Comparative Analysis of Collaborative Filtering on GraphLab, MLlib and Mahout

Abdul Samad[1], Dr. Saif-ur-Rahman[2]

[1,2]*Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST) Karachi, Pakistan*

[1]abdulsamad22@hotmail.com

[2]saif.rahman@szabist.edu.pk

Abstract—Recommendation systems are used to recommend items or products to the user based on their previous purchases, visits, interests, ratings, wish-lists or reviews to develop interest and to display the accurate and suitable items on board. Recommendation systems are used in various online shops (E-Commerce application) and decision making systems. Recommendation is a particular form of information filtering. It falls under the Data Mining and Machine Learning. Collaborative Filtering is the key technique used in this system. In this study, the data loading, model generation, recommendation implementation and accuracy of same algorithm on some major tools and libraries (GraphLab, Mahout-Hadoop, Mahout-Spark and MLLib) has been discussed. To serve the purpose, a well-known algorithm Alternating Least Square ALS for collaborative filtering was used. Netflix Prize (training) data set was used in this research with the listed tools and libraries. At the end of this research a factual comparative analysis of the tools was carried out.

***Keywords*—Recommendation System, Machine Learning, Collaborative filtering**

I. INTRODUCTION

Recommendations make decisions easy and fruitful. People are confused occasionally when buying variety of items for example CDs, Books, Electronic Items, Mobile Phones, Grocery and etc.. They need user reviews, ratings or proper representation of the top selling items. Here, recommendation system becomes handy which provide users with relevant, rated or reviewed items similar to the users searched item of interest. There are tools and libraries available that implement accurate recommendation systems with known algorithms which heavily use modern CPU architecture and works with Disk or in Memory based read/writes and designed for clusters of computers for distributed processing of large scale data. Model generation and Data load time of these tools are variable with respect to their data models and efficiency of algorithms. Tools that have been selected for this study are based on Hadoop file system and selected works with in-memory RDD based processing. Hadoop file system achieves fault tolerance with the help of replication mechanism over the distributed nodes [1]. While discussing about in-memory based processing that is Apache Spark which is used by MLlib and Mahout, the fault tolerance is achieved by Lineage mechanism or recovers lost data sets over the distributed nodes [2]. Following are the approaches to achieve recommendations.

1) Collaborative filtering
2) Content based filtering
3) Hybrid Recommendations

The point of discussion in this research is collaborative filtering and the methodologies, tools and their performance, data models and algorithms to achieve accuracy. Some of the tools that have been selected in this study are using iterative algorithmic approach during execution and data model generation. Iterative Algorithms are used to solve certain problems in a cyclic manner to improve or boost its relative results. It is a mathematical approach to nonlinear or at times linear equations. It is also known as Convergent that is to merge the results of cyclic operations with the intermediary results of the operations performed and stops at the point which is approximate to the given initial predictions or approximations. Direct-method is used to solve specific definite solution of the given equations and there are finite steps involved in its solution [3]. In this study, single ALS (Alternating Least Count) algorithm has been selected to be used on tools.

II. COLLABORATIVE FILTERING

Collaborative filtering is a technique used in recommendation systems which facilitate to produce a list of recommended items for the user with proper representation. In collaborative filtering, the predictions are made from users to users' basis or users to items basis i.e. certain user has reviews regarding an item and the other user may also have the same review for that item. The reviews are collaborated and the recommendation accuracy increases as a result of this collaboration. On the other hand, if various user rates an item with good points then its recommendation accuracy will increase and it is in the list of recommended items matched with the searched items. Automatic predictions can be made in collaborative filtering techniques by comparing the interests of similar users to each other [4].

A. Problems

There are some problems or challenges in collaborative filtering. Few of them are discussed below.

1) Sparsity (cold start): User to Item matrix i.e. When new items are added in system, they have no ratings, reviews and purchases. This cause sparsity (sparse matrix) in recommendations and the users are confused to purchase such items. If they belong to known brand then they have been purchased; otherwise, marketing and comparisons are required to make user interested in those items. No ratings, reviews and no purchase history may damage the recommendation accuracy and make data model sparse [5-6].

