DISTRIBUTED RECOMMENDER SYSTEM USING MULTI-AGENT FOR SOCIAL NETWORKS

by

Lubaid Ahmed

M.Sc. Computer Science, Ryerson University, Toronto, Canada, 2010

M.Sc. Computer Science, NED University of Engineering & Technology, Karachi, Pakistan, 2002

M.Sc. Applied Physics with Specialization in Electronics, University of Karachi, Pakistan, 1996

B.Sc. University of Karachi, Pakistan, 1993

A dissertation
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in the Program of
Computer Science

Toronto, Ontario, Canada, 2016

Author’s Declaration

**DISTRIBUTED RECOMMENDER SYSTEM
USING MULTI-AGENT FOR SOCIAL NETWORKS**

Lubaid Ahmed, 2016

Doctor of Philosophy

Computer Science

Ryerson University

Abstract

Social networks have become significant tools due to the vast and useful information existing in them. The social platforms also act as the storage of entered choices of millions of users for various applications such as political surveys, research studies, marketing product preferences and many more. Social network recommender systems exploit this information and direct users in selecting their choices. It is clear that recommender systems should be efficient enough to be able to process the huge magnitude of data that has been generated in recent years by social network users. This research proposes a foundation of an efficient and scalable recommender system to be able to process large amount of data (i.e. Big data) in a short amount of time. The main goal is providing scalability and efficiency of the recommender system. The simulation of the prototype of such a distributed recommender system by using multi-agent based technologies shows promising results. These prototypes provide recommendations to users about other users with the similar interests in online and distributed manner as real recommender systems. The agents can simulate users or can be used as the containers of algorithms for comparing the similarity between users by different approaches, such as cosine similarity and clustering methods for testing and examining real scenarios. To be able to test these prototypes in agent-based simulation environment an agent-based framework is developed. This framework has three modules named social network crawler, social network simulator and employed prototype of the distributed recommender system that use different text and data mining algorithms. Finally, newly developed performance metric (called Scalability Factor) is introduced that shows the minimum number of servers needed to be able to run the agent systems in parallel. This thesis shows using a distributed and parallel model for recommender systems is the key to increase the speed of recommendation convergence and as a result to provide scalability. Multi-agent based simulation results, coupled with numerical analysis affirm that the proposed solution provides scalability and efficiency for recommender systems.

Acknowledgements

It is my honour to express my deep gratitude to the people who made this challenge possible for me. I would like to thank my supervisor Dr. Abdolreza Abhari, for giving me the opportunity to work under his supervision. He offered me such a challenging topic. With his support and guidance, I am finally able to achieve my goals. I would also like to express my gratitude to my thesis committee members, Dr. Alex Ferworn, Dr. Isaac Woungang, Dr. Denis Hamelin, Dr. Kosta Derpanis and Dr. Alagan Anpalagan for their time and effort they spent reviewing my thesis. My sincere appreciation is extended to Dr. Shervin Shirmohammadi, University of Ottawa, for his valuable comments and agreeing to be in my committee.

I also thank all the members of our research group in the Distributed Systems and Multimedia Processing lab (DSMP) for their support.

Finally, and most importantly, my family. A special and deep gratitude to my mother Mrs. Lubna Farooqui who taught me to cherish knowledge and seek for it. Also, I would like to thank my brothers Saad and Munad, and my wife Sadaf, they are always being cooperative and patient. It is their prayers and support that gave me courage to achieve my goals.

Dedication

To my Mother

I would not be here without her support, encouragement and strong belief in me.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACL	Agent Communication Language
ANNS	Approximate Nearest Neighbour Search
ATC	Automatic Text Categorization
CBF	Content-Based Filtering
CDRS	Clustering-based Distributed Recommender System
CF	Collaborative Filtering
DCSP	Distributed Constraint Satisfaction Problems
DF	Demographic Filtering
DS	Distributed Systems
EM	Expectation Maximization
GAAC	Group Average Agglomerative Clustering
GPU	Graphic Processing Unit
HAC	Hierarchical Agglomerative Clustering
HAC	Hierarchical Agglomerative Clustering
HC	Hierarchical Clustering
HF	Hybrid filtering
IA	Initializer Agent
IDF	Inverse Document Frequency
IR	Information Retrieval
JADE	Java Agent DEvelopment framework
JMSD	Jaccard Mean Squared Differences
kNN	k-Nearest Neighbours
MAS	Multi Agent System
NLP	Natural Language Processing
NNS	Nearest Neighbour Search
OU	Organizing Unit
RA	Recommender Agent

RS	Recommender System
RU	Recommender Unit
SDRS	Similarity-based Distributed Recommender System
SM	Similarity Measure
TA	Tweeting Agent
TARS	Trust-Aware Recommender System
TF	Term Frequency
TFIDF	Term Frequency Inverse Document Frequency
TPU	Text Processing Unit
UA	User Agent
UGC	User Generated Contents
URL	Universal Resource Locator
VSM	Vector Space Model
WC	Word Count

List of Appendices

Chapter 1

Introduction

1.1. Motivation

Recently web researchers have proposed several data mining and web mining methods which are used for building recommendation systems for the Web 2.0 social networking sites such as Facebook, Twitter, YouTube, and LinkedIn. Unfortunately, the large amount of data feeding in social networks by millions of users causes efficiency problems in the online employment of these methods for real time recommender systems. Therefore, scalability and efficiency of recommender systems are open questions and hot research topics.

One of the known and common methods for improving the scalability and efficiency of massive data processing applications is to develop them as parallel and distributed systems. Currently, the use of cloud computing and common use of distributed data processing tools such as Hadoop and MapReduce, and availability of hundreds of servers make it possible to turn the central applications to parallel or distributed applications at low cost. However, there are limited studies that show the benefits of transforming a central data processing system to distributed system. Many studies that proposing methods or guidelines for the transformation of a central text processing recommender system into a parallel and distributed system are either suggest using distributed algorithms instead of central algorithms or employing distributed file system for data processing which needed change of the program or software and data modelling of the applications. The simplest proposed method for recommender system which does need change of application, is dividing the data in smaller granularity (called chunks) to provide scalability. This thesis studies the benefits of simple transformation of a central recommender system by replicating an entire recommender system on a distributed and parallel model without changing used algorithms, adding additional software layer or reducing the granularity of the data needed to be processed by a recommendation system.

This thesis will first define a formal model of the problem and then introduce a framework implemented by multi-agent based systems (MAS) to be able to conduct a simulation. Multi-agent based prototypes used in the simulations of turning central text processing recommender applications to distributed systems, determine the benefits based on number of processed tweets. These multi-agent based systems will be used as data crawlers to input real data for examining the recommender system’s efficiency as well as recommending the users with other similar users while they are entering data in Web 2.0 social networking platforms. The main feature of the distributed recommender system is the capability of using distributed storages of multiple servers instead of one server to distribute the load of data when performing data processing, which significantly reduces the running time. The use of distributed multi-agent based framework with text based recommender systems in social network sites is a novel idea. To measure performance of such a framework two variations of distributed recommender systems have been developed and compared with each other. Besides comparing the accuracy, new performance metrics (specifically to measure distributed processing effectiveness and scalability) are used for examining the employed Web mining methods in these recommender systems.

1.2. Problem Statement

The problem is to design a scalable and efficient recommender system to perform the recommendations by text processing based data mining algorithms (i.e. functions). In this design rather than inventing a new data mining or recommending algorithm the common algorithms (i.e. clustering and cosine similarity) employed in the recommender systems are used in parallel to provide scalability. Given that such algorithms are fully replicated in a distributed environment including servers containing local databases, the number of servers (i.e. scalability factor) of algorithms are unknown. The general form of the problem is explained as follows: Assume a recommender system has the Accuracy of *A* and is consisting of a set of algorithms that are shown as the functions:

$$F = \{ f_1, f_2, ..., f_h \}$$

Where,

F = Set of function or algorithm of the recommender system.

f = Functions or algorithms.

h = Total number of functions.

For processing the information of k users, each of the users generates tweets shown by the set of

$$U_i T = \{U_i t_1, U_i t_2, ..., U_i t_j\}$$

Where,

U_i = User i.

T = Set of Tweets.

j = Number of tweets.

Let $\hat{S} = \{S_1, S_2, ..., S_m\}$ be the sets of servers that are hosting functions *F*, where S_i represents the server number which is hosting function *F*. Each server *m* has a limited storage capacity (C_i) for storing in its local database which is required to be processed by a set of functions *F*, given that each function uses the tweet vectors which are locally stored on each server's local database. The global storage constraint is the total numbers of stored tweet vectors $\sum_{i=1}^{k} |U_i\, T|$ which should be equal to the total number of tweet vectors on the centralized recommender system (where, *k* is the number of users). For the distributed versions of recommender systems (presented in this thesis) this constraint will be distributed constraints and for simplicity defined as tweet vectors distributed almost equally on each server. It means assuming the tweet vectors of each user stored entirely on one server. This constraint is defined as follows

$$\sum_{i=1}^{l} |U_i\, T| \leq \sum_{i=1}^{k} |U_i\, T|/m$$

Where,

l = Number of users whose tweet vectors are stored in local servers.

m = Number of servers.

k = Total number of users.

For the given number of tweets it is required to find the minimum number of servers (i.e. scalability factor) in the set of $\hat{S}$, which is a set of *m* servers that host recommendation system consisting of function *F*, provide scalability for the recommender system.

That shall:

- Reduce the running time of recommendation process by running the functions in parallel.

- Produce the recommendations for a user U_i with the same degree of accuracy of *A* by recommending the similar users who have the highest possible similar tweets as in the central version upon receiving a request from each user U_i.

- Considering the amount of message passing in the system should be in the range of $|U_iT| * (\sum_{i=1}^{m} | S_i| - 1)$ and the system should be capable of providing recommendations upon receiving a request from each user U_i. Where, $|U_iT|$ is the cardinality of the set U_iT.

The variable in the general form of the problem is to find the number of replications (i.e. replication factor) of each function on the *m* hosting servers and the local constraint can be number of tweets and the processing power of each server. However, in reality the number of servers is limited, so in this thesis the simplified model is used for simulation. In the simplified simulation model, we assume all servers have the same processing power and approximately equal number of tweets and all functions are replicated just one time on each server. In Chapter 3, the relation of time complexity models of the functions will be examined against the number of tweets to be able to define the scalability factor for each function and finally for the whole recommender system. These models will be implemented by agents for simulations discussed in Chapter 4 and Chapter 5.

1.3. Recommender Systems

Recently, the recommender system that is based on social network data, has gained importance in the applications of data mining. The internet users constantly share their views and ideas using these social networking sites. This valuable information is increasing every day. The recommender systems are based on user's preferences, needs, and desires, which are the inputs to the recommender system. Textual messages in microblogs directly relate to sentiments of public opinion, which are measured from the polls. Data that is present in social networking sites are growing exponentially. The recommender system that uses collaborative filtering techniques reduces the information overload [1]. The application of this well-known technique on e-commerce shows its limitation of scalability [2]. The purpose of recommender system is to acquire previous knowledge from different inputs and gives suggestions for the given problem. The effectiveness of the recommender system mainly depends on how it makes good suggestions for those who are seeking recommendations. The recommender system uses information from different sources as their input. Nowadays, many recommender systems are internet dependent, as we have huge significant data (text, multimedia) that can be used for commerce, politics and security. The recommender system emerges as an excellent technique, which enables us to find the best solution.

The idea is to develop a recommender system that uses distributed and parallel algorithms to process big social data in real time. For such development a multi-agent based system is designed to simulate users in the Twitter environment. The Twitter social network, users' behaviors and their communication with the recommender system and the combination of recommender system modules are considered as complex environment, which can be effectively simulated by multi-agent based systems.

In the proposed system, the Twitter user data, such as tweets are analyzed to find user interest. However, it is a challenge due to three main reasons: the tweets have a maximum of 140 characters, unstructured language is used in the tweets and the volume of tweets is huge as millions of user tweets in a given time. These challenges guarantee the scalability issues.

Recently, the recommender system that is based on social network data has gained importance in the field of information retrieval. But the fact is that it is difficult to find valuable and useful information from big social data for recommendation purposes. For example, for the Twitter due to the challenges mentioned above, it is a very time consuming technique to use central algorithms available for recommender systems. Therefore, efficiency of social recommender system is dependent on the processing power of the server. There are many recommender systems available for social networks, but the efficiency of the recommender system in real time is an open question. These central algorithms are time consuming and cannot work in an online manner. In this thesis, a framework is proposed that employs common recommender system algorithms as distributed and parallel components working on processing tweets which are distributed on the servers. The proposed framework is called distributed recommender system and addresses the scalability problems of the algorithms used in current distributed systems.

1.4. Methodology

In this thesis, a distributed and parallel model is proposed to provide scalability for a recommender system. An agent-based model is developed and the algorithms of TF, TFIDF, k-mean and cosine similarity are used for agent-based simulation of simplified model. By building prototypes of many agents, each of them acts as the representative algorithm (function) used in text-based recommender systems simulation. The communication between the agents can be performed by message passing between their hosting servers. In Chapter 3, the theoretical foundation of building a distributed recommender system is presented by examining common algorithms and showing the benefits in terms of reducing the running time for these common algorithms when used in recommender systems. Also, cost is defined in terms of message passing when using a distributed system consisting of these algorithms in parallel. For the simplified problem addressed in the thesis, numerical analysis and multi-agent based simulation are used to compare theoretical and practical solutions. In Chapter 4, prototyping, specification and implementation of multi-agent based framework are detailed. Two variations of distributed recommender systems are implemented by using agent systems, one based on clustering and the other based on similarity of data. In Chapter 5, we explain how we measure the accuracy to be able to get the same accuracy in distributed or parallel version of the examining recommender system as their central variation. To measure the accuracy of each system the collected data of the Twitter follower and followees information will be measured by using the mean average error performance metrics. Measuring the benefits and cost in recommender systems by reporting the reduction in processing time of the system when implementing as distributed recommender system together with five real data sets is presented. MAS is used for prototyping and implementing a distributed recommender system and simulating the users who are generating text based information collected from real data. Using real data by collecting user selections from user network information makes it possible to assign an accurate value to the examining recommender systems and examine them against distributed versions of the system. For all these steps, only the Twitter public information of user tweets and user network information are required. Finally by comparing the achieved results in Chapter 3 (by doing numerical analysis) and simulation results in Chapter 5, the scalability factor (i.e. number of servers that make the recommender system scalable) for five data sets (maximum 25,000 of tweets) is determined theoretically and in practice for the above problem.

1.5. Contributions

Three main contributions are made in this thesis, as follows:

- <u>Distributed Recommender System</u>: Implementation of the novel distributed recommender systems by using multi-agent based system and providing analytical and experimental values for efficiency.

- <u>Implementing the recommender system by using multi-agent system</u>: Two variations of the distributed recommender system with different architectures are developed in this thesis by using multi-agent based technology. The first architecture uses a similarity approach called Similarity-based distributed recommender system (SDRS) and the second uses clustering method called Cluster-based distributed recommender system (CDRS). Implementation of both systems by using MAS has been introduced and detailed in this thesis.

- <u>Providing analytical and experimental values for efficiency (i.e. scalability factor):</u> In Chapter 3, the scalability factors for two variations of developed distributed recommender systems are presented by finding the number of servers that produce the linear relation of running time and number of tweets for consisting algorithms. The effectiveness of both memory and disk storage-based scenarios for processing data have been tested and together with the accuracy of each system are provided in Chapter 5.

- <u>For all used datasets:</u> We found that for processing of up to 25,000 of tweets adding minimum of two servers and transforming a central common recommender system into a parallel and distributed system provides scalability. The recommender systems that use clustering-based algorithms benefit more from distributed model explained in the thesis than the ones using similarity-based algorithms.

Contrary to the centralized recommender system, the suggested distributed systems architecture uses specific numbers of servers to improve the performance by running parallel algorithms to process massive data. The number of servers can be given as the constraint, which can be determined by the finding of this thesis. In the last chapter, by comparing the two developed distributed systems with memory and disk data storage, the values for scalability factor and performance improvement are shown for different algorithms use in the distributed recommender system. These values can be used as the guideline for transforming current central recommender systems to distributed ones.

Transferring this prototype to a real world distributed recommender system is a straight forward task. A user agent in form of very light weight app. can be installed in user social network account. This app can communicate with other distributed components (i.e. agents) which are running in parallel in a cloud or multi-server environment. The following are the publications related to this work.

- L. Ahmed and A. Abhari. "Agent-Based Simulation of Twitter for Building Effective Recommender System." In *Proceedings of 17th Communications and Networking Simulation Symposium (CNS14) of SCS/ACM*, 1-7, Tampa, Florida, USA, 2014.
- L. Ahmed and A. Abhari. "A Multi-Agent-Based Simulator for a Transmission Control Protocol/Internet Protocol Network." *SIMULATION: Transactions of The Society for Modeling and Simulation International*, *vol.* 90, no. 5 (May 1, 2014): 511-21, 2014.

- L. Ahmed and A. Abhari. "Distributed Recommender System for Online Processing of Big Social Data." In *Spring Simulation Multiconference (SpringSim'15) of SCS/ACM*, 699-700, Alexandria, Virginia, USA, 2015.
- L. Ahmed and A. Abhari. "Information Retrieval using Multi-Agent Distributed System for Social Networks" In *Information Retrieval Journal, 2015 (under review).*

1.6. Thesis Organization

This thesis is composed of the following chapters:

Chapter 2: Background and Related Research

In this chapter, we discuss previous works on the subject and their limitations. We then discuss the motivations behind our work.

Chapter 3: Modelling the proposed Distributed Recommender Systems

This chapter constitute of the core of this thesis. In this chapter, we show the relation of running time of the parallel run of the algorithms for each distributed recommender system against a number of tweets employed based on typical information retrieval algorithms. This chapter also details the scalability degree for each participating algorithm and distributed processing effectiveness of the combined distributed recommender system architecture.

Chapter 4: Multi-Agent based Framework, Specification and Implementation

This chapter details the implementation of the multi-agent based simulation framework and prototypes. We describe a simulation framework, including a development of the Twitter simulator and Twitter data crawler based on multi-agent systems. We also present various algorithms used in the distributed recommender system prototypes. This chapter also provides an insight into the theoretical aspects on which our work is based. The papers discussing the architecture of agent based system and the initial results of this work are published in [3, 4, 5].

Chapter 5: Simulation and Experimental Results

In this chapter, the prototyped multi-agent based system will be used in simulation. Measuring the efficiency performance of the proposed framework is of course an essential part of this research work. In this chapter, we describe the simulation setup, scenario, performance parameters and the results. Also validation and verification techniques together with the effects on accuracy and comparison with a similar work discussed.

Chapter 6: Conclusions and Future Works

We conclude our work and present future possible directions that can be done to extend the scope of the work we carried in this thesis.

Chapter 2

Background and Related Research

This chapter presents the studies and research that are related to this thesis. This chapter is divided into two main sections. In Section 2.1, background information about the design issues and components of recommender systems are explained. In Section 2.2, related research and state of the art recommender systems are discussed. Section 2.1 is further divided into five subsections. Section 2.1.1, discusses information filtering techniques that are used in recommender systems. Section 2.1.2 explains the similarity measures that are used in recommender systems, Section 2.1.3 discusses clustering method. Section 2.1.4 shows recommender systems that are used for social network. Section 2.1.5 discusses multi-agent systems with their use in recommender systems. In Section 2.2.1 state of the art systems are presented. At the end, in Section 2.3 a summary of this chapter is presented.

2.1. Background Information

2.1.1. Filtering Techniques used in Recommender Systems

Many techniques are proposed to build recommender systems. These techniques are used in different applications and domains. The main component of any recommender system is its filtering technique. The recommendation provided by a recommender system is dependent on the precision and accuracy of filtering technique. There are four main filtering techniques [6, 7], which are Content-based filtering, Collaborative filtering, Hybrid filtering and Demographic filtering techniques.

2.1.1.1. Content-Based Filtering

Content-based filtering (CBF) is discussed in [8]. This filtering method uses information about the selection or viewed items by a user and then recommends new item to that user on that information. The recommender system that is built on CBF needs to select the right set of features to be used in recommendations.

