ITEM-BASED RECOMMENDATION ALGORITHM USING HADOOP

by

Chetan Prakash Somani

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 12, 2015

Keywords: Hadoop, HDFS, MapReduce, Recommendation Systems, Collaborative Filtering, Item-based Similarity, Distributed Computing, Cloud Computing, Cluster

Approved by

Xiao Qin, Chair, Professor of Computer Science and Software Engineering
Daniela Marghitu, Professor of Computer Science and Software Engineering
Jeffrey Overbey, Assistant Professor of Computer Science and Software Engineering

Abstract

Recommendation systems are used to provide solutions to the problem of making personalized recommendations. They are achieving widespread success in E-commerce, social networking and advertisements. In E-Commerce, day-to-day growth of customers and products poses key challenges for recommendation systems as they are responsible for producing high quality recommendations. In addition, they are even needed to perform many recommendations per second, for millions of customers and products. Hence, new recommendation system technologies are required to produce high quality personalized recommendations. Implementing a recommendation algorithm using a sequential approach for a large dataset has large performance issues. We address the performance issues by implementing a parallel algorithm to derive recommendations by using Hadoop map-reduce framework along with an item-based similarity collaborative filtering technique. Map-Reduce is a programming framework used for processing and generating large datasets. In map-reduce framework, input and output is represented in terms of key-value pairs. Users specify a map function that processes a key-value pair to generate a set of intermediate key-value pairs, and a reduce function merges all intermediate key-value pairs associated with the similar intermediate key to produce final output key-value pair. Similarity among item pairs followed by deriving recommendations is computed using map-reduce programming approach. Similarity among item pairs is configured by finding out the similarity ranking which is calculated by using different similarity measures. The item ratings with highest degree of similarity with any given item are given highest priority for recommendation for the given item. Map jobs are responsible for gathering the information from the input dataset and then, compute relationship among items on multiple nodes in parallel to generate item pairs and reduce jobs combine all item pairs from all nodes to generate the recommendation list by computing the

similarity among the items using similarity measurement techniques. In this study, we will focus on item-based collaborative filtering technique, which is a well known technique used in recommendation systems using Hadoop map-reduce framework. They are two collaborative filtering methods: User-based and Item-based collaborative filtering methods. User-based collaborative filtering focus on computing relationship among users i.e. they find out how similar two users are and based on their similarity recommendations are made. Item-based collaborative filtering focus on computing relationship among items i.e. they find out how similar two items are and based on their similarity recommendations are made. Experiments prove that the implementation of item based collaborative recommendation algorithm on Hadoop using map-reduce framework has higher degree of performance with the increase in number of nodes within a cluster when compared to the results of implementation in a single node cluster.

Acknowledgments

This thesis would not have been completed without invaluable guidance, experience sharing, constant support and encouragement from my adviser, people in our research group and family members during my study at Auburn University.

First and foremost, I offer my sincerest gratitude to my adviser, Dr. Xiao Qin, who has supported me throughout my thesis with his patience and knowledge while allowing me the room to work in my own way. I attribute the level of my Masters degree to his encouragement and effort and without him this thesis would not have completed or written. One simply could not wish for a better or friendlier adviser.

I am also grateful to Dr. Daniela Marghitu and Dr. Jeffrey Overbey for serving as members of my advisory committee. I would like to thank the Department of Computer Science and Software Engineering and Auburn University for providing such great resources and facilities.

A very special thanks goes out to my research group members without their support and help, I would not have been able to finish my research. I doubt that I will ever to able to convey my appreciation fully, but I owe everyone of the group my eternal gratitude.

Finally, I would like to thank my family for the love, encouragement and support they provided me through my entire life to achieve my goals.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, recommendations have become extremely prevalent and are applied in variety of applications. Applications can be confined to movies, music, news, books, research articles, social tags, and e-commerce products. The process of recommendation algorithms for finding recommendations starts by finding a set of items, purchased and rated by users which overlap with the other set of items which have been purchased and rated by the similar user. The algorithm then aggregates and eliminates items the user has already purchased or rated, and then recommends the remaining items to the user which have not yet being purchased or rated. In order to generate accurate recommendations, the system have to process large datasets with required information. As greater the information being analyzed, greater the accuracy of the results being generated. Applications which are data-intensive need to access datasets ranging from a few gigabytes to several terabytes or even petabytes. For example, Google, uses the map-reduce model to process twenty petabytes of data per day in a parallel fashion. The map-reduce programming framework simplifies the complexity of running parallel data processing functions across multiple computing nodes in a cluster, as scalable map-reduce helps programmers to distribute programs across nodes in a cluster and have them executed in parallel. Map-Reduce automatically gathers the results generated across multiple nodes and returns a single result or a set of results. In addition, the map-reduce platform offers fault tolerance entirely transparent to programmers. Even if one of the node in the cluster breaks, the processing of the task is not effected. Map-Reduce is a programming model for parallel data processing in high-performance cluster computing environments.

1.1 Recommendation Systems

Recommendation systems are part of information filtering system which aims at predicting the preference the user would give to an item. Recommendation systems have changed the way websites interact with their users. They eliminate the static experience, in which users search for static information for potentially buying products, by increasing interaction among users to provide rich user experience dynamically. Recommendation systems derive recommendations for each individual user based on their past purchases, searches and on other users' purchase and search behaviors. Recommendation systems personalize the experience of each user by randomizing content that is particularly relevant to their experienced interests, instead of providing a static experience to every user. Recommendation systems can be extremely effective on large scale if they are implemented correctly. Many of the world's top used websites, such as Facebook, Twitter, LinkedIn, Amazon etc use recommendation systems to engage their users with relevant content.

1.2 Collaborative Filtering

Recommendation systems use knowledge discovery techniques to provide solution to the problem of making personalized recommendations. They solve the problem by taking in account the users past experiences, such as music they have listened or rated, articles they have read or products they have purchased or movies they have watched or rated to identify potential user preferences. Recommendation algorithms are software techniques providing suggestions for items that would be of an use to a specific user. Suggestions such as what music to listen, what products to buy, what movies to watch or what news articles to read. Many algorithmic approaches have been applied to the problem of making accurate and efficient recommendation systems. Collaborative filtering is one of among many algorithmic approaches.

Collaborative filtering is the most successful recommendation system technique till date. It is used in many of the most successful recommendation systems on the web. Collaborative filtering systems recommend products to a target user based on the behaviors of other users. Collaborative filtering systems uses statistical techniques to find a set of users known as neighbors, that have a similar experiences as that of the target user i.e. either they have rated different products similarly or they tend to buy similar set of products or either they have watch same set of movies or listened to similar kind of music. Once a neighborhood of users is formed, the system uses several algorithms to derive recommendations. In order to understand the algorithm and the recommendation process, its good to introduce basic terms and then get familiar with approaches, methods, tasks, challenges and evaluation of recommendation systems.

Items and users are the two important entities engaged in every recommendation system. An item refers to any product such as a music song, a movie, a product, an article that recommendation system is to recommend. User is a person ready to accept recommendations when providing opinions about various items. The goal of collaborative filtering algorithms is to either make suggestions about new items or to make prediction about the acceptance of a certain item for recommendations when providing opinions about various items. In addition, it even aims to either make suggestions of new items or to make prediction about the acceptance of a certain item for a particular user based on users past experiences and similarity with others users. Prediction is a numeric value expressing the affinity of an item for the active user. Recommendation is a list of items that the active user will like the most. This is also known as top N recommendations where the list contains top N liked items.

1.3 Hadoop Map-Reduce Framework

Map-Reduce is a programming framework designed for processing large amounts of data, in parallel, by dividing a complete task into a set of independent tasks where each independent task performs the similar computation. The data in the map-reduce framework

is not shared across the nodes. Instead, the data elements in the map-reduce are immutable i.e. the data once written cannot be written twice and it can only be read many times. All the data used as an input and the resultant data post processing is stored in HDFS (Hadoop Distributed File System). The data read from the input files, stored in HDFS are processed and converted into intermediate results and are further processed to generate final results.

Map-Reduce programs process the input data into two steps: Map and Reduce. In the map step, the mapper takes in one element at a time from input list of data elements fetched from the HDFS. It then, performs the required computation as defined within the map step on the input data element and emits the required output to an intermediate output data element. As the input file set is first split into several sets called file splits or input splits before assigning it to a mapper, all the map operations are paralleled as every mapper would have exactly one input split. Thus, the number of mappers created is dependent on the number of input splits. Splitting the input dataset into smaller datasets helps in paralleling the processing. In turn, as each mappers performs computation on its own dataset, they do not have to synchronize with one another and are independent of one another. Every mapper that receives the input split processes the data into a specified format.

The Record Reader known as input split parser in the mapper parses the input split and generates the required key-value pairs as each mapper takes in input in the key-value format. All the key-value pairs are processed in parallel by the mappers, to generate intermediate key-value pairs. The output key-value pair of the mapper serves as input to the reducer. Once the map step is complete, the intermediate key- value pairs are shuffled between nodes in a cluster to send all values with the same key to a single reducer. The reducer receives the intermediate key-value pairs generated by the mapper as input, combines the values of all key-value pairs and generates a single output key-value that is unique to each other. Reducers are similar as mappers in and generates a single output data corresponding to the input data fetched by the mapper. Reducers are independent of one another and they do not have to communicate with each other as that of mappers.

1.4 Contribution

In this research, item-based collaborative filtering algorithm, a specific type of collaborative filtering algorithm is researched and a new parallel approach for item-based collaborative recommendation algorithm is proposed for computation of item similarity. The item-based collaborative filtering algorithm is implemented using Hadoop map-reduce framework by writing two map-reduce jobs to derive the recommendation list of items. The proposed algorithm uses the Pearson Coefficient, Cosine Similarity, Jaccard Similarity and Tanimoto Similarity to find similarity among items and based on the similarities, the most accurate recommendations for any type of data is found with increased performance and also increased load.

The entire dataset is first pre-processed to identify the relationship among users and items. This algorithm initially computes similarities between items from the dataset and recommends recommendations based on similarities among items. Map-Reduce jobs are written to process the data in parallel on a cluster. Map jobs run on each cluster simultaneously and pass on the results to the reduce jobs. The reduce jobs combine the results from map jobs and form the final results. The map-reduce approach enables the algorithm to compute the recommendations in less amount of time as mappers and reducers run in parallel to perform the computation.

Experiments are executed by varying number of nodes in a cluster and increasing the size of the dataset. For each experiment, the required results are recorded. The results are compared with a single node cluster which shows significant improvement in performance with the use of map-reduce jobs on a cluster of varying nodes.

1.5 Organization

The organization of documentation is as follows: we provide a general introduction to map-reduce and HDFS in chapter 2. In chapter 3, we discuss the motivation for the Hadoop

map-reduce version of item-based collaborative filtering algorithm. In chapter 4, we present the design of the algorithm and in chapter 5 we discuss the implementation details of map-reduce jobs to derive the recommendation for items. In addition, we also describe the input and output formats at each map-reduce phase of execution. In chapter 6, we provide the experiments along with the results of this algorithmic approach. In chapter 7, we discuss the future work that can be done in this research. In chapter 8, we provide the conclusion of our research. In chapter 9, we summarize the main contributions and findings of this research.

Chapter 2

Background

Almost all large-scale E-Commerce websites recommend products to users. Recommendation systems are responsible for analyzing large amounts of information in order to identify potential user preferences. Most of the recommendation systems uses collaborative filtering as the approach for deriving recommendations. Most of the recommendations algorithms available for recommendation systems are from the field of machine learning. It is a subfield of artificial intelligence that produces algorithms applicable to learning, prediction, and decision-making.

Recommendation algorithms are a form of unsupervised learning aims at finding a structure within a set of random data i.e. they work by identifying similarities among items by calculating their distance from other items within feature space. Feature space represents item data. In broad terms, any software system which actively recommends an item either to purchase or to subscribe or to invest can be regarded as a recommendation system. An advertisement can also be a recommendation. Many online advertisement systems such as Google Adwords, uses recommendation systems to advertise products based on user profile. Collaborative filtering is one of the approaches to personalized recommendation systems.

2.1 Collaborative Filtering

Collaborative filtering provides recommendations based on model of prior item preference level. The model can be constructed either from a single item preference level or more effectively also from the preferences of other items who have similar attributes. When other items preference levels are taken into consideration, collaborative filtering uses group knowledge to form a recommendation based on likewise items. Recommendations are based

on an automatic collaboration of multiple items and filtered on those who exhibit similar attributes. Collaborative filtering techniques rely heavily on simple similarity measures (Cosine Similarity, Pearson Correlation, Jaccard Similarity, Tanimoto Similarity) to match similar items together. If we have a huge matrix with users in one dimension and items in another, with the cells containing either votes or likes, then collaborative filtering techniques use similarity measures on two vectors either rows or columns of such a matrix to generate a number representing similarity.

2.1.1 Collaborative Filtering Algorithm

Collaborative Filtering algorithm is a classic personalized recommendation algorithm which is widely used in many commercial recommendation systems [6]. Collaborative Filtering algorithm is an algorithm based on the following assumptions:

1. Users preferences and interests for items do not change.
2. User choices for items can be predicted based on their past preferences.

Because of the above assumptions, the collaborative filtering algorithm is based on the comparison of one item with other set of items, to find how similar items are, and according to the items similarity, its easy to predict users interests or preferences for an item.

Collaborative filtering systems are categorized into three subgroups: Memory-based, Model-based and Hybrid. Memory-based methods construct a rating matrix and uses the rating matrix as a reference to issue recommendations based on the similarity computed between users or items. Neighborhood based recommendation algorithms, user-based recommendation algorithms or item-based recommendation algorithms are some of the examples of memory-based collaborative filtering systems. Model-based methods are used for generating a model of the data under consideration for recommendation for generating predictions for a set of users or items. It issues recommendations based on the model. Clustering methods for recommendations is one of the example of model-based collaborative filtering. Hybrid based is the collection of both memory-based and model-based collaborative filtering systems. The

most popular memory-based collaborative filtering methods are neighborhood-based methods, which predict ratings by referring to users whose ratings are similar to the queried user, or to items that are similar to the queried item. This is motivated by the assumption that if two users have similar ratings on some items they will have similar ratings on the remaining items. Alternatively, if two items have similar ratings by a set of users, the two items will have similar ratings by the similar set of users. Specifically, user-based collaborative filtering methods identify users that are similar to the queried user, and compute the desired rating to be the average ratings of these similar users. Similarly, item-based collaborative filtering takes the average of the ratings of these similar items. Neighborhood methods vary considerably in how they compute the weighted average of ratings. The first step of collaborative filtering algorithm is to obtain the items history profile, which can be represented as a ratings matrix with each entry the rate of a user given to an item [18]. A ratings matrix for item-based collaborative filtering algorithm consists of a matrix where each row represents an user, each column represents a specific item, and the number at the intersection of a row and a column represents the rating given by the user for that item. The absence of a rating score at this intersection indicates that user has not yet rated the item. The next step is to calculate the similarity between the items and identify the similar items. The calculation of similarity among items is done by considering columns of the rating matrix. There are many similarity measure methods. We have used Pearson Coefficient, Cosine Similarity, Tanimoto Similarity and Jaccard Similarity. The calculation process of Collaborative Filtering algorithm would consume intensive computing time and computer resources. When the dataset is very large, the calculation process would continue for several hours or even longer. Therefore, we propose new method that is to implement the collaborative filtering algorithm on Hadoop platform to reduce the computation time of the similarity.

