User's Manual
&
Research Results

Version 2.0

2 May, 2016

Revision Signatures

By signing the following document, the team member is acknowledging that he has read the entire document thoroughly and has verified that the information within this document is, to the best of his knowledge, is accurate, relevant and free of typographical errors.

Name	Signature	Date
Sushant Ahuja		
Cassio Lisandro Caposso Cristovao		
Sameep Mohta		

Revision History

The following table shows the revisions made to this document.

Version	Changes	Date
1.0	Initial Draft	1 May, 2016
2.0	Final Minor Changes	2 May, 2016

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide information to the user on how to run the movie recommenders we have built for both systems Hadoop and Spark. Furthermore, we have a detailed breakdown of the files used in this project as well as a graphical summary of each test and results performed on environments - Hadoop and Spark, both single node and cluster.

1.2 Overview

This document includes the following four sections.

Section 2 - Hadoop: Gives a detailed overview of the tests we conducted (Word Frequency Count, Matrix Multiplication, K-Means Clustering, Recommender) on our Hadoop cluster and single node, and the graphs we put together to show our results.

Section 3 - Spark: Gives a detailed overview of the tests we conducted (Word Frequency Count, Matrix Multiplication, K-Means Clustering, Recommender) on our Spark cluster and single node, and the graphs we put together to show our results.

Section 4 – Using Our Recommenders: In this section we discuss how the user should use the recommendation systems we built for both systems; Hadoop and Spark.

Section 5 - Glossary of Terms: Lists all the technical terms that are mentioned in this document with their definitions.

2 Hadoop

2.1 Word Frequency Count

2.1.1 What is it?

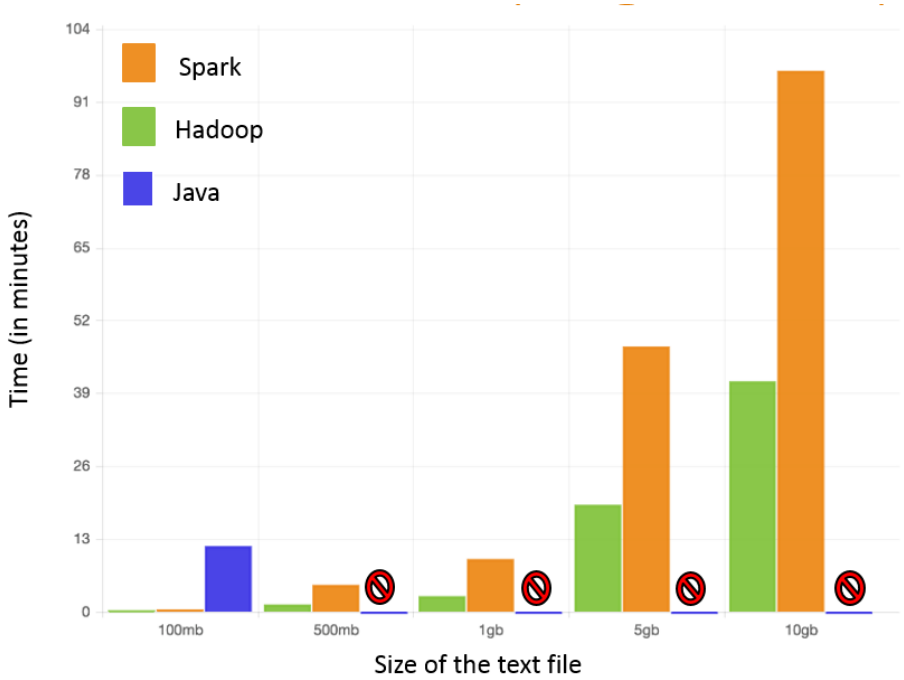
- Word count is called the "Hello World" program of the Big Data development
- It is exactly what you think. We will have a text file as the input and we write a MapReduce code to count the number of each word in that input file.

2.1.2 Files Used

- For the word count we performed the test with the following file sizes (all files can be found in the following link):
  - 100 MB
  - 500 MB
  - 1 GB
  - 5 GB
  - 10 GB
- Inside the 'Word Count' folder, choose the appropriate file, example *100M.txt* to run the test. The format is just random strings of words each word follow by a new line.