2.1.1.2. Collaborative Filtering

Collaborative filtering (CF) technique uses collaboration of previous user ratings histories. This means that when a user purchases an item, the system asks the user to rate the purchased item. On the basis of this collected information, the item is suggested to the target user, as it is highly rated by similar users. The collaborative filtering does not require item knowledge; therefore, it can be applied to any kind of items and is used in many online recommendation systems, including websites, songs, videos, jokes, stocks, books and news articles. For example, CF is used in Amazon [9] for recommending items. Additionally CF technique is used for filtering emails as in [10].

According to [11], k-Nearest Neighbours (kNN) is the most commonly used algorithm in collaborative filtering. kNN is the reference algorithm for recommendation systems which use collaborative filtering for its simplicity and accuracy [12]. kNN uses a similarity approach to calculate recommendations. The similarity between the two users is calculated based on their item ratings. This algorithm uses the following steps for recommendations: 1) the set of k neighbour users for active user A is generated using selected similarity measures (such as the Pearson Correlation, Cosine, Constraint Pearson Correlation, Mean Squared Differences, Euclidean etc.). The resulted set represents the users who are similar to a user A. 2) an aggregation approach (such as average, weighted sum and adjusted weighted aggregation) is used to predict item I for the user A. From the list of recommendations, top n items are recommended to the active user A. The main limitations of this algorithm are that it lacks in scalability [13] and sparsity [14] in recommendation system database.

2.1.1.3. Demographic Filtering

Demographic filtering (DF) technique uses the information about the user, which depends on the user's age, country, sex etc. In [15, 16] a framework that uses user's profile to recommend restaurants is presented. That framework uses DF along with CBF and CF.

2.1.1.4. Hybrid Filtering

Hybrid filtering (HF) technique is the combination of two or more filtering techniques [17, 18]. In [19], a hybrid filtering technique that combinations CF and DF is presented. In [20], CF and CBF are combined. In [21], a recommendation system is presented for maintaining the accuracy and scalability of CF. In this recommendation system, a hybrid fuzzy-genetic approach is used.

2.1.2. Similarity Measures

Similarity measure (SM) is the metric which is used to determine the similarity of two users or two items [22]. In RS context, similarity measure can be used to compare all the items ratings of two users. The widely used similarity metrics are discussed in [7, 23, 24], which are Pearson Correlation, Cosine Similarity, Adjusted Cosine, Constrained Correlation, Mean Square Differences, Euclidean, and Jaccard Mean Squared differences.

In [23], Jaccard Mean Squared Differences (JMSD) is introduced. This similarity measure uses both numerical information of rating (using Mean Squared Differences) as well as non-numerical information (using Jaccard). This method uses Pareto dominance for pre-filtering, which is useful in removing users with less representation value, in turn, keeps most valuable users in the k nearest neighbours process. In [25], a collaborative filtering SM Sing is proposed. This method uses vote information for all users and related information for two users or two items. In [26], genetic SM GEN is proposed, this similarity metric is based on a model generated by using genetic algorithm.

Due to significant increase in Web 2.0 applications and a huge number of users of social media websites, new types of social information are introduced, such as friends, followers, etc. These types of information can be used in social recommender systems. This information is based on user trust, reputation and their credibility [27]. In [28, 29], trust information is extracted using user’s rating set. In [30], it is shown that it is more effective to use the specific similarity metric which includes some recommendations related parameters than using traditional similarity metrics.

In [31], a framework for evaluating collaborative systems is presented. This framework considers the evolving process of the evaluated collaborative system. Validation of recommender system can be performed by using a cross validation technique of random sub-sampling and K-fold cross validation [32].

2.1.3. Clustering Methods

Clustering problem is defined as partitioning a set of data points into different groups, where the points in each group are as similar as possible [33]. The most common clustering techniques are: K-mean Clustering and Group Average Agglomerative Clustering (GAAC).

2.1.3.1. K-mean Clustering

K-mean clustering is a flat clustering technique that clusters the data point to a predefined number of clusters [34]. In k-mean, the following steps are applied in order to cluster n data points in euclidean space: 1) initially and randomly select *k* points that represent centriod of the clusters, 2) perform the following steps for all other remaining *(n-k)* points in Euclidean space: a) Check centriod of all clusters against each data point and assign that point to the nearest cluster. b) Calculate the mean of all data points and assign the new value to the centriod. c) For all data points in the clusters, compute the Euclidian distance from the centroid of all clusters. Move the data point to its nearest cluster and then update cluster's centroid. d) Repeat step c, until convergence is achieved.

2.1.3.2. Group Average Agglomerative Clustering

Group average agglomerative clustering is a hierarchical clustering (HC) algorithm [34]. There are two types of HC algorithms: namely agglomerative (bottom-up) and divisive (top-down). Hierarchical clustering algorithm generates a set of clusters which are organized in the form of a tree. Bottom-up hierarchical clustering is also known as Hierarchical Agglomerative Clustering (HAC), where each document is a cluster considered as a singleton and each cluster pair merge together until all pairs are merged to form a single cluster which contains all the documents.

2.1.3.3. Vector Space Model

Most of the well-known clustering algorithms use vector space model (VSM) to represent the document corpus [33, 34]. In [35], a document corpus is represented by a vector in the term space. In order to find out the similarity of two documents and differentiate them, the term weight algorithm is used as in [36, 37, 38, 39]. Term frequency inverse document frequency (TFIDF) is the widely used term weight algorithm [40, 41]. In this algorithm, term frequency (TF) of a term is calculated as the number of times that term is found or appears in the given document. Inverse document frequency (IDF) is calculated as the number of documents in which the term under consideration is found [42]. The TFIDF of a term can be calculated by multiplying the TF by IDF of that term [39]. There are many variants of TFIDF algorithms such as automatic text categorization (ATC) [37], LTU [36] and Okapi [38]. In ATC, maximum term frequency is used as an extra parameter, while LTU and Okapi use two extra parameters, which are the document length and average document length.

In [43], term frequency inverse corpus frequency (TFICF) algorithm is proposed. This algorithm does not require the TF of other documents. It can be used in data streams. Table 2.1 shows the formulas used in each of these methods.

Table 2.1: Different Term Weighting Schemes [43].

Name	Term Weighting Scheme
TF-IDF	$w_{ij} = \log(f_{ij}) \times \log(N / n_j)$
MI	$w_{ij} = \log \dfrac{\frac{f_{ij}}{N}}{\frac{\sum_{i=1}^{N} f_{ij}}{N} \times \frac{\sum_{j=1}^{M} f_{ij}}{N}}$
ATC	$w_{ij} = \dfrac{\left(0.5 + 0.5 \times \frac{f_{ij}}{max\_f}\right) \log\left(\frac{N}{n_j}\right)}{\sqrt{\sum_{i=1}^{N} \left[\left(0.5 + 0.5 \times \frac{f_{ij}}{max\_f}\right) \log\left(\frac{N}{n_j}\right)\right]^2}}$
Okapi	$w_{ij} = \left(\dfrac{f_{ij}}{0.5 + 1.5 \times \frac{dl}{avg\_dl} + f_{ij}}\right) \log\left(\dfrac{N - n_j + 0.5}{f_{ij} + 0.5}\right)$
LTU	$w_{ij} = \dfrac{(\log(f_{ij}) + 1.0) \log\left(\frac{N}{n_j}\right)}{0.8 + 0.2 \times \frac{dl}{avg\_dl}}$

The main limitation of these algorithms is that they require updating the values of term weighting when a new document is included. Therefore, we can say that they are not very efficient in online data streams in real time. Utilizing the strength of parallel processing can increase the performance of these algorithms. Therefore, in this thesis, we use multi-agent systems as core components of the proposed framework.

2.1.4. Recommender System for Social Networks

The applications of a recommender system are used in social networks. Many works are available in the literatures that discuss the extraction of data and user's information from network media sites. The first recommender system for the web was developed in 1992 by Tapestry [10]. The emergence of Web 2.0 allows the internet users to easily access and use social media sites, such as, Wordpress [44] for blogging, Twitter [45] for micro-blogging, Facebook [46] for social networking, YouTube [47] for video sharing, Flickr [48] for photo sharing, Digg [49] for social news reading, Delicious [50] for bookmarking [51] , furl [52] for searchable copies of webpages, CiteULike [53] for research papers, and Pinterest [54] for keep things. Internet users come to these social media sites to express their experiences in daily life, browse, watch videos, play songs etc. In the last few years, researchers have been taking interest in exploiting the valuable information found in the social media sites, and they have managed to come up with many interesting models and frameworks to utilize the huge data available on these sites. Therefore, recommender systems using social media data have emerged as a significant trend in the world of social media applications. For example, many recommender systems have been developed to recommend films, books, and music to the users [55]. In [56], the correlation model between social media users with e-commerce is discussed. This study shows the close loop model exists between e-commerce and social media sites.

2.1.4.1. Classification of Social Recommender System

Recommender systems can be classified by using three major factors, which are:

1. Source of data used by recommender system, which includes websites (traditional way), social networks or Web 2.0 (two way), and internet of things or Web 3.0 (health signals, GPS locations).
2. Targeted data of user and item.
3. Data extraction methods which include explicit method and implicit method.

2.1.4.2. Factors Affecting Social Recommender System

The two main factors which affect the quality of recommendation in social recommender systems are user profile and trust issues.

User profiling is the top factor when addressing the quality of the recommendation. This information can be gathered by mining social networks. Most of the time, users enter their profile explicitly when using social media sites. This information can also be gathered by using software agents that capture user behaviours. There are two types of methods used to get user profile information.

1. Explicit Method.
2. Implicit Method.

In explicit method, information is entered by users where they rate the items according to their liking. This method is more accurate as it was registered personally by the user. The implicit method uses system agents to capture the behaviours of the user to extract their interests. In online recommender systems, user log data of click streams and navigation patterns are used as implicit inputs. This method of inference is not always valid (erratic), but it is transparent to the user, and the user is not required to input any information. In both methods, we need sufficient information about the user profile and his/her behaviour in order to give recommendations [34].

The second main factor which affects the quality of recommendations is the trust of the user. When a user has to select between recommendations from a friend and a recommender system, a user always prefers their friend's recommendation [57]. Therefore, a recommender system should have trust feature that gives more reliability to the recommendations.

2.1.5. Recommender System using Multi-Agent based System

In this section, we discuss the fundamentals and architecture of an agent and the multi-agent systems. In the last few years, agents and multi-agent based systems have been growing very fast in the field of computer science. Multi-agent based systems are significantly beneficial in the decentralized and distributed computation environments, such as distributed data mining and information retrieval [58, 59], sensor networks [60, 61], social sciences, artificial life, computer games, simulations, and soccer robots applications. In MAS environment, one or more agents coexist and collaborate in order to achieve a predefined goal. The soccer playing robots and software agents on the Internet are examples of multi-agent system environments. In a MAS environment, all agents have to communicate and coordinate with each other. Because of their ability of parallel processing, MAS is utilized in social recommender systems [62, 63].

In [64], a framework for parallel crawlers for online social network is proposed. In this work, many crawlers operate independently in order to increase the reliability of the system; such that, if one crawler fails, it does not influence the operations of the others. In [65], a collaborative filtering technique that depends on simultaneous clustering of users and items is presented. In that work, a design of incremental and parallel versions of the co-clustering algorithm is used. This version of the co-clustering algorithm is utilized in order to build an efficient real-time collaborative filtering framework.

In [62], a multi-agent based system called "Infonorma" is proposed to recommend its users for legal normative instruments. This system uses content-based similarity analysis of web documents. In [63], a mobile application that uses multi-agent system is presented in order to provide personalized and quick recommendations to the social network users. In [66], a multi-agent recommender system is presented for e-tourism. It is based on a reputation collaborative filtering algorithm and deals with the cold start problem mostly found in tourism domain.

2.2. Related Research

2.2.1. State-of-the-art Social Recommender Systems

Many factors play important roles in the development of social recommender systems such as novelty, accuracy and stability [30]. The biggest challenge for online social recommender systems are the complete user profile and efficiency of recommendations. A sophisticated user profile can be created by applying the above mentioned techniques on user generated content. It is possible to predict opinion of users for a particular item by using information available through user generated contents (UGC). By using UGC data, an accurate user profile can be created. These profiles can reduce the problems of cold start and malicious rating.

In [67], researchers developed a program which can search the user blogs and recommend new blogs on the basis of user blogs. This program uses similarity clustering and tagging for recommendations. In [68], a new blog crawler called RetriBlog for Blogsphere was developed. It deals with the variations typically found in blogs. In [69], the trust-based social recommender systems are discussed. Trust rating is used to define the level of trust of the user. This method has three main properties of trust: transitivity, asymmetry, and personalization. It is feasible for small social network, but it is very difficult to define trust for all users of a large social network. Trusting a user or not, depends on the personal opinion. In [70], a method is proposed to improve accuracy of group recommendation using group personality composition along with trust between members of a group. Researchers define the trust that should be personalized, where users can have the ability to define different levels of trust to other users on the social network. In [71], a probabilistic matrix factorization framework is employed. This framework uses both user's social trust network and user-item matrix for recommendations. This framework outperforms state-of-the-art model of collaborative filtering and social trust-based recommendation algorithms. The experiments have been done on Epinions datasets. Their results were promising when few ratings were available in the dataset. In [72], a trust-aware recommender system (TARS) is proposed. They use trust metric weight estimation which is able to measure the trustworthiness of the trusted network. This method is capable of using large datasets. Using a larger dataset increases number of ratings and not compromise on accuracy. In [73], a trust-aware recommender system was proposed. This system is able to solve the data sparseness problem. They proposed a new routing protocol for their TARS. This protocol has a higher prediction compared to classical routing protocol of free-scale network. Trust is the personal opinion about the user. In [74], it has been found that similar interests of people lead to create trust between them. In [75], a probabilistic modelling approach is presented for Digg website [49]. This website provides services for its user to submit their favourite articles and also rate the articles on the website. The rating criteria are based on votes by the user. Digg only uses article's vote counts, and then puts that highly voted article on the front page of their website, no user preferences are considered for recommendations. If the recommendations are based on user preferences, then it is possible that the user can vote for articles without searching for their interested articles.

In [75], a personalized recommender system is presented. This model can be very useful for the cold-start and warm-start problem faced by many recommender systems. This model also recommends the low score relevant articles with equal importance as other high score articles. An algorithm is proposed named EM algorithm. This algorithm is able to learn proposed probabilistic model parameters. A generalized probabilistic latent semantic indexing is used, which is similar to the technique used in [76].

In [77], model based algorithms predict user`s preferences for unknown products or items. Precision of recommendation can be increased when using a probabilistic model.

Netflix [78] uses a probabilistic matrix factorization technique for movie recommendations.

Tags are very effective information that can be used in social recommender system. Tags are keywords which can be added to any digital object such as website, pictures, video, songs, and movies. These objects are explored by using these keywords. Tags are used in many websites such as Delicious [50], Flickr [48], Technorati [79], and CiteULike [53].

Social tagging analysis was done on the Delicious bookmarking system in [80]. The result shows improvement while searching. In [81], distribution of tags in Delicious is studied. This work suggests a generative model of collaborative tagging. The tag information is gaining more importance due to the effect of information available in tags as compared to other user information like a click stream or data logs. Tag is a small piece of information, but very powerful because it is contributed by human intelligence. There is no restriction of language, words, etc. for tagging. Tags have different meanings for different users. Almost 60% of the tags are personal tags. Tags can be used to profile the user's topic preferences accurately. This brings more challenges to the researchers for example, how to solve freestyle vocabulary of tags. In [82], a model uses standard expert ontology or item taxonomy for each user tag. This model tries to minimize the noise from the tags. Item taxonomy is the controlled vocabulary terms or topics to classify items. For example, Amazon uses book taxonomy.

People-tags based recommender system uses no explicit information from users. It is based on people’s relationships and user tags information. In [51], a people-tag based personalized recommender system for enterprise is proposed. This method is based entirely on aggregating social network data. In [83], it was shown that aggregating social network information from different sources will have accurate and better results. Suggested item for the user is based on the item-people and user-tags aggregated relationship for the target user. In [51], people and tags are used for recommendations in social aggregation system [84, 85]. First, the relationship between people, tags, and items are aggregated. They compared their results with tags-based recommender system, people-based recommender system, people-or-tag-based recommender system, people-and-tags-based recommender system, and popularity-based recommender system. The tags-based recommender system that uses incoming tags and user tags has achieved better recommendation results (success rate of 70%) as compared to others recommender systems. The compared people-based recommender system and tags-based recommender system only have 2% overlapping of recommendations. A hybrid people-or-tag-based recommender system, which includes explanations, gives slightly better results. The outcome, by combining peoples and tags relationship to user profile does not produce much larger change as compared to only tags-based recommender system. The recommendation provided by this method gives item diversity and lower recommendation of already known items to the target user.

The main and critical outcomes are to define user profiles accurately and the system should have enough information that can be used in recommendations. These social networking sites have rich information which is related to the users. This information can be extracted using latest data mining techniques. Personalised recommender systems can use tagging, blogs mining, and trust techniques to strengthen their recommendation.

2.2.2. Recommender System using Twitter Data

Recommender system using the Twitter data is discussed in [86, 87]. A comprehensive and detailed Twitter data analysis can be found in [86]. Followee-recommender system for Twitter is proposed in [87]. This system is based on the user profile, followees, and followers. The dataset of 20,000 Twitter users is used for experimental purposes. User tweets and tweets from followees and followers are used in that research. New system architecture, which is described as a Twittomender recommender system, is introduced as a web service. The user has two basic modes, namely: user search and user recommendation. The user search mode is used to handle the queries entered by the user, while in user recommendation mode user profile is used as query source. User profiling is done using tweets and social connections. Five different methods are used for profiling: user’s own tweets, user`s followee tweets, user`s followers tweets, user’s followee ids, and user`s follower ids. Lucene platform, which depend on TFIDF weighting metric is used. If content sources such as user tweets, followee tweets, follower tweets are used, then it can act as content-based recommender system. On the other hand, if indexing the users by their followee and followers then collaborative-based recommender system can be formed. The collaborative filtering method performs better than content filtering method from the perceptive of precision. Precision result ranges from 0.15 to 0.2, which means 2 out of 10 recommendations for followees is correct. In position of results for recommendation-list, content filtering performs better than collaborative filtering. That research shows that social media data such as Twitter, which is noisy data, can be applied for recommendation purposes. In [88], an approximate TFIDF is implemented using the graphic processing unit (GPU), which shows better results as compared to TFIDF using a brute force algorithm.

For reducing the total processing time of the tweets different types of approaches are used by researchers. In [89], uses a locality sensitive hashing method [90] to speedup first story detection task. In [91], which is a fairly new development uses chuck of tweets to reduce the total processing time. This method is able to reduce the total processing time by limiting the amount of tweets to be processed at a given time. There is no other work found to reduce the total processing time for social network data either by using parallelization or distributed architecture.

As a part of this thesis, the Twitter data crawler tool is presented. The proposed crawler is MAS based which downloads Twitter data and stores them in the database. In [3], we presented the Twitter simulated environment that can be used for the distributed recommender system. This Twitter environment is capable of simulating Twitter users as agent in multi-agent environment. In [5], a multi-agent simulator for TCP/IP network is presented. We use the message passing between agents that is presented in [3, 4, 5] and the Twitter simulated environment presented in [3, 4] as the major components of the proposed framework. Also, the method which is referred to as data grouping method is compared with the proposed distributed recommender system in the Chapter 5 Section 5.7.

2.3. Summary

In this chapter, we presented the key background research material for this thesis. In the first section, we introduced the task that we are dealing in more detail, by providing detailed information on filtering techniques and the methods used in recommender systems. In the second section, we explained the similarity methods used in the recommendation process. We then presented the current state-of-the-art approaches in the area of recommender systems. Furthermore, we explained different state-of-the-art social recommender system along with different Twitter recommender systems. Finally, we explain the concepts of multi-agent based system, which is one of the key techniques in this work. In the rest of this thesis, we develop algorithms for distributed recommender system that address all the key challenges, by reducing the online processing time for big social data.

Chapter 3

Modelling the proposed Distributed Recommender Systems

In this chapter, a simulation model will be presented for the distributed recommender system. The objective of such model is finding the relation between the input variable number of tweets and output variable running time of distributed recommender system. The proposed distributed recommender system has the components running in parallel, referred to as processing units. Another part of the distributed recommender system is the distributed storage of databases used by these units. In summary, these units that may include other units are categorized as follows:

- A set of *text processing units*.
- A set of *recommender units*.
- An *organizing unit*.

that is explained in the following sections.