2.2 Apache Hadoop Framework

Apache Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers.[10] It is designed to scale from a single server to thousands of servers, with high degree of fault tolerance. In place of relying on high-end hardware, the flexibility of these clusters comes from the software's ability to detect and handle failures at the application layer. Hadoop enables a computing solution that is scalable, cost effective, and fault tolerant.[10]

2.2.1 Hadoop Top Architecture

Hadoop is implemented using master slave design pattern. Masters are responsible for controlling the slaves across the cluster. One of the masters is the Name Node, which is responsible for managing the HDFS (Hadoop Distributed File System) and controlling the slaves that store the data. The other master is the Job Tracker, responsible for handling the parallel processing of data in slave nodes using the map-reduce programming model.[10] The rest of the architecture contains all the slave nodes, which run both Data Node and Task Tracker daemons. Data Nodes are responsible for obeying the commands from the Name Node and store the partitioned data decoupled from the meta-data in the Name Node. Task Trackers are responsible for obeying the instructions from the Job Tracker and does all the computing work assigned by the Job Tracker. Finally, client machines are neither Master nor Slaves. Client machines are responsible for assigning tasks to the masters to load data into HDFS, submit map-reduce jobs describing how that data needs to be processed, and then retrieve or view the results of the task once finished.

The above figure shows the basic organization of the Hadoop cluster. The client machines communicate with the Name Node to perform basic file operations i.e. add, move, manipulate, or delete files in HDFS. The Name Node in turn calls the Data Nodes to store, delete or make replicas of data being added to HDFS. When the client machines want to process the data in the HDFS, they communicate to the Job Tracker to submit a job that

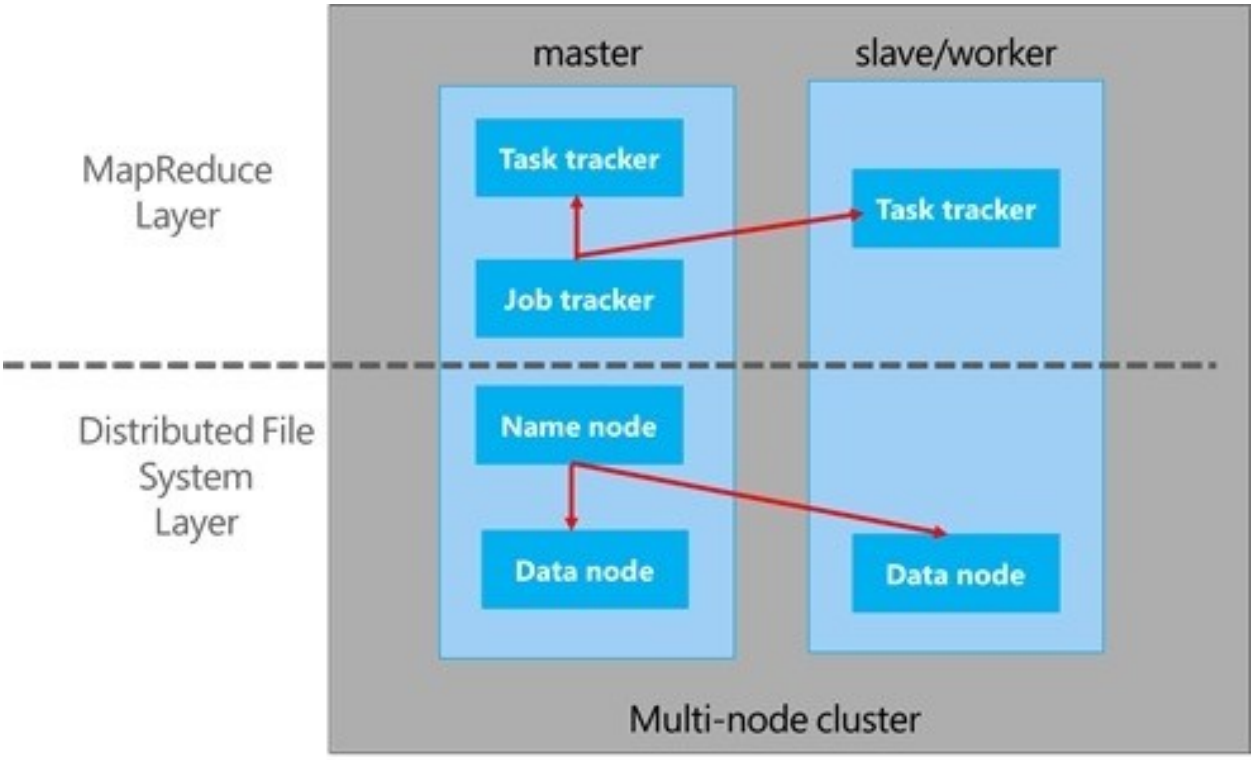

Figure 2.1: High-level Hadoop Architecture

uses map-reduce. Job Tracker divides the jobs to map and reduce tasks and assigns it to the Task Tracker to process it. Typically, all nodes in Hadoop cluster are arranged in the air-cooled racks in a data center. The racks are linked with each other with the help of rack switches, which runs on TCP/IP.

2.3 Hadoop Distributed File System

The Hadoop Distributed File System or HDFS is a distributed file system designed to run on commodity hardware.[11] HDFS is the main distributed storage used by Hadoop applications on clusters. Although HDFS has many similarities with existing distributed file systems, the differences between HDFS and other systems are significant. For example, HDFS is highly fault-tolerant and is designed to deploy on cost-effective clusters. HDFS offers high throughput access to application data which is suitable for applications that have large datasets.

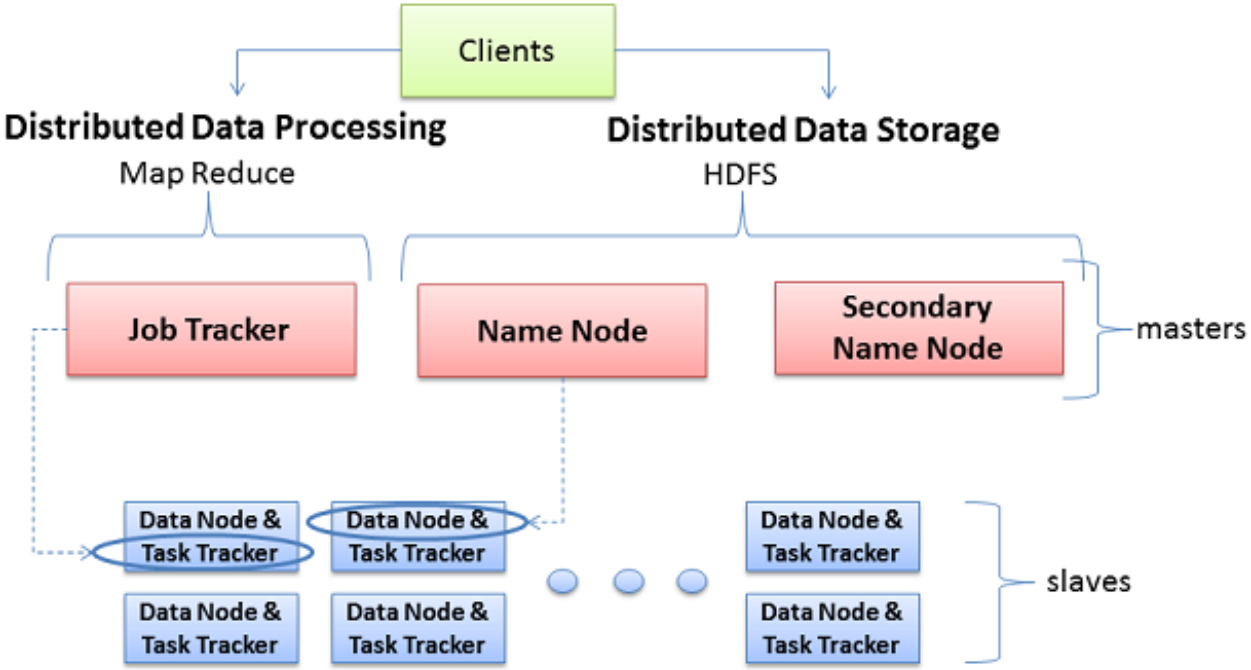

Figure 2.2: Hadoop Architecture

2.3.1 Architecture

HDFS uses master-slave architecture, in which a master is the Name Node and slaves are Data Nodes. The below figure shows a diagram representing the architecture of HDFS. Basically, an HDFS cluster consists of a single Name Node, which manages the file system namespace and regulates access of clients to files. In addition, there are a number of Data Nodes. Usually, each node in a cluster has one Data Node that manages storage of the node on which tasks are running. HDFS exposes file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks stored in a set of Data Nodes.

2.3.2 Name Node

The Name Node is the master of HDFS that maintains and manages the blocks present on the Data Nodes. It keeps the information of all files in the file system, and keeps track of the location of the file across the cluster. It does not store the data of these files itself.

There is just one Name Node in Hadoop, which is the single point of failure in itself. The HDFS architecture is built in such a way that the user data is never stored in the Name Node.

These are the following functions of a Name Node: The Name Node maintains and executes the file system namespace. If there are any modifications in the file system namespace or in its properties, this is tracked by the Name Node. It directs the Data Nodes to execute all the low-level I/O operations. It keeps a record of how the files in HDFS are divided into blocks, in which nodes these blocks are stored. In total, Name Node manages cluster configuration. It maps a file name to a set of blocks and maps a block to the Data Nodes where it is located.

It records the metadata of all the files stored in the cluster, e.g. the location, the size of the files, permissions, hierarchy, etc. With the help of a transactional log, that is, the Edit Log, the Name Node records each and every change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the Name Node will immediately record this in the Edit Log. The Name Node is also responsible to take care of the replication factor of all the blocks. If there is a change in the replication factor of any of the blocks, the Name Node will record this in the Edit Log. Name Node regularly receives a heartbeat and a block report from all the Data Nodes in the cluster to make sure that the Data Nodes are working properly. A Block Report contains a list of all blocks on a Data Node. In case of a Data Node failure, the Name Node chooses new Data Nodes for new replicas, and balances disk usage and also manage the communication traffic to the Data Nodes.

2.3.3 Data Node

A Data Node stores data in the HDFS. A functional file system has more than one Data Node with data replicated across them. On start up, a Data Node connects to the Name Node; spinning until that service comes up. It then responds to requests from the Name Node for file system operations. These are the following functions of a Data Node: Data

Nodes perform the low-level read and write requests from the file systems clients. They are also responsible for creating and deleting blocks and replicating the same based on the decisions taken by the Name Node. They regularly send a report on all the blocks present in the cluster to the Name Node. Data Nodes also enables pipelining of data. They forward data to other specified Data Nodes. Data Nodes send heartbeats to the Name Node once every 3 seconds, to report the overall health of HDFS. The Data Node stores each block of HDFS data in separate files in its local file system. When the Data Nodes gets started, they scan through its local file system, creates a list of all HDFS data blocks that relate to each of these local files and send a block report to the Name Node.

2.3.4 Secondary Name Node

The Name Node is the single point of failure for the Hadoop cluster, so the HDFS copies the namespace in Name Node periodically to a persistent storage for reliability and this process is called check pointing. Along with the namespace it also maintains a log of the actions that change the namespace, this log is called journal. The checkpoint node copies the namespace and journal from Name Node to applies the transactions in journal on the namespace to create most up to date information of the namespace in Name Node. The backup node however copies the namespace and accepts journal stream of namespace applies transactions on the namespace stored in its storage directory. It also stores the up-to-date information of the namespace in memory and synchronizes itself with the namespace. When the Name Node fails, the HDFS picks up the namespace from either Backup Node or Checkpoint Node.

2.4 Map-Reduce Model

The Map-Reduce model was mainly designed for unstructured data processed by large clusters of commodity hardware; the functional style of map-reduce automatically parallelizes

and executes large jobs over a computing cluster. The map-reduce model is capable of processing many terabytes of data on thousands of computing nodes in a cluster. Map-Reduce automatically handles the messy details such as handling failures, application deployment, task duplication's, and aggregation of results, thereby allowing programmers to focus on the core logic of applications. Each map-reduce application has two major types of operations - a map operation and a reduce operation. Map-Reduce allows parallel processing of the map and reduce operations in each application. Each mapping operation is independent of the others so all mappers can be performed in parallel on multiple machines. Similarly, a set of reduce operations can be performed in parallel during the reduction phase. All outputs of map operations that share the same key are presented to the same reduce operation. Map-Reduce can be applied to process significantly larger datasets than commodity servers. For example, a large computing cluster can use map-reduce to sort a petabyte of data in only a few hour.

Parallelism also offer some possibility of recovering for partial failure of computing nodes or storage units during the operation. In other words, if one mapper or reducer fails, the work can be rescheduled, assuming the input data is still available. Input datasets are available even in presence of storage unit failures, because each dataset normally has three replicas stored in three individual storage units. A map-reduce program has two major phases - a map phase and a reduce phase. The map phase applies user specified logic to input data. The results, called as intermediate results, are then fed into the reducer phase so the intermediate results can be aggregated and written as a final result. The input data, intermediate data and final data, all are represented in key-value pair format [3]. Figure 2.3 shows an executional example of the map-reduce model. As shown by the diagram during their respective phases multiple map and reduce jobs are executed in parallel on multiple computing nodes. Map-Reduce is also usually described in the form of the following functions summarized in Table 2.1.