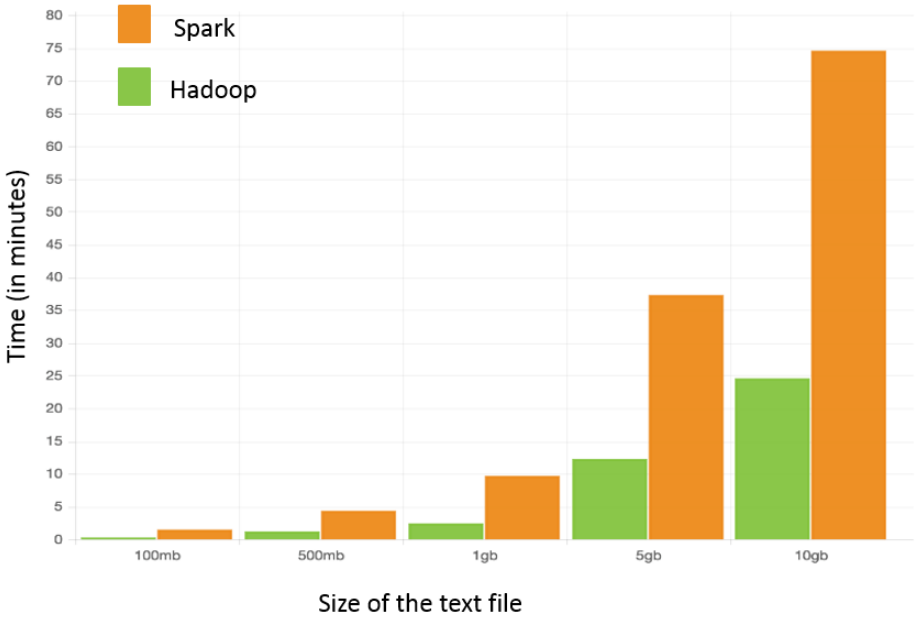
2.1.3 Results

- **Single Node**



Observation: We observe that Java fails when the size of the input file is 500MB due to heap size and feasibility problems. Hadoop performs better than Spark as Hadoop is better at batch-processing.

- **Cluster**



Observation: We observe that Hadoop performs better again in the cluster and the time for processing these files is considerably low as compared to single-node.

2.2 Matrix Multiplication

2.2.1 What is it?

- Matrix Multiplication it is an operation where we take a pair of matrices and produce a new matrix
- Matrix Multiplication is very useful to test application performance.
- Matrix Multiplication is an integral part of a recommendation system.

2.2.2 File Used

- For matrix multiplication, we used the following set of matrix sizes, first being A.txt and the second being B.txt:

	A.txt	B.txt
1	**2x5**	**5x3**
2	**10x10**	**10x10**
3	**50x50**	**50x50**
4	**100x200**	**100x200**
5	**1000x800**	**800x1000**
6	**5000x6000**	**6000x5000**

- Sample File and sample output file

```
1 A,0,0,5.0
2 A,0,1,1.0
3 A,0,2,2.0
4 A,0,3,3.0
5 A,0,4,4.0
6 A,1,0,5.0
7 A,1,1,6.0
8 A,1,2,7.0
9 A,1,3,8.0
10 A,1,4,9.0
```

Matrix A
(2*5)

```
1 B,0,0,1.0
2 B,0,1,1.0
3 B,0,2,2.0
4 B,1,0,3.0
5 B,1,1,4.0
6 B,1,2,5.0
7 B,2,0,6.0
8 B,2,1,7.0
9 B,2,2,8.0
10 B,3,0,9.0
11 B,3,1,10.0
12 B,3,2,11.0
13 B,4,0,12.0
14 B,4,1,13.0
15 B,4,2,14.0
```

Matrix B
(5*3)

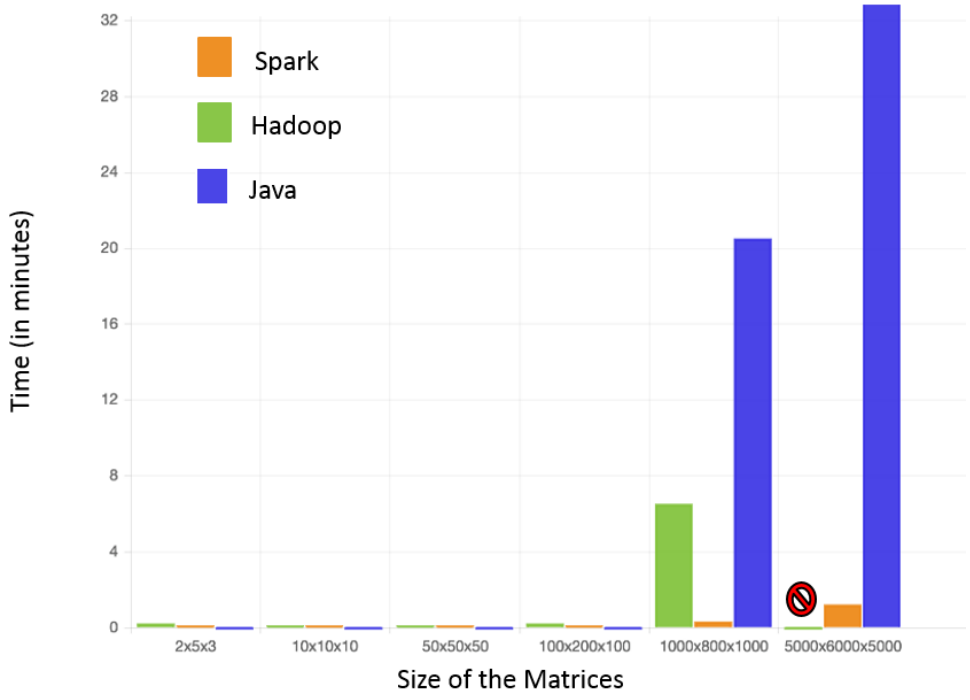
```
1 0,0,95.0
2 0,1,105.0
3 0,2,120.0
4 1,0,245.0
5 1,1,275.0
6 1,2,310.0
```