2) Scalability Issue: A Data-set and user rating increases with time gradually which may take long processing times. More processing power need highly configured CPU and RAM consumption and all the clusters are so busy that extra hardware is added at runtime to maintain the processing accuracy which is costly and need maintenance [6].

3) Biased Ratings from users: Users rate their liked products and brands they use and rate them on the basis of their experiences and some of them gives low rating and reviews to their opponents [6].

The scalability issues is the point of discussion to increase the performance on large data-sets and to find better tools and libraries available for these issues with the help of an activity on local system. To produce benchmarks for the interested communities, business use cases.

III. GRAPHLAB

GraphLab is the high performance framework designed for distributed and parallel processing. It is an open source project by Apache. Originally, it was designed for Machine Learning or Data Mining task for asynchronous shared memory distributed environment. It works in an iterative manner. As the amount of data is growing rapidly and needs high computing power; therefore, Multi Core CPUs, Clouds, and Clusters are developed so that data may fit on more than one node (Graph) and the distributive/parallel algorithms can be applied on that large scale data. GraphLab provides application programming interface for rapid deployment of algorithms for distributed environment i.e. for machine learning written in C++ which enhanced the execution of Graphlab programs, Multithreading and disk IO usage. Number of machine learning and data mining toolkits are also integrated in framework [7]. TCP/IP is used for inter communication of the clusters of computers over the network. Message Passing Interface MPI is a standard which is used in distributed/parallel processing to manage or launch GraphLab programs. HDFS direct access and reads writes are allowed in GraphLab. Every process or program in graphLab is multithreaded so that it can use all multicores available in modern CPU architectures. (Figure 1)

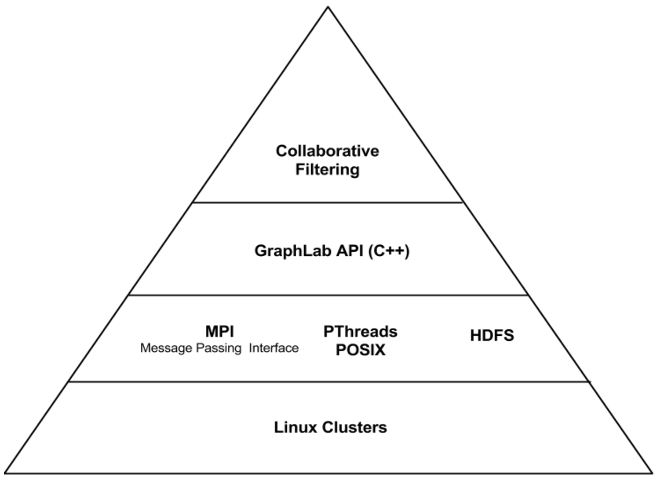


Fig. (1). GraphLab Architecture Diagram [7]

IV. DATA MODEL IN GRAPHLAB

There are vertices and edges in data graph which holds data that is mutable. In graphLab every vertex and edge is associated with user data. GraphLab executes programs on vertices in parallel; each vertex can access nearby edges and vertices and can initiate other nearby vertex programs.GraphLab vertex programs are divided in three main phases which is Gather Apply and Scatter (GAS) Model also known as GAS vertex programs. In Hadoop there are MapReduce stages through which data passed through result calculated but in GraphLab this MapReduce stages are mapped on Gather, Apply and Scatter phases in such way that Gather and Scatter works similar to Map state and Apply stage works similar to Reduce[7].

A. GAS Model

1) Gather Phase: In this phase, a gather (vertex, edge) function in vertex class which is called on each edge linked to nearby vertex which gathers values from each edge to the connected vertex. Small squares (M) are the messages passed to nearby connected active vertices along edges. White vertices are the active vertices. Figure 2 demonstrates the activity performed in gather phase.

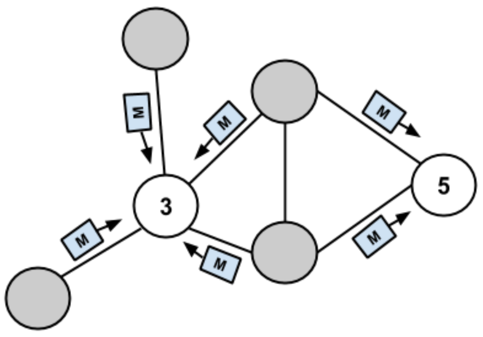


Fig. (2). GAS Gather Phase

B. Apply Phase

Apply (vertex, total) function written in vertex class is executed in this phase which apply the summation of gathered data from the edges on nearby connected active vertices so that it update its value. Figure 3 demonstrate the activity performed in apply phase.