Input	Output
map(k1,v1)	list(k2,v2)
reduce(k2,list(v2))	list(k3,v3)

Table 2.1: Map Reduce Functions

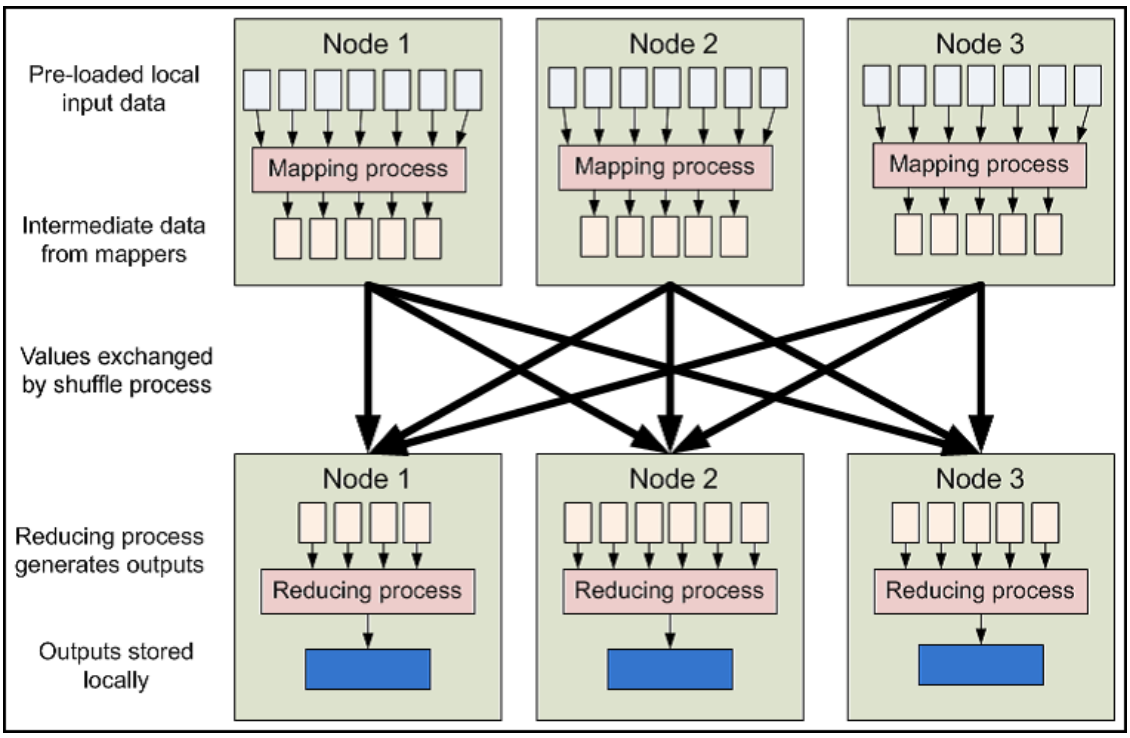

Figure 2.3: Map Reduce Model

Chapter 3

Motivation

Data-intensive applications are popular and are increasing day by day. Applications not limited to social networks, web auctions sites, e-commerce sites are data-intensive as they generate data on day-to-day basis. Data-intensive applications need access to ever-expanding datasets ranging from a few gigabytes to several terabytes or even petabytes. Google, for example, leverages the map-reduce model to process twenty petabytes of data per day in a parallel fashion [9]. Map-Reduce is an attractive model for processing large data in parallel, as it breaks the larger task into smaller tasks, in high-performance cluster computing environments. As map-reduce operates by partitioning large task into numerous small tasks running on multiple machines in a large-scale cluster, its scalability is proven to be high.

Collaborative Filtering algorithm is a widely used personalized recommendation technique in commercial recommendation systems [17], [18]. A lot of work has been carried out in the field to improve the performance. Cloud computing has been one of the focus to overcome the problem of large-scale computations. Cloud computing is the provision of dynamically scalable and often virtualized resources as a service over the Internet [15]. There is no requirement for the users to have knowledge of, expertise in the technology infrastructure in the cloud that supports them.

Cloud computing services provide common business applications online which are accessed from a client browser, while the software and data are stored on the servers. In order to solve scalability problem and improve the performance of the recommendation system, we implement the collaborative filtering algorithm on the cloud-computing platform. There are several cloud computing platforms available, for example, the Dryad [24] of Microsoft, the

Dynamo [25] of amazon.com and Netune [23] of Ask.com etc. In this paper, we choose the Hadoop platform as the base of our implementation since, the Hadoop platform [9], [14] is an open source cloud- computing platform. It implements the map-reduce framework that has been successfully evaluated by Google.com. The Hadoop platform uses a distributed file system, Hadoop Distributed File System (HDFS) [15], to provide high throughput access to application data. Using the Hadoop platform, we can execute a program in parallel. Map-Reduce framework allows user to split a large task into many small tasks. All the small tasks are then handled by the Hadoop platform, thus improving the computation speed. The Hadoop map-reduce framework solves the scalability issue for systems dealing with large datasets. The ability of Hadoop framework to process huge data very fast motivated us to utilize its capability to generate recommendations by converting the sequential approach to parallel that processes the huge dataset. The algorithm is divided into multiple parts to identify the components that can take the advantage of parallelization. Motivation to select the item-based collaborative filtering approach to solve the recommendation system problem is that the item-based similarities derive the most accurate predictions to the item based on the similarities of similar items.

3.1 Problem Statement

Let A be I1 x I2 matrix holding all known interactions between a set of items I1 and a set of items I2. An item i is represented by his item interaction history Ii, the i-th row of A. The top-N recommendations for this item correspond to the first N items selected from a ranking r of all items according to how strongly the similarity measure between the items are. This ranking is inferred from patterns found in A.

3.2 Contributions

In this research, we study the implementation details of item based collaborative filtering algorithm on cloud computing platform. The work we have done is summarized as follows.

Firstly, we designed an item based collaborative filtering algorithm for the map-reduce program framework, and implemented the algorithm on the Hadoop platform. Secondly, we tested our implementation under several configurations. Two map-reduce jobs are written to derive recommendations and these jobs are run on Hadoop cluster. The first map-reduce job is responsible for computing the similarity among items. The second map-reduce job is responsible for deriving recommendations based on the similarity computed using first map-reduce job. The item-based collaborative filtering algorithm based on items similarity is implemented using parallel programming environment of Hadoop map-reduce framework. The algorithm processes the input dataset to compute similarities and then generate recommendations for each item based on similarities computed. The idea for map reduce algorithm is based on basic map reduce paradigm i.e. to split the task into smaller sub tasks and solve each sub-task in parallel. The data is partitioned in a way to support the parallel similarity computation. Thus taking the advantage of map-reduce the similarity computation which is the key for resource consumption is parallelized.

Chapter 4

Design

In this chapter, we discuss about item-based collaborative filtering algorithm and its respective map-reduce version. This section discusses the step-by-step development of our algorithmic framework. We start with discussion on how to conduct distributed item similarity calculation for our simple model for input data. After similarity calculation, we discuss how to derive recommendations based on the similarity values being calculated among items. We can add more nodes to process the data, in order to achieve linear scalability with a growing size of dataset

4.1 Approach

In order to relate a relationship between items and users, we define the value as 1 for an item, if the user has purchased an item else the value for the item is defined as 0. Here, we are considering the purchase history of the user. If we consider the rating given by the user to the item, we can have some other parameter values assigned to build the relationship between the item and the user. Let U be the user, I1 be a first item and I2 be a second item, if user has purchased an item I1, the value of relationship between the user U and item I1 is 1 say R(U,I1) = 1. If the user has not purchased an item I2, the value of the relationship between the user U and item I2 is 0 say R(U,I2) = 0.

In order to compare two items, a dot product of item vectors is computed, where a vector of the item contains the relationship of the item with all the set of users i.e. the vector will contain the values given by all the users to the item. Based on the computed value of two item vectors, we can say how similar two items are. When computing recommendations for a particular item with item-based collaborative filtering, we fetch all the computed similarity

values of the item with other set of items. Among all the fetched computed similarity values, the top N values will be considered. The top N items are then recommended for the item.

In order to compute dot product of item vectors, we generate an item-item matrix where each column represents an item. Each row has a value a user has given to the items represented by the columns.

Notation Hints: Va denotes the vector for item a where Va contains a list of values of item a given by all the users. Vb denotes the vector for item b where Vb contains a list of values of item b given by all the users. Vab denotes the similarity value of item a with item b.

4.2 Sequential Approach

As we are using different set of formulae to compute similarity between two items, the sequential approach for computing the similarity of items Si varies based on the computation of the formulae.

The standard sequential approach [21] for computing the similarity of items Si(u, v) is defined as

$$Sim\_(u, v) = dot(u, v)/ItemsPurchased\_u + ItemsPurchased\_v - dot(u, v)$$

$$dot(u, v) = vector(u).vector(v)$$

ItemsPurchased(u) indicates the total count of users purchased item u.

To get the similarity of items we need to compute the dot product of each item vector of U with each another item vector of V.

Algorithm to compute similarity of two items sequentially:

Step 1: Compute total count of users purchased item u:

Algorithm 1 Count of users purchased item i

1: **for** each user u who purchased item i **do**
2: Ci = Ci + 1
3: **end for**

Step 2: Compute the dot product of two vectors of items u and v

A be the matrix containing all the items and the values indicating whether the user has purchased item a or not.

Vi be the dot product value of item a with item b

Ia be the value indicating whether user has purchased item a or not

Ib be the value indicating whether user has purchased item b or not

Suv be the similarity value of item u with item v

Algorithm 2 Dot Product of Two Item Vectors

1: **for** each item a in matrix A **do**
2: **for** each item b in matrix A **do**
3: Vi = Vi + (Ia * Ib)
4: Suv = Vi / (Ca + Cb - Vi)
5: **end for**
6: **end for**

The sequential algorithm takes more time with the increasing number of users and items. The components calculated in the previous algorithm are used here to compute the similarity. The computation for each item with other item is computed more than once. The algorithm to compute the dot product is the most resource consuming part of similarity computation and has to be addressed to improve the performance. In order to improve the run-time speed-up proportional to the number of machines in the cluster, the algorithm should be modified to a parallel version.

One solution to this problem is to pre-process the data in a way that the computation can take place by column-wise of the matrix across multiple machines and can achieve parallelism of the similarity computation. After applying the pre-processing technique we design a map reduce program to compute the similarity and derive recommendations using the similarity computed between two items.

4.3 Parallel Approach

We need to devise a parallel algorithm in order to improve the complexity of the sequential algorithm for the similarity computation to make its run-time speed-up proportional to the number of machines in the cluster. This is not possible with the standard sequential approach, as it requires random access to the rows and columns of matrix A in the inner loops of algorithm for similarity computation, which cannot be efficiently realized in a distributed, shared-nothing environment where the algorithm has to work on partitioned data. We can efficiently compute the similarity by the use of a map-reduce paradigm which is compatible to partitioned data.

Multiple map-reduce jobs can run mappers in parallel to compute different components of the equation and reduce jobs finally compute the similarity and ratings. Each row of the item-item matrix represents the value given by user to the item i. We can compute the similarity of one item with another item by just processing each row of matrix for each item. The data is pre-processed such that all items ratings of set of users is available on one node so that the computation happens with all items. Data is partitioned by Hadoop row wise, each row creates a mapper, the computation for each item is parallelized and all the results are combined in reducer to get the overall similarity of all items for each item pair. The item pairs with all the similarity values are processed by another map-reduce job to derive recommendations for each item. We discuss more details of the algorithm like data pre-processing, data partitioning, similarity computation and deriving recommendations in the further sections.

4.3.1 Data Pre-processing

In order to parallelize the similarity calculation, the computation has to be carried out in small parts on multiple machines. The key is to pre-process data such that all the data required to complete small tasks specific to a machine can be present within that machine. In our problem, we need to compute the item-similarity for each pair of items. So all the

```
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
1,0;2,0;3,0;4,0;5,1;6,0;7,0;8,1;9,1;10,0;11,0;12,0;13,0;14,0;15,0;16,0;17,0;18,0;19,1;20,0;21,0;22,1;23,1;24,0;25,0;26,0;27,0;28,0;29,0;30,0;31,1;32,1;33,0;34,0;
35,0;36,0;37,0;38,1;39,1;40,0;41,0;42,0;43,0;44,0;45,0;46,0;47,0;48,0;49,1;50,0;
```

Figure 4.1: Input Dataset

item data needs to be available on one node. Hence, the data partition can be user based. Each node can just have the data of some users. The results of these small tasks can be joined together to compute similarity for a pairs of items.

The dataset we have is a user-item matrix, which has ratings for each item given by all the users. Each row represents a user followed by a set of items along with their ids and the ratings given by the user to the respective item. In order to compute the similarity of items across multiple machines, we can split the whole dataset randomly based on the total number of machines used. As we require the data to be in the format where each row represents a single user with all the set of items, we don't have a requirement of reorganizing the data to a new format. Thus, we can randomly split the data row wise so that each individual machine gets a split of the whole pre-processed dataset.

The sample dataset is shown in figure: 4.1. Each line in the dataset represents a user id followed by an item id and the rating given by the user for that item. The value 0 represents the user has not given any rating to the item and hence, by default, it is set to 0.

The data shown in figure: 4.1 is reorganized as shown in figure: 4.2 to be available for Hadoop for processing. The data reorganized is similar to what we represent the data in a matrix form. The conversion of data into matrix format is really easy.

The dataset is divided into two partitions partition one is shown in figure: 4.3 and partition two is shown in figure: 4.4, each of which contains a set of items and their ratings. The reorganization of partition data is shown in figure: 4.5 and figure: 4.6 respectively.

Users	Item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8	Item9
User1	1	1	2	5	4	2	5	2	3
User2	3	2	3	4	3	4	3	4	2
User3	2	4	2	3	2	3	2	3	3
User4	4	3	1	2	1	1	4	2	4
User5	1	2	2	1	3	2	1	4	1

Figure 4.2: Reorganized Dataset

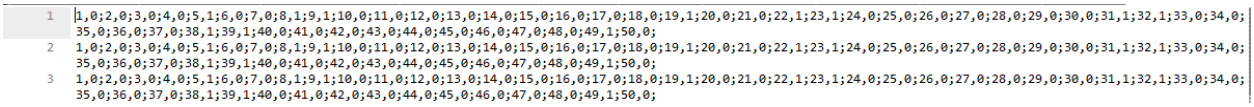

Figure 4.3: Partition Dataset 1

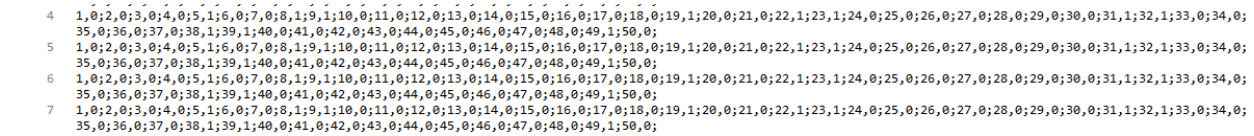

Figure 4.4: Partition Dataset 2

Users	Item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8	Item9
User1	1	1	2	5	4	2	5	2	3
User2	3	2	3	4	3	4	3	4	2

Figure 4.5: Reorganized Partition 1

Users	Item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8	
User3	2	4	2	3	2	3	2	3	3
User4	4	3	1	2	1	1	4	2	4
User5	1	2	2	1	3	2	1	4	1

Figure 4.6: Reorganized Partition 2

Here, we partitioned the input dataset by user-based, so that each row will contain all the items and the ratings given by a particular user for all the items. Now map-reduce jobs will merge the similarity results with the raw data partitioned using user-based strategy. This enables to leverage the map-reduce programming advantage at more higher level.