Result Matrix
(2*3)

- To run the test for example; 100x200 and 100x200 go inside folder – matrix multiplication/4/
- To generate your own matrices of any sizes, refer to the source code on the following link: http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html

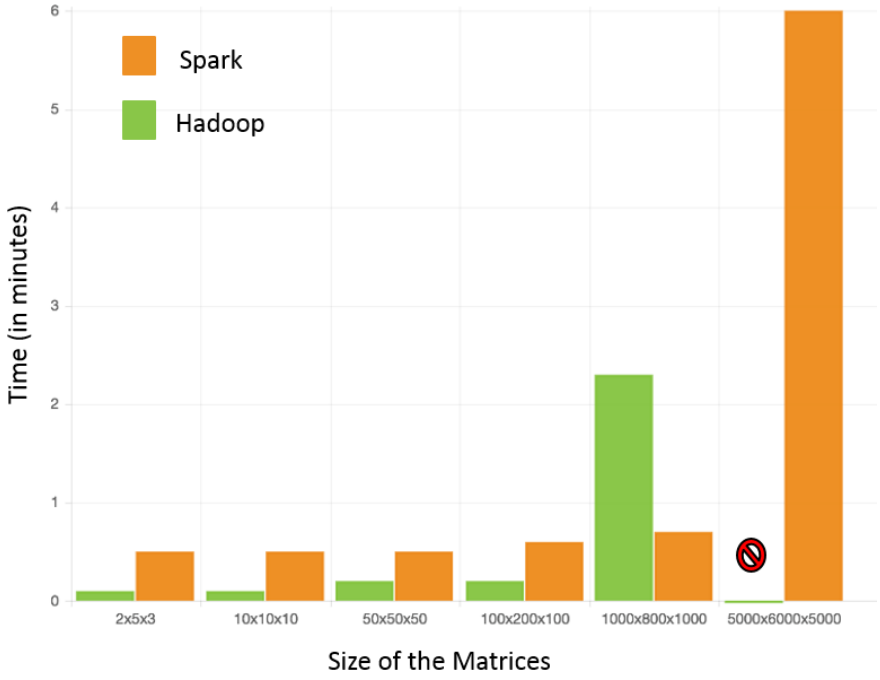
2.2.3 Results

- **Single Node**



Observation: We observe that Java performs well when there is a sequential matrix multiplication and the size of the input files is not large. Spark also performs really well as it has great computational capability. Hadoop's performance is unsatisfactory due to less computational power.

- **Cluster**



Observation: We observe that the performance gets really better in a cluster. However, Hadoop crashes when the size of matrix is large due to less number of nodes in its cluster and its low disk-processing power.

2.3 K-Means Clustering

2.3.1 What it is?

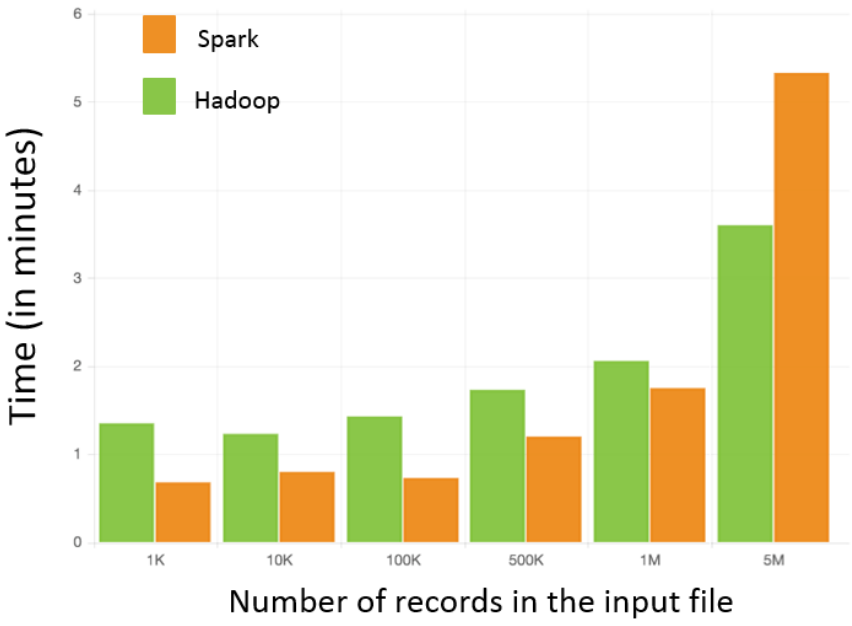
- As the name suggests, Clustering implies grouping items together. But on the basis of what?
- We use clustering in Big Data industry to group similar items/users together for better data management.
- It has numerous applications in the business and helps them to target their customers in a better way.
- K-Means algorithm is a simple, but widely used algorithm used for clustering. All objects need to be represented as a set of numerical features. Also, the user has to specify the number of groups (k) he wants

2.3.2 Files Used

- For this test we used files with different number of points, those points where randomly generated, each point value ranges from 5 to 100 as floats. Below are the files by number of records.
  - 1 K
  - 10 K
  - 100 K
  - 500 K
  - 1 M
  - 5 M

2.3.3 Results

- **Cluster**



Observation: We observe that Spark performs a little better for smaller files, but as the size of the file is huge (5 million records in a file), Hadoop performs better as it has good rate of data-processing for huge datasets.

