## PERFORMANCE COMPARISON OF HADOOP AND SPARK



**Group Members:**

**1) Imanuel Drexel (170582641)**

**2)** **Deni Setiawan (170357407)**

**3) Ouphachay Thongsamouth (170587602)**

**4) Sarang Kharche (150740395)**

Big Data Processing

**Table of Contents**

[PERFORMANCE COMPARISON OF HADOOP AND SPARK 0](file:///Z:\Academia\Big%20Data%20Processing\CW2\Report.docx#_Toc503114869)

[Abstract 2](#_Toc503114870)

[Introduction 3](#_Toc503114871)

[Context of Performance of distributed computation 3](#_Toc503114872)

[Explanation of the Selected Techniques 4](#_Toc503114873)

[Implementation Details 5](#_Toc503114874)

[Discussion of the Obtained Results 6](#_Toc503114875)

[References 7](#_Toc503114876)

# 

# **Introduction**

Big Data is currently a hot topic for companies and scientists around the world, due to the emergence of new technologies, devices and communication mean like social network sites, which led to a noticeable increase of the amount of data produced every year, even every day. As it said the total data used to double every million years during 1100, then eventually it went down to 1000, then 500, 100, 50, 10 years and now presently the data doubles every 2 years as of 2017. We believe this number will even go down further to 1 year and one point it will 1 month which I believe will be very interesting times.

In addition, traditional algorithms and technologies are inefficient to process, analyse and store this vast amount of data. So, to solve this problem, Big Data frameworks are needed. In this report, we present and discuss a performance comparison between two popular Big Data frameworks. Hadoop and Spark, which are used to efficiently process the vast amount of data in parallel and distributed mode on a large cluster. Hibench benchmark suite is used to compare the performance of these two frameworks based on the criteria as execution time, throughput and speedup. Our experimental results show that Spark is more efficient than Hadoop to deal with a large amount of data. However, spark requires higher memory allocation, since it loads processes into memory and keeps them in caches for a while, just like standard databases. So, the choice depends on performance level and memory constraints.

Our motivation and the purpose for this is to compare performance behaviour of Hadoop MapReduce Vs. Spark. This study will show how distributed computing algorithms run in practice, and further understand the obtained performance. Results have compared the resource utilisation of cluster elements, as well as the execution time.

In this report, further, we have gone through with the process of comparing performance between Hadoop and Spark, demonstrated context of performance of distributed computation and have presented the results accordingly.

|  |  |
| --- | --- |
| **Hadoop Ecosystem** | **Spark Ecosystem** |
| **Component** | **Component** |
| HDFS | Tachyon |
| YARN | Mesos |
| **Tools** | **Tools** |
| Pig | Spark Native API |
| Hive | Spark SQL |
| Mahout | MLib |
| Storm | Spark Streaming |
| Giraph | GraphX |
| HUE | Spark Notebook/ISpark |

# **Context of Performance of distributed computation**

With rapid development of information and communication technologies, the data are created and consumed heavily, a report from IDC suggests around 2.7 Zetabytes of data exist in the digital universe today. Distributed computing creates preconditions for analysing and processing such Big Data by distributing the computations among many compute nodes. In this work, performance of distributed computing environments based on Hadoop and Spark frameworks is estimated for real clusters. For a task, we used the twitter dataset and h1b visa petitions dataset by kaggle. It was found that the running times grow very fast with the dataset size and faster than a power function even. As to the real and virtual versions of cluster implementations, this tendency is the similar for both Hadoop and Spark frameworks. Moreover, speedup values decrease significantly with the growth of dataset size, especially for virtual version of cluster configuration. The problem of growing data generated by various means have accelerated the need to figure more efficient and faster distributed computation platform. The current observations as to the running times and speedup on Hadoop and Spark frameworks in real cluster configurations can be very useful for the proper scaling-up and efficient job management, especially for machine learning and deep learning applications, where Big Data are widely present.

# **Explanation of Implementation and Selected Techniques**

**Numerical summarisation:**

**1. Hadoop**

Mapper gets each line of input data and split it into Array of String using splitter”;”. The goal of it is to get the tweet itself, which is the 3rd element of the array. After getting the tweet content, using the length method of String in Java, we can get the length of the content. But, for avoiding any faulty and error in the splitter result, the length method is called just when the length of Array String is equalled 4 (the complete part of each line).

