.movieId AND
        m.movieId = r.movieId AND
       t.tag like 'visually appealing'
GROUP BY m.title;
```

The third query includes an aggregation:

Count the number of records where the movie rating is greater than or equal to 4.5 and the movieId is greater than or equal to 96000:

```
SELECT count(*)
FROM movie m, ratings r
WHERE m.movieId = r.movieId AND
        R.rating >= 4.5 AND
        m.movieId >= 96000;
```

The final query uses a distinct function:

Select all distinct movie titles where the genre has the phrase "Act":

```
SELECT distinct (m.title)
FROM movies m, rating r
WHERE m.movieId = t.movieId AND
        (m.genres like 'Act%' or m.genres
        like '% | Act%');
```

After completing each job two to three times on both clusters separately to maximize performance, the differences are substantial. As seen from Figure 2 and Figure 6, Spark utilizes much more processing power to run all queries, 25% vs. about 8%. This can be

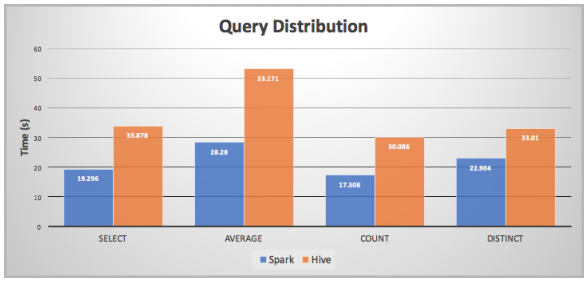

Figure 8: Distribution of execution time based on each query for Hive and Spark in seconds.

present because of the amount of data that is processed in memory and spilled over into disk. This criss-cross effect could need more resources to achieve a fast execution time. Using the acyclic data flow also utilizes much more processing power in order to pump out data at such a high rate. Now looking at Figure 3 and Figure 7, it can be said that Hive utilizes much more disk usage than Spark. Obviously this result was assumed because of Hive's reliability only on storage of the data and not memory. Spark primarily uses memory and in some cases if the data is too large, it will spill into storage, which can be concluded by its nearly half amount of disk work.

5. Conclusion

Testing each system out on all four queries led to some interesting yet known results. Spark is definitely faster and much more efficient, but can consume much more processing resources. Hive seems very sluggish and cannot really compare, but if speed is not what you are looking for then maybe Apache Hive is a safe option. When it comes to just querying a large set of data, if you would like a robust and reliable data system that needs no streaming capabilities then choose Hive. It's easy to set up and utilize its HQL querying language. On the other hand, if you want speed and lightning fast result from a huge dataset, then use spark. Figure 8 shows how Spark performed nearly 1.5x better than Hive with a dataset that is only about 50mb. Think if we had gigabytes of data stored and needed to get results as soon as possible, Spark would definitely be the data system for the job.

6. References

Hadoop: *http://hadoop.apache.org/*
Hive: *https://hive.apache.org/*
Hive Setup: *http://holowczak.com/getting-started-with-hive-on-google-cloud-dataproc/*
Spark: *http://spark.apache.org/*
Spark Setup: *http://holowczak.com/getting-started-with-apache-spark-on-google-cloud-services-using-dataproc/*
Spark use-cases: *https://www.qubole.com/blog/big-data/apache-spark-use-cases/*