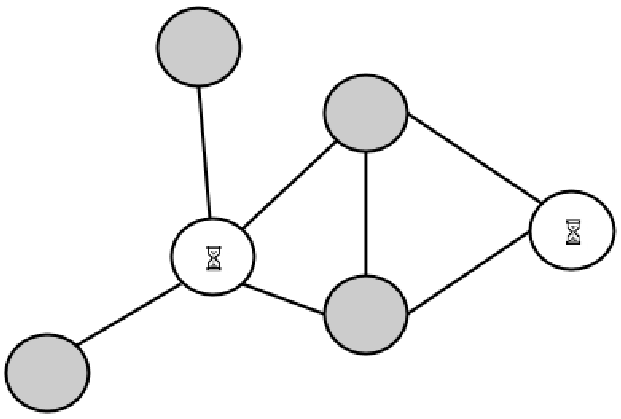


Fig. (3). GAS Apply Phase

C. Scatter Phase

Once again, scatter (vertex, edge) function written in vertex class is called on each edge to update its neighboring vertices from the processed value by apply phase. Figure 4 demonstrates the activity performed in scatter phase.

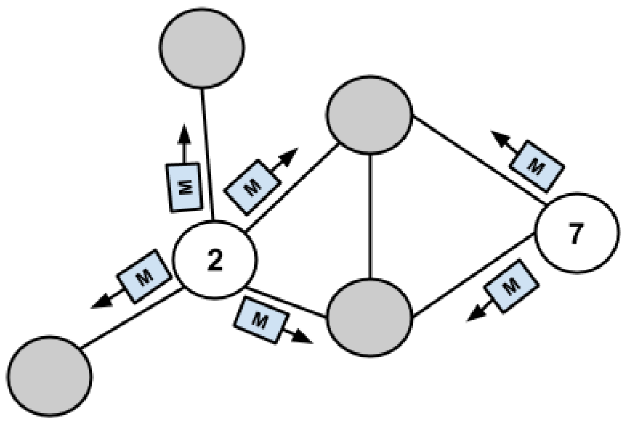


Fig. (4). GAS Scatter Phase

V. APACHE HADOOP

The open source Apache Hadoop framework used on clusters of computers for distributed processing of large scale data and designed in such a way that it scale up to many server in live mode. Hadoop framework contains modules such as Hadoop Common package, Hadoop Distributed File System (HDFS used on machines in a cluster to store data and provide high network bandwidth across the cluster), MapReduce (Data processing model) and Yarn (Resource manager for clusters). These abstractions are designed or defined in such a way that Hardware failures are common and the failures are detected at the application layer automatically which is handled by the framework so that delivering service availability more rather than hardware availability.

VI. MAPREDUCE

In depth there are some other steps involved in MapReduce task.

1) Data Input
2) Data Splitting
3) Data Mapping
4) Data Shuffling
5) Reduce
6) Result

At data input phase, large data set is inserted into the system then it enters in Data Splitting phase where it is divided into many data blocks which are stored on commodity machines (Nodes) with HDFS then Map function is applied on that data chunks where it is sorted or shuffled (Data Shuffling phase) then Reduce function is applied on that sorted data for processing. After that final result, set is reconstructed from all data processed chunks. Figure 5 below explains the data flow in Hadoop MapReduce.