3.1. Text Processing Unit

The text processing unit (TPU) is responsible for receiving the tweets published from the user. After capturing the tweet, a text processing unit performs text processing based on general text filtering criteria, which is used in common text processing algorithms that are designed for processing post or tweets in online social websites. The criteria include stopwords removal, retweets removal and stemming of words. Tweet have a maximum of 140 characters. These tweets contain lots of noises, which needs to be removed. The example of the noises are non-English words, stopwords and URLs. The first step is the conversion of tweet sentences into a sequence of words/terms by tokenizing each sentence. This will separate words or terms from the sentence that can be used for further processing. The following are the steps involved in the processing of the raw tweets to produce a clean set of tweets.

1. Stopword Removal: Each word is compared and checked against a standard stopword list of English terms. The stopword list from the Natural Language Tool Kit's is used to remove least informative terms. This process is used to rectify articles (such as 'an', 'a', 'the') from the sequence of words in a tweet. The remaining words are the alphanumeric words which are then used. This process also removes URLs and a word which starts with '@' (aka mention or reply) to be removed from the tweets, however, the words with '#' sign (aka hashtags) are retained.

2. Number of Words: This process counts the number of English words present in the clean tweet text. The lower limit for the number of words in one tweet is set to three words, which means a tweet should have at least 3 words. Once a tweet passes this criteria, then the clean tweet text will be taken for further processing. Once the word filtration process is complete, each tweet is split into a set of tokens or features and included in the vector space model [92]. Considering the above explanation for a common text processing algorithm, the following algorithm is used for a text processing unit.

Algorithm 1 Pseudo-Code for text processing unit

```
1.  procedure stopWordRemovel(Tweets)
2.  Declare:
3.      N = number of tweets
4.      L = number of stops words in stopWord array.
5.      cleantweetText = temp string variable to store clean tweet
6.      wordCount = number of words in cleantweetTweet
7.
8.  for i = 1to N            // remove all stop words from all tweet
                             // tweetText is tokenized by using the space between words
9.      wordtokens ← tokenize(tweetText[i])
10.     W = number of words in raw tweetText.
11.     for j = 1 to W            // for all words in tweetText
12.         word ←wordtokens[j]
13.         flag = 0
14.         for k = 1 to L    // remove all stop word from tweet
15.             if(word = stopWord[k]
16.                 flag = 1
17.         end
18.         if (flag = 0 AND wordCount >= 3)
19.             Append in cleantweetText  += word                // word and space
20.     end
21. end
```

N is the total number of tweets to analyse the complexity of the above algorithm. The worst case scenario is considered and time complexity (worst case) of Algorithm 1 for line 8-21 is calculated as:

$$= O(N * W * L)$$

Where, N = Number of tweets.

W = Number of words in each tweet.

L = Number of stopwords.

It is clear that the number of tweets (N) is the dominating factor, so the complexity of typical text processing unit is $O(N)$.

3.2. Recommender Unit

Recommender Unit (RU) is responsible for performing two types of functions. First, the name indicates itself, it provides recommendations to the requesting users upon their requests and the second, it calculates weight for words of tweets for all users connected to the recommender unit. When the clean tweet arrives at the recommender unit, it is stored in a local database along with a user name. There are in total four different algorithms which are implemented to build a common recomender unit. These algorithms can be configured and initialize according to the architecture of the distributed recommender system under test. The recommender unit is composed of the following, term frequency unit, term frequency inverse document frequency unit, cosine similarity unit, and k-mean clustering unit.

3.2.1. Term Frequency Unit

When a request is initiated from a user, recommender unit process all tweets present in the unit by using Algorithm 2. A word bag is created for each user separately. This word bag contains all the words it receives from the user. If the word is not present in user word bag, it will create an entry for that word and if the word is previously present, the word count is increased by one. Each user's word bag is then used in finding that user's interests. A general method of term frequency [92] can be used which is defined as follows:

$$tf(t,d) = \ Number of\ repeated\ terms\ in\ the\ document$$

Term frequency represents the number of terms present in the document. The following Algorithm 2 is used for calculating the term frequency for one user tweet.

Algorithm 2 Pseudo-Code for calculating term frequency for creating wordbag

```
1.  procedure wordBag(cleantweetText)
2.  Declare:
3.      Variable Tweets contains N tweets
4.      wordBag ← {word, wordCount}
5.
6.  for i = 1to N            // for all clean tweets received for one user
                             // W = number of words in cleantweetText
                             // cleantweetText is tokenized by using the space between words

7.      wordtokens ← tokenize(cleantweetText[i])
8.      for j = 1 to W               // for all words in tweetText
9.          word ←wordtokens[j]
10.         if (word not in wordBag)           // wordCount = 1
11.             Insert in wordBag (word, wordCount)
12.         else
13.             Increment wordCount of word in wordBag

14.         if (word not in wordList)
15.             Insert in wordList(word, wordCount)
16.         else
17.             Increment wordCount of word in wordList
18.     end
19. end
```

Time complexity (worst case) of Algorithm 2 is calculated as:

$$= O(N * W)$$

Where, N = Number of tweets

W = Number of words in each tweet.

Obviously, the number of tweets (N) is the dominating factor, so the complexity of term frequency can be approximated by $O(N)$.

3.2.2. Term Frequency Inverse Document Frequency Unit

Recommender unit can also include term frequency inverse document frequency weight calculation. When a request is initiated from a user, recommender unit process all tweets present in the unit by using Algorithm 3. The term weight or term importance in a tweet depends on the following three factors [93].

- Term Frequency *(tf)*.
- Inverse Document Frequency *(idf)*.
- Length Normalization *(ln)*.

The term frequency inverse document frequency algorithm is the most widely used term weight algorithm for information retrieval. It is the product of two statistics methods, term frequency and inverse document frequency. Inverse Document Frequency is the number of documents counted in a searched collection and is indexed by the term. The *idf* is also called a Global term weight [94] and represents the importance of terms in the corpus [95].

$$idf(t, D) = \log_2 \frac{N}{df}$$

Where,

N = Total number of documents searched.

df = Document frequency (number of documents with same keyword).

Inverse document frequency for all tweets has to be calculated every time, when a user tweets. Therefore, TFIDF is calculated when a new tweet received by the unit using following formula:

$$tf * idf(t, d, D) = tf(t, d) * idf(t, D)$$

There are short tweets as well as long tweets. To give importance to both types of tweets, we need to use normalization. If the tweet is long and has the same term available repeatedly, then this tweet will have more influence on term frequency weight. If there are more terms in the tweet, then again this will influence the search criteria. To reduce the effect of long tweets (higher term frequencies and more terms), we need to employ normalization on the basis of tweet length. We use cosine normalization in the vector space model [35]. We use tweet vector to calculate cosine normalization. For a tweet t_i, we formally define

$$cosNorm\,[i, 1] = TID_i$$

$$cosNorm\,[i, 2] = \sqrt{\left(w_1^i\right)^2 + \left(w_2^i\right)^2 + \cdots + \left(w_n^i\right)^2}$$

$$cosNorm\,[i, 2] = \sqrt{\sum_{j=1}^{m} \left(w_j^i\right)^2}$$

Where, $w_i = tf_i \times idf$

By using cosine normalization, we can normalize both higher term frequencies as well as more terms in the long tweet. The output is a tweet feature vector which consists of tweet ids and their TFIDF values. The following Algorithm 3 shows the pseudo code for TFIDF unit.

Algorithm 3 Pseudo-Code for calculating TFIDF values by creating wordBag and tweetVector

```
1.  procedure wordBag(cleantweetText)
2.  Declare:
3.      Tweets contains N tweets called tweet chunk
4.      wordBag ← {word, wordCount}
5.      tweetVectorTable ← {tweetID, wordList}    where, wordList ← list of {word, wordCount}

6.  for i = 1 to N                  // for all clean tweets received
                                    // W = number of words in cleantweetText
                                    // tweetText is tokenized by using the space between words

7.      wordtokens ← tokenize(tweetText[i])

8.      for j = 1 to W              // for all words in tweetText
9.          word ←wordtokens[j]
10.         if (word not in wordBag)            // wordCount = 1
11.             Insert in wordBag (word, wordCount)
12.         else
13.             Increment wordCount of word in wordBag

14.         if (word not in wordList)
15.             Insert in wordList (word, wordCount)
16.         else
17.             Increment wordCount of word in wordList
18.     end
19.     Insert in tweetVectorTable (tweetID, wordList)
20.
21.  end

22. tweetCount = Total number of tweets in tweetVectorTable

23. for i = 1 to N                  // calculate TFIDF for all tweets
24.     tweetVector_Info ← tweetVectorTable[i]
25.     for j = 1 to W              // n is number of words in wordList
26.         wordList_Info ← wordList[j]
27.         totalwordCount  ←  (wordlist_Info(i,1), wordBag)
28.         wordidf  = wordCount  * log ( N / totalwordCount )
29.         wordtfidf = wordCount * wordidf
```

```
30.                 tweettfidf += (wordtfidf ) $^2$
31.         end
32.         tfidfvalue = $\sqrt{}$ tweettfidf
33.         tweettfidfVector[i] $\leftarrow$ {get_TweetID(tweetVector_info), tfidfivalue}
34.  end
```

Time Complexity (worst case) for Algorithm 3 is

$$= O(N^2) + O(W)$$

3.2.3. Cosine Similarity Unit

Cosine similarity is a similarity measure of two vectors. It is the measure of cosine angle between two vectors not their magnitude. If two vectors are exactly the same it means cosine similarity is 1, because both vectors have no difference in their angle (i.e. a direction which is zero, cos 0 = 1). The calculated similarity is bounded between [0, 1]. By using following Euclidean dot product formula, the cosine of two vectors can be calculated as follows:

$$A \cdot B = \|A\|\|B\| \cos\theta$$

$$cosSim(A, B) = \frac{\vec{V}(A) \cdot \vec{V}(B)}{|\vec{V}(A)||\vec{V}(B)|}$$

$$= \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

Where,

A, B = Two users' word vectors.

n = Number of words in the vector.

The following Algorithm 4 is used to calculate the cosine similarity of requesting user against each user present in the unit.

Algorithm 4 Pseudo-Code for finding user interest using cosine similarity

```
1.  procedure cosSimilarity(userReq, listofuser)
2.  Declare:
3.      cosSim ← {username, cosValue}
4.      U = list of users
5.      userReq = requesting user
                                    // rUV = requesting user word vector
                                    // UV = temp word vector, use for temp user words
                                    // generate vector for requesting user (rUV)
6.      for i = 1 to U              // all users in listofuser
                                    // generate vector for temp user.
7.          for j = 1 to W          // all words in requesting user vector
8.              value += rUV[j] * UV[i][j]
9.              mag1 += $(rUV[j])^2$
10.             mag2 += $(UV[i][j])^2$
11.         end
12.         mag1 = sqrt(mag1)
13.         mag2 = sqrt(mag2)
14.         if  both mag1 and mag2 is not zero
15.             cosValue = value / (mag1 * mag2)
16.         else
17.             cosValue = 0.0

18.         // save cosine similarity value with username in cosSIM
19.         Insert in cosSim(username, cosValue)
20.     end
```

Time Complexity (worst case) Algorithm 3 can be calculated as:

$$= O(rUV(rUV + UV))$$

Where,

rUV = Vector size of requesting user.

UV = Vector size of all user in the server.

The complexity of cosine similarity in the worst case is that if the number of users (U) is equal to the number of tweets (N) then it can be calculated by using $O(N^2)$.

3.2.4. K-mean Clustering Unit

The K-mean clustering unit is used for finding recommendations. Clustering is performed on all the tweets vectors present in the recommender unit's local database. By using k-mean clustering algorithm, two clusters (*k=2*) are created as *C1* and *C2*. Each TFIDF value of a tweet is checked against the mean value of both tweet clusters. The tweetID of TFIDF value is assigned to the cluster whose cluster mean distance is less than the other cluster mean. The method is divided into two steps, which are, assignment, and update steps. In the first step, random mean values in two clusters are defined and used as the centriod for calculation by using the following formula:

$$C_i^{(t)} = \{x_p : || \, x_p - m_i^{(t)} \, ||^2 \leq || \, x_p - m_j^{(t)} \, ||^2 \, \forall_j \, , 1 \leq j \leq k\}$$

Where x_p is assigned in one or more than one $C^{(t)}$ clusters.

In the next step, which is the update step, mean is again calculated and set as the new centriod, by using the following formula:

$$= m_i^{(t+1)} = \frac{1}{\left|C_i^{(t)}\right|} \sum_{x_j \in C_i^{(t)}} x_j$$

The time complexity of k-mean (Lloyd's algorithm) is given by $O(nkdi)$, where *n* is the number of *d*-dimensional vectors, *k* is the number of clusters and *i* is the number of iterations needed. In this thesis, the number of dimensions is already defined for *k* and *d*. Where *k* = 2 (clusters) and *d* = 1 (one *TFIDF* value for each tweet). Therefore the time complexity can be solved exactly in $O(n^{dk+1} \log n)$ time, where *n* is the number of tweetIDs to be assigned. The following Algorithm 5 is used by the recommender unit to perform clustering when the request from user is received. K-mean clustering algorithm is a stochastic in nature, therefore, each clustering result varies depending on initialization of centriod (which is pseudo random).

Algorithm 5 Pseudo-Code for Distributed Algorithm (Finding similar user interest using k-mean)

```
1.  procedure clusters(tweettfidfVector)
2.  Declare:
3.        tweettfidfVector ← {tweetID, tfidfvalue}
4.        N = Total number of tweet vectors in tweettfidfVector
5.        k = 2,                      // k is the number of clusters to create similar interest user list
6.        C1 = {} and C2 = {}         // C1 and C2 are two lists that represent cluster 1 and cluster 2.
                  // calculate random r1 and r2 that represent the centre of two clusters K1, K2.
                  // r1 and r2 are in the range of 0 to maximum value in tweettfidfVector(tfidfvalue)

7.  maxtfidf ← MAX(tweettfidfVector(tfidfvalue))
8.  r1 = RANDOM(0: maxtfidf )
9.  r2 = RANDOM(0: maxtfidf)

10. while ( mean(C1) & mean(C2) )
                                      // calculate distance between all tweettfidf and r1 and r2 for each cluster
11.       for i = 1 to N              // all vectors in tweettfidfVector
                                      // Calculate distance $d_1$ b/w each tfidf(i) and cluster mean $r_1$.
                                      // Calculate distance b/w each tfidf(i) and cluster mean $r_2$.
12.               d1 = Distance ( tweetVector (i,2), r1 )
13.               d2 = Distance ( tweetVector (i,2), r2 )
14.               if $d_1$ >= $d_2$
15.                       ADD in cluster C1 ( tweettfidfVector(i,:) )
16.               else
17.                       ADD in cluster C2 ( tweettfidfVector(i,:) )

18.               Recalculate the new cluster's centre $r_1$ and $r_2$ by using formula.
19.                       $r_m = (1/l_m)\sum_{j=1}^{l_m} K_m(j:2)$   where $l_m$ is the number of tweets in mth cluster
20.               if (mean value of cluster is unchanged then exists while loop)
21.       end
22. end
23. result = {C1, C2}
24. // sort the interest list using merge sort
```

Time Complexity (worst case) for Algorithm 5 is

$$= O(N^2 \log N) + O(W)$$

3.3. Organizing Unit

Third unit is Organizing Unit (OU). This unit is responsible to receive request generated by user and sent the request to all recommender units. This unit does not have any data processing overheads.

Time complexity is calculated to show the relation between running time and number of tweets for each unit. By increasing the number of tweets and showing the time complexity of above discussed units the relation between time complexity and number of tweets is analysed. The effects of number of users on time complexity is not included and assumption is that the users are distributed evenly based on total number of their tweets on each server. From Figure 3.1 to Figure 3.5 shows the time complexity of five algorithms that are used to show the increase in the number of tweets in the above discussed units and their performance improvement when running them in parallel by multiple servers. It can be seen all the algorithms that have higher time complexities benefits more from having more processing units (i.e. running them in a distributed environment). However, there is a limit for that which is determined by additional time of message passing which is described in Section 3.6 and is measured in the experiments of Chapter 5, Section 5.4.3 and Section 5.5.3. For example, the recommender unit benefits a lot from the multi-processing of its units because all the consisting four units have the algorithms with running times approximated as $O(N^2)$ and in Figure 3.2 to Figure 3.4 also shows that when using them with two servers their running times are significantly reduced (around 75%). Text processing unit also benefits from multiple processing, for example 40% reduction in running time with two servers and so forth.

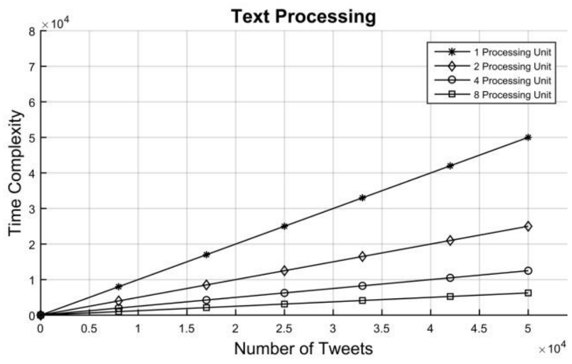

Figure 3.1: Shows the time complexity of the text processing algorithm.

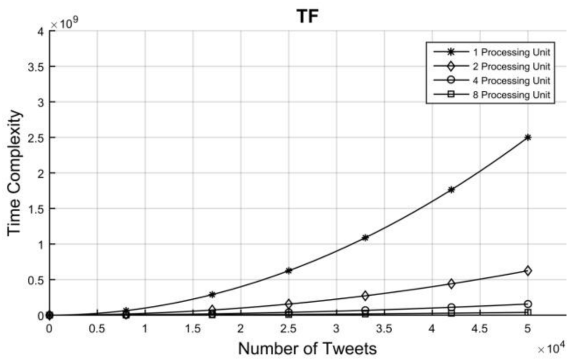

Figure 3.2: Shows the time complexity of the term frequency algorithm.

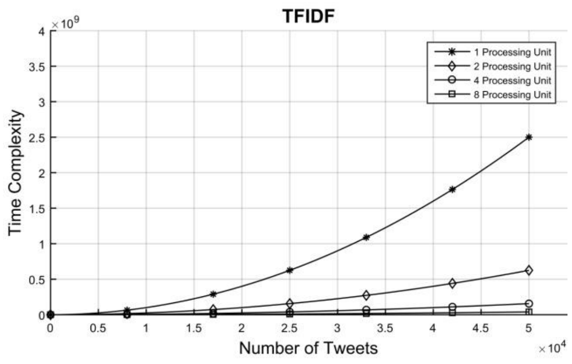

Figure 3.3: Shows the time complexity of the TFIDF algorithm.

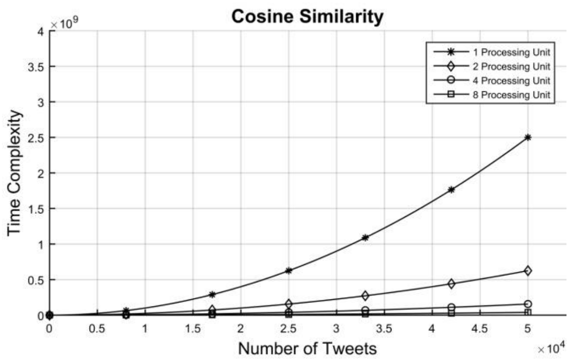

Figure 3.4: Shows the time complexity of the cosine similarity algorithm.

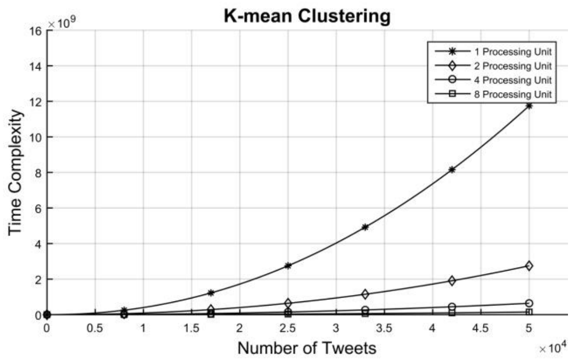

Figure 3.5: Shows the time complexity of the K-mean clustering algorithm.