4.3.2 Similarity Computation

Once the data is being partitioned, the similarity values among the items needs to be calculated. We have used four different formulae to compute similarity among items. Each formulae is different in its own sense. The process for computing the similarity between items is the same in different experiments. The only change we have made in different experiments is the use of computational formulae to compute similarity.

Process to compute similarity among items:

1. For each user u, generate item pairs with ratings given by the user for that item.

2. Once, we have item pairs, merge all the similar items pairs into a single large item pair.

3. For each large item pair, compute the similarity among the two items by using the respective similarity computational formulae.

4. Once similarity among item pair is computed, we can then use the similarity values to recommend the top N items for a particular item based on the similarity value.

All the nodes that have item data follows the above process and results for each item pairs are saved. The saved results are passed on to be computed for similarity based on different similarity formulae.

The above task is handled in parallel as the data for each pair can reside on any data node and each data node will compute the similarities of the item-pair whose items rated values are present on the node.

The similarity measures used for computing similarity among item pairs are Jaccard Similarity, Cosine Similarity, Tanimoto Similarity and Pearson's Coefficient. Figure 4.8

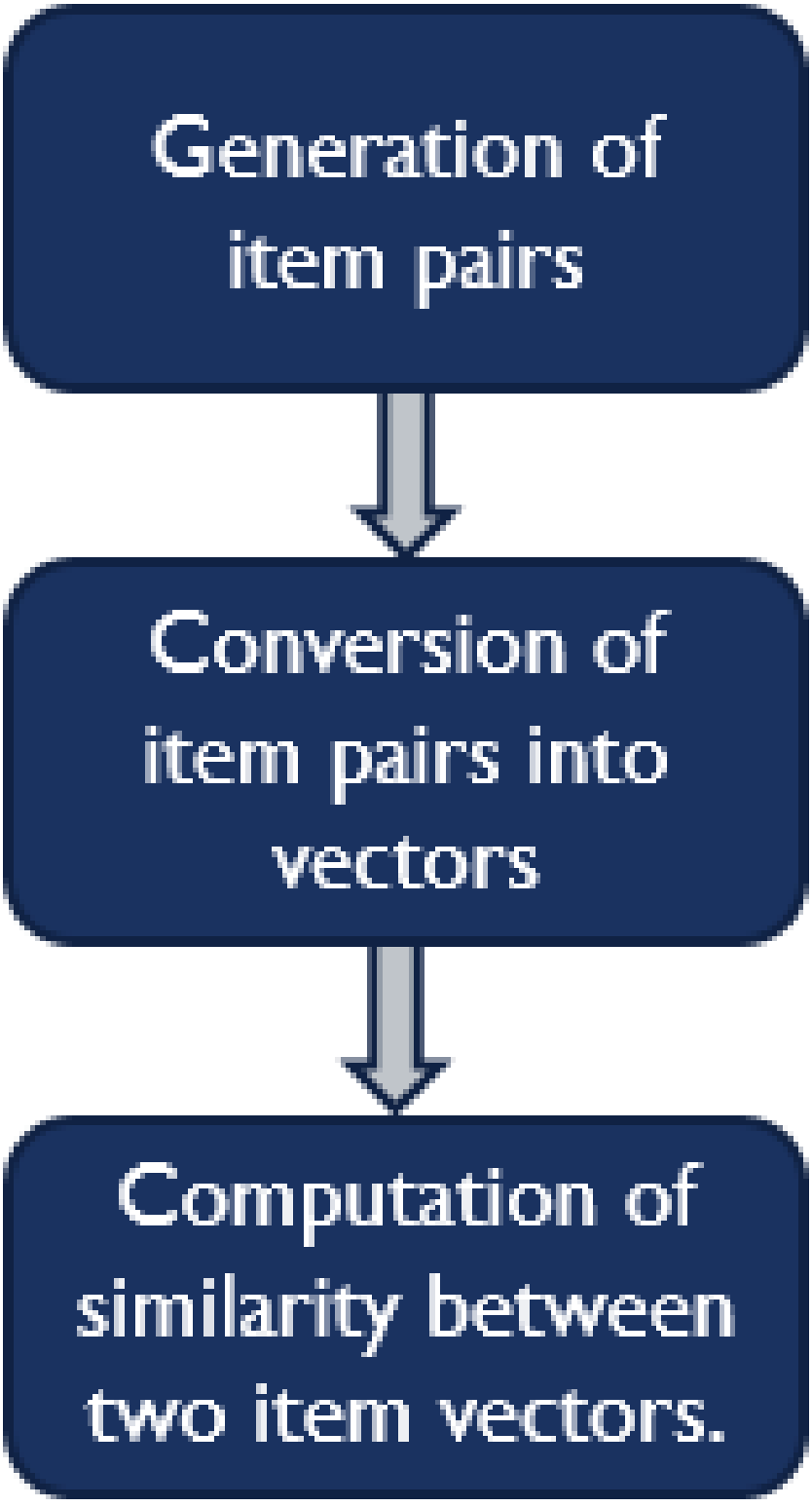

Figure 4.7: Similarity Computation

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Cosine Similarity

$$P_{x,y} = \frac{\sum_{i=1}^{n}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})} * \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})}}$$

Pearsson's Coefficient

$$J(\mathbf{x}, \mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)},$$

Jaccard Similarity

$$f(A, B) = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$

Tanimoto Similarity

Figure 4.8: Similarity Measures for Computing Similarity.

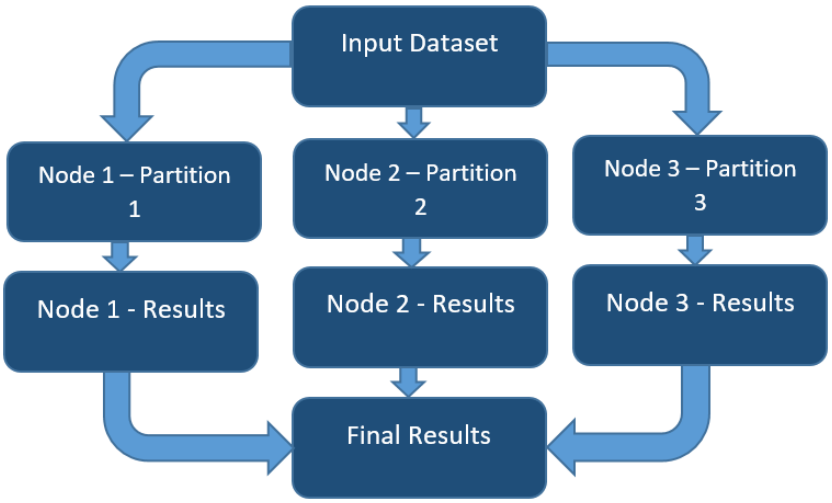

Figure 4.9: Parallel Processing of data by multiple nodes

provides the required similarity measure formulae used for computing the similarities among items.

4.3.3 Deriving Recommendations

The next part of the problem is to derive recommendations by using the similarity values computed in the previous step for a pair of items. We can generate recommendations for all the items at a time.

To derive recommendations for each item, the ranking among the item pairs (i,j) needs to be computed. For each item pair (i, j) ranking is computed based on the descending order

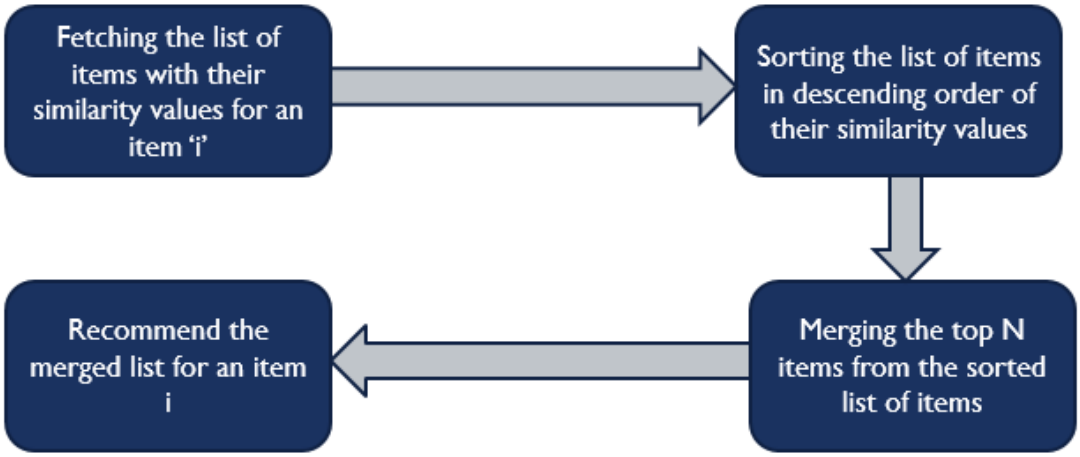

Figure 4.10: Deriving Recommendations using the similarity computation

of similarity value of item i with rest of the items. This process should be repeated for all item pairs and all the individual item ratings are computed. From all the items, items with top N ranking are selected and provided as the recommendation to the user for an item i. In simpler terms, we can list the steps of deriving recommendations for each item as follows:

1. Fetch the list of item-pairs containing the target item i with similarity values.
2. Sort the list of item-pairs in descending order of their similarity values.
3. Once sorted, merge the top N items from the sorted list of item-pairs.
4. Once merged, recommend the merge list of items for the target item i.

Chapter 5

Implementation

5.1 Map-Reduce Jobs

Map-Reduce jobs run in parallel on a Hadoop cluster. Hadoop cluster is a collection of two or more nodes and each node has Hadoop installed on it. A map-reduce job is divided into smaller map-reduce jobs, where in each job executes the same task but on a different dataset. The input dataset which is processed by Hadoop is partitioned into several smaller datasets and are distributed across several data nodes and the map-reduce job on each node uses the dataset respective to the node as input and performs the required computation as specified within the map-reduce job. Each map task after processing the input dataset generates a (key, value) pair as intermediate output which is stored on the local disk. The reduce tasks will collect all the intermediate results with same key and performs the required computation as specified within the reduce job to generate the final output. Map-Reduce jobs use text files as input and output the final results into text files.

5.2 Input Data

In Hadoop, for map-reduce jobs to process dataset, dataset should be in the form of key-value pairs. The input data is created as a text file by querying the movie lens database system which provides us with the required input data. We can pre-process the data so that it can processed by Hadoop map-reduce jobs. The text file contains the data where each row contains all the items and the rating given by the user for all the items. As map-reduce jobs works based on key-value pairs, we assume that the input data is pre-processed in the required format.

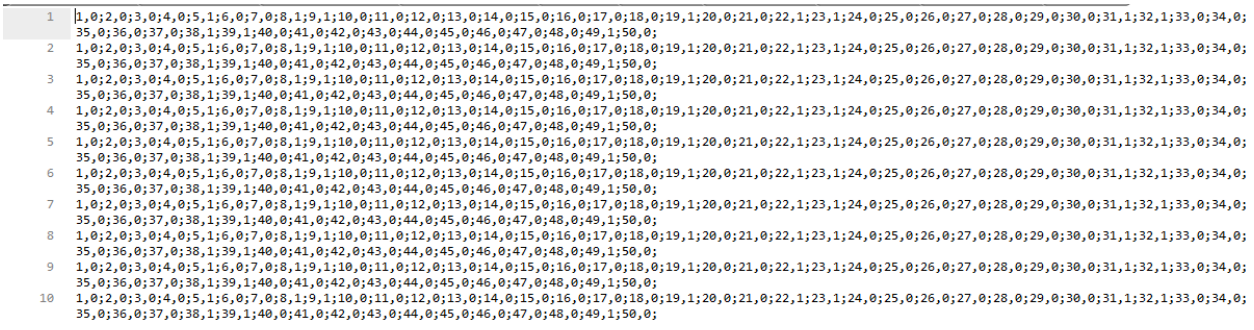

Figure 5.1: Input data format after pre-processing of data. Here, the item and rating for the item is separated by ','. Collection of items are separated by ';'

The hash-code generated internally by Hadoop for each input line of the text file is used as key and entire single line within the text file(containing the item id and rating) as value. When a map-reduce job executes, each row data is processed by an individual mapper. Each line within the input text file represents an user and the line contains the information about all the items represented by item id and the rating given by the user for all the items. All the items within the input file is separated by delimiter ';' and the rating for each item given by the user is separated by delimiter ','. When Hadoop runs the map-reduce job, it processes each line as a different mapper and uses the data to create intermediate results as key-value pairs.

As the recommendation problem is divided into two parts: building relationship among item pairs then computing the similarity among item pairs and deriving the recommendations among item pairs, map-reduce jobs are written to finish each part of the problem.