3 Spark

3.1 Word Frequency Count

3.1.1 What is it?

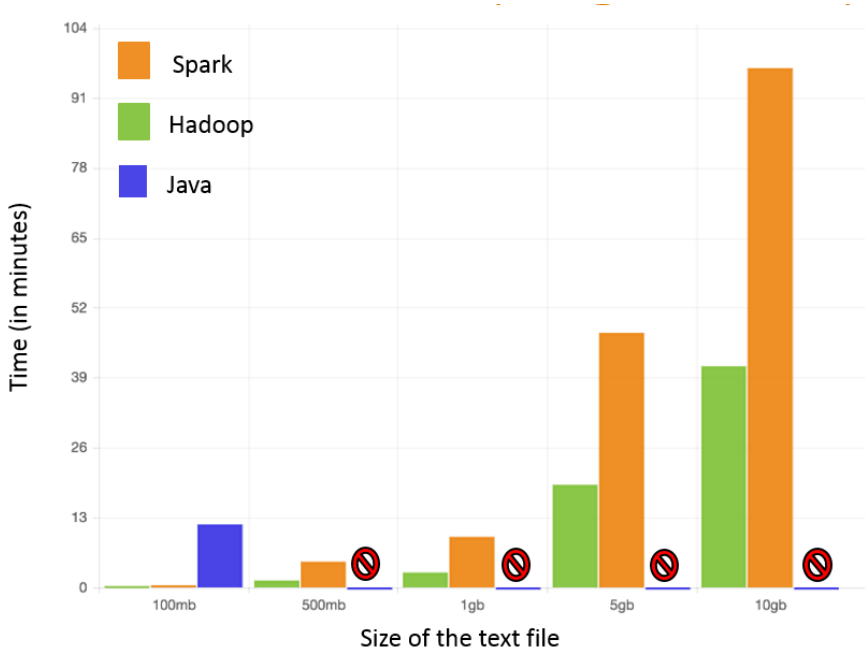
- Word count is called the "Hello World" program of the Big Data development
- It is exactly what you think. We will have a text file as the input and we write a MapReduce code to count the number of each word in that input file.

3.1.2 Files Used

- For the word count we performed the test with the following file sizes (all files can be found in the following link):
  - 100 MB
  - 500 MB
  - 1 GB
  - 5 GB
  - 10 GB
- Inside the 'Word Count' folder, choose the appropriate file, example *100M.txt* to run the test. The format is just random strings of words.

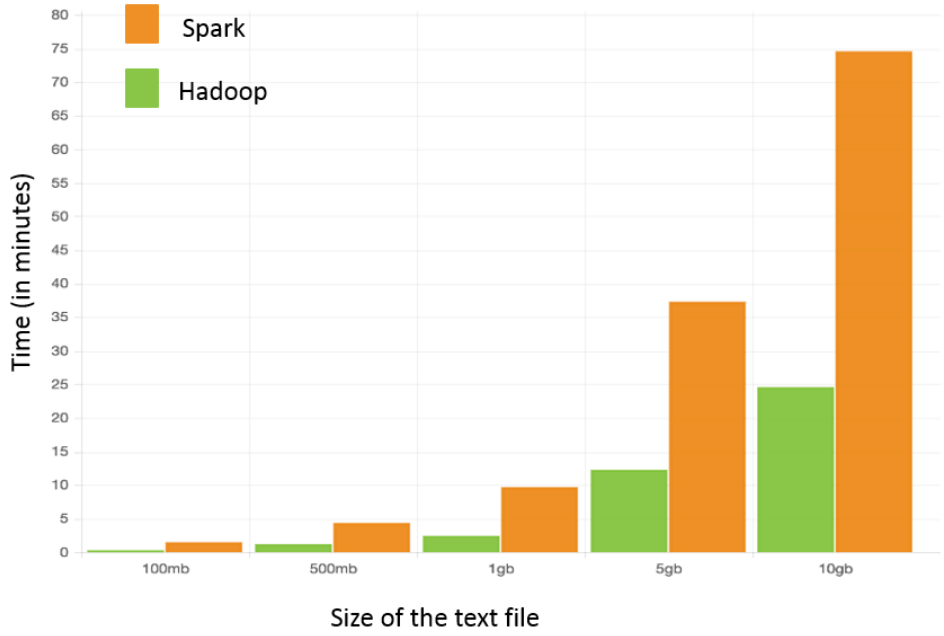
3.1.3 Results

- **Single Node**



Observation: We observe that Java fails when the size of the input file is 500MB due to heap size and feasibility problems. Hadoop performs better than Spark as Hadoop is better at batch-processing.

- **Cluster**



Observation: We observe that Hadoop performs better again in the cluster and the time for processing these files is considerably low as compared to single-node.