For providing the bin, the received length is divided with 5 and then multiply it with 5 again. This will give the result of an Integer with multiplies of 5 as the maximum value of each bin. The Key and Value of Mapper are sent just if the tweet length less than equals 140. With key contained the Bin (Text) and the value contained a counter, IntWritable with a value of 1.

As they are sent into reducer, the Key and Value of reducer had to have similar input with the Map Output. In reducer, the value will be iterated into a loop that counts the sum of data from similar bins

**2. Spark (Scala)**

Firstly, RDD of splitting the dataset given by using “;” and filter the lines that contain more than 140 characters. A transformation of RDD using Math.Ceil was done after that. It is used because we want to get the bin of every 5 characters given. So, in the end, it should be 28 bins. After getting the bin, there’s RDD for making it as a tuple, and reduce it by its key using reduceByKey. For completing the job, an action of saveAsTextFile is called at the end of the stages.

**Expected performance:** Our expectation was that Spark would be much more efficient and faster than Hadoop, after the experiment we came to the conclusion that Spark is much faster in certain condition as it processes data in-memory.

**Iterative computation:**

**Hadoop 1:**

Algorithm Description

1. We made 8 dimension features to calculate the euclian distance of each line in the dataset. We made making 8 different hashtables for translating into numerical order. Those are case\_status, employer\_name, SOC\_name, job\_title,full\_time\_position, prevailing\_wage, year, worksite. Thus, while have translated the data set into numerical number, for example:

original line:

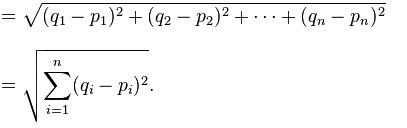
"CERTIFIED","CARBONITE, INC.","MARKET RESEARCH ANALYSTS AND MARKETING SPECIALISTS","SENIOR DATABASE MARKETING ANALYST","Y",74006,2016,"BOSTON, MASSACHUSETTS"

after translating (assumption):

1, 1, 31, 22,482, 4,5,5

2. we tried to classify each line to find them.

The formula is:



The problem we faced:

We just can translate case status into a hashtable, but after adding the second hashtable we faced some errors so we cannot continue the project.

**Spark 2:**

With this question, we tried to implement the k-means algorithm to measure the multiple iterations of spark and Hadoop to compare their performance using h1b visa petition dataset, the reason we choose this dataset because it contains quite a sufficient number of record and contains many features that we believe suitable for measuring the performance. In order to perform these tasks first step, we have to clean up the dataset and extract the features from the dataset, however after implement different methods for features extraction for both spark and Hadoop such as: for spark we have tried StringIndexer and Dataframe but we cannot produce the result as we expected instead we got the errors without any result which we cannot record and compare the result between both of them spark and Hadoop. However, some source code of the implementation has been included in this report.

**Join between datasets:**

1. **Hadoop**

It is implementing Hashtable for saving the information got from the joint dataset. For extracting

information, the additional dataset also is split by (“, ”) because it’s a comma-separated values file. However, the result should be filtered just array of string that has length equals nine would be processed to save the extracted information.

In the mapper class, a Set<String> class of java is implemented to get the all of Keys from the hashtable. After that, a for loop iteration would be set the value of every key received, and check if it’s on the tweet or not. If it is true, the key which is the name of the athlete would be set as the key mapper and an IntWritable(1) also be assigned as the value of mapper, to mark the appearance of each key.

In the reducer, key and value would be received. The key will pass directly as the key of the output after the shuffle and sort process by the Hadoop, but the value would be summed up for each same key received. And in the end, that sum would be set as the value of the key.

1. **Spark (Scala)**

Joining dataset in Spark can be implemented by using a broadcast variable. After getting and cleaning the dataset, using the “/data/olympictweets2016rio” and cleaning it with giving a filter to use the tweet that contains less than 140 characters only, we get the second dataset (“/data/athletesrio.csv”) and get the value of athlete name. Then, that value is collected using collect(). Spark will take this command as action and process the RDD. After being collected, the result is broadcasted as a new RDD.

At this time, the value of athletename is shared with the all the workers so that they can process the next action. To get the athlete name in all the tweet dataset, an RDD that doing iteration using condition was done. The command tells spark to iterate all the value of athlete name to all line, and if there’s an athlete name in that line, yield that name. The next step is to map all of the results and make it tuple (name, 1) for being reduceByKey after that. To complete all of this RDD transformation, an action saveAsTextFile is taken.