The important observation is for the units with the algorithms of *O(N^2)* the running time has non-linear relation to the increase in the number of the tweets (i.e. Figure 3.2 to Figure 3.5) which means they are non-scalable. Whereas, when using the same units with two servers this relation became linear relation for upto 50,000 tweets. Based on this observation, we define scalability factor as the following: For the units of recommender systems with *O(N^2)* running time complexity, using two servers (i.e. running two units in parallel) make them scalable for upto 50,000 tweets. Of course it is just a theoretical result with no consideration of length of tweets, effects of message passing, disk storage latency, and other delays of organizing all components of recommender system.

In the next section, we provide a computational model to be used in different scenarios, but before that we need to discuss distributed storage of the database and data distribution.

3.4. Distributed Storage and Data Distribution

So far, the theoretical improvements of the parallel running of the units which are in proposed recommender distributed systems have been discussed. Another feature of the distributed recommender system proposed in this thesis is the use of a distributed database for the storage of tweets that are processed by parallel units. For example, each recommender unit uses its local database for providing recommendations; therefore, the storage of user tweets is distributed across distributed databases. In the distributed database (disk storage), the performance of the distributed recommender system depends on disk latency.

Let m be the number of recommender units used in the distributed system, U be the number of users and N be the tweets of the user which are given as input to the system.

The tweets that need to be distributed belong to different users, so we divide the tweets based on users. This way, all tweets from each user will be stored in one recommender unit's local database. To do this, users are assigned to a specific recommender unit and local database. Moreover, data distribution (or load balancing) of tweets makes it possible for the distributed local databases have approximately equal number of tweets, this process of data distribution is done by initializer agent. Data distribution in the distributed recommender system is done by dividing the total number of processed tweets by *m*. Thus, we should get a reduction in disk seek time in the amount of total seek time in central version divided by *m*. This calculation will be done in all three phases of the data process.

In Phase 1, each user U_i from the list of user U is assigned to one text processing unit (p_i). Each text processing unit processes all tweets of that user. In Phase 2, the text processing unit is assigned to a specific recommender unit, such that each text processing unit stores the clean tweets in a local database connected to a recommender unit. If we assume the system knows how many tweets are stored in each server at a time, then it can move all the tweets of one user if it exceeds the calculated capacity of the local database of that server. Since the system depends on some parameters (e.g., the geographical location of each user) at the beginning each user can be assigned to specific server. The idea is to divide tweets available in the local databases equally to all recommender units, but sometimes it may not be possible to have equal tweets on each server because we want to process all tweets of a user by one recommender unit. Also, the tweets of one user should not be fragmented in different servers. It is possible some users have more tweets than the others. In this case, some recommender unit may get more than the specific amount of tweets, so we may add more recommender units. The main benefit that we expect to get from data distribution is to reduce the total disk seek time for processing the tweets in the local databases instead of the central database. Although, the load balancing (i.e. data distribution) would not be 100%, as the number of tweets from each user may differ from the other users. We expect to reduce the total seek times in half for the distributed system with two servers compared to the central system.

3.5. Distributed Recommender Systems

In this section, we will explain two simple configurations that can be easily made by using this framework. These configurations are as follows:

1. By using text processing, term frequency units (i.e. algorithms) referred to as similarity-based distributed recommender system (SDRS).
2. By using text processing, term frequency inverse document frequency units referred to as clustering-based distributed recommender system (CDRS).

For simplicity in both models, the number of units is assumed to be one instance (i.e. unit) in each server. Thus, *m* refers to the processing units or servers in both the following system architectures:

3.5.1. Similarity-based Distributed Recommender System (SDRS)

In first the distributed recommender system configuration, three different units (i.e. algorithms) are used. The following Figure 3.6 shows the distributed recommender system architecture with the initial number of the above units on two servers. These initial values are used to solve the problem with a theoretical approach in the following section. For each unit, common and existing algorithms of text processing, term frequency calculation, and cosine similarity are used.

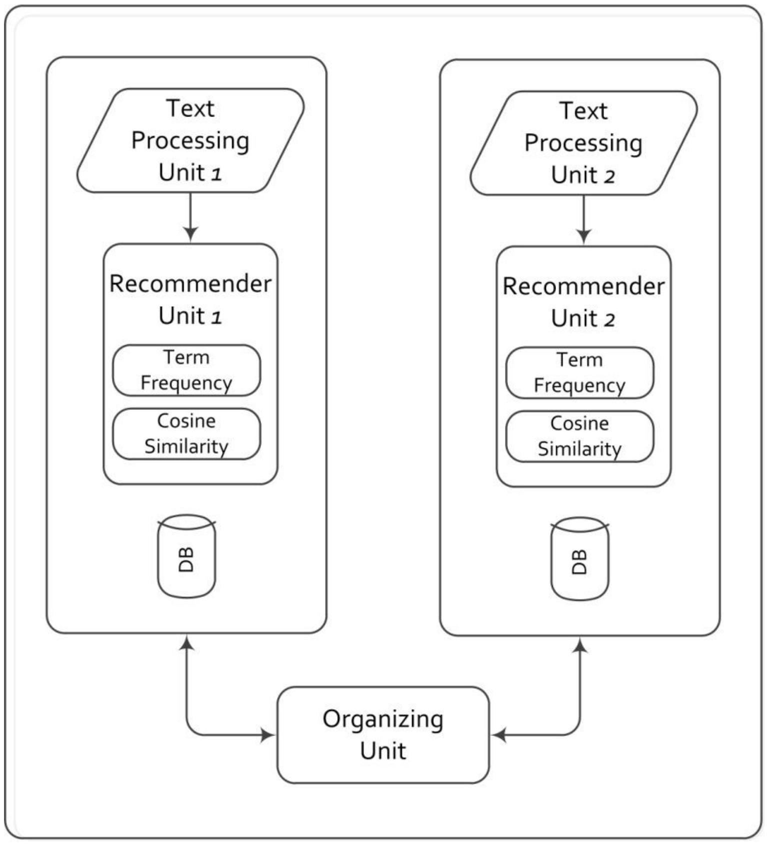

Figure 3.6: Distributed recommender system using TF and cosine similarity.

The text processing unit receives the raw tweets and gives output in clean tweet text to the recommender unit. There are two functions of the recommender unit. First, this unit is responsible for receiving cleantweets and storing them locally in the database. Second, when user request arrives, it calculates term frequency by using Algorithm 2, and find, the similar user by using a cosine similarity algorithm (Algorithm 3) as explained previously.

3.5.1.1. Examining the Effect of Parallel Processing on SDRS

The time complexity of the distributed recommender system can be calculated by the number of text processing units (p) connected to the recommender unit (m). The time complexity of processing text is the time required by each text processing unit to complete the processing of all user tweets. From the previous section, the time complexity of the text processing unit is calculated as $O(N)$. To determine the time complexity of the text processing unit when having p processing units can be calculated by the following formula:

$$= O\left(\frac{N}{p}\right)$$

Where,

N = Number of tweets

p = Number of text processing units

By using the above formula, the effects of having multiple text processing units are examined and shown in the Figure 3.1. We can observe from the graph, when the number of tweets increased, the time increased exponentially when using central processing. Using two processing units reduces the time significantly. The following formula shows the time complexity by using multiple recommender units (m).

$$= O\left(\left(\frac{N}{m}\right)^2\right)$$

Where,

N = Number of tweets received by recommender unit.

m = Number of recommender units.

We can observe from the graph when the number of server increases, the time decreases approximately in half. The time complexity of the recommender unit for cosine similarity (Algorithm 3) is calculated. The following graph shows the time complexity using multiple units (*m*). The complexity of cosine similarity in the worst case is that if the number of users (U) is equal to the number of tweets (*N*) then the time complexity can be calculated by $O(N^2)$. Therefore, the time complexity of cosine similarity for one unit can be defined by

$$= O\left(\left(\frac{N}{m}\right)^2\right)$$

Where,

N = Number of tweets.

m = Number of recommender unit.

The total time complexity of the distributed recommender system can be calculated by using the following formula with the initial number of one text processing for each recommender unit.

$$= O(Text\ Processing + Term\ Frequency + Cosine\ Similarity)$$

$$= O(N + N^2 + N^2)$$

$$= O(N^2)$$

For *m* recommender units, the total time complexity can be expressed as

$$= O\left(\frac{\mathrm{N}}{\mathrm{m}} + 2\left(\frac{N}{m}\right)^2\right)$$

Where,

N = number of tweets.

m = number of recommender units.

Following is the graph for total running time complexity reduction for SDRS as the number of processing units (i.e. servers) increased.

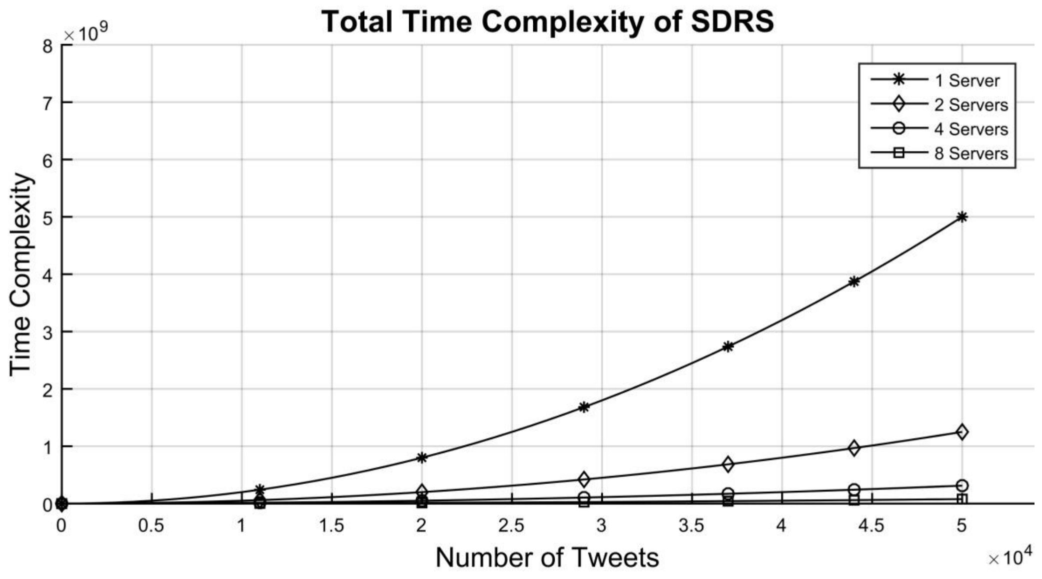

Figure 3.7: Total time complexity of the similarity-based distributed recommender system (SDRS).

The graph also shows that in theory, parallelization of recommender units contributes more by reducing the total running time. Figure 3.7, shows the increase in time complexity of central variation has a power law relation with the increase of the number of tweets. Figure 3.7 indicates that the time complexity of SDRS is non-linear (i.e. non-scalable) when using one server and it is linear (i.e. scalable) when using at least two servers for upto 50,000 tweets. Figure 3.7 also shows that for the small number of tweets (i.e. around 50,000 tweets) using two servers reduces running time almost 75% for SDRS configuration. For calculating total benefits of such parallelization, we need to know the cost which means the time of message passing (calculated in the Chapter 3 Section 3.6) in SDRS.

3.5.2. Clustering-based Distributed Recommender System (CDRS)

In the second distributed recommender configuration, we use text processing, TFIDF and k-mean clustering units (i.e. algorithms). The Figure 3.8 shows the distributed recommender system architecture with the initial number of the above units on two servers. For each unit, common and existing algorithms of text processing, TFIDF and k-mean clustering are used. In this configuration, text processing is similar to the previous configuration as explained before.

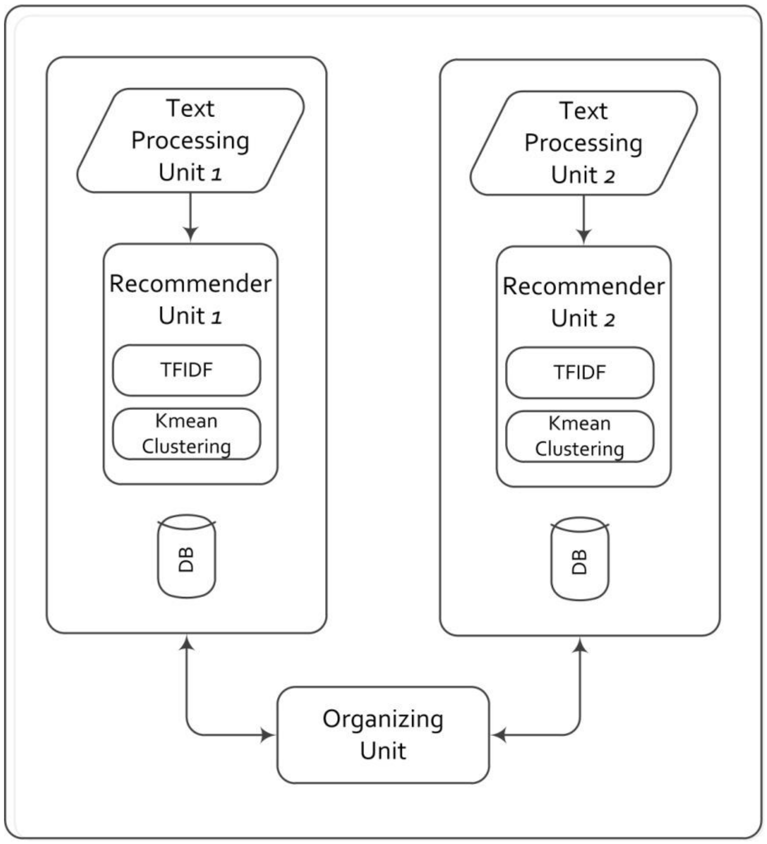

Figure 3.8: Distributed recommender system using TFIDF and k-mean clustering.

Similar to the previous configuration, recommender unit uses two functions, namely TFIDF and k-mean clustering. When the user request arrived, the recommender unit calculates TFIDF by using Algorithm 3 for all users present in the local database, and then to find similar users by using k-mean clustering (Algorithm 5).

3.5.2.1. Examining the effect of parallel processing on CDRS

The time complexity of the text processing unit is $O\left(\frac{N}{p}\right)$ as explained in first configuration. Where, N is the number of tweets and p is the number of text processing units. The time complexity of the recommender unit of TFIDF function is calculated as

$$= O\left(\left(\frac{N}{m}\right)^2\right)$$

Where,

N = Number of tweets received by recommender unit.

m = Number of recommender unit.

The time complexity of k-mean algorithm for one server can be defined by

$$= O\left(\left(\frac{N}{m}\right)^2 \log\left(\frac{N}{m}\right)\right)$$

Where,

N = Number of tweets vector to cluster.

m = Number of Recommender unit.

The total time complexity of the distributed recommender system can be calculated by using the following formula:

$$= O(Text\ Processing + TFIDF + k\ mean)$$

$$= O\left(\frac{\mathrm{N}}{\mathrm{p}} + \left(\frac{\mathrm{N}}{\mathrm{m}}\right)^2 + \left(\frac{\mathrm{N}}{\mathrm{m}}\right)^2 \log\left(\frac{\mathrm{N}}{\mathrm{m}}\right)\right)$$

Where,

N = number of tweets.

p = number of text processing units.

m = number of recommender units.

Figure 3.9 shows the power law relation between increase in time complexity and number of tweets for the central variation of CDRS configuration. Figure 3.9 shows that by using two servers the relation between time complexity of CDRS and the number of tweets become linear and scalable. Figure 5.39 also shows that for a small number of tweets (i.e. around 50,000 tweets) for making the system scalable the total time complexity reduction by using two servers is approximately 77% for CDRS which is more than the reduction gained by two servers in SDRS. Thus, we require more reduction in time complexity of CDRS in comparison to SDRS to make the system scalable. However, using scalability factor of two servers still it is working for CDRS.

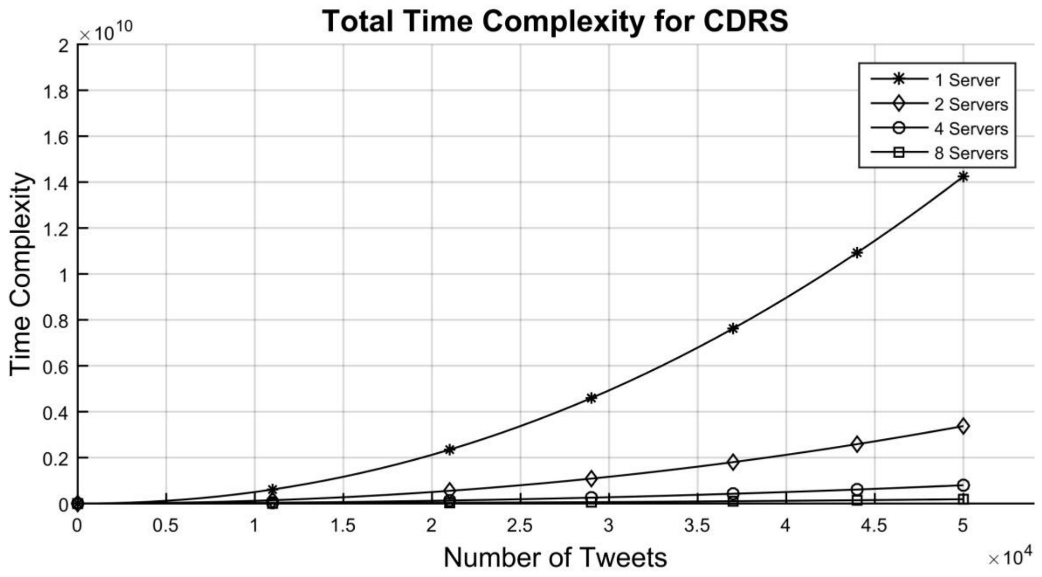

Figure 3.9: Total Time Complexity of the clustering-based distributed recommender system (CDRS).

In Chapter 5, we examine these results with simulation of the system prototype implemented by using multi-agent system.

3.6. Message Passing in Distributed Recommender System

To calculate a general message passing in such system, this can be formally represented as follows. Suppose we want to find, what is the worst case scenario for message passing, which can be calculated by assuming that after each tweet, a user requests for recommendations. It means that the system should transfer N messages to $(m-1)$ processing units on each server, considering m is the number of servers present in the distributed environment. So the cost is the number of message passing when a user asks a question after each tweet (worst case) is

$$= (m-1) * \sum_{i=1}^{N} i$$

$$= (m-1) * \frac{N(N+1)}{2}$$

It means that the time of message passing is proportional to N in the power of 2 times m. The graph of the message passing is shown in the following figure:

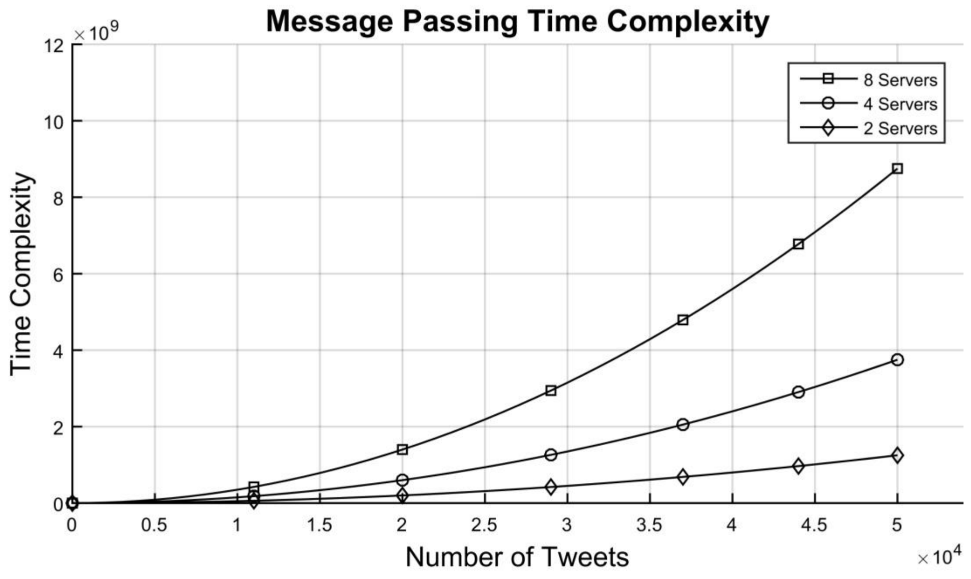

Figure 3.10: Shows the time complexity of message passing.

It can be observed from the Figure 3.10 that the total cost of the system will increase by adding the number of servers (*m*). Please note that message passing time has the opposite trend compare to total running time, Figure 3.10 shows adding more servers increases message passing time by adding the number of tweets. It is clear that if the message passing time increased and becomes more than achieved reduction in running time, then having many units running in parallel is not useful.