3.2 Matrix Multiplication

3.2.1 What is it?

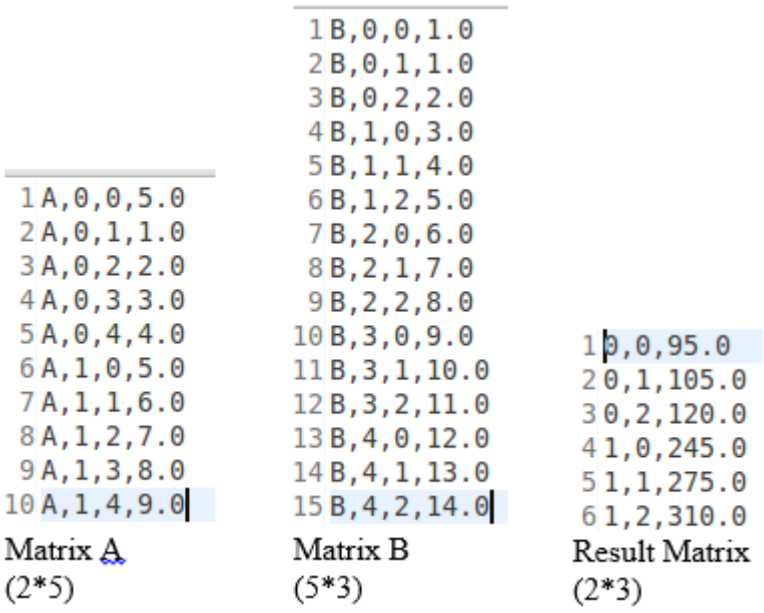
- Matrix Multiplication is an operation where we take a pair of matrices and produce a new matrix
- Matrix Multiplication is very useful to test application performance.
- Matrix Multiplication is an integral part of a recommendation system.

3.2.2 Files Used

- For matrix multiplication, we used the following set of matrix sizes, first being A.txt and the second being B.txt:

	A.txt	B.txt
1	2x5	5x3
2	10x10	10x10
3	50x50	50x50
4	100x200	100x200
5	1000x800	800x1000
6	5000x6000	6000x5000

- Sample Files and sample output file

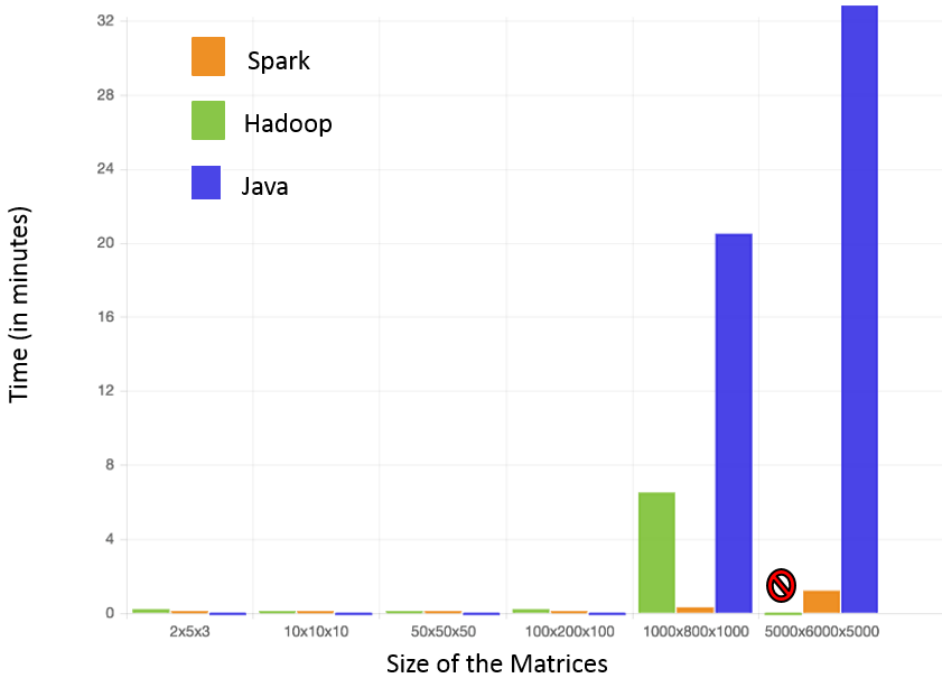


- To run the test for example; 100x200 and 100x200 go inside folder – matrix multiplication/4/

- To generate your own matrices of any sizes, refer to the source code on the following link: http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html