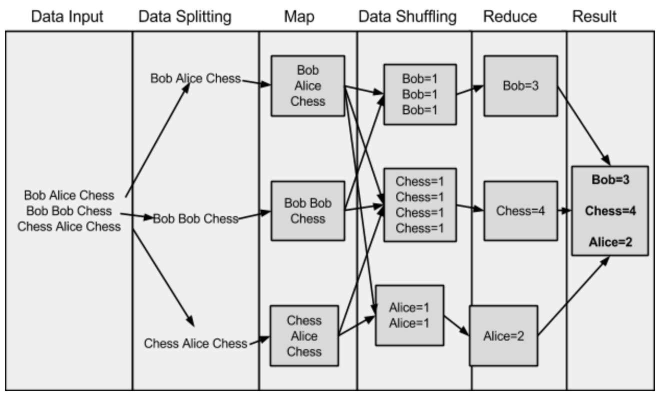


Fig. (5). Apache Hadoop

VII. APACHE MAHOUT (HADOOP)

Mahout is a set of scalable machine learning java libraries focused on collaborative filtering, clustering and classification subproject of Apache Lucene started in 2008. Most of its implementation uses Apache Hadoop (MapReduce). It provides libraries for math operations like linear algebra and statistics. Single/Multi-Node or non-Hadoop clusters are allowed to work with its implementations. It supports business use-cases as it is free under Apache software license, scalable for large data sets that may run on single or multiple machines (nodes). Works on the top of Hadoop implementation, that is in key-value pairs model [8].

A. Apache Mahout (Spark based Implementation)

Previously, Mahout was a machine learning library for Hadoop but now Mahout team works on Spark to allow Mahout for Spark and for this purpose, they have introduce Scala based DSL (Domain Specific Language) with algebraic expressions added to work with Mahout. Scala Language supports operator overloading and functional programming which support to build distributed code with linear algebra on Scala shell which is customized to work with Mahout to run Spark on it with defined Mahout DSL [8].

VIII. ALTERNATING LEAST SQUARE

Alternating least square falls under the umbrella of Least Squares and the goal of least squares is to find the evaluation of the parameters of a function f (x) on data set A1 , , , , , An . Least Squares method is the standard approach to find the close solution of over-determined systems. Over-determined systems the systems in which the number of known equation is more than unknown equations. Let a system have 2 unknown variables and 3 known equations [9] as shown in figure 6.

$$2x1+x2=-1$$

$$-3x1+x2=-2$$

$$-x1+x2=1$$

Fig. (6). Equation: Alternating Least Square

Least Square Method is sub-divided in two parts

1) Linear Least Squares
2) Non Linear Least Squares

Further, Least Squares can be classified in weighted least squares (WLS), alternating least squares (ALS), ordinary least squares (OLS) and partial least squares (PLS).

IX. ALS ALGORITHM

ALS uses the sparse rating matrix R which holds the ratings given by the users to movies. It builds a linear data model users (U) and movies (M) low dimensional matrices from that sparse rating matrix. Rating Matrix R is the product of multiplication of low-rank User (U) and Movies (M) latent factors. ALS is an iterative algorithm and it minimizes the errors of summation every iteration. In every iteration, algorithm fixes one of the latent factor matrixes and solve for other by least square method and this process continues until it converges as show in figure 7.

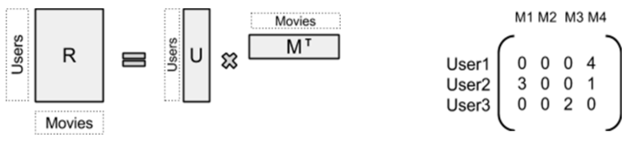


Fig. (7). ALS Algorithm

X. APACHE SPARK

Spark is an open source framework for data analytics on clusters of computers. It is used for large-scale data processing i.e. Big Data and builds on Hadoop Distributed File System (HDFS) and it also provides advance execution of graphs with in-memory processing and data storage (RDD, Resilient Distributed Data-sets). It has many advantages over MapReduce execution stages and enhances performance up to 100 times faster than Hadoop [10]. Apache Spark process data with in-memory rather than on disk to save IO of disk read write cost and network bandwidth during data processing on different nodes within the cluster of computers with multi- pass Iterative algorithms that are required to be common on machine learning or graph processing tasks as shown in figure 8. In Hadoop, HDFS is an intermediate storage used to save the processed data set for replication at the completion of every state and read that output back in immediately to start the new state and fault tolerant is achieved by this replication mechanism over the machines in network. This is the overhead in between all these stages, similarly the interactive queries, each query have to face this network overhead and disk I/O and the distributive file system is slow by this disk I/O but it is required to achieve fault tolerance. In contrast to Apache Spark, the data will be processed in Memory or cached on disk drive that is locality of reference for faster access of data reads. Apache Spark works on Resilient Distributive Dataset (RDD) as HDFS in Hadoop for intermediate storage but no overhead is created like network or disk I/O in Hadoop. Fault tolerance is achieved in Spark by using Lineage, a terminology which describes the fault tolerance in such a way that on failure recomputed the lost partitions until success at certain timeout interval. If there is no lost then no cost for re-computation. Questions arises here that if any memory leakages are found in any of the nodes then what to do with the lost RDDs [2, 10].