3.7. Summary

In this chapter, the time complexities of algorithms of a typical recommender system are analysed. Then, the relation between running time and number of tweets are shown by time complexity. Arguably, the most important challenge that modern recommender systems face is the high volume of online data that they are dealing with. In this chapter, we found that using at least two servers (i.e. the scalability factor of two) will scale the common recommender systems by improving the time complexity of typical units involved in them. The core of our approach is to reduce the running time of the algorithms by running them in parallel. We further showed that how the reduced processing time can be used to find scalability factor for each recommender system. We found that by using at least two servers we can make two variations of common recommender systems to be scalable for upto 50,000 tweets. By running recommender units in parallel, we expect that the running time of the proposed distributed systems to be reduced by the 75% when using at least two servers in comparison to one server. In the next chapters, we examine the theoretical result that was found here with multi-agent based simulation of discussed recommender systems.

Chapter 4

Multi-Agent based Framework, Specification and Implementation

Agent based development is one of the recent approaches in software engineering for development of complex applications. Agent based development is similar to object based development, but it is more suitable for distributed and complex applications. The environment for which the proposed distributed recommender system is developed is a complex environment. Agents are also very useful tools to simulate complex environments such as social networking. In this thesis, a group of the agents is developed as a prototype of the proposed distributed system that can be turned into a real system in the future and some other agents are developed to simulate a user (called user agent) which communicates with other agents on behalf of a Twitter user. Therefore, the whole developed agents system is referred to as multi-agent based framework in this thesis. For now, the user agent is very simple and just relays user's tweets. In the future, it can be very complicated by analysing the user's interests and providing suggestions for its users based on the recommendations it receives from recommender agents. The user agents communicate with the distributed recommender system and can be located in different geographical locations.

This chapter presents the design issues of a multi-agent based framework that can process the huge amount of social data of Web 2.0 sites in an online manner. This framework consists of multi-agent systems, making three different applications (they are referred to as modules in the rest of this chapter) which are: Data crawler, Twitter simulator, and distributed recommender system. All of these applications have been developed to be able to test proposed distributed recommender system. The main focus of this chapter is to build multi-agent recommender system that consists of developing agents to do the tasks of the algorithms explained in Chapter 3. These tasks are, text processing, tweet analysis using Term Frequency, providing recommendations by using cosine similarity algorithms and etc.

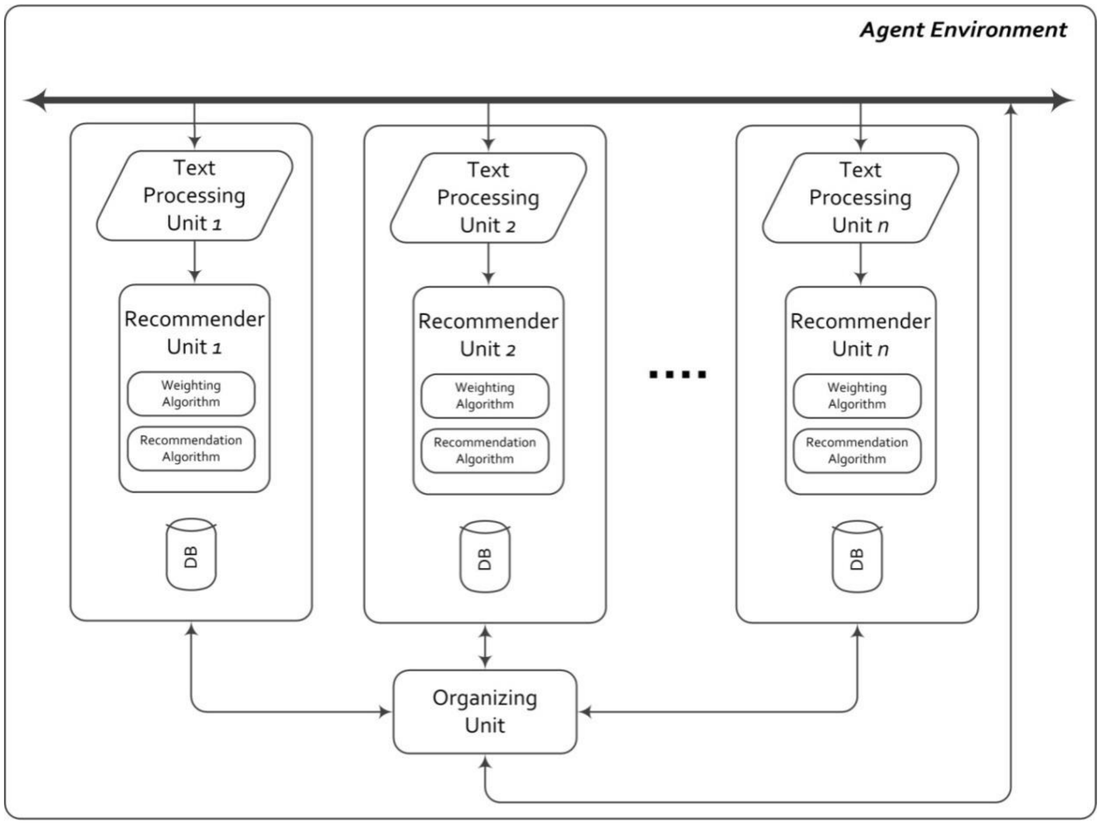

Figure 4.1: Shows MAS based Distributed Recommender System.

4.1. Structure of the Proposed Multi-Agent based Framework

The proposed multi-agent based framework consists of the following modules, as shown in Figure 4.2.

1. Data Crawler Module.
2. Twitter Simulation Module.
3. Distributed Recommender System Module.

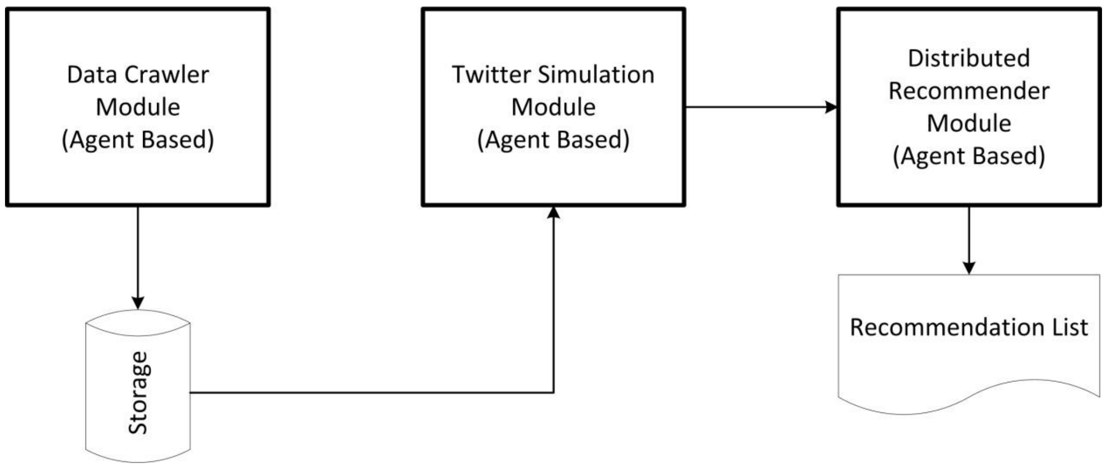
Data Crawler Module (Agent Based)
Twitter Simulation Module (Agent Based)
Distributed Recommender Module (Agent Based)
Storage
Recommendation List

Figure 4.2: Multi-Agent based proposed framework.

1. Data Crawler Module: This module is based on a multi-agent system which is used to extract tweets from the Twitter web site and store them in "usertweet" database, more details about how this module work is presented in Section 4.1.1.

2. Twitter Simulation Module: This module is a multi-agent system that utilizes tweet text, date time stamp, tweet ID, and user ID, which are stored in "usertweet"

database to simulate the Twitter environment. Later, this environment is used to test our proposed distributed recommender system. The description of this module is in Section 4.1.2.

3. Distributed Recommender System Module: Distributed recommender system module, is also based on a multi-agent system. This is a core module of the proposed framework. The task of this module is to recommend a user with similar users based on their interest. The detail of this module is presented in Section 4.1.3.

4.1.1. Data Crawler Module

Data crawler module reads the input from a text file called “userdatadownload”. This file contains a set of Twitter users IDs along with a number of friends’ and followers’ tweets to be downloaded. Data crawler uses these inputs in order to extract tweets along with their attributes from the Twitter website. The outputs of this module are the tweets of the user, user’s friends, and user’s followers. The list of Twitter user IDs is used to create data crawler agent. Each agent is responsible for downloading tweets of its Twitter user from the Twitter website. These agents work in distributed environment, so it is possible to download tweets in parallel. The downloaded tweets are then stored in “usertweet” database. In the crawler module, the following processes are performed:

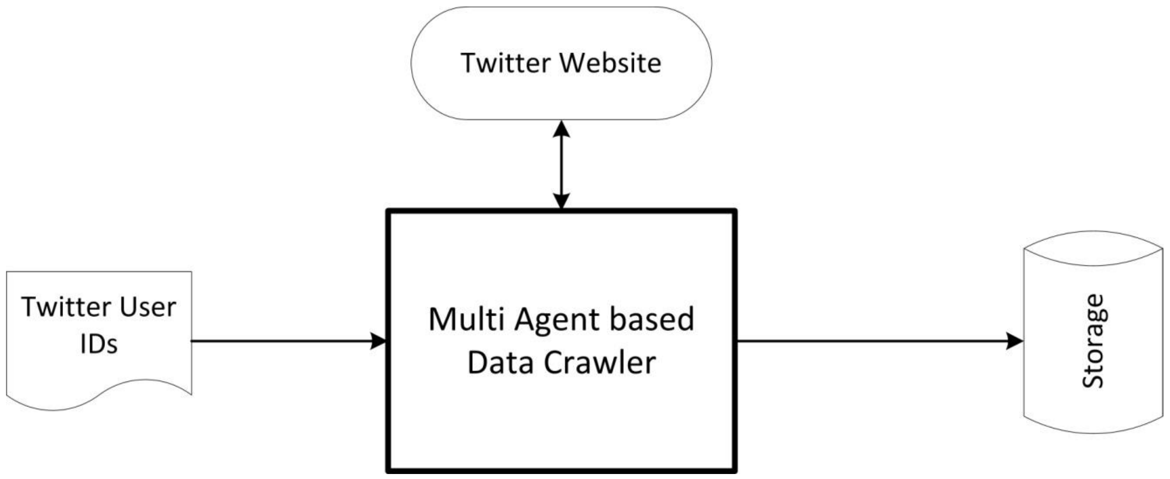

Figure 4.3: Multi-Agent based Twitter Data Crawler.

1. Authentication: In order to get the user data from the Twitter website, first we need to create the Twitter application development account. This account is used to get authentication in the form of bearer token from the Twitter website. To authenticate the data crawler agents, a bearer token agent is created. This bearer token agent is responsible for getting the bearer token and distribute it to all data crawler agents.

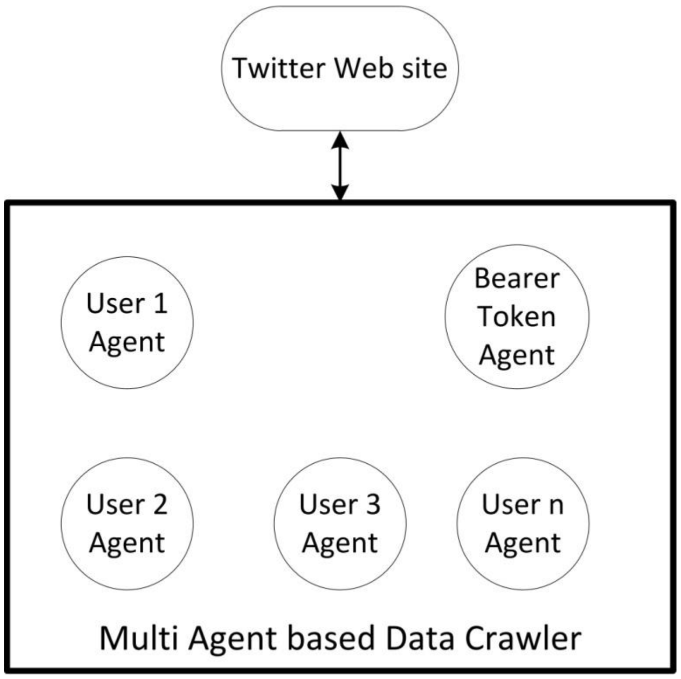

Figure 4.4: Twitter Data Crawler Module showing user agents and bearer token agent.

2. The token is passed to a set of data crawler agents in a multi-agent environment. A data crawler agent is created for each Twitter user ID which is defined in “userdatadownload” text file. For example, if we have ten Twitter user IDs in “userdatadownload” file, the data crawler tool will create ten data crawler agents. The data crawler agent uses a bearer token to download tweets of these users and stored them in the “usertweet” database.

4.1.2. Twitter Environment Simulator Module

This module is used to simulate Twitter users in a multi-agent based environment. The input of this module is a text file called “simulateusers”. This text file has a set of Twitter user IDs. This list of users IDs are used to create Twitter simulator user agents. Each

Twitter simulator user agent reads their tweets from the “usertweet” database. In the Twitter simulator environment, the following processes are performed:

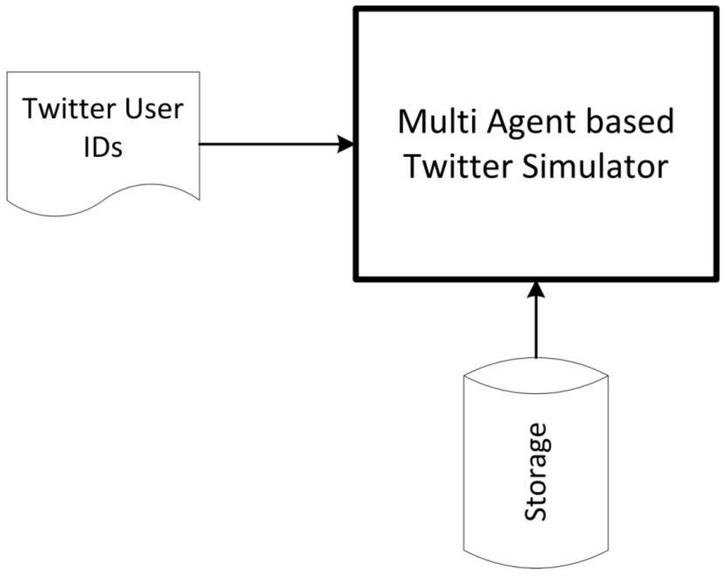

Figure 4.5: Multi-Agent based Twitter Environment Simulator Module.

1. Read the tweets from the database depending on the following parameters: start date, start time, end date and end time.

2. Each tweet is published sequentially based on the date time stamp. Also the tweets are timed as per their tweet date time stamp received from the Twitter website.

4.1.3. Distributed Recommender System Module

The distributed recommender system is the module responsible for generating recommendations for Twitter user about the other users with similar interests. The input to this module is a set of Twitter users’ tweets information published by the Twitter simulator module. The output is a set of the recommendation list for the user. Note that, in real Twitter environment tweets will be captured directly from the Twitter website. Each element of this set of tweets includes the following attributes:

- User ID: shows the identification of the Twitter user who has published the tweet.
- Tweet ID: is the number that is the unique identifier for the tweet.
- Date/Time Stamp: shows the date and time that a tweet has been published.
- Tweet text: includes the text content of the tweet. It also includes URLs, retweets, hashtags etc.

The proposed distributed recommender system architecture in this thesis is based on MAS, where each agent is used to simulate node in the distributed environment. The node in the distributed environment can be a Twitter user agent, recommender agent and organizing agent. To evaluate this platform we use the Twitter website for real text data from actual users. In a real environment, server machines have a lot of resources as they are designed to handle many incoming client requests. The request from the light weight Twitter user agent is redirected to the recommender agent by the organizing agent for the recommendations. This organizing agent sends the user agent request to all available recommender agents in the network environment. In return, recommender agent replies to it with the response message. This message contains the list of similar users, which is sent to the requesting user agent. There are two types of agents designed in this simulation model are as follows.

1. Administrative Agents;
2. Runtime Agents.

4.1.3.1. Administrative Agents

This type of agent can communicate with all under laying agents, but only be controlled by the system administrator (a person). The administrator provides test scenarios for the agents in a simulation environment and logs all the information processed by other agents. This agent is called initializer agent. Initialzer Agent is an administrative agent, which is responsible for deploying and initialling the complete test scenario. The runtime environment consists of user agents, recommender agents, and the organizing agent. The user of the simulator can only interact with the initializer agent. The initializer agent uses property file, and reads all the start-up and configuration information. This includes connection information related to the user agents to all agents in the environment. Each agent is initialized with a separate handler. For instance, the recommender agent will be created and initialized by the recommender agent handler.

4.1.3.2. Runtime Agents

These agents participate in the simulation environment by processing data from different agents. They implement the behaviour and functionality of each text based algorithm in the distributed environment. They are able to communicate with other agents; however, there is a provision that the administrative agent could also directly communicate with the runtime agents. To evaluate the simulation tool, three types of runtime agents are designed, which are as follows:

1. Text processing agent.
2. Recommender agent.
3. Organizing agent.

The complete configuration of the distributed simulation network is defined in a property file that is uploaded during the system start-up process. This start-up process is initiated by the initialize agent. Once a property file is uploaded, the complete runtime environment is created in the multi-agent based environment. Runtime agents' behaviors are based on the text processing and message passing algorithms. These algorithms are initialized by an administrative agent. After the initialization process is complete, agents can use these algorithms' behavior as they are needed.

4.1.3.2.1. Text Processing Agent

The text processing agents are responsible for receiving the tweets published from the user or tweeting agent in the Twitter simulator module. After capturing the tweet, the text processing agent performs text processing based on the criteria which are detailed in Chapter 3 Section 3.1. Once the tweet is cleaned and passes selection criteria of the tweets, it will be sent for further processing.

4.1.3.2.2. Recommender Agent

The recommender agent is responsible for performing two types of functions. These functions are implemented as behaviours of the agent. The first behaviour of this agent is the receiving of the cleantweets and storing them in a local database. The second behaviour is the generation of a recommendations list for requesting users. This behavior is used when a request is initiated by a user agent for finding the similar users. This behaviour has four different functions that can be selected during initialization.

1. Term frequency weighting function.
2. Term frequency inverse document frequency weighting function.
3. Cosine Similarity recommendation function.
4. K-mean Clustering recommendation function.

These functions are implemented as a separate behaviours and the user has the option to select any combination of these functions. In this thesis, we selected term frequency with cosine similarity as the first configuration, and TFIDF with k-mean clustering for second configuration. These configurations are used in the experiments, detailed in the next chapter.

4.1.3.2.3. Organizing Agent

The organizing agent is a runtime agent and is also created by the start-up process. The responsibility of this agent is to receive request generated by user agents and send the request to all recommender agents. This process is called the recommendation process, which is explained in the next section.

4.2. Message Passing in Distributed Recommender System

In order to generate recommendation list of similar user the following processes are performed:

1. User agent sent a request for recommendation of similar user to organizing agent.

2. The organizing agent uses the sender ID information to find out which user has requested for the recommendation. The user name (sender ID) is then broadcast to all recommender agents. The goal of this broadcast is to request all recommender agents for the recommendation list.

3. Each recommender agent checks whether the requesting user is present in its local user database. Only one recommender agent will have a requesting user connected to it. The first step is if recommender agent found requesting user in its local database, then recommender agent will obtain the requesting user tweets vector (*tw*), which is then sent to other recommender agents. The second step is to find the interest list for the requesting user from its own server by using cosine similarity or k-mean clustering algorithm (depending on the initial configuration). The interest list of users is generated, this list has two columns: first column is for user name and second column includes the interest percentage. This list is then sent to the organizing agent.

4. The recommender agent who does not have the requesting user in its local database will not respond to the broadcast of the organizing agent as in step 2. It will only respond to the message which is sent from recommender agent who has requested user connected. This message contains tweet vectors of the requesting user. When other recommender agent receives the tweet vectors, they will store them temporarily in the recommender agent (servers) memory for one time processing. The received tweets will not be stored permanently on recommender agent (server) database. Once all tweet vectors of the requesting user are received, the similarity or clustering algorithm (explained in previous Chapter 3) are applied and recommendation list is generated.