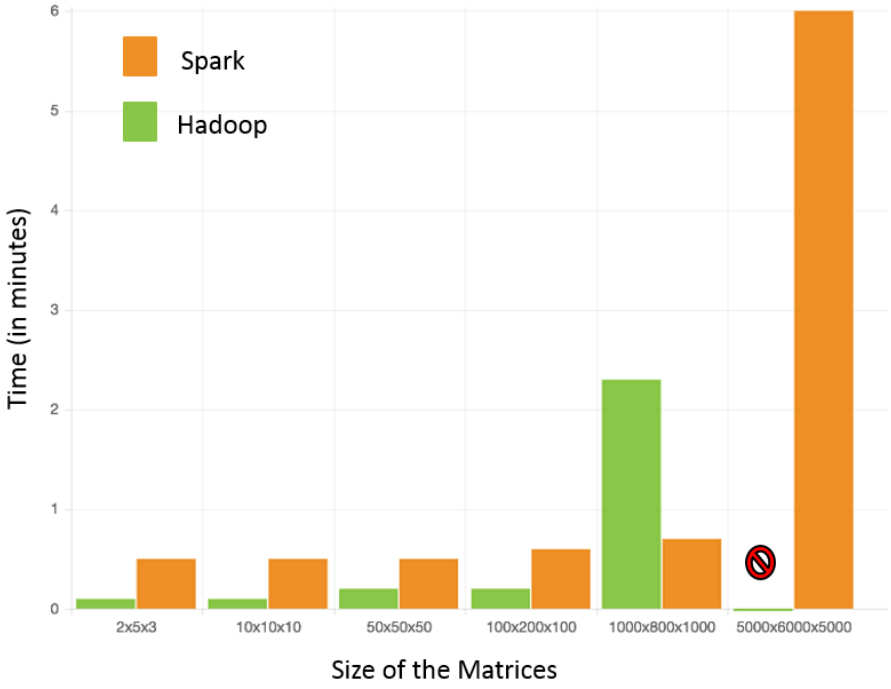
3.2.3 Results

- **Single node**



Observation: We observe that Java performs well when there is a sequential matrix multiplication and the size of the input files is not large. Spark also performs really well as it has great computational capability. Hadoop's performance is unsatisfactory due to less computational power.

- **Cluster**



Observation: We observe that the performance gets really better in a cluster. However, Hadoop crashes when the size of matrix is large due to less number of nodes in its cluster and its low disk-processing power.

3.3 K-Means Clustering

3.3.1 What is it?

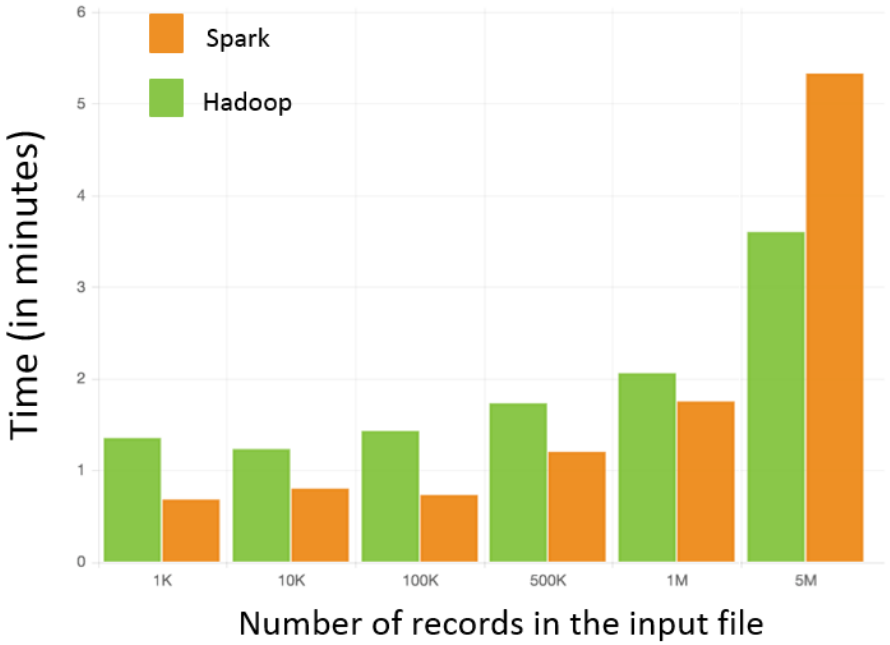
- As the name suggests, Clustering implies grouping items together. But on the basis of what?
- We use clustering in Big Data industry to group similar items/users together for better data management.
- It has numerous applications in the business and helps them to target their customers in a better way.
- K-Means algorithm is a simple, but widely used algorithm used for clustering. All objects need to be represented as a set of numerical features. Also, the user has to specify the number of groups (k) he wants

3.3.2 Files Used

- For this test we used files with different number of points, those points where randomly generated, each point value ranges from 5 to 100 as floats. Below are the files by number of records.
    - 1 K
    - 10 K
    - 100 K
    - 500 K
    - 1 M
    - 5 M

3.3.3 Results

- **Cluster**



Observation: We observe that Spark performs a little better for smaller files, but as the size of the file is huge (5 million records in a file), Hadoop performs better as it has good rate of data-processing for huge datasets.

4 Using our Recommenders

4.1 Hadoop

4.1.1 Co-occurrence Algorithm

In Hadoop, we used the co-occurrence algorithm using the Apache Mahout library. This type of algorithm is Item-based which means the items will be suggested to the user based on finding similar items to the ones the user already likes, again by looking to others' apparent preferences.