5.3 Compute Similarity between Item Pairs

The similarity computation phase of the recommendation problem is sub-divided into a single map-reduce job. Mapper of the map-reduce job is responsible for building relationships among items i.e. it is responsible for emitting the key-value pairs where each key is a set of two items and the value represents the rating given by a user for both the items. Reducer of the map-reduce job is responsible for computing the similarity among the items being emitted

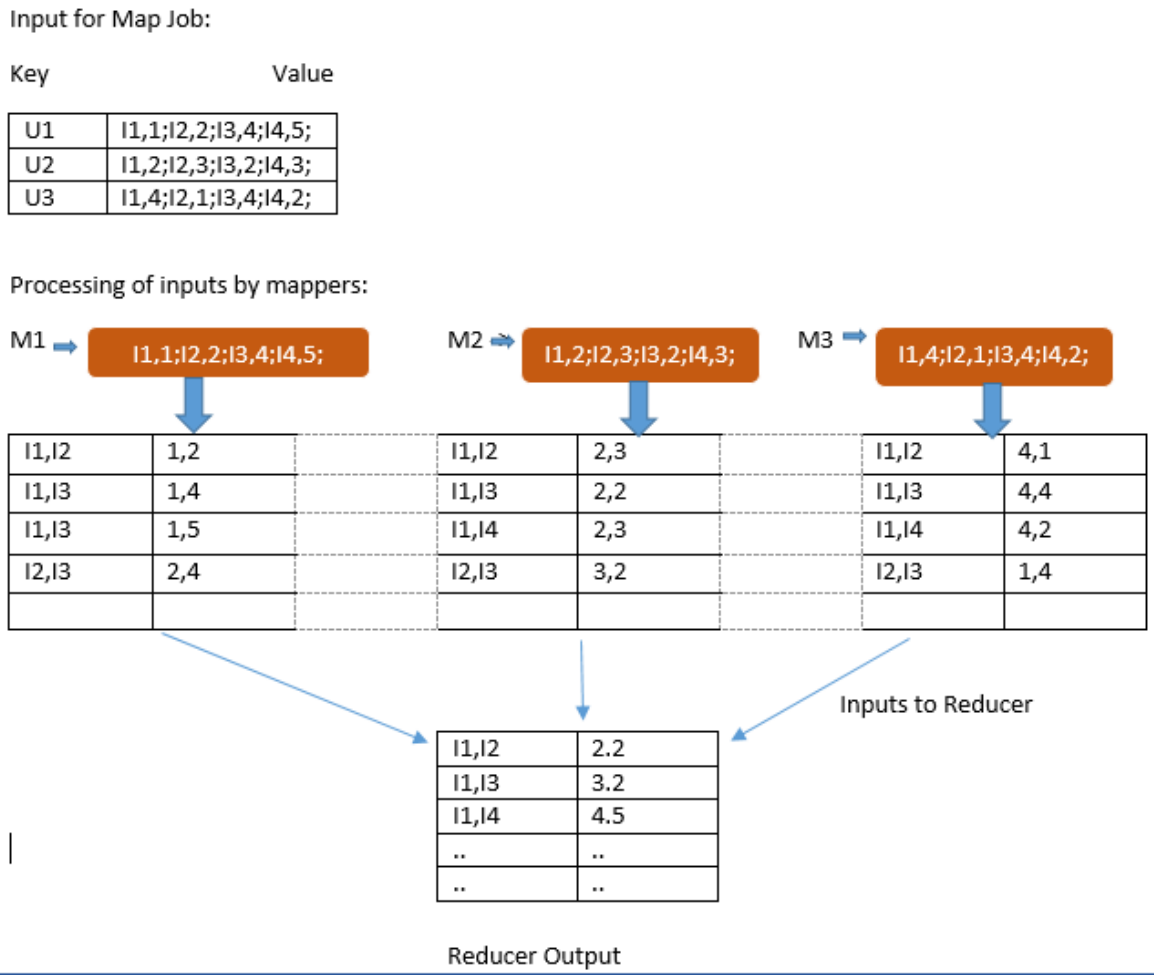

Figure 5.2: Computation of Similarity between Item Pairs. Input key for Map Job is a Hash code automatically generated by Hadoop. Input value for Map Job is a single row of the input dataset. Input key of the reduce job is (I1,I2) where I represents an Item. Input value is (R1,R2) where R represents the rating for item I.

by the mapper i.e. it is responsible for computing the similarity of each key represented by pair of items.

5.3.1 Map-Reduce Job

As mentioned earlier, each row of data is processed by a mapper, the map job creates item-item pair keys and values represents the ratings given by the user for item-item pair. The map job generates the required (key,value) pair by splitting the input data by delimiter ';'. The output data is again split into sets by the use of delimiter ','. The final output data

is paired in a way to form item pairs as keys and the corresponding values i.e. ratings for each item is paired to form as value for the key. The map task produces intermediate results in the form of (key, value) pairs as discussed earlier, which are then sent as input to the reduce job.

The reduce job collects all the item-item keys and values representing the ratings given by the user for item-item keys. Post collection, similarity for each unique key collection is calculated. In order to calculate similarity, reducer first converts the value of the key collection into a vector. Two vectors are constructed where in each vector representing an item. Then, we use four different similarity calculation formulae to calculate similarity among the item pairs by considering the vectors being generated. The four different formulas are used based on different experiments. Post, the computation of similarity, the final result for the item pair is outputted.

Figure 5.2 shows the input to the map job and the corresponding output of the map job. It even shows the input and output to the reduce job.

Four different formulae used for the similarity calculation are Jaccard Similarity, Pearsson's Coefficient, Cosine Similarity and Tanimoto Similarity. Figure 5.5 shows the Jaccard Similarity to compute similarity between two vectors. Figure 5.6 shows the Cosine Similarity to compute similarity between two vectors. Figure 5.7 shows the Tanimoto Similarity to compute similarity between two vectors. Figure 5.8 shows the Pearsson's Coefficient to compute similarity between two vectors.

The computation time taken to compute similarity using map-reduce job is greatly reduced when compared to sequential computation of the similarity among item pairs as the tasks is divided into smaller tasks and each smaller tasks is performed by a map-reduce in parallel.

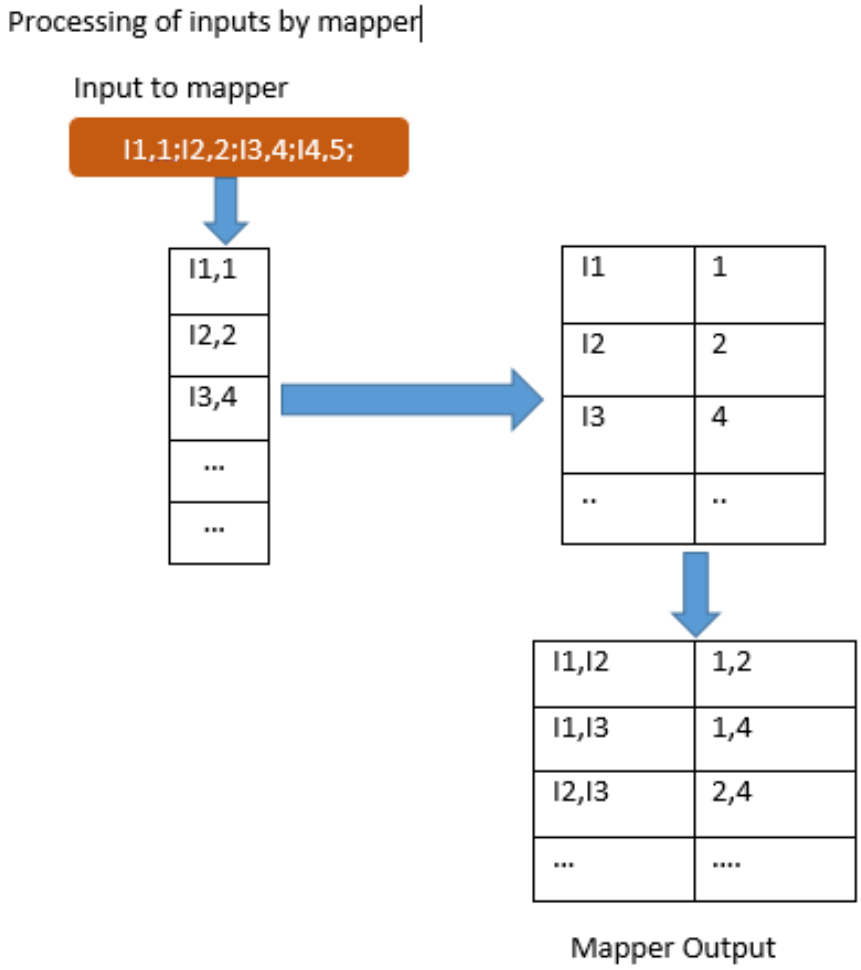

Figure 5.3: Map Job for Similarity Computation. Input key for Map Job is a Hash code generated by Hadoop. Input value for Map Job is a single row of the input dataset. Output key of the map job is (I1,I2) where I represents an Item. Output value of the map job is (R1,R2) where R represents the rating for item I.

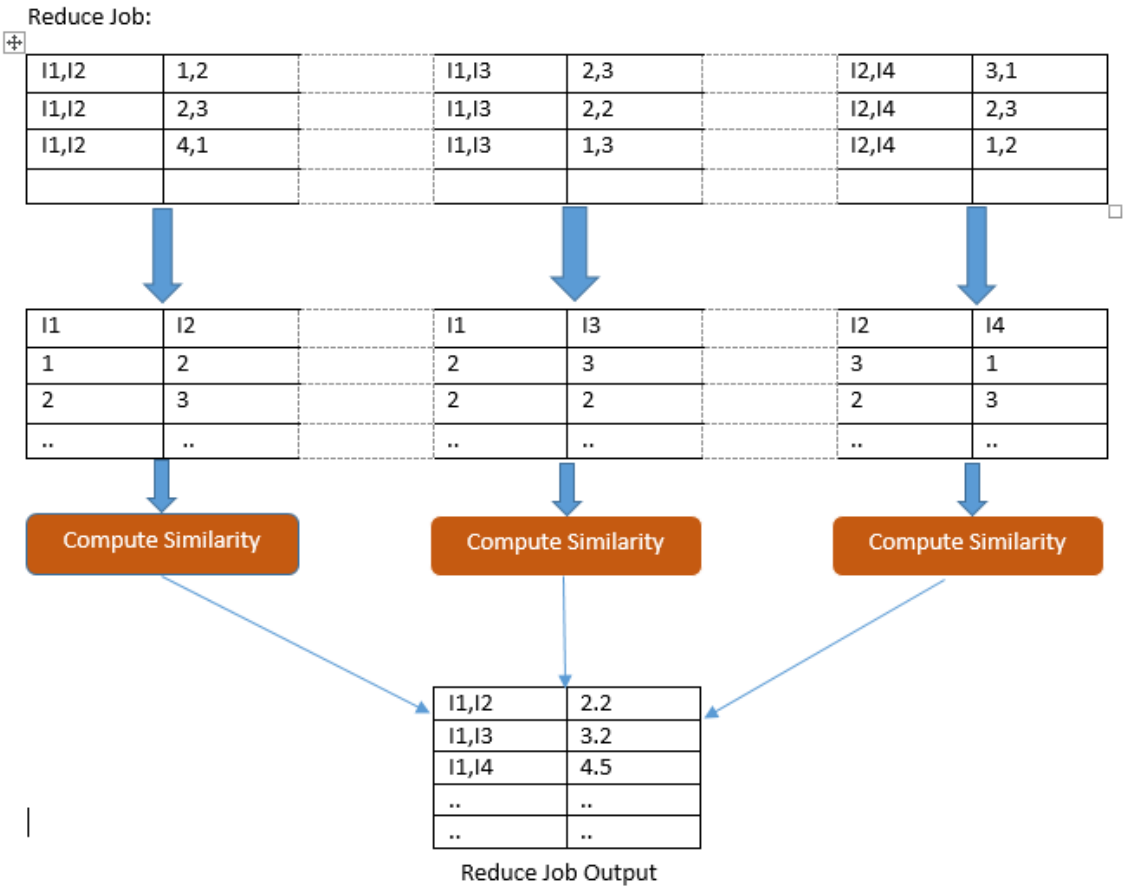

Figure 5.4: Reduce Job for Similarity Computation. Input key for Reduce Job is (I1,I2) and the input value is (R1,R2). Output key of the reduce job is (I1,I2) and the output value is S1. I represents an item, R represents the rating associated with an item and S represents similarity value for an item pair.

$$J(\mathbf{x},\mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)},$$

Figure 5.5: Jaccard Similarity to compute similarity between two vectors. X and Y represents two vectors.

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Figure 5.6: Cosine Similarity to compute similarity between two vectors. A and B represents two vectors.

$$f(A,B) = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$

Figure 5.7: Tanimoto Similarity to compute similarity between two vectors. A and B represents two vectors.

$$P_{x,y} = \frac{\sum_{i=1}^{n}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})} * \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})}}$$

Figure 5.8: Pearsson's Coefficient to compute similarity between two vectors. X and Y represents two vectors.

5.4 Deriving Recommendations

The final part of the implementation is to derive recommendation for items in the system based on the similarity value being calculated for the item pairs. The item pair which has the highest similarity value for the selected item is considered highest in ranking and its given the top most priority in the recommendation list. In order to compute this, we need to take in consideration the similarity values among the item pairs. Then sort the item pairs in decreasing order of their similarity values and select top N item pairs and provide the list for the selected item.

5.4.1 Map-Reduce Job

The second map-reduce job is responsible for deriving the recommendations based on the similarity values between item pairs computed using the first map-reduce job. The second map-reduce job takes in the input as the output of the reduce job of the first map-reduce job. The map-reduce job is responsible for emitting an item pair where each item within the pair act as a key. The reduce job takes in collection of values of the item key. It sorts the collection in decreasing order of the similarity values. Once, we get the list of decreasing order, the items associated with the similarity values are outputted in the order of their similarity values as the recommendation list for the item key.

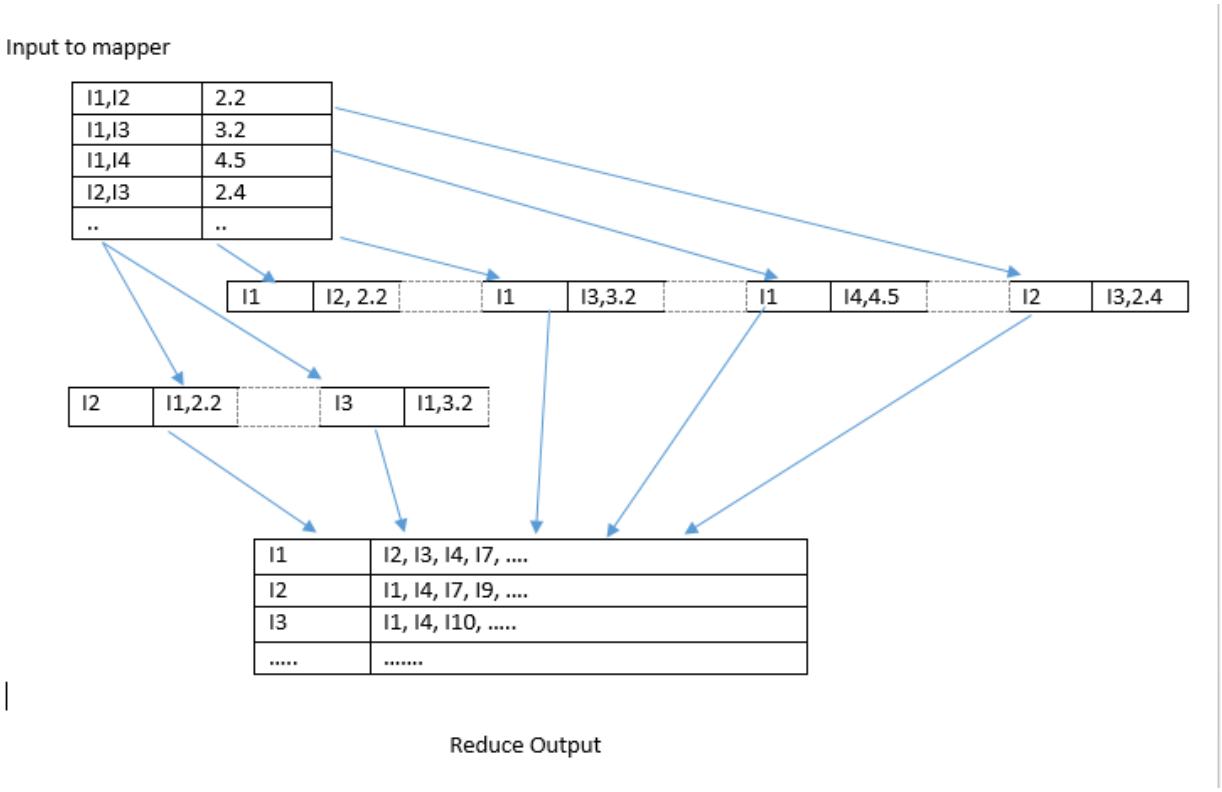

Figure 5.9: Deriving Recommendation. Input key for the map job is an item pair (I1,I2). Input value is a similarity value S1. Input key for the reduce job is an item I1 and value is a pair of item with similarity value (I2,S1).