5. Next, recommender agent will prepare the recommendation list with the first column for the user name and interest percentage in the second column and send the recommendation list to the organizing agent.

6. The organizing agent receives the interest list from all recommender agents. The lists from all RAs are combined and send to the requesting user agent.

4.3. Agent Development Environment

The multi-agent based distributed recommender system is implemented by using Java Agent Development framework (JADE). It is a software framework written in Java language and is use for development of intelligent agent. This framework was developed by Telecom Italia and can be downloaded freely under GNU Lesser General Public License (LGPL). The JADE frame uses the Foundation for Intelligent Physical Agent –

Asynchronous Communication Language (FIPA-ACL). FIPA-ACL is used for coordination and communication between the agents [96]. JADE uses a container to host agents in distributed agent platform. The platform provide different tools for debugging, agent mobility and execution of different agents in parallel [58].

4.4. Verification and Validation

The verification and validation of the multi-agent based distributed recommender system was done by two methods, which is previously presented in [5]. The first method is done, by validating the complete results of the simulation. The second method is the verification of the behaviour of each agent by checking the message sent to other agents (information flow). The validation of the MAS simulator was done by comparing its results to the central recommender system based simulator using the same selected scenario. For that reason, the same algorithms are used by the proposed multi-agent based simulator [3] and [4]. This is the extension of the previous work [3], a simulator application written in java language.

For the verification of the multi-agent system, we investigated the information flow of message passing from each agent to a recommender unit. Java agent development framework's test feature is used, which is compliant with the Foundation for Intelligent Physical Agent standards. The proposed simulator used the same compliance for interaction between the agents as detailed in [58]. For the verification of the information flow (message passing), the JADE Sniffer Agent was used. Sniffer agent, when activated in the MAS platform, can track all messages sent and received by any agent in the MAS environment. This agent can sniff or track a single agent or a group of agents. Messages exchanged in the agent platform can be recorded and viewed using the sniffer agent window [58, 96]. We verified each agent that sends information to the next agent by using the sniffer agent. This verification approach is similar to the verification approach of Virtual Overlay Multi-Agent System (VOMAS) used in [97]. The verification is also done manually by comparing all the message passing used in the simulation with the results of the sniffer agent which detailed the complete flow of information (message) between the agents.

4.5. Summary

In this chapter, we develop the multi-agent framework to be used for prototyping the recommender system for various simulation scenarios. The implementation is done in three modules. In the first, the data (tweets) is downloaded from selected users from the Twitter website. This is accomplished by implementing a multi-agent based data crawler module. In the second module, a multi-agent based Twitter Simulator module is developed that can mimic the Twitter environment. This Twitter simulator uses tweets from database (downloaded by data crawler module) and simulating users publishing the tweets. In the third module, a multi-agent based distributed recommender system is developed. This distributed recommender system is capable of using different agents, including text processing agents and recommender agents based on algorithms and models discussed in Chapter 3. In the next chapter, two configurations of distributed recommender system will be used in multi-agent based simulation to examine the theoretical results found in Chapter 3 about scalability factors and running time reductions.

Chapter 5

Simulation and Experimental Results

The intent of this chapter is to evaluate the efficiency of the proposed distributed recommender system by conducting simulation and comparing it against a central recommender system or one server recommender system. Also, we will show results obtained from the simulation of four different types of experiments on each dataset. A total of four experiments are performed on each dataset by using the central recommender system, two server recommender system, four server recommender system and eight server recommender system. In total five different algorithms are implemented and simulated in the experiments.

As explained in Chapter 3, main modules that have been implemented to develop the framework are data crawler module, Twitter environment simulator module, and distributed recommender system module. The data collection is done by using MAS based data crawler module that is explained in Chapter 4.1. There are five seed Twitter users that are used for data collection and to construct the Twitter social network graph. The data set for each seed user is collected in the interval of every 15 days for six months. The last data set is used for evaluating the recommended results.

The second module is the MAS based Twitter environment simulator module, this module uses the data sets collected by the data crawler module. Twitter environment simulator module as explained in Chapter 4.2, is a multi-agent based system. Each Twitter user has its own agent in a multi-agent environment which simulated as a tweet text generator.

5.1. Datasets

To evaluate the proposed framework we need to build the Twitter social network graph, which is available on the Twitter website. Data crawler module builds the local repository for each user by downloading the following data:

User Profile (user name, location, number of follower, number of followee, etc.)

1. Tweets and retweets of user.
2. List of Followers of user.
3. List of Followees of user.
4. Tweets and retweets of user's followers.
5. Tweets and retweets of user’s followees.

Information which was downloaded for each tweets and retweets of any user includes the userID, tweetID, date and time tweet is published and tweet text. Following are the total data (tweets and user) downloaded by using the data crawler module.

Table 5.1: Shows the user graph information downloaded by using data crawler.

Attributes	Total
Users	6095
Tweets	931031
Followers	12898
Followees	1961
Restricted Users	293

The five different datasets are downloaded. The datasets are collected in different time frame. The main reason to use 5 different datasets is that each dataset shows users following specific twitter user therefore capturing different types of tweets from different set of users. The network graph is created for all users based on specific user followed by them. For example the data of all users who followed Ryerson user in a specific time frame captured in dataset 4. Tweets text, tweetIDs, etc. are collected for all the followers of the user for specific date, so that comparison can be done in the recommendation process. The recommendation process uses all the tweets one day before the real date when the users start following the main user. The following table shows five datasets used in the experiments for two different sets of the distributed recommender system.

Table 5.2: Shows five datasets used in the experiments.

Data Set	Total Tweets	Total Users
1	8003	37
2	9760	35
3	16009	56
4	21766	15
5	25533	41

5.2. Simulation Platform

All three proposed modules are implemented using Java programming language in Eclipse IDE for Java EE Developers compiler on the Windows 7 platform. System configuration used in this work is shown in Table 5.3. All modules are based on multi-agent system, therefore, we use a development tool known as Java Agent Development framework [58, 98]. The data crawler module connects to the Twitter API 1.1, different functions are developed to get user content from the Twitter website. Twitter API is freely available for all developers and researchers. There are many functions such as get user time line or get followers of user by implementing GET connection. The output of each GET function is in a simple JSON (JavaScript Object Notation) format. The JSON output is then parsed and stored in the database. We used mySQL Server 5.6 Database, which is a lightweight relational database management system and also use mySQL Workbench 6.3 Community Edition, both are available freely under GPL Licenses. Different tables are created in the database named “twdatabase”. Tables which are created are userTweets, userFollowers, userFollowees, and userRestricted.

Table 5.3: System Configuration

CPU	Intel core i5-430M
Speed	2.26GHz, 3MB L3 Cache
RAM	4GB DDR3
Platform	Windows 7

The implementation of the Twitter simulator module is also based on JADE. One agent is used for each Twitter user. The user agent reads their own tweets from mySQL database and is responsible to tweet based on the date time stamp of the tweet. This tweet is then published the Twitter simulator environment.

5.3. Experimental Setup and Scenarios

In this section, two main simulation scenarios are considered which are as follows:

- Varying the number of distributed servers to minimize the running time of the recommendation process.
- Maximize the degree of accuracy by performing simulations using different configurations of proposed system architecture.

For each experiment, dynamic programming is used to find how many users can be placed on one server, which depends on the number of servers available in the distributed environment. Each server will have almost equal number of tweets to process, but the total number of tweets on each server may not be 100% equal, because every user may have a different number of tweets. The number of tweets in each dataset is less than 50,000 so the results can be compared with theoretical results of time complexity provided in Chapter 3.

5.4. Experimental Results for SDRS

In this section, the first experimental results for each of the consisting algorithms of SDRS will be shown and then entire SDRS including all algorithms.

5.4.1. Running Time of each Algorithm used in SDRS

In this section, we show the results obtained on five datasets for all algorithms used namely, text processing, tern frequency, and cosine similarity. We compare the central algorithm with 2 servers, 4 servers, and 8 servers distributed systems on the basis of the performance metrics stated in Chapter 3. The results which are obtained, validates the theoretical proofs provided in Chapter 3.

Following are the experiments, where the number of servers is changed whose range are 1, 2, 4, and 8, when only one server is used, it is known as a central recommender system, while from 2 to 8 (in power of 2) are the distributed recommender systems. When the number of server varied, we find that there is a great impact on processing time as mentioned in Chapter 3, which justifies the theoretical foundation of this thesis. The results of all five datasets for all server configurations are depicted in the following figures:

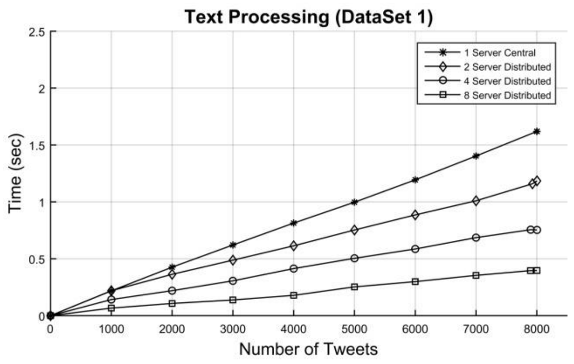

Figure 5.1: Shows text processing units used in 4 different server configurations for dataset 1.

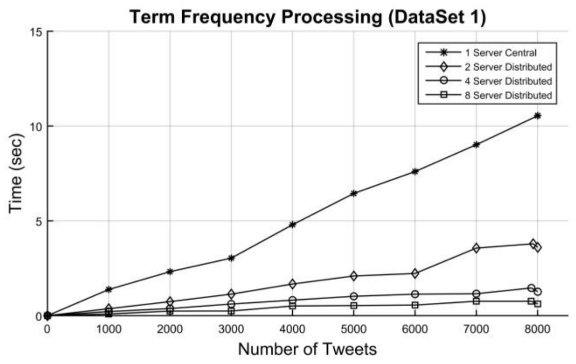

Figure 5.2: Shows term frequency processing used in 4 different server configurations for dataset 1.

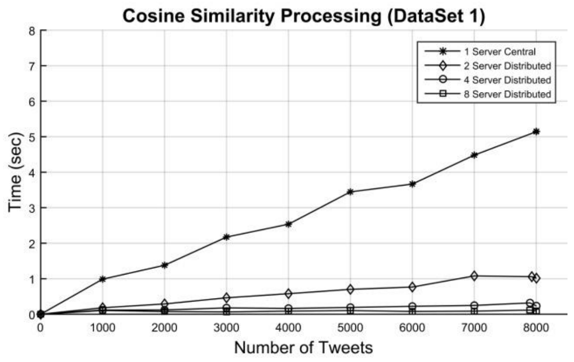

Figure 5.3: Shows cosine similarity units used in 4 different server configurations for dataset 1.

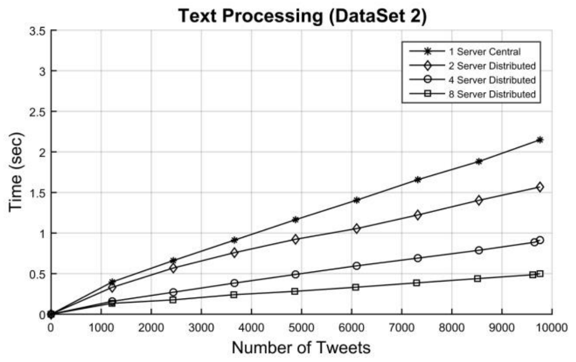

Figure 5.4: Shows text processing units used in 4 different server configurations for dataset 2.

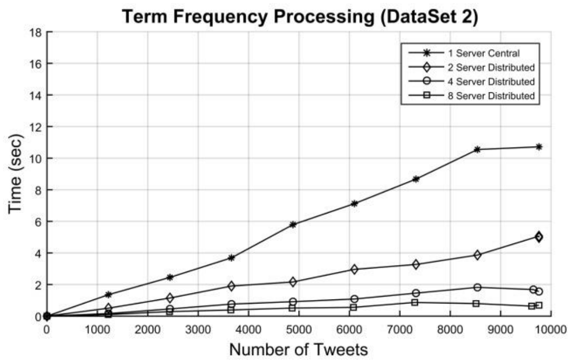

Figure 5.5: Shows term frequency processing used in 4 different server configurations for dataset 2.

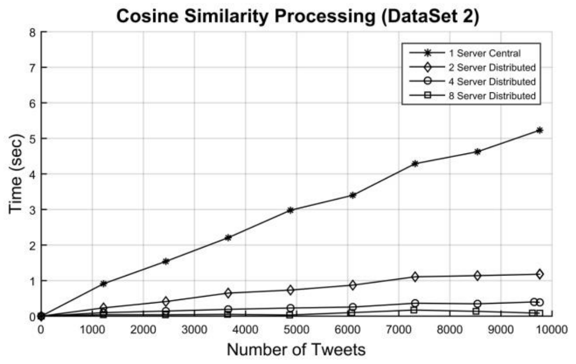

Figure 5.6: Shows cosine similarity units used in 4 different server configurations for dataset 2.

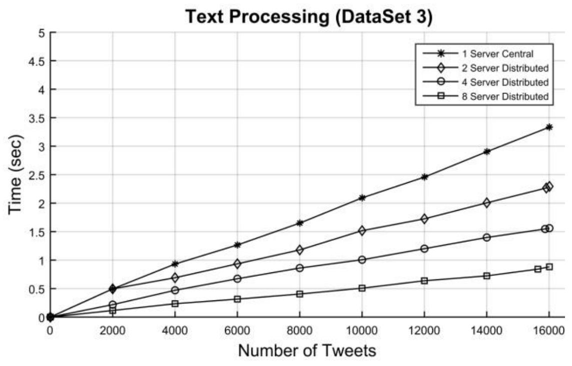

Figure 5.7: Shows text processing units used in 4 different server configurations for dataset 3.

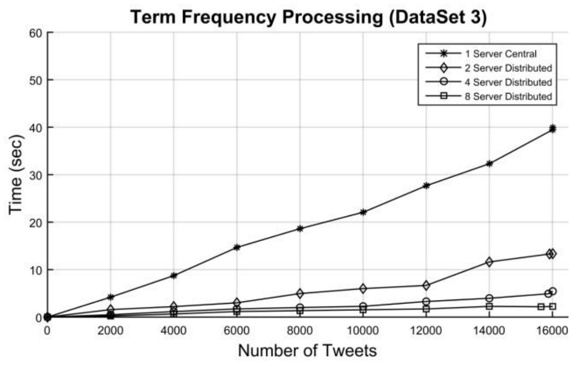

Figure 5.8: Shows term frequency processing used in 4 different server configurations for dataset 3.

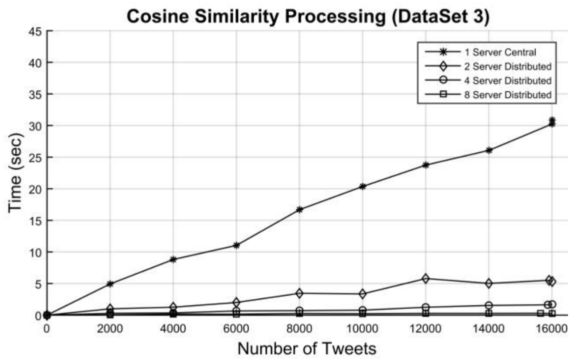

Figure 5.9: Shows cosine similarity units used in 4 different server configurations for dataset 3.

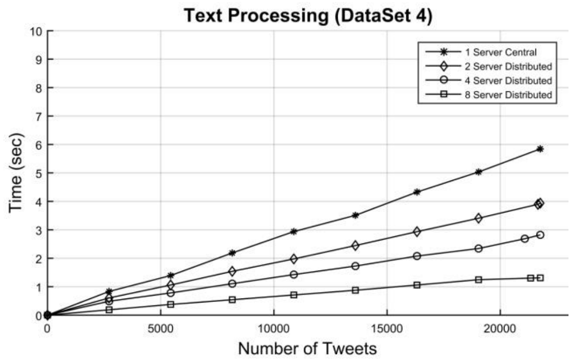

Figure 5.10: Shows text processing units used in 4 different server configurations for dataset 4.

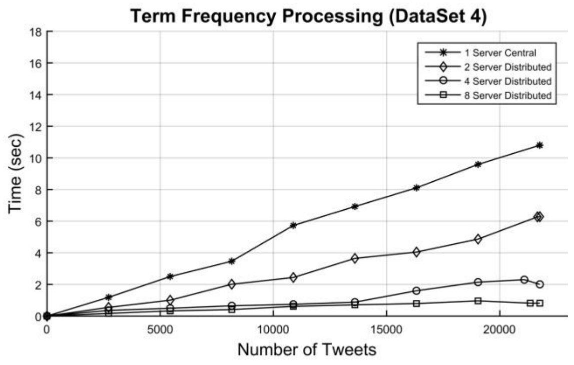

Figure 5.11: Shows term frequency processing used in 4 different server configurations for dataset 4.

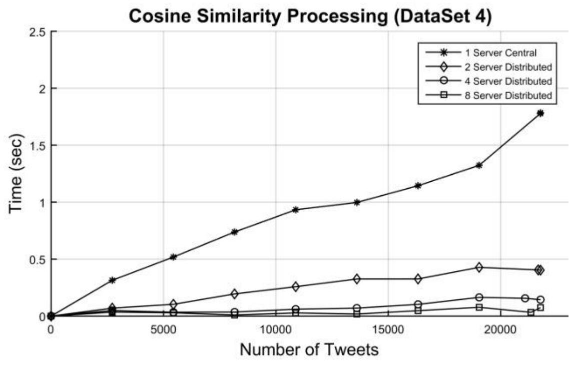

Figure 5.12: Shows cosine similarity units used in 4 different server configurations for dataset 4.

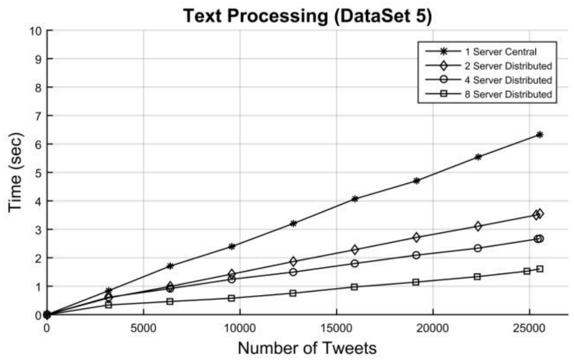

Figure 5.13: Shows text processing units used in 4 different server configurations for dataset 5.

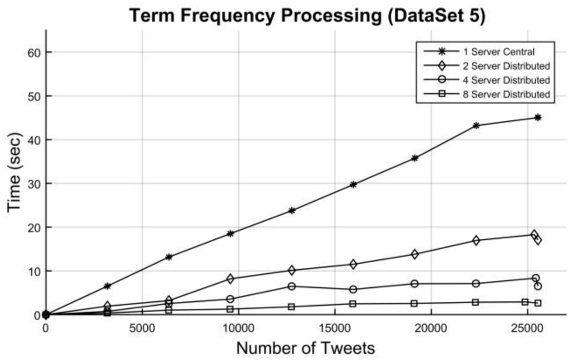

Figure 5.14: Shows term frequency processing used in 4 different server configurations for dataset 5.

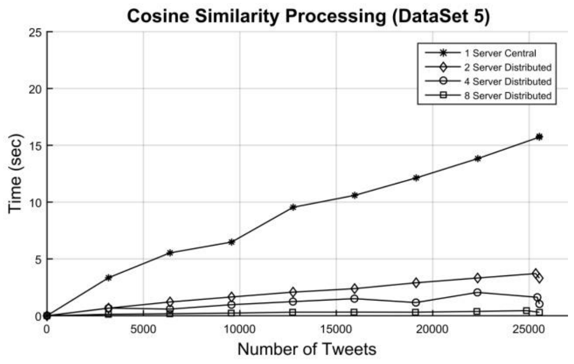

Figure 5.15: Shows cosine similarity units used in 4 different server configurations for dataset 5.