- In this algorithm, we start by creating a co-occurrence matrix. It is not as hard as it sounds!
- We begin by finding some degree of similarity between any pair of items. Imagine computing a similarity for every pair of items and putting the results into a giant matrix. This is called a co-occurrence matrix.
- This matrix describes associations between items, and has nothing to do with the users. It computes the number of times each pair of items occur together in some user's list of preferences.
- For example, if there are 17 users who express liking for both items A and B, then A and B co-occur 17 times.
- Co-occurrence is like similarity, the more two items turn up together, the more related/similar they are.
- The next step is to compute a user vector for each user. This vector tells which movie has a specific user watched and which ones has he not.
- The final step to get the recommendations is to multiply the co-occurrence matrix with the user vector.

4.1.2 Implementing Co-occurrence recommender

- We used the movie lens data for our recommender. Here is a link of the movie lens data: http://grouplens.org/datasets/movielens/
- The original format of the movie lens data is:
  userId *movieId* *rating* *timestamp*
- We convert* this format to the following format:
  userId,movieId

  ***Note** – The source code to convert from the movie lens format to our format can be found on the link:
  http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html
- We do this because our recommender in Hadoop is not rating-based, rather it is item-based. It only takes into account the movies watched by the user.

- In the next step, you would have to upload the input file with the users and the movies they have watched on the HDFS using the hadoop 'put' command as mentioned earlier in the manual.
- You can find the source code of our whole recommender on the link: http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html
- The command to run our recommender is as follows:

```
hadoop jar path_of_jar_file frogbdata.tcu.recommendereg.RecommenderJobRun
path_of_input_file_HDFS path_of_output_file_HDFS num_of_recommendations
```

4.2 Spark

In Spark, we used collaborative filtering algorithm using Apache MLlib library. According to Apache Spark documentation on MLlib collaborative filtering, "Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. ***spark.mllib*** currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.mllib uses the **alternating least squares (ALS)** algorithm to learn these latent factors."

To learn more about ALS algorithm, refer to this link.

4.2.1 Spark recommender Source Code

To get the source code of our recommender, follow this link:
http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html

Name of the file: MovieLensALSMain.py

4.2.2 Data Files for Spark Recommender

To get the data files, follow this link:http://grouplens.org/datasets/movielens/

Here, you will find different sizes of the real movie ratings (100K, 1M, 10M, 20M and 22M).

The format of the files are as follows:

- **100K: 100,000 ratings (1-5) from 943 users on 1682 movies.**
  - **ua.base**: Format: User_Id Movie_Id Movie_Rating Time_Stamp

```
1	1	5	874965758
1	2	3	876893171
1	3	4	878542960
1	4	3	876893119
1	5	3	889751712
1	6	5	887431973
1	7	4	875071561
1	8	1	875072484
1	9	5	878543541
1	10	3	875693118
1	11	2	875072262
1	12	5	878542960
```

- **u.item**: Sample Format

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?
Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?
GoldenEye%20(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-
exact?Four%20Rooms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-
exact?Get%20Shorty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?
Copycat%20(1995)|0|0|0|0|0|0|1|0|1|0|0|0|0|0|0|1|0|0
```

4.2.3 1M

1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.

- **movies.dat**: Sample Format

```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
5::Father of the Bride Part II (1995)::Comedy
```

- **ratings.dat:** Format User_Id::Movie_Id::Movie_Rating::Time_Stamp

```
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
1::1197::3::978302268
```

4.2.4 10M

This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

- **movies.dat:** Sample format

```
1::Toy Story (1995)::Adventure|Animation|Children|Comedy|Fantasy
2::Jumanji (1995)::Adventure|Children|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama|Romance
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children
9::Sudden Death (1995)::Action
10::GoldenEye (1995)::Action|Adventure|Thriller
```

- **ratings.dat:** Format: User_Id::Movie_Id::Movie_Rating::Time_Stamp

```
1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
1::292::5::838983421
1::316::5::838983392
1::329::5::838983392
1::355::5::838984474
1::356::5::838983653
1::362::5::838984885
1::364::5::838983707
1::370::5::838984596
```

4.2.5 20 M

It contains 20000263 ratings by 138493 users across 27278 movies.

- **movies.dat:** Sample format

```
1::Toy Story (1995)::Adventure|Animation|Children|Comedy|Fantasy
2::Jumanji (1995)::Adventure|Children|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama|Romance
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children
9::Sudden Death (1995)::Action
10::GoldenEye (1995)::Action|Adventure|Thriller
```

- **ratings.dat:** Format: User_Id::Movie_Id::Movie_Rating::Time_Stamp

```
1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
1::292::5::838983421
1::316::5::838983392
1::329::5::838983392
1::355::5::838984474
1::356::5::838983653
1::362::5::838984885
1::364::5::838983707
1::370::5::838984596
```

4.2.6 22M

It contains 22884377 ratings by 247753 users across 34208 movies

- **movies.dat:** Sample format

```
1::Toy Story (1995)::Adventure|Animation|Children|Comedy|Fantasy
2::Jumanji (1995)::Adventure|Children|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama|Romance
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children
9::Sudden Death (1995)::Action
10::GoldenEye (1995)::Action|Adventure|Thriller
```

- **ratings.dat:** Format: User_Id::Movie_Id::Movie_Rating::Time_Stamp