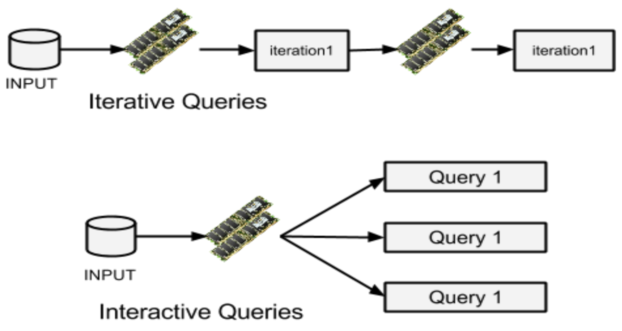


Fig. (8). Apache Spark Work Flow

XI. MACHINE LEARNING LIBRARY (MLLIB)

MLLib is a library for machine learning algorithms and utilities used with spark to support machine learning problems like classification, regression, clustering, collaborative filtering, dimensionality reduction and optimization primitives but MLlib was used and discussed with respect to collaborative filtering (ALS Alternating Least Square). MLLib is a standard component of Spark which provides machine learning primitives on Spark. MLlib provides Application Programming Interface in Scala, Java and Python languages currently. It executes 100x faster programs in memory than Hadoop MapReduce, but on disk it executes 10 times faster.

XII. NETFLIX DATA SET

To optimize or enhance the existing recommendation algorithm of Netflix, a competition for data mining named Netflix prize was held on large scale data set of their own DVD rental system. The goal was to improve the accuracy of the recommendations or predictions of users ratings of movies data format = user ID, movie ID, ratings. Netflix training data set consists of 2 files:

1) smallnetf lixm m.train

2) smallnetf lixm m.validate

There were some challenges in Netflix data set such as the data volume and data limit. Data volume of this training set is small composed of user to movie ratings which spans of around 52M B disk size i.e., (smallnetf lixm m.train : 44.1M Bandsmallnetf lixm m.validate : 7.4M B) which on processing phase consumes high CPU usage so that it should be run on modern multicore CPU architecture with sufficient amount of RAM (min4GB). This training data set is limited but sufficient and accurate for the prize competition activity. Small Netflix data set must be altered according to the tool. While using with GraphLab, both files were in space separated and while using with Apache Mahout, they were in tab separated.

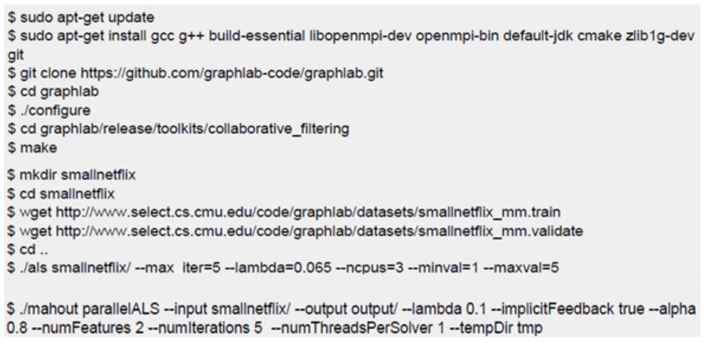
XIII. EVALUATION

A. Installation Machine Specs

LENOVO B5400 machine with Ubuntu latest version was used to install all the selected tools and it has Intel Core-i5 2.50GHz processor and 8 GB of system Memory. Prerequisites are necessary for the installation and proper execution.