In all of the above experiments, the results show significant reduction in the processing time when different number of servers are used in a distributed configuration. The total time of text processing decreases as the number of text processing unit increases. Therefore, each text processing unit as they are in parallel will work simultaneously and the total text processing time decreases. In a similar way, the time of term frequency calculation will decrease by the increase in the number of recommender units. This performance improvement is from sharing tweets workload to multiple recommender units, by reducing the amount of data that each parallel recommender unit process. All

Figure 5.1 to Figure 5.15 confirm the theoretical results achieved in Section 3.3 of Chapter 3 which shows cosine and term frequency algorithms in central version running time tend to show non-linear relation with the increase in the number of tweets. We can see by using a minimum of two servers, it results linear relation between running time and number of tweets which makes the all algorithms to be scalable. And also when we use 4 servers instead of 2 servers the performance increases in all datasets. It is interesting to note that for small dataset such as dataset 1, when the number of servers increase to 8 instead of 4 the performance improves only marginally compared to 4 servers scenarios.

5.4.2. Performance Improvement of each Algorithm used in SDRS

In the experiment for dataset 1, the results shows that the average tweet time is reduced for text processing from 0.202 milliseconds on the central to 0.05 milliseconds on the distributed recommender system with 8 recommender units, Similarly, the average tweet time reduction for term frequency is from the central 1.319 milliseconds to the 8 server distributed 0.078 milliseconds and the cosine similarity time reduction from central 0.644 milliseconds to the 8 server distributed 0.011 milliseconds. The following Table 5.4 and Table 5.5 shows the average tweet time for central and distributed respectively for all five datasets.

Table 5.4: Shows the average tweet time (in milliseconds) for TF and cosine similarity for central server by using SDRS.

Data Set	**Central (1 Server)**		
	Text	**TF**	**Sim**
1	0.202	1.319	0.644
2	0.220	1.098	0.536
3	0.208	2.494	1.928
4	0.269	0.496	0.082
5	0.248	1.767	0.618

Table 5.5: Shows the average tweet time (in milliseconds) for TF and cosine similarity for 2, 4, and 8 server configurations using SDRS.

Data Set	**2 Servers**			**4 Servers**			**8 Servers**		
	Text	**TF**	**Sim**	**Text**	**TF**	**Sim**	**Text**	**TF**	**Sim**
1	0.148	0.450	0.127	0.094	0.157	0.029	0.050	0.078	0.011
2	0.161	0.510	0.122	0.094	0.160	0.040	0.051	0.071	0.009
3	0.143	0.835	0.332	0.097	0.340	0.107	0.055	0.140	0.016
4	0.181	0.289	0.019	0.130	0.092	0.007	0.060	0.037	0.003
5	0.139	0.668	0.129	0.105	0.255	0.040	0.063	0.104	0.012

The Table 5.6 shows the performance improvement calculation for each algorithm by using 2, 4, and 8 server configurations for all datasets. In dataset 1, we have gained 75.25% improvement in the text processing by using the 8 server configuration. The performance improvement for term frequency and cosine similarity calculation in dataset 1 are 94.09% and 98.29% respectively by using the 8 server configuration.

Table 5.6: Shows the performance improvement of the average tweet (in percentage) for 2, 4, and 8 recommender server configurations using SDRS.

Data Set	**Performance Improvement (in %)**								
	2 Servers			**4 Servers**			**8 Servers**		
	Text	**TF**	**Sim**	**Text**	**TF**	**Sim**	**Text**	**TF**	**Sim**
1	26.73	65.88	80.28	53.47	88.10	95.50	75.25	94.09	98.29
2	26.82	53.55	77.24	57.27	85.43	92.54	76.82	93.53	98.32
3	31.25	66.52	82.78	53.37	86.37	94.45	73.56	94.39	99.17
4	32.71	41.73	76.83	51.67	81.45	91.46	77.70	92.54	96.34
5	43.95	62.20	79.13	57.66	85.57	93.53	74.60	94.11	98.06

5.4.3. Message Passing Time in SDRS

This experiment shows the total time for message passing for different configurations of servers (i.e. 2, 4, and 8). The message passing time is the total time to send and receive messages from one server to other(s). This total time also includes the time required to send a request message from organizing agent. In the Figure 5.16, the message passing time is shown for similarity-based distributed recommender system.

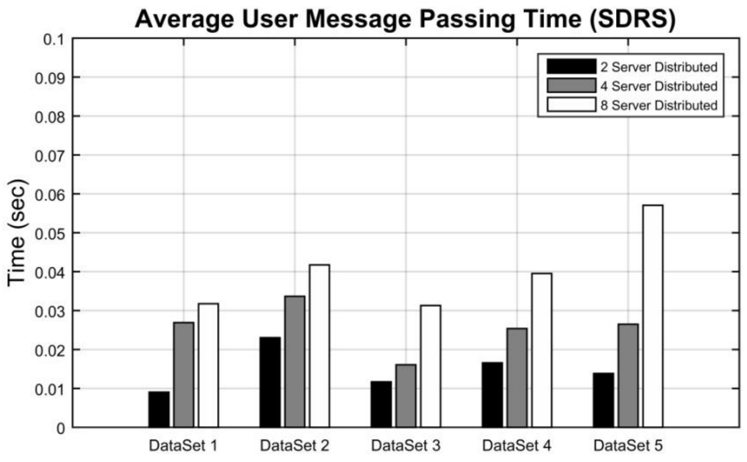

Figure 5.16: Shows average user message passing time in similarity-based distributed recommender system.

In the above Figure 5.16, it can be seen that the average message passing time for a user is negligible. It is only in the range from 0.03 seconds to 0.06 seconds.

5.4.4. Total Running Time of SDRS using Distributed Database

As explained in Chapter 3, another component of the distributed recommender system is that it reduces disk seek time when using distributed databases. To calculate such reduction we have performed experiments for the proposed distributed recommender system by using distributed database (disk storage).

$$Total\ Time = Time\ (Algorithms) + Message\ Passing\ Time + Disk\ IO$$

The following Table 5.7 shows the total running time (in seconds) when distributed database is used for data processing. It can be seen from Table 5.7 below for dataset 1, when using the central recommender system, the total running time is 41.18 seconds which is reduced to 9.21 seconds by using 8 recommender units. Total time for all datasets using SDRS are shown in Figure 5.17 to Figure 5.21. These figures show that in all datasets (except for dataset 5) the central configuration is scalable. The reason for the non-linear behaviour of dataset 5 is because the last dataset has more number of tweets therefore, running time of SDRS contributes more than disk IO in total SDRS time. In the other datasets improvement of using two servers reduces running time, but does not change linearity because with a low number of tweets the disk IO is dominating factor that has a linear relation with the increase in the number of tweets.

Table 5.7: Total time (in second) for central, 2, 4, and 8 server configurations for SDRS using distributed database.

Data Set	**Total Tweets**	**Total Users**	**Total Time for SDRS (in sec)**			
			1 Server	**2 Server**	**4 Server**	**8 Server**
1	8003	37	41.18	26.08	14.03	9.21
2	9760	35	48.41	33.82	17.53	10.68
3	16009	56	121.63	59.07	29.33	16.91
4	21766	15	90.82	70.84	38.85	21.96
5	25533	41	148.40	89.74	45.27	26.68

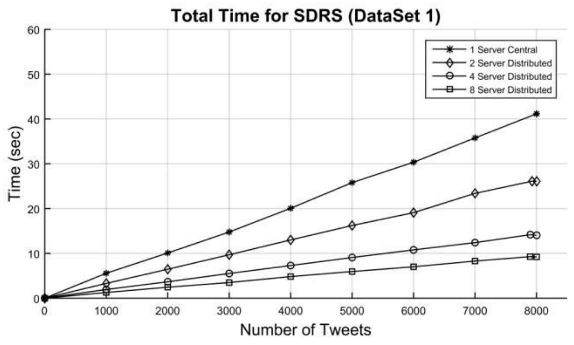

Figure 5.17: Shows Total time for SDRS in 4 different server configurations for dataset 1.

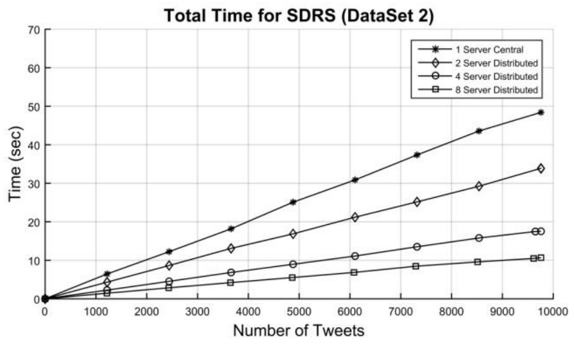

Figure 5.18: Shows Total time for SDRS in 4 different server configurations for dataset 2.

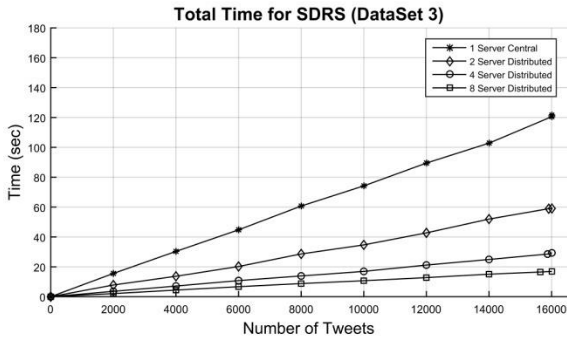

Figure 5.19: Shows Total time for SDRS in 4 different server configurations for dataset 3.

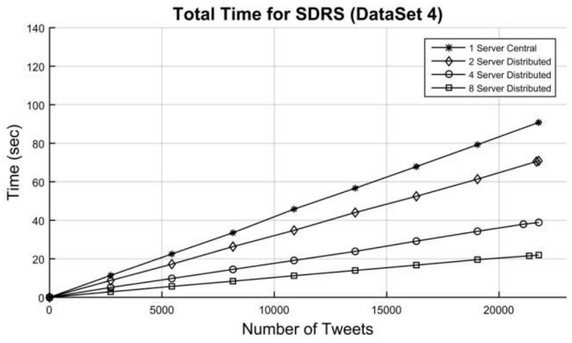

Figure 5.20: Shows Total time for SDRS in 4 different server configurations for dataset 4.

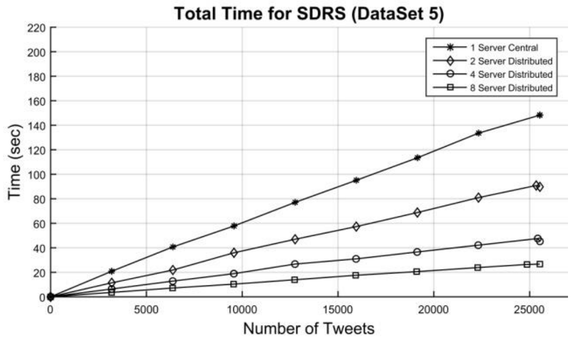

Figure 5.21: Shows Total time for SDRS in 4 different server configurations for dataset 5.

The Table 5.8 shows that when using eight servers the reduction of running time for SDRS is in the range of 75.82% to 86.1%. The results of performance improvement by using 2, 4, and 8 server configurations are shown in the Figure 5.22.

Table 5.8: Shows performance improvement (in %) for 2, 4, and 8 server configurations for SDRS using a disk-based distributed database.

Data Set	Performance Improvement for SDRS (in %)		
	2 Server	**4 Server**	**8 Server**
1	36.67	65.93	77.63
2	30.15	63.79	77.93
3	51.44	75.89	86.10
4	22.00	57.22	75.82
5	39.53	69.50	82.02

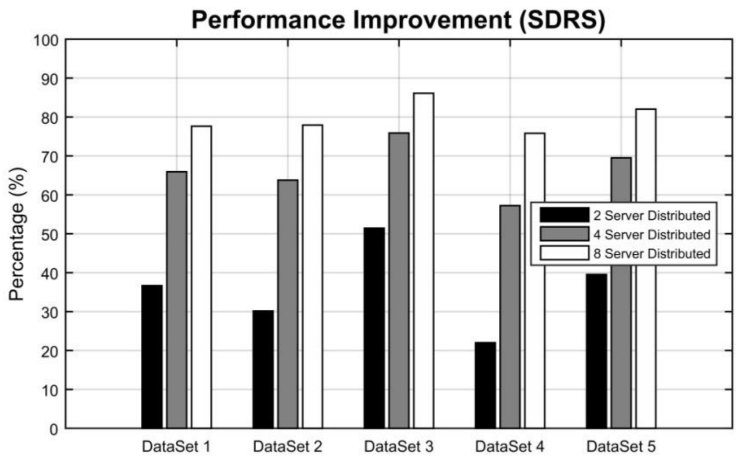

Figure 5.22: Shows the performance improvement (in %) for all five datasets using a disk-based distributed database for SDRS.

5.5. Experimental Results for CDRS

5.5.1. Running Time of each Algorithm used in CDRS

In this section, we will show the results obtained on five datasets for all algorithms used namely, text processing, term frequency inverse document frequency and k-mean clustering algorithm. We compare the central, 2 servers, 4 servers and 8 servers distributed system on the basis of the performance metrics stated in Chapter 3. The results which are obtained, validates the theoretical concepts provided in Section 3.3 clearly shows the central variation in k-mean running time for all dataset is non-linear whereas using two servers it is linear and scalable.

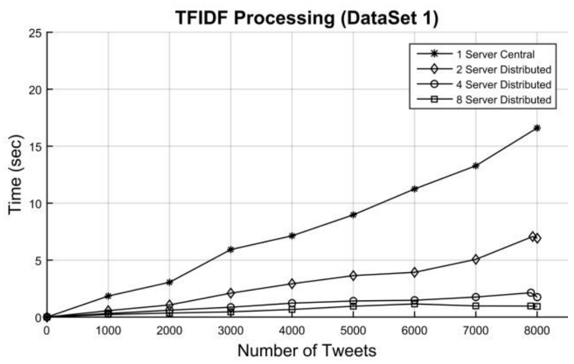

Figure 5.23: Shows TFIDF processing used in 4 different server configurations for dataset 1.

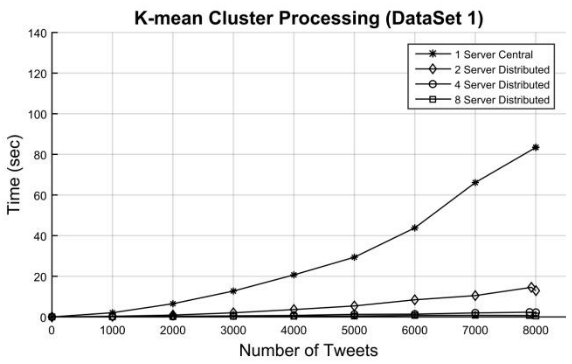

Figure 5.24: Shows k-mean clustering units used in 4 different server configurations for dataset 1.

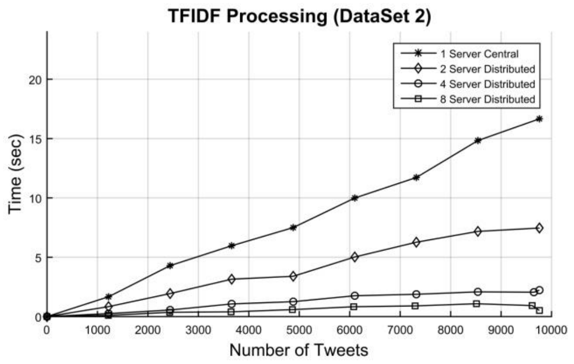

Figure 5.25: Shows TFIDF processing used in 4 different server configurations for dataset 2.

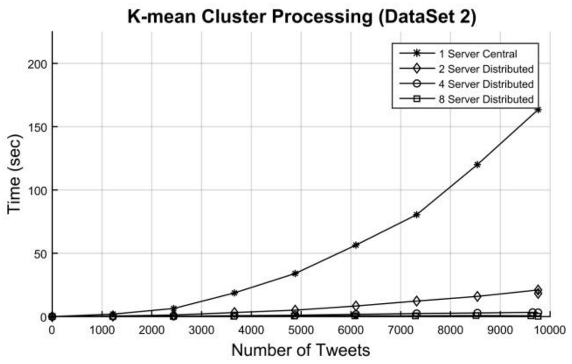

Figure 5.26: Shows k-mean clustering units used in 4 different server configurations for dataset 2.

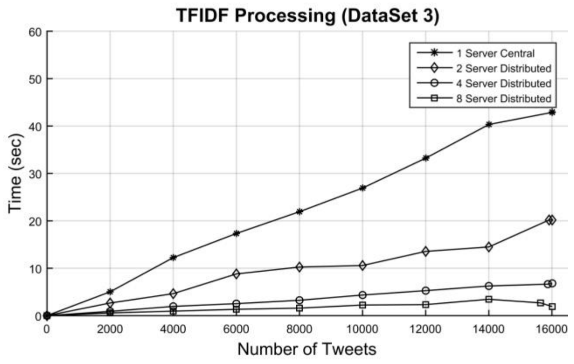

Figure 5.27: Shows TFIDF processing used in 4 different server configurations for dataset 3.

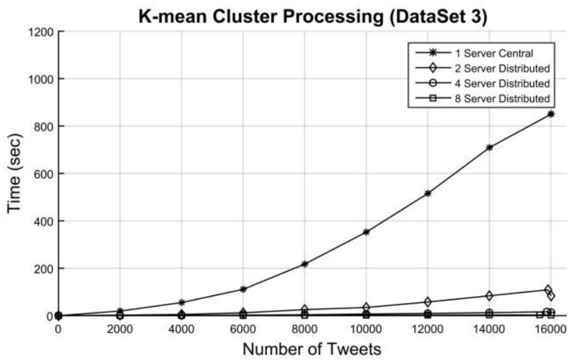

Figure 5.28: Shows k-mean clustering units used in 4 different server configurations for dataset 3.

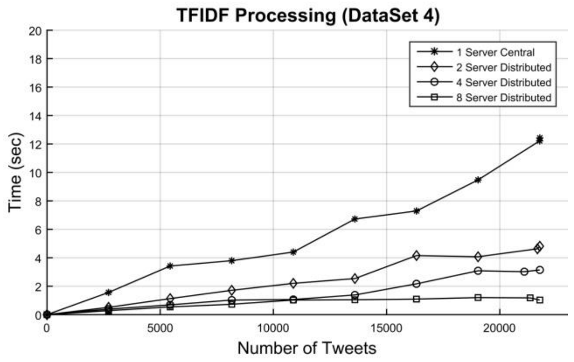

Figure 5.29: Shows TFIDF processing used in 4 different server configurations for dataset 4.

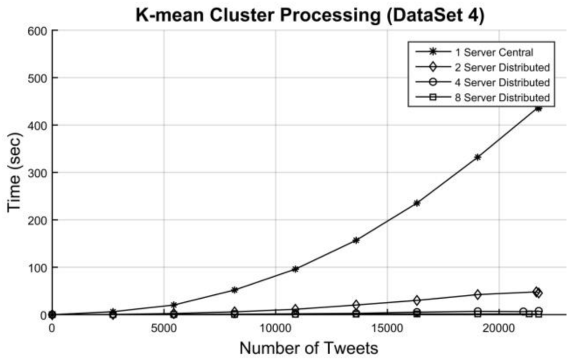

Figure 5.30: Shows k-mean clustering units used in 4 different server configurations for dataset 4.

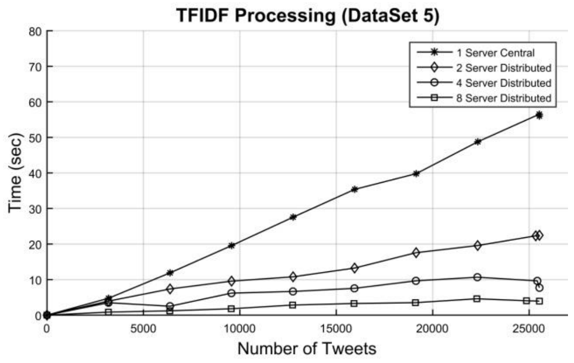

Figure 5.31: Shows TFIDF processing used in 4 different server configurations for dataset 5.

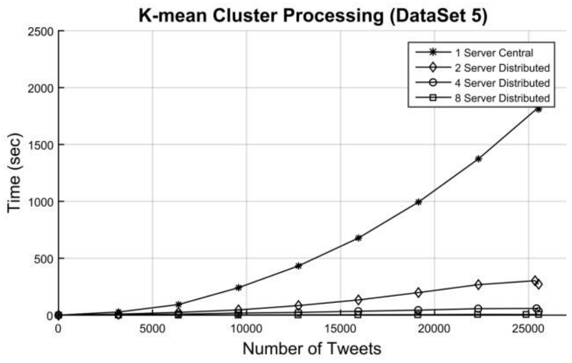

Figure 5.32: Shows k-mean clustering units used in 4 different server configurations for dataset 5.