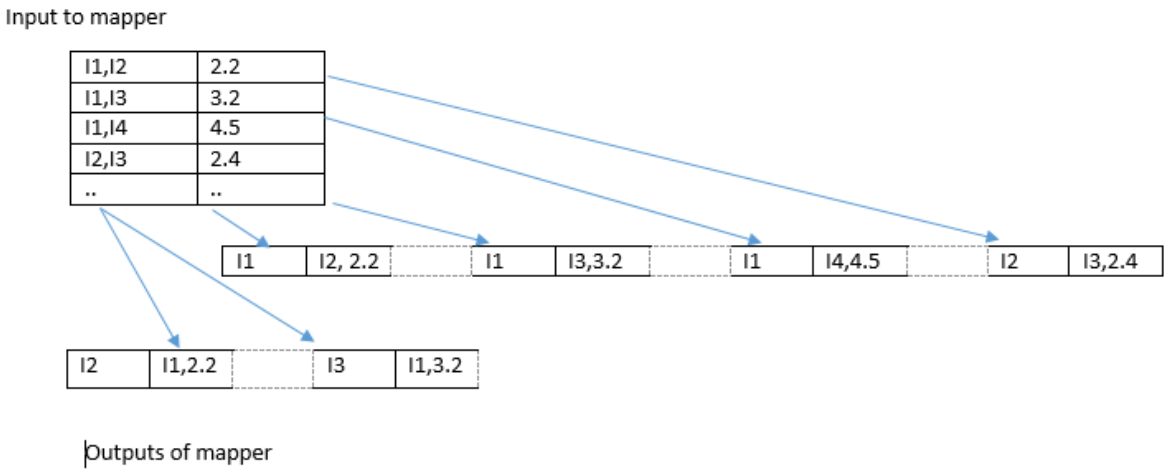

Figure 5.10: Mapper Job Deriving Recommendation. Input key for the map job is an item pair (I1,I2). Input value is a similarity value S1. Output key of the map job is an item I. Output value of the map job is a pair of item with similarity value (I2,S1).

The map job reads each key which is an item pair and the value which is the similarity value among the item pairs. It then emits the key into two individual items i.e. if the key is say (I1,I2), it emits the pair into I1 and I2. After emitting the key pairs, it merges the items in a way such that each item behaves as an independent key and the value for each item is an another item of the item pair along with the similarity value between the item pair i.e. say (I1,I2) is an item pair and the similarity value among the item pair is 4.5, the mapper output for the item pair (I1,I2) with similarity value 4.5 is (I1, I2-4.5) and (I2, I1-4.5).

The reduce job takes in collection of values of the item key. It splits each value pair of the collection by the delimiter ','. Once splitting is done, a matrix is formed with set of items and the corresponding similarity value for that item. After the matrix is formed, the matrix is sorted in decreasing order of the similarity values. After sorting the matrix in decreasing order, all the items associated with the similarity values within the sorted matrix are merged based on the number of items we need to recommend. After merging, the final merged list of items are outputted in the order of their similarity values as the recommendation list for the item key.

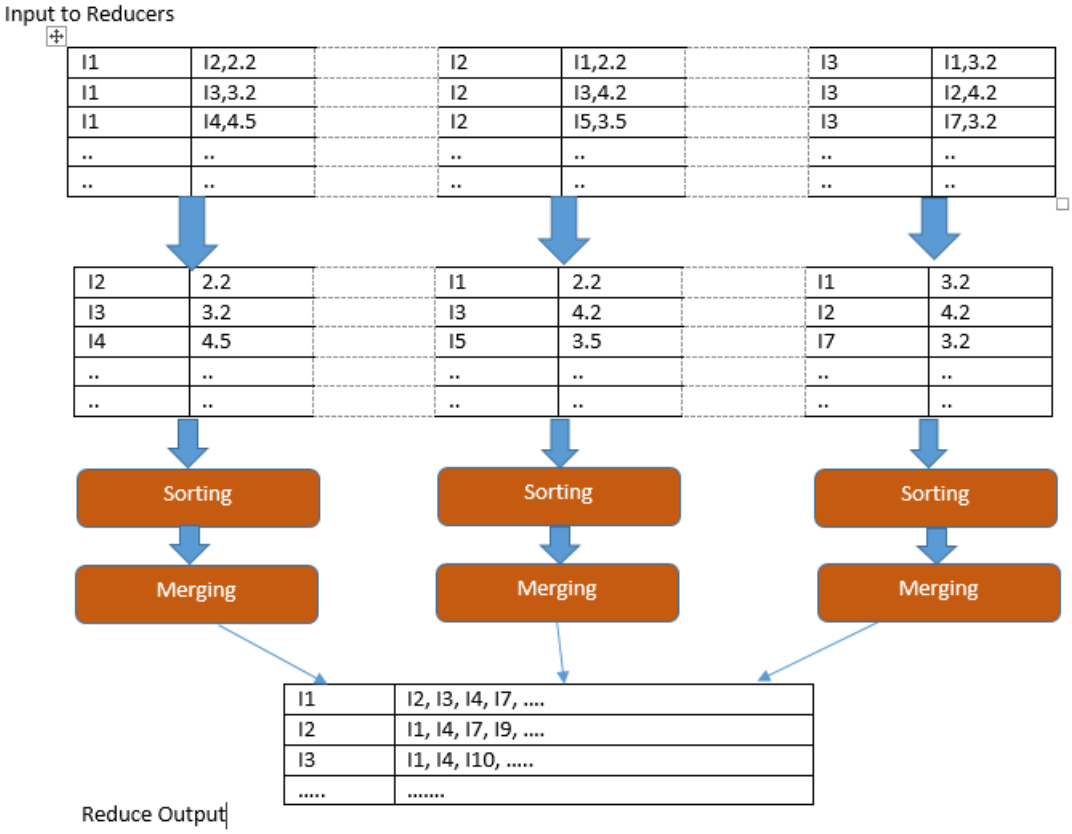

Figure 5.11: Reduce Job Deriving Recommendation. Input key for reduce job is an Item. Input value of the reduce job is a pair of item with similarity value (I2,S1). Output key of reduce job is an Item. Output value of the reduce job is a collection of items.

Figure 5.5 shows the input to the map job and the corresponding output of the map job. It even shows the input and output to the reduce job for deriving recommendations.

Chapter 6

Experiments

The algorithm is designed to derive recommendations to the items based on their similarity with set of items. To derive recommendations we used movie-lens dataset as input data for experiments. We use both the binary notation data and non-binary notation data i.e. rating data to derive recommendations. The similarity computation for both the dataset does not change as the difference is just the value representation.

Experiments are done on both single node and multi-node clusters with varying data sizes and performance metrics are recorded. Both the map-reduce jobs run in sequence as the output of first map-reduce job is the input for the second map-reduce job. All the experiments run both the map-reduce jobs to perform the required computations and output the required recommendations for all set of items.

Table 6.1 and 6.2 provides the hardware and software configurations of the system used for experiments.

6.1 Map-Reduce Job 1

Map-Reduce job 1 is responsible for computing the similarity among item pairs. As said earlier, we are using four different similarity measures i.e. Cosine Similarity, Pearson Coefficient, Tanimoto Similarity and Jaccard Similarity to compute similarity among item pairs. Computation of similarity is done on similar data sets. Based on the results from

Computer	HP ProLiant ML110 G6
CPU	Intel Xeon X3430 @ 2.4 GHz
Memory	2GB

Table 6.1: Hardware Configurations of the System

Operating System	CentOS 6.5 Linux Kernel 2.6.32-431
Software Framework	Hadoop 2.6.0

Table 6.2: Software Configurations of the System

the computation, the time taken by different similarity measures is different i.e. Jaccard Similarity takes less time out of four similarities. The order of time taken in ascending order is Jaccard Similarity, Tanimoto Similarity, Cosine Similarity and Pearsson's Coefficient.

6.1.1 Single Node Cluster

Single Node Cluster has one of each HDFS components: JobTracker, TaskTracker, NameNode, Secondary NameNode and DataNode.

A series of steps need to be executed to completely test the algorithm. We can either write a shell script to execute them in a row or manually enter the list of commands sequentially to execute the algorithm.

Command to start Hadoop Distributed File System:

sbin/start-dfs.sh

Command to start Node Manager:

sbin/start-yarn.sh

Command to create a directory in Hadoop File System to store input data:

bin/hadoop fs -mkdir ¡directory-name¿

Command to copy the test data from local system to Hadoop File System:

bin/hadoop fs copyFromLocal ¡local-file-path¿ ¡hdfs-path¿

Command to run a job on Hadoop cluster:

bin/hadoop jar ¡path-of-jar-file¿ ¡class-name¿ ¡input-data-path¿ ¡output-data-path¿

Command to view the results on Hadoop cluster:

bin/hadoop fs -cat /¡output-directory¿/¡output-file-name¿

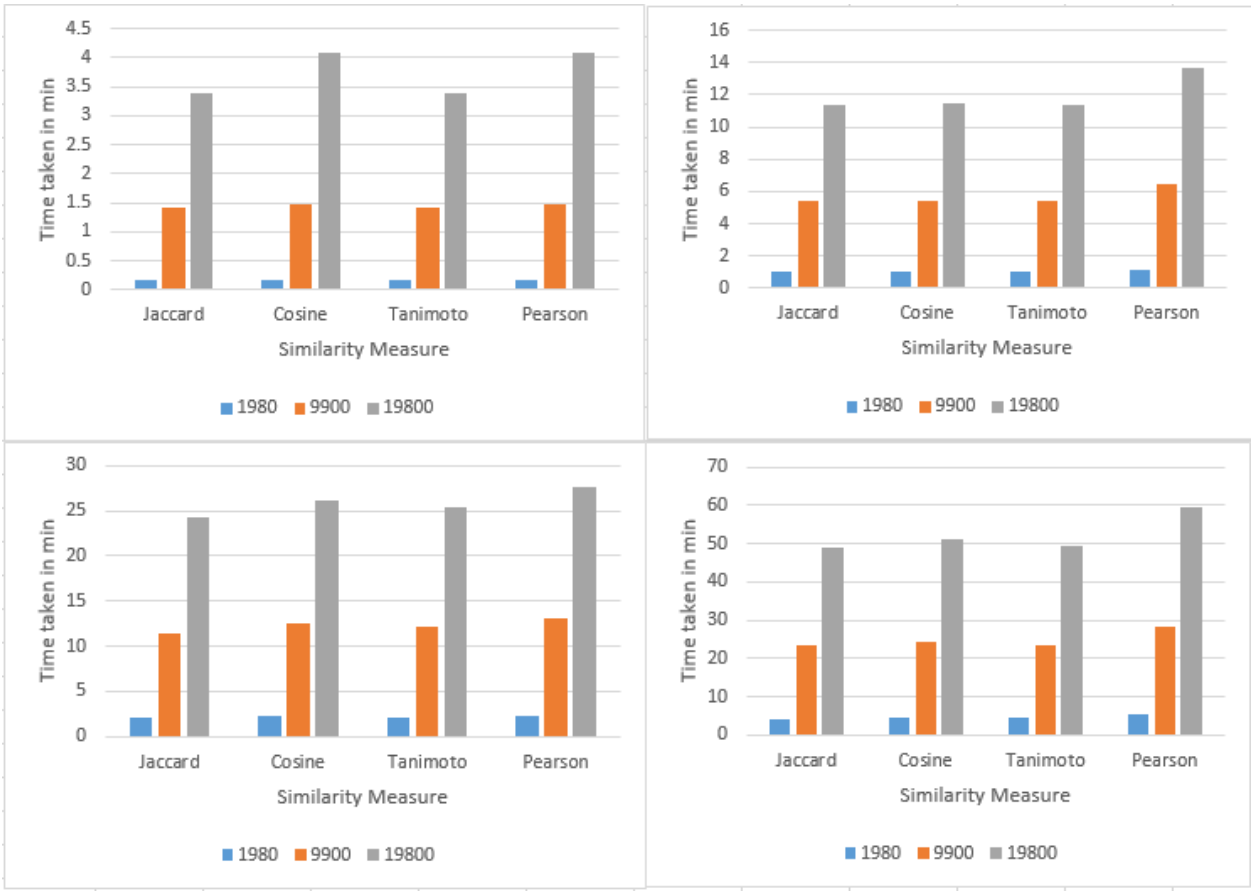

Figure 6.1: Single Node results for different datasets using different similarity measures.

Command to output the results from HDFS to local file system: **bin/hadoop fs -getmerge /¡output-directory¿/¡output-file-name¿ ¡local-file-system-file-name-path¿**

On a single node-cluster, experiments are conducted using different input data sets. The experiments are even conducted for different similarity calculation formulas.

Figure 6.1 provides the results in terms of time taken to compute the similarity among items using different similarities. Different dataset is used for each computation. Dataset includes different set of movie items with different set of users who have given ratings to the movie items. Figure contains graphs for each similarity measure. X-axis represents similarity measure used for computing the similarity among items. Y-axis represents the time taken in minutes to compute the similarity. Legend represents the total number of users who have rated the movie items.