**Expected Performance:** As with previous we concluded Spark is faster, efficient and wasy to use. This task proved it right again, Spark performs better when all the data fits in the memory, especially on dedicated clusters.

# **Discussion of the Obtained Results**

**Numerical summarisation**

|  |  |
| --- | --- |
| **Hadoop Jobs** | **Duration (In Second)** |
| Iteration 1 | 61 |
| Iteration 2 | 68 |
| Iteration 3 | 44 |
| Iteration 4 | 41 |
| Iteration 5 | 102 |
| Iteration 6 | 40 |
| Iteration 7 | 96 |
| Iteration 8 | 91 |
| Iteration 9 | 110 |
| Iteration 10 | 113 |
| **Average** | **76.6** |

|  |  |
| --- | --- |
| **Spark Jobs** | **Duration (in second)** |
| Iteration 1 | 21 |
| Iteration 2 | 13 |
| Iteration 3 | 11 |
| Iteration 4 | 10.5 |
| Iteration 5 | 14 |
| Iteration 6 | 11 |
| Iteration 7 | 10 |
| Iteration 8 | 10 |
| Iteration 9 | 15 |
| Iteration 10 | 15 |
| **Average** | **13.05** |

Table of duration for spark and Hadoop jobs measured in seconds.

For the result, the result is spark jobs is 5.8 times faster averagely than the Hadoop jobs. This is because MapReduce in Hadoop performs disk-based operations, while the spark is processing an in-memory cache. Thus, the spark is faster because it does not need to access HDFS frequently as Hadoop does after a map or reduce action. Moreover, for spark, it is a ‘cheap’ computation because it is only collect and count the values. So, it can perform maximally.

**Join between datasets:**

|  |  |
| --- | --- |
| **Hadoop Job** | **Time (in second)** |
| Iteration 1 | 107 |
| Iteration 2 | 132 |
| Iteration 3 | 143 |
| Iteration 4 | 172 |
| Iteration 5 | 210 |
| Iteration 6 | 204 |
| Iteration 7 | 178 |
| Iteration 8 | 166 |
| Iteration 9 | 172 |
| Iteration 10 | 198 |
| **Average Time** | **168.2** |

|  |  |
| --- | --- |
| **Spark Job** | **Time (in seconds)** |
| Iteration 1 | 1335 |
| Iteration 2 | 677 |
| Iteration 3 | 916 |
| Iteration 4 | 1511 |
| Iteration 5 | 1399 |
| Iteration 6 | 671 |
| Iteration 7 | 1459 |
| Iteration 8 | 971 |
| Iteration 9 | 739 |
| Iteration 10 | 973 |
| **Average Time** | **1065.1** |

Spark and Hadoop job duration on joining dataset measured in seconds.

On using 2 datasets and join them for being operated, Spark did much slower than Hadoop. Hadoop can do 6.67times faster than spark job for this particular job. This happened because in spark, before being broadcast, the smaller dataset should be collected using collect () and costs several seconds to do it. Moreover, in the map stage on spark, there was iterative computation there which needed huge memory because every single line of olympictweets2016rio dataset should have alliteration of the athleterio dataset. If the full dataset consists of 2.578.144 lines and the athletesrio dataset had 11.539 lines, the total operation would be 2.578.144 \* 11.539 = 29.749.203.616 times. Spark is depended on the memory of the cluster. If the RDD operation exceeded the memory, it will use the disk for the data that doesn’t fit in the memory, which increases the time to travel the data between them.

# **References**

(n.d.). Retrieved from Datamation: https://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html

(n.d.). Retrieved from Cloudera: http://blog.cloudera.com/blog/2015/05/working-with-apache-spark-or-how-i-learned-to-stop-worrying-and-love-the-shuffle/

(n.d.). Retrieved from SCNSOFT: https://www.scnsoft.com/blog/spark-vs-hadoop-mapreduce

(n.d.). Retrieved from Quora: https://www.quora.com/Is-Apache-Spark-faster-than-Hadoop-processing

(n.d.). Retrieved from Stcloudstate.edu: http://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1008&context=csit\_etds

*Analytics Week*. (2018, Jan). Retrieved from Analytics Week: https://analyticsweek.com/