In all above experiments, the result shows significant reduction in processing time when distributed configuration of server is used. In a similar way as explained above, the time of TFIDF calculation is decreases with the increase in the number of recommender units.

5.5.2. Performance Improvement of each Algorithm used in CDRS

In the experiment for dataset 1, the results show that the average tweet time is reduced for TFIDF processing from 2.074 milliseconds on the central to 0.116 milliseconds on the distributed recommender system with 8 recommender units. Similarly, the average tweet time reduction for k-mean clustering is from central 10.436 milliseconds to 0.058 milliseconds using 8 servers. The following Table 5.9 shows the average tweet time for the central recommender system and Table 5.10 shows the average tweet time for 2, 4, and 8 servers distributed recommender system for all five datasets.

Table 5.9: Shows the average tweet time (in milliseconds) for TFIDF and k-mean clustering for central server using CDRS.

Data Set	**Central (1 Server)**	
	TFIDF	**K-mean**
1	2.074	10.436
2	1.707	16.751
3	2.684	53.205
4	0.571	19.980
5	2.193	70.893

Table 5.10: Shows the average tweet time (in milliseconds) for TFIDF and k-mean clustering for 2, 4, and 8 server configurations using CDRS.

Data Set	**2 Servers**		**4 Servers**		**8 Servers**	
	TFIDF	**K-mean**	**TFIDF**	**K-mean**	**TFIDF**	**K-mean**
1	0.865	1.612	0.218	0.278	0.116	0.058
2	0.766	1.875	0.228	0.330	0.053	0.028
3	1.260	5.208	0.426	0.855	0.119	0.153
4	0.221	2.096	0.144	0.353	0.047	0.047
5	0.880	10.642	0.302	1.343	0.154	0.265

The Table 5.11 shows the performance improvement for 2, 4, and 8 server configurations. We have able to achieve the improvement of 96.90% for TFIDF by using the 8 server configuration (in dataset 2). The performance improvement for clustering (k-mean) calculation is 99.83% by using 8 servers distributed recommender system (in dataset 2).

Table 5.11: Shows the performance improvement for average tweet (in percentage) for 2, 4, and 8 recommender server configurations using CDRS.

Data Set	**Performance Improvement (in %)**					
	2 Servers		**4 Servers**		**8 Servers**	
	TFIDF	**K-mean**	**TFIDF**	**K-mean**	**TFIDF**	**K-mean**
1	58.29	84.55	89.49	97.34	94.41	99.44
2	55.13	88.81	86.64	98.03	96.90	99.83
3	53.06	90.21	84.13	98.39	95.57	99.71
4	61.30	89.51	74.78	98.23	91.77	99.76
5	59.87	84.99	86.23	98.11	92.98	99.63

5.5.3. Message Passing Time in CDRS

This experiment shows the total time for message passing for different configurations of servers (i.e. 2, 4, and 8) using CDRS. In the Figure 5.33, the message passing time is shown for clustering-based distributed recommender systems using all five datasets. In the Figure 5.33, it can be seen that the message passing time for a user is low. It is only in the range from 0.05 seconds to 0.18 seconds.

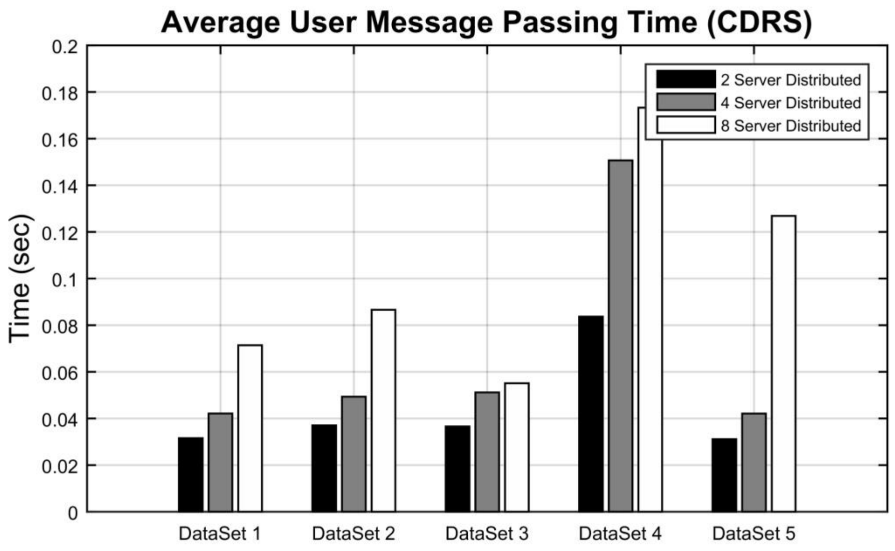

Figure 5.33: Shows average user message passing time in cluster-based distributed recommender system.

5.5.4. Total Running Time of CDRS using Distributed Database

We have performed the experiment using distributed database for CDRS. The following Table 5.12 shows the total time when distributed database is used. When using two servers the reduction of total running time for CDRS is 40.99 seconds while the central recommender system is 125.6 seconds, which confirms the theoretical results shown in Chapter 3. Total time for all datasets using CDRS are shown in Figure 5.34 to Figure 5.38.

Table 5.12: Shows the total time (in second) for central, 2, 4, and 8 server configurations for CDRS.

Data Set	**Total Tweets**	**Total Users**	**Total Time for CDRS (in sec)**			
			1 Server	**2 Server**	**4 Server**	**8 Server**
1	8003	37	125.60	40.99	15.56	8.79
2	9760	35	212.61	52.66	19.90	9.33
3	16009	56	945.55	143.34	41.84	17.07
4	21766	15	525.55	114.44	47.29	22.71
5	25533	41	1953.61	363.02	78.69	32.22

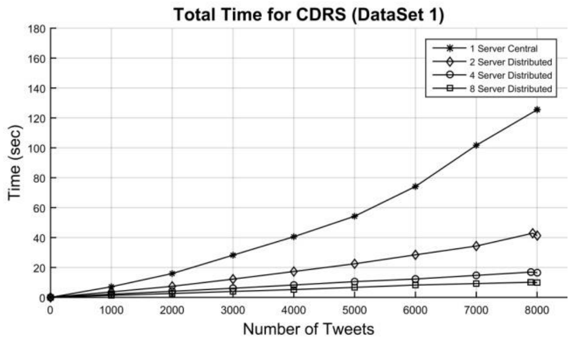

Figure 5.34: Shows Total time for CDRS in 4 different server configurations for dataset 1.

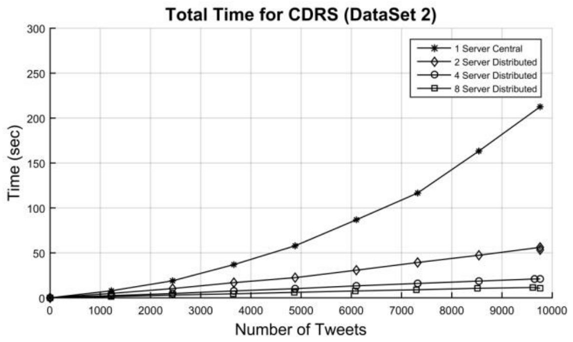

Figure 5.35: Shows Total time for CDRS in 4 different server configurations for dataset 2.

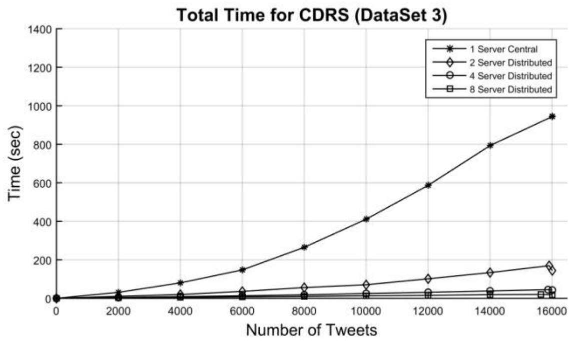

Figure 5.36: Shows Total time for CDRS in 4 different server configurations for dataset 3.

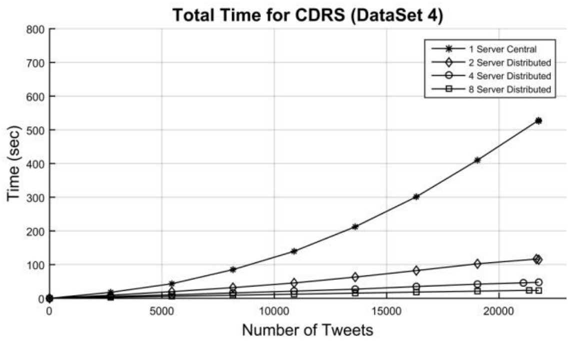

Figure 5.37: Shows Total time for CDRS in 4 different server configurations for dataset 4.

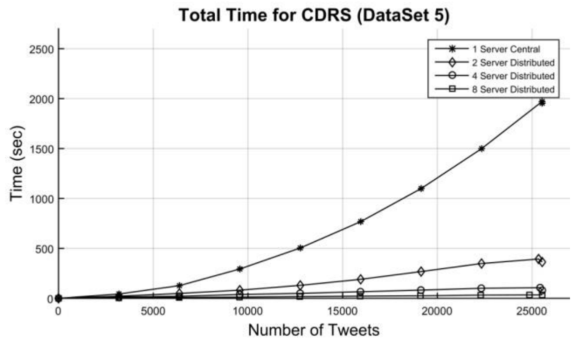

Figure 5.38: Shows Total time for CDRS in 4 different server configurations for dataset 5.

Table 5.13 shows that, when using two servers the reduction of total time for CDRS is in the range of 67.37% to 84.84%. The results of performance improvement by using 2, 4, and 8 server configurations are also shown in the Figure 5.39. With regards to scalability using at least two servers makes the system to have a non-linear relation between run time and number of tweets which confirms the theoretical results (time complexity) of CDRS in Section 3.5.2.1. It is interesting to note that in spite of SDRS in CDRS disk IO is not the dominating factor and non-scalable behaviour in the central recommendation system and the change it to linear relation by using two servers for such low data is shown. We expect for larger datasets the effectiveness of distributed solution will be obvious for all configurations of similarity or clustering based recommender systems.

Table 5.13: Shows performance improvement (in %) for 2, 4, and 8 server configurations for CDRS using a disk-based distributed database.

Data Set	Performance Improvement for CDRS (in %)		
	2 server	4 server	8 server
1	67.37	87.61	93.00
2	75.23	90.64	95.61
3	84.84	95.57	98.19
4	78.23	91.00	95.68
5	81.42	95.97	98.35

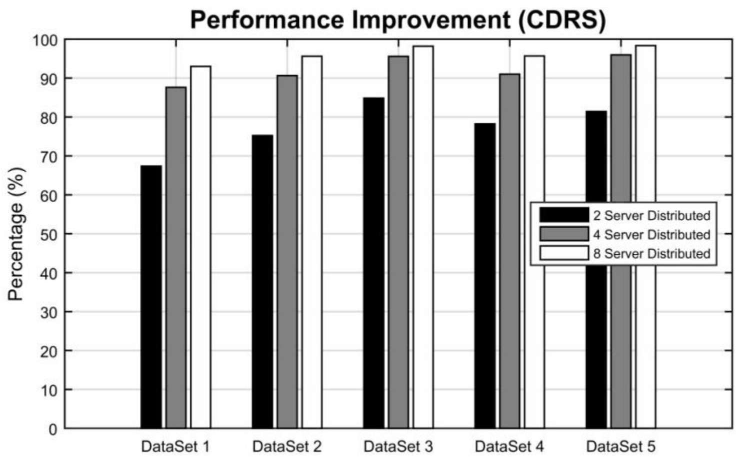

Figure 5.39: Shows the performance improvement (in %) for all five datasets using a disk-based distributed database for CDRS.

5.6. Evaluation of Distributed Recommender System

In this section, we look at the accuracy of recommender systems because our solution for efficiency will be useful when it does not change the accuracy of recommender systems. In this section, a common evaluation method of the mean absolute error for recommender systems is used for evaluating the accuracy of the proposed distributed recommender system. The mean absolute error method (MAE) is used in evaluating the accuracy of recommender systems by finding the prediction accuracy of the recommendations. MAE is calculated using the average absolute deviation of the predicted values from the actual values. If MAE value is small, then this means less error and the system is more accurate.

The MAE is an average of the absolute errors $|e_i|$, this can be calculated by the following formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|R_i - A_i|$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

Where, R = Recommended value.

A = Actual value.

e = Error between recommended value and actual value.

The smaller value of MAE shows that the prediction accuracy is higher and the highest value of MAE is 1. The MAE for SDRS using different server configurations is calculated for all five datasets and is shown in Table 5.14. As discussed, we checked the accuracy by using user network information, including followees and followers, which is cross checked with the recommendation of the proposed system in different time periods.

Table 5.14: Shows the Prediction Accuracy for central, 2, 4, and 8 server configurations using SDRS for five datasets.

MAE top k	**Prediction Accuracy of SDRS**				
	DataSet1	**DataSet2**	**DataSet3**	**DataSet4**	**DataSet5**
3	0.97	0.97	0.95	0.93	0.95
10	0.72	0.65	0.67	0.43	0.88
20	0.36	0.32	0.45	0.00	0.80
30	0.03	0.00	0.27	0.00	0.55
40	0.00	0.00	0.09	0.00	0.00

In above Table 5.14 when k=10, the prediction accuracy is in the range of 0.43 to 0.88 for central, 2, 4, and 8 server configurations. Please note that the prediction accuracy of 2, 4, and 8 servers are same as of central configuration. This is dues to the fact that in recommendation process cosine similarity is used; the cosine similarity depends on term frequency which is same for all distributed configurations of 2, 4, and 8 servers.

Table 5.15: Shows the Prediction Accuracy for central configuration using CDRS for five datasets.

MAE	**Prediction Accuracy of CDRS**				
top k	**DataSet1**	**DataSet2**	**DataSet3**	**DataSet4**	**DataSet5**
3	0.08	0.03	0.72	1.00	0.86
10	0.00	0.00	0.68	1.00	0.86
20	0.00	0.00	0.68	0.07	0.79
30	0.00	0.00	0.29	0.07	0.79
40	0.00	0.00	0.29	0.07	0.65
50	0.00	0.00	0.00	0.07	0.00

The above Table 5.15 show the prediction accuracy of the CDRS. When k=3, the prediction accuracy is in the range of 0.03 to 1 for central server configuration. The prediction accuracy for 2, 4, and 8 server configurations can be found in Appendix II. All the results show that, except for datasets 1 and 2 that have low numbers of tweet and users, the change in accuracy by increasing the number of servers in CDRS is negligible and in SDRS there is no change and the proposed method does not lower the accuracy of common recommender systems.

5.7. Comparison with Related Works

In this section, we compare our distributed recommender system with the closest method to ours used in the literature. In most of the similar works (for example [91] and [89]) data grouping (called a data chunk) are used to speedup the processing time for the large number of tweets. The following experiments compare data grouping technique and our proposed method for central, 2, 4, and 8 server configurations. Since the central approach is used in all related works, it is used as the compared method here and is examined with different chunk sizes together with our proposed methods. In other words, these experiments examine parallelization and distribution of data (used in proposed method) compared to grouping of data.

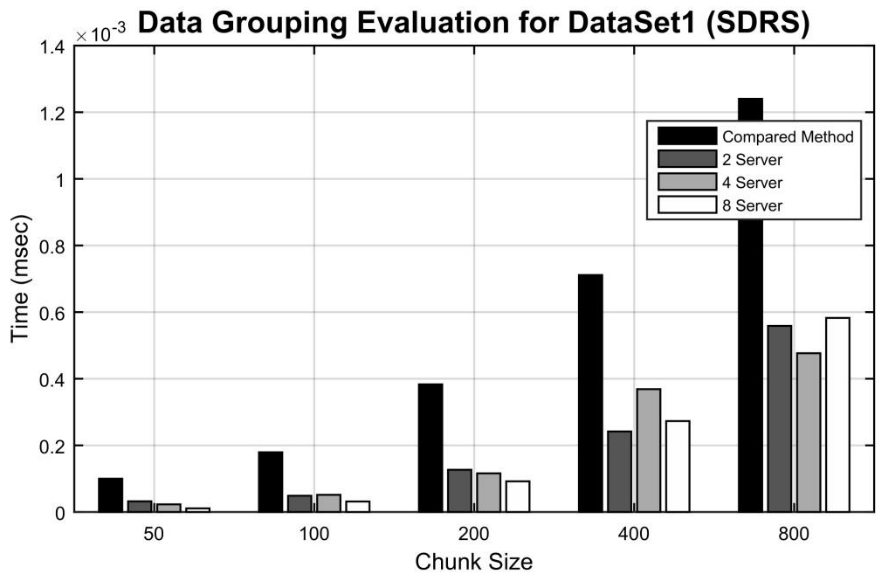

Figure 5.40: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for SDRS (Dataset1).

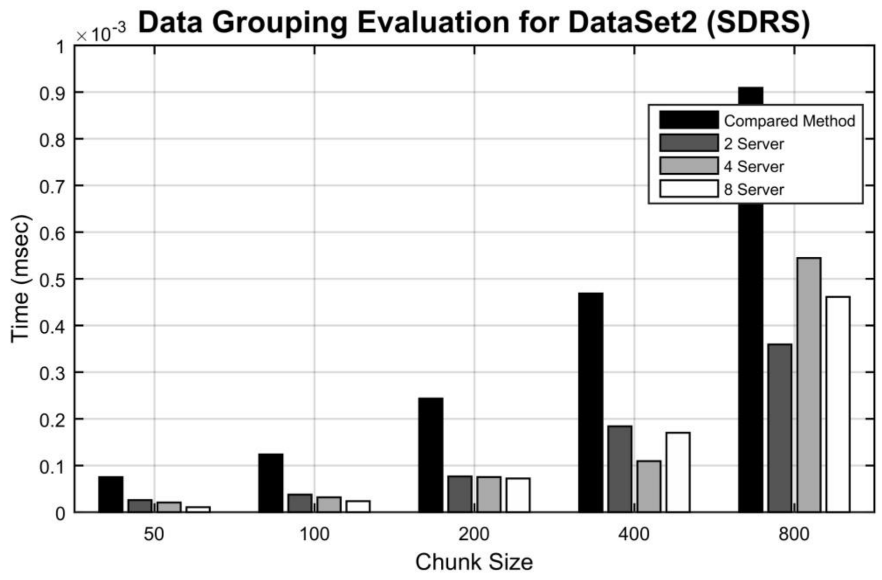

Figure 5.41: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for SDRS (Dataset2).

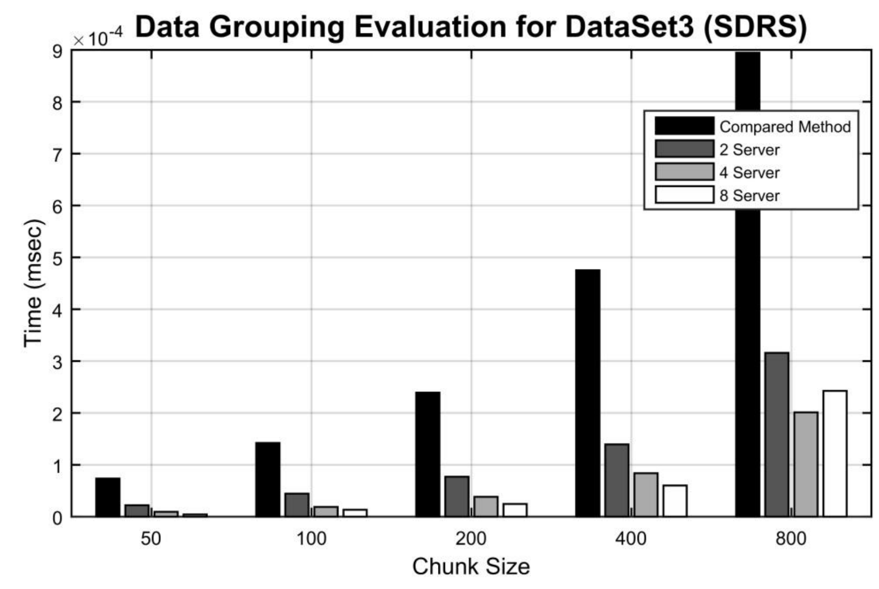

Figure 5.42: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for SDRS (Dataset3).