B. GraphLab Installation and Execution Steps

There is an official GIT repository of GraphLab from which a checkout on local machine was made. Commands listed below are used for git checkout. Before checkout, all prerequisites for GraphLab execution were install as shown in figure 9 below. Data set that had been used in this execution is same as used in Apache Mahout.



```
$ sudo apt-get update
$ sudo apt-get install gcc g++ build-essential libopenmpi-dev openmpi-bin default-jdk cmake zlib1g-dev
git
$ git clone https://github.com/graphlab-code/graphlab.git
$ cd graphlab
$ ./configure
$ cd graphlab/release/toolkits/collaborative_filtering
$ make
$ mkdir smallnetflix
$ cd smallnetflix
$ wget http://www.select.cs.cmu.edu/code/graphlab/datasets/smallnetflix_mm.train
$ wget http://www.select.cs.cmu.edu/code/graphlab/datasets/smallnetflix_mm.validate
$ cd ..
$ ./als smallnetflix/ --max  iter=5 --lambda=0.065 --ncpus=3 --minval=1 --maxval=5

$ ./mahout parallelALS --input smallnetflix/ --output output/ --lambda 0.1 --implicitFeedback true --alpha
0.8 --numFeatures 2 --numIterations 5  --numThreadsPerSolver 1 --tempDir tmp
```

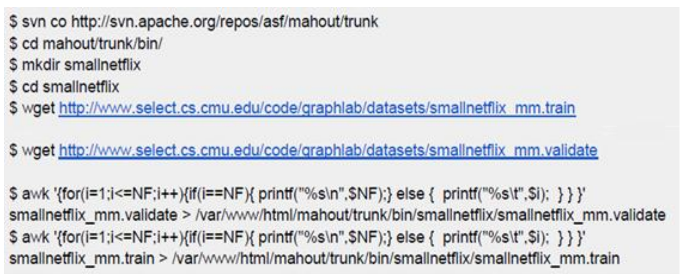
Fig. (9). GraphLab Installation and Execution Steps

C. Observations of GraphLab Execution

GraphLab execution involves Remote Procedure Call (RPC) connections with local machine by obtaining its IP and subnets to use Message Passing Interface (MPI) for data flow during execution. Firstly, it establish the connection (TCP layer) with the host machine successfully then it creates a cluster depending on the machine connected as in this study only one after this successful. Connection started loading the graph from the given input files smallnetf lixm m.train and smallnetf lixm m.validate and read it line by line, then it display graph loading time 16.3477 seconds in this execution which is important and vary with respect to input parameters in execution command and CPU architecture of the machine. It then passes the data in graph and allocating memory for the execution and start ALS iterations on that data in graph producing RMSE (Root Mean Square Error) with respected time periods in seconds.

D. Apache Mahout Installation and Execution Steps

Installation of Apache Mahout required any Linux (Ubuntu) machine with GIT installed on it to checkout from its official GIT repository. Small Netflix train and small Netflix validation files needed to be converted in Tab separated user I D < Tab > item I D < Tab > Ratings by Ubuntu console as shown in figure 10 below.



```
$ svn co http://svn.apache.org/repos/asf/mahout/trunk
$ cd mahout/trunk/bin/
$ mkdir smallnetflix
$ cd smallnetflix
$ wget http://www.select.cs.cmu.edu/code/graphlab/datasets/smallnetflix_mm.train

$ wget http://www.select.cs.cmu.edu/code/graphlab/datasets/smallnetflix_mm.validate

$ awk '{for(i=1;i<=NF;i++){if(i==NF){ printf("%s\n",$NF);} else {  printf("%s\t",$i);  } } }'
smallnetflix_mm.validate > /var/www/html/mahout/trunk/bin/smallnetflix/smallnetflix_mm.validate
$ awk '{for(i=1;i<=NF;i++){if(i==NF){ printf("%s\n",$NF);} else {  printf("%s\t",$i);  } } }'
smallnetflix_mm.train > /var/www/html/mahout/trunk/bin/smallnetflix/smallnetflix_mm.train
```