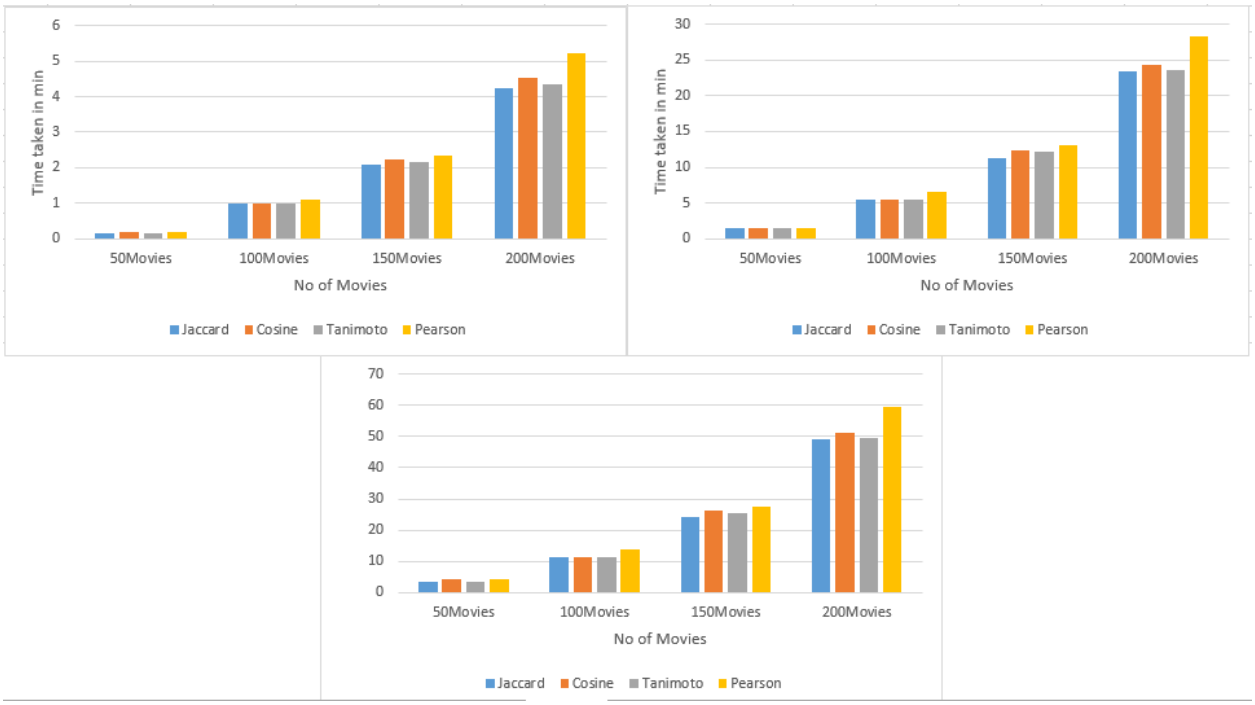

Figure 6.2: Comparison of results for different set of movies using different similarity measure on a Single Node cluster.

From the single node results, all we can say is the time taken to compute the similarity for smaller set of movies using different similarities is almost the same. As the movies increase and the number of users increase, the time taken by the respective similarity measure changes.

Figure 6.2 shows the results for different set of movies using different similarity measures. The number of users for each graph is the same. We can say from Figure 6.2, as the number of movies increases though the users remain the same, the time taken to compute similarity increases.

Figure 6.3 shows the output results using Cosine Similarity computed among different item pairs. Figure 6.4 shows the output results using Jaccard Similarity computed among different item pairs. Figure 6.5 shows the output results using Tanimoto Similarity computed among different item pairs. Figure 6.6 shows the output results using Pearson Coefficient computed among different item pairs.

Based on the output results, we can say the similarity value among item pairs differ among similarity measures. As each similarity measure has a different formulae, the output

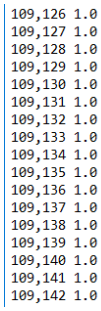

Figure 6.3: Similarity Computation Results using Cosine Similarity. The results shows the item pairs and the corresponding similarity values for item pairs.

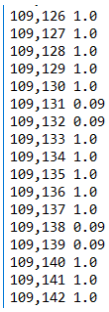

Figure 6.4: Similarity Computation Results using Jaccard Similarity.The results shows the item pairs and the corresponding similarity values for item pairs.

computation among item pairs will differ based on the rating given by different users for the item pairs.

6.1.2 Multi-Node Cluster

A multi-node cluster is a collection of single node clusters, in which one of the nodes act as master node and rest of the nodes act as slave nodes.

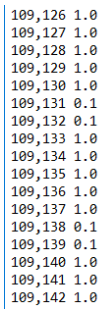

Figure 6.5: Similarity Computation Results using Tanimoto Similarity.The results shows the item pairs and the corresponding similarity values for item pairs.

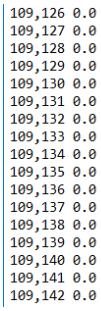

Figure 6.6: Similarity Computation Results using Pearson Coefficient.The results shows the item pairs and the corresponding similarity values for item pairs.

There exits a DataNode and TaskTracker on each slave node and a NameNode and JobTracker on master node. Master node can also have DataNode and TaskTracker in case if it needs to handle the processing itself. A master can also act as Secondary NameNode. In addition, we can even have a single node behave as a Secondary NameNode.

Following experiments were conducted on different set of node clusters with a map reduce implementation of our approach.

Experiments with different datasets are conducted on 2-node, 3-node, 4-node and 6-node clusters.

2-Node and 3-Node Cluster

A multi-node cluster is set up to test the algorithm performance on large data sets. For a 2-Node cluster, the cluster has one master node and one slave node. The master node triggers the task and assigns the tasks to the slave. The master node runs the Job Tracker and NameNode and Secondary NameNode. The slave node runs the Task Tracker and DataNode.

For a 3-Node cluster, the cluster has one master node and two slave nodes. The master node triggers the task and assigns the tasks to the slave. The master node runs the Job Tracker and NameNode and Secondary NameNode. The slave nodes run both the Task Tracker and the DataNode.

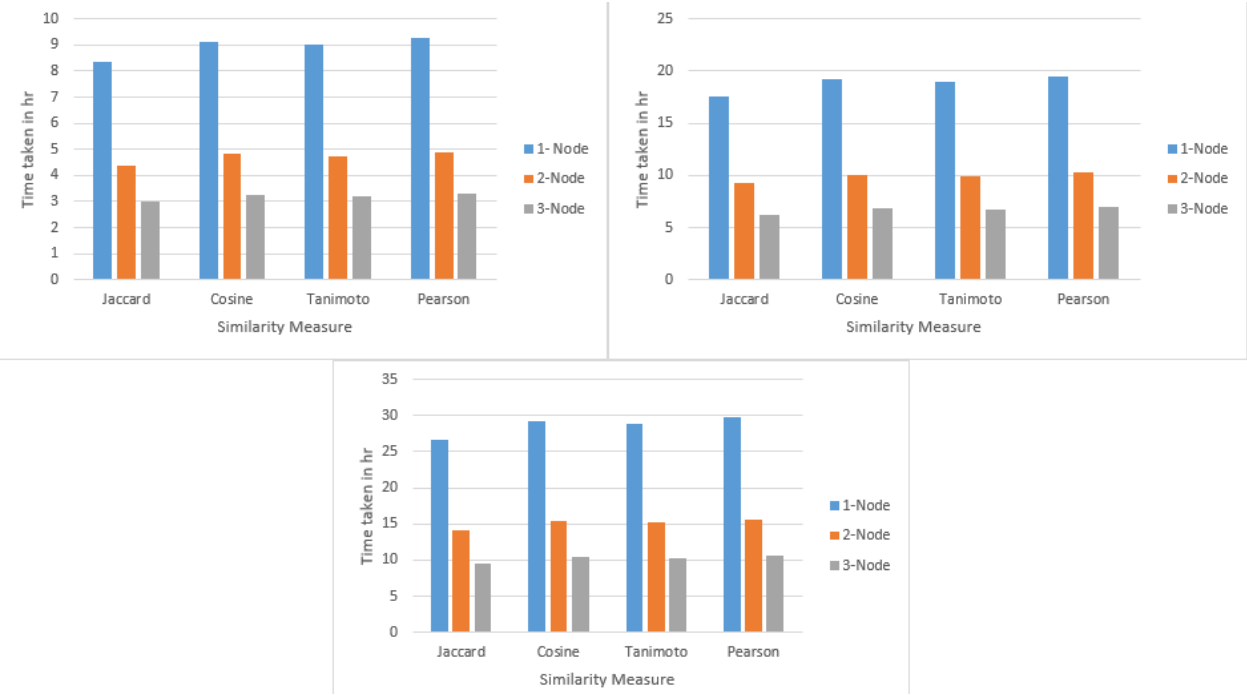

Figure 6.7: Results for different similarity measures on a 1-Node, 2-Node and 3-Node Cluster.

Figure 6.7 shows the results for 1-Node, 2-Node and 3-Node cluster for different similarity measures. The experiments on 1-Node,2-Node and 3-node cluster were carried out on different sizes of the dataset. Based on the results, we can say as the number of nodes increases, the time taken to compute the similarity decreases. The time taken by Jaccard Similarity is relatively less when compared to other similarity measures. The ascending order of similarity measures based on time taken to compute similarities is Jaccard Similarity, Tanimoto Similarity, Cosine Similarity and Pearson Coefficient.

4-Node and 6-Node Cluster

A multi-node cluster is set up to test the algorithm performance on large data sets. For a 4-Node cluster, the cluster has one master node and three slave nodes. The master node triggers the task and assigns the tasks to the slave. The master node runs the Job Tracker and NameNode and Secondary NameNode. The slave nodes run both the Task Tracker and the DataNode.

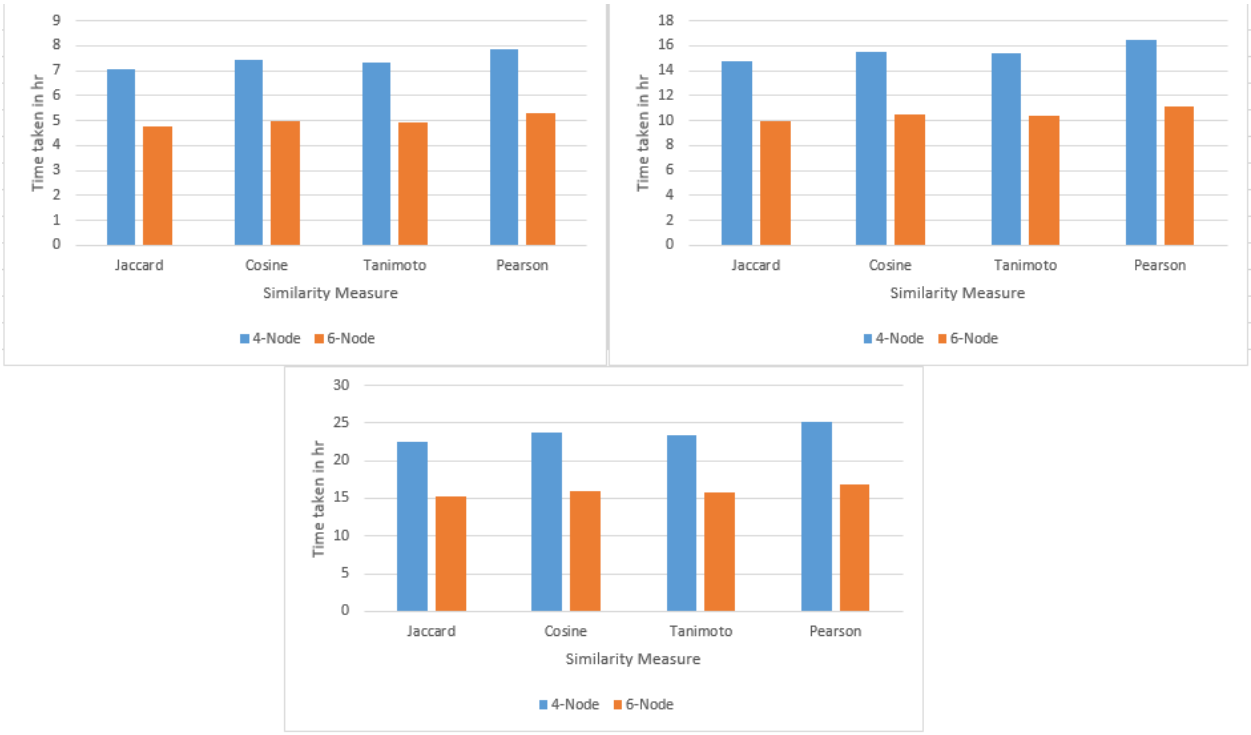

Figure 6.8: Results for different similarity measures on a 4-Node and 6-Node Cluster.

For a 6-Node cluster, the cluster has one master node and five slave nodes. The master node triggers the task and assigns the tasks to the slave. The master node runs the Job Tracker and NameNode and Secondary NameNode. The slave nodes run both the Task Tracker and the DataNode.

Figure 6.8 shows the results for 4-Node and 6-Node cluster for different similarity measures. The experiments on 4-Node and 6-Node cluster were carried out on different sizes of the dataset. Based on the results, we can say as the number of nodes increases, the time taken to compute the similarity decreases. The time taken by Jaccard Similarity is relatively less when compared to other similarity measures. The ascending order of similarity measures based on time taken to compute similarities is Jaccard Similarity, Tanimoto Similarity, Cosine Similarity and Pearson Coefficient.

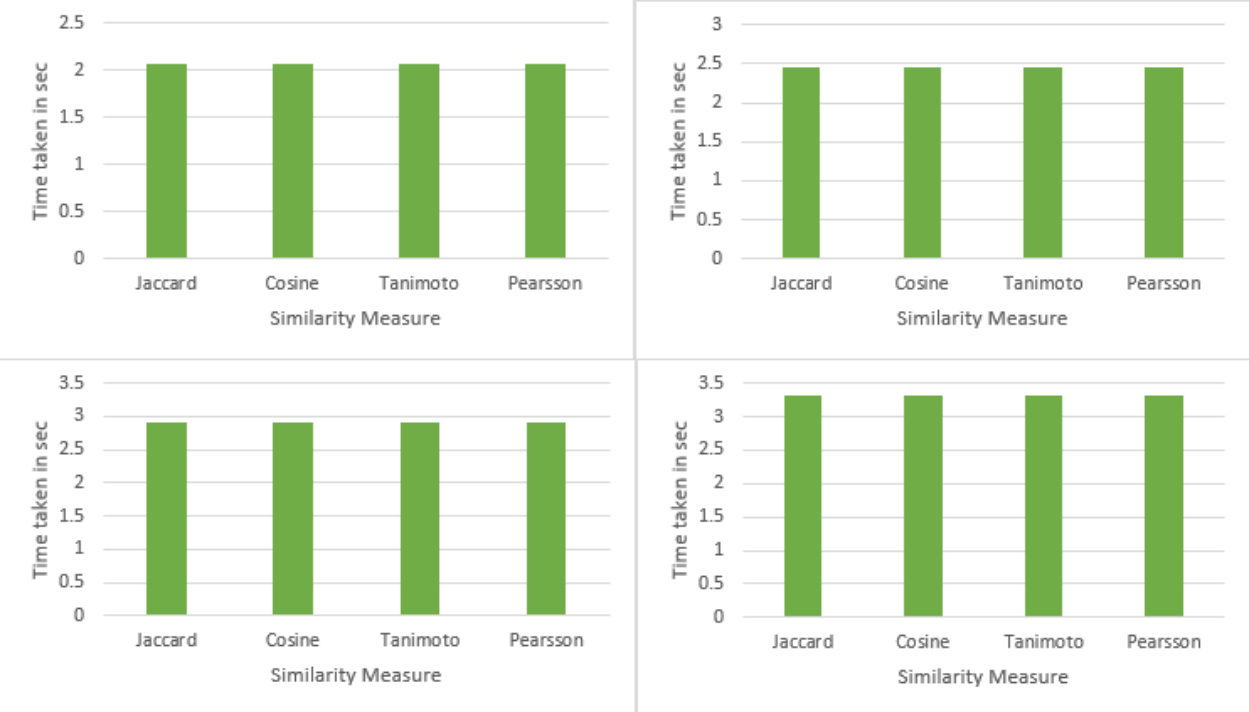

Figure 6.9: Deriving Recommendation for Different Similarity Measures

6.2 Map-Reduce Job 2

Map-Reduce job 2 is responsible for computing recommendation based on the similarity value among item pairs. Based on the results from the computation of similarity, the time taken to compute recommendation for different similarity measures is almost the same for same set of items. The main reason behind it is the output results of map-reduce job 1 will have same number of item pairs for defined number of items. The only difference in the output results is the similarity value among the item pairs.