```
1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
1::292::5::838983421
1::316::5::838983392
1::329::5::838983392
1::355::5::838984474
1::356::5::838983653
1::362::5::838984885
1::364::5::838983707
1::370::5::838984596
```

4.2.7 Instructions

- To give the personalized ratings for any number of users, run the python script named "rateMovies" and follow the instructions: Keep in mind that the movies.dat file (movies.dat file should be the same file for which you want to run the recommender) should be in the same directory as the python script. This will generate the file called "personalRatings.txt" that contains the personal ratings of the user.
- **Note: To start the recommender on a cluster of 2 nodes, run the following commands:**

start-dfs.sh

start-yarn.sh

hadoop fs -mkdir -p /user/username1/spark_recommender/

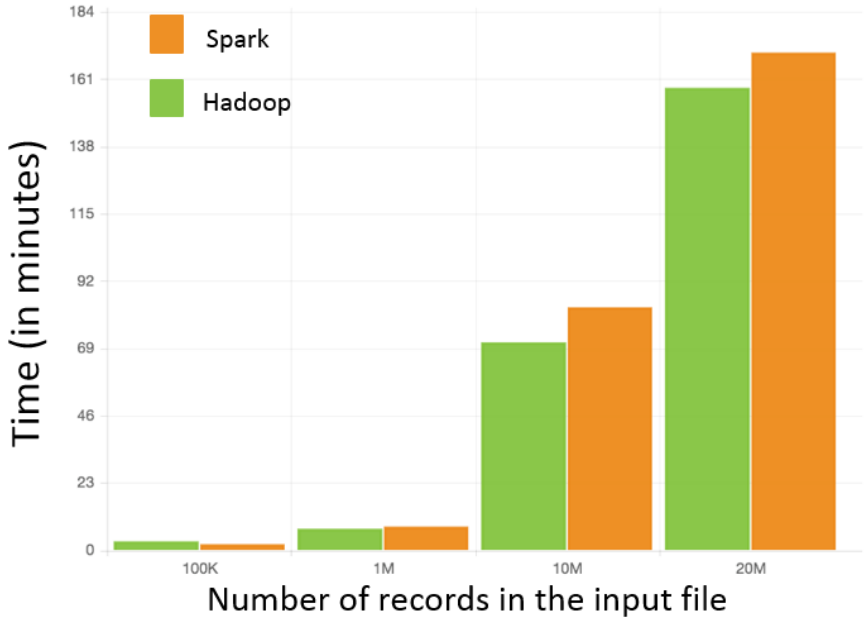
hadoop fs -put /home/username/Downloads/ml-latest/ /user/username1/spark_recommender/ml-latest/

hadoop fs -put /home/username/Desktop/personalRatings.txt/ /user/username1/spark_recommender/personalRatings.txt

./spark/bin/spark-submit --master yarn-cluster --num-executors 5 --executor-cores 1 --executor-memory 3G /home/username/workspace/MovieLensALS/src/MovieLensALSMain.py hdfs://HadoopSMaster:9000/user/username1/spark_recommender/ml-latest/ hdfs://HadoopSMaster:9000/user/username1/spark_recommender/personalRatings.txt hdfs://HadoopSMaster:9000/user/username1/spark_recommender/output22M

4.2.8 Recommender results

- **Cluster**



Observation: We observe that both Hadoop and Spark took almost the same amount of time for various sizes of input files. The performance of both the frameworks can be much better if you have more than 2 nodes in your cluster and better algorithm implementation.

5 Glossary of Terms

Apache Hadoop: Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets.

Apache Hadoop Yarn: YARN (Yet Another Resource Negotiator) is a cluster management technology. It is characterized as a large-scale, distributed operating system for Big Data applications

Apache Mahout: An Apache software used to produce free implementations of distributed scalable machine learning algorithms that help in clustering and classification of data.

Apache Maven: A build automation tool for projects that uses XML to describe the project the project that is being built and its dependencies on other external modules.

Apache Spark: Apache Spark is an open source cluster computing framework which allows user programs to load data into a cluster's memory and query it repeatedly.

Big Data: Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions

Collaborative Filtering: Method to predict the interests of a user based on interests of other users.

Co-occurrence algorithm: Counting the number of times each pair of items occur together and then predicting the interests of a user based on the user's previous interests and most co-occurred items.

HDFS: Hadoop Distributed File System is a Java based file system that provides scalable and reliable data storage.

IDE: Integrated Development Environment.

K-means clustering: A way of vector quantization used for cluster analysis in data mining.

Map Reduce: A programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

MLlib: Apache Spark's scalable machine learning library that consists of common learning algorithms and utilities including classification, clustering, filtering etc.

PyDev: A Python IDE for Eclipse which is used in Python Development.

Root Access: Access to install various software and related items on Linux machines.

Scala: A programming language for general software applications.

XML: XML stands for Extensible Markup Language that defines the protocol for encoding documents in a format that is both, human and machine-readable.