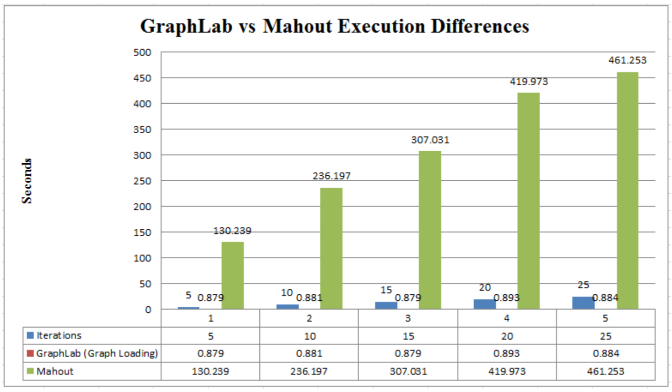
Fig. (10). Apache Mahout Installation and Execution

E. Observations of Mahout Execution

Mahout is observed with respect to Hadoop so Map and Reduce task are involved in this execution. When the command is executed on console, it finds the mahout local and Hadoop configuration directory first class path. If no Hadoop configuration directory is defined then it runs locally on the machine. Mahout then loads the input files smallnetf lixm m.train and smallnetf lixm m.validate from the given path in command and launches the Map tasks to split the data and create jobs to process those chunks and create the item ratings partial files by Reduce function. It merge the data in tmp folder and completes the jobs one by one while maintaining fault tolerance by replication in run-time and took total time to complete 133062ms (Minutes : 2.2177).

F. GraphLab vs. Mahout Execution Differences

GraphLab and Apache Mahout Execution samples have been taken by fixing the number of maximum iterations at the interval of 5 and fixing lambda to 0.065 with 3 CPU cores so that 5 separate readings are noted down for each tool as shown in Figure 11. It should be noted that execution of each tool was perform at a time on installation on machine so that correct reading should be noted otherwise false reading would be produced by the system.



	1	2	3	4	5
🟦 Iterations	5	10	15	20	25
🟥 GraphLab (Graph Loading)	0.879	0.881	0.879	0.893	0.884
🟩 Mahout	130.239	236.197	307.031	419.973	461.253

Fig. (11). GraphLab Execution Steps

XIV. CONCLUSION

Certain popular and useful tools have been studied in this paper. These tools were installed and Netflix Prize data set was used with different formatting with respect to each tool. The data set was executed and results were observed. The findings include time to complete full task and data load time or RMSE in some tools. There are fewer studies available that are based on comparative analysis which covers all major tools with data models and flow details with known Netflix Prize data-set with same Alternating Least Count Algorithm. This study is imperative to the companies or individuals who are interested to build systems which can provide recommendation feature with Collaborative filtering in terms of hardware cost, processing power needed, disk I/O, network overhead, maintenance, usability and deployments. Tools discussed in this study are all comprehensively usable in the industry with worthy community support and examples available online. Some tools works in Memory while others uses disk based or graph based approach to process data and implement fault tolerance with respect to architectural support and methodologies. Distributed processing over the nodes in clusters is easy with the help of these tools and every tool have specific feature with respect to scenarios occurred in real world.

REFERENCES

[1] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. “Distributed graphlab: A framework for machine learning and data mining in the cloud”. In *Proceedings of VLDB Endowment*, 2012, vol. 5, no. 8, pp: 716-727.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing”. *In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI’12)*, 2012, pp: 2-2.

[3] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[4] B. Sarwar, G. Karypis, J. Konstan and John Ried. “Item-based collaborative filtering recommendation algorithms”. In *Proceedings of the 10th International Conference on World Wide Web, (WWW ’01)*, 2001, pp: 285-295.

[5] M. Grcˇar, D. Mladenicˇ, B. Fortuna and Marko Grobelnik. “Data sparsity issues in the collaborative filtering framework”. In *Proceedings of the 7th International Conference on Knowledge Discovery on the Web: Advances in Web Mining and Web Usage Analysis (WebKDD’05)*, 2006, pp: 58-76.

[6] Collaborative filtering (2014), Wikimedia Foundation, [Online]. Available: http://en.wikipedia.org/wiki/Collaborative_filtering

[7] GraphLab Inc. Graphlab. 2014.

[8] The Apache Software Foundation. Apache mahout. 2014.

[9] G. Taka´cs and D. Tikk. “Alternating least squares for personalized ranking”. In *Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys ’12)*, 2012, pp: 83-90.

[10] The Apache Software Foundation. Apache spark, mllib. 2014.