Figure 6.9 provides the results in terms of time taken to compute the recommendation among items. Though we have used different similarity measures for computing similarity, all the similarity measures outputs the same set of items pairs for set of items irrespective of input dataset size. X-axis represents Similarity Measure and Y-axis represents time taken in computing recommendation among set of items.

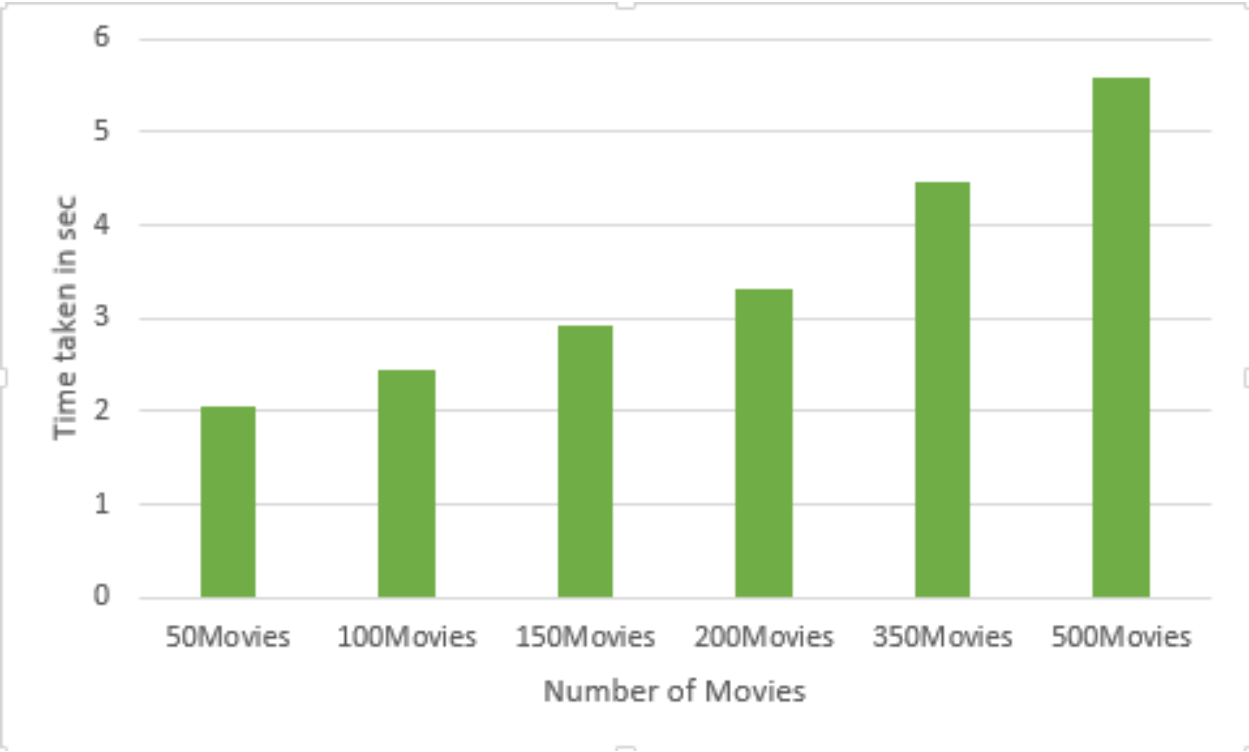

Figure 6.10: Deriving Recommendation for Different Set of Movies

As the number of movies i.e. items increases, the time taken to compute recommendations among items increases. The reason behind it is as the number of items increases, the number of pairs among items increases. The recommendation is derived by considering the similarity value among the set of items and by comparison of similarity values among the items pairs for a specific item.

Figure 6.10 provides the results in terms of time taken to compute the recommendation among different set of items. X-axis represents Similarity Measure and Y-axis represents time taken in computing recommendation among set of items.

Figure 6.7 shows the recommendation results for Cosine Similarity . Figure 6.8 shows the recommendation results for Jaccard Similarity. Figure 6.9 shows the recommendation results for Tanimoto Similarity. Figure 6.10 shows the recommendation results for Pearson Coefficient.

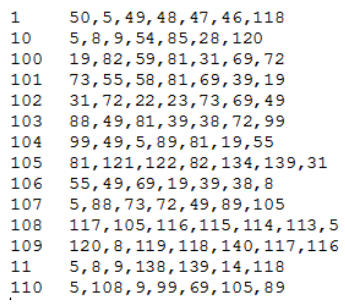

Figure 6.11: Recommendation results for Cosine Similarity. The results show an item and a recommended list of items for the item.

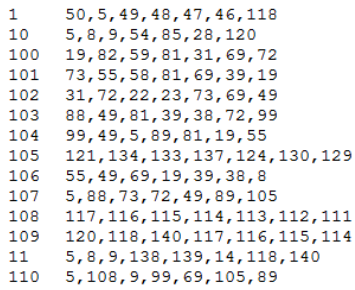

Figure 6.12: Recommendation results for Jaccard Similarity. The results show an item and a recommended list of items for the item.

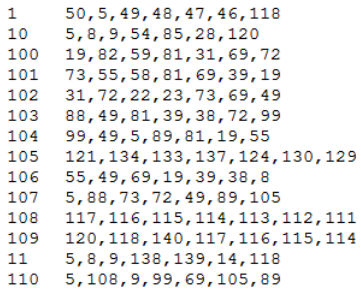

Figure 6.13: Recommendation results for Tanimoto Similarity. The results show an item and a recommended list of items for the item.

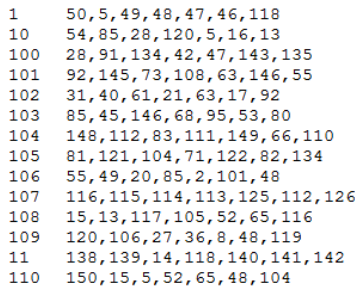

Figure 6.14: Recommendation results for Pearson Coefficient. The results show an item and a recommended list of items for the item.

All the experiments for map-reduce job 2 were carried out in a single node cluster due to the dataset size. If the dataset size is huge, we can extend the computation of deriving recommendations on multi-node cluster similar to map-reduce job 1 in order to maximize the efficiency of time taken to compute recommendations. If the dataset is small enough as in the case of experiments we carried out, we can derive the recommendations on a single node cluster.

Chapter 7

Future Work

The current implementation of the algorithm uses four different similarity measures for computing similarity among items to derive recommendations. Based on the similarity measures used, we were able to decide which similarity measure performs best over the other in terms of computing time. We need to decide which similarity measure is more accurate over the other. In addition, there might be other similarity measures which can be parallelized to find similar items.

The algorithm uses only the rating of the item given by the user to compute the similarity among items but, there can be other parameters which can be considered to improve the accuracy of the recommendations.

There have been two more proposals proposed for deriving item-based recommendations using Hadoop. The proposals can be implemented and thus, can be compared with the current implemented algorithm in order to find which proposal behaves the best in terms of performance and accuracy.

Chapter 8

Conclusions

In this thesis, we discussed about recommendation system algorithms and map-reduce programming model by Hadoop. A parallel algorithm to compute item-based similarity using Hadoop map-reduce framework is explained. The reason behind parallelization of the computation is inspired by word count example using map-reduce. The computation is split into multiple small tasks and each of the small task is handled independently by a map-reduce job. In the thesis, we showed details of the approach including the inputs and outputs produced at each phase of execution.

Different experiments are conducted to show the performance improvement by using a multi-node cluster. Test data for experiments is taken from movie-lens databases. The results show that the performance of the algorithm improves as the number of nodes within a cluster increases with constant dataset size. The Hadoop map-reduce approach saves a lot of resources in computing similarities and generates recommendations in short time.

Chapter 9

Bibliography

1. Yahoo! The Hadoop Distributed File System. https://developer.yahoo.com/hadoop/tutorial/module2.h
2012. [Online; accessed April 2014].

2. Yahoo! Hadoop Tutorial. https://developer.yahoo.com/hadoop/tutorial/module3.html,2012.
[Online; accessed April 2014].

3. Yahoo! MapReduce. http://developer.yahoo.com/hadoop/tutorial/module4.html, 2012.
[Online; accessed April 2014].

4. Yahoo! Advanced MapReduce Features. http://developer.yahoo.com/hadoop/tutorial/module5.html,
2012. [Online; accessed April 2014].

5. Yahoo! Managing Hadoop Cluster. http://developer.yahoo.com/hadoop/tutorial/module7.html,
2012. [Online; accessed April 2014].

6. Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. (2012). GroupLens: Applying Collaborative Filtering to Usenet News. Communications of the ACM, 40(3), pp. 77-87. [Online; accessed May 2014].

7. Shardanand, U., and Maes, P. (1995). Social Information Filtering: Algorithms for Automating Word of Mouth. [Online; accessed May 2014].

8. GroupLens Research Group. Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering. [Online; accessed May 2014].

9. Jeffrey Dean and Sanjay Ghemawat, Google, Inc. MapReduce: Simplified Data Processing on Large Clusters. [Online; accessed June 2014].

10. Apache Software Foundation. Hadoop.. http://hadoop.apache.org/hadoop. [Online; accessed June 2014].

11. Apache Hadoop. Hadoop Distributed File Systems (HDFS). http://hadoop.apache.org/docs/r0.17.1/hd

accessed June 2014].

12. Yahoo. Yahoo! launches worlds largest Hadoop production application.. http://tinyurl.com/2hgzv7. [Online; accessed July 2014].

13. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews.. [Online; accessed August 2014].

14. Dean J, Ghemawat S. (2007). Distributed programming with Mapreduce.[Online; accessed August 2014].

15. Ghemawat S, Gobioff H, Leung ST. The Google file system.[Online; accessed August 2014].

16. A. Metwally and C. Faloutsos. (2012). V-smart-join: A scalable MapReduce framework for all-pair similarity joins of multisets and vectors.. [Online; accessed September 2014].

17. J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems.. [Online; accessed September 2014].

18. Adomavicius G., Tuzhilin A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.. [Online; accessed October 2014].

19. Douglas Thain, Todd Tannenbaum, and Miron Livny. (2005). Distributed computing in practice: the condor experience: Research articles. [Online; accessed October 2014].

20. Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans arno Jacobsen, Nick Puz, DanielWeaver, and Ramana Yerneni. (2008). Pnuts: Yahoo!s hosted data serving platform.. [Online; accessed November 2014].

21. Shang Ming-Sheng, Zhang Zi-ke. (2009). Diffusion-Based Recommendation in Collaborative Tagging Systems.China. [Online; accessed November 2014].

22. Shang Ming-Sheng, Jin Ci-Hang, Zhou Tao, Zhang Yi- Cheng. (2009). Collaborative

filtering based on multi-channel diffusion. Physics A: Statistical Mechanics and its Applications. [Online; accessed November 2014].

23. Chu LK, Tang H, Yang T, Shen K. (2003). Optimizing data aggregation for cluster-based Internet services. [Online; accessed January 2015].

24. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad. (2007). Distributed data-parallel programs from sequential building blocks. [Online; accessed February 2015].

25. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. (2007). Dynamo: Amazons highly available key-value store. [Online; accessed February 2015].

26. Brad Hedlund. Understanding Hadoop Clusters and the Network. www.ibm.com/software/data/infosp
[Online; created March 2015].

27. IBM. IBM Hadoop. http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html, 2012. [Online; accessed March 2015].

28. Apache Software Foundation. The hive project http://hadoop.apache.org/hive. [Online; accessed April 2015].

29. Apache Software Foundation. The pig project.. http://hadoop.apache.org/pig. [Online; accessed April 2015].

30. Apache Software Foundation. Apache Hadoop.. http://hadoop.apache.org/zookeeper. [Online; accessed April 2015].

31. Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl. (2001) Item-Based Collaborative Filtering Recommendation Algorithm. Grouplens Research Group. [Online; accessed January 2015].

32. Charu C. Aggarwal, Alexander Hinneburg, Daniel A. Keim. On the Surprising Behaviour of Distance Metrics in High Dimensional Space. IBM T.J. Watson Research Center. [Online; accessed April 2015].

33. Charu C. Aggarwal. Re-designing Distance Functions and Distance-Based Applications for High Dimensional Data. IBM T.J. Watson Research Center. [Online; accessed April

2015].

34. Mohammad Kolahdouzan ad Cyrus Shahabi. Voronoi-Based K Nearest Neighbour Search for Spatial Network Databases. Proceedings of the 30th VLDB Conference. [Online; accessed April 2015].

35. Alexander Hinneburg, Charu C. Aggarwal, Daniel A. Kein. What is the nearest neighbor in high dimensional spaces?. Proceedings of the 26th VLDB Conference. [Online; accessed May 2015].

36. Yang Song, Lu Zhang and C.Lee Giles. Automatic Tag Recommendation Algorithms for Social Recommender Systems. Microsoft Research. [Online; accessed April 2014].

37. Maurice W Benson, Paul O. Frederickson. Fast Parallel Algorithms for the Moore-Penrose Psuedo Inverse. (1986). Second Conference on Hypercube Multiprocessors. [Online; accessed April 2014].

38. Joseph w. H. Liu. A Graph Partitioning Algorithm by Node Separators. ACM Transactions on Mathematical Software. [Online; accessed May 2014].

39. Francois Foss, Alain Pirotte, Jean-Michael Renders, Marco Saerens. Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation. (2007). IEEE Transactions on Knowledge and Data Engineering. [Online; accessed June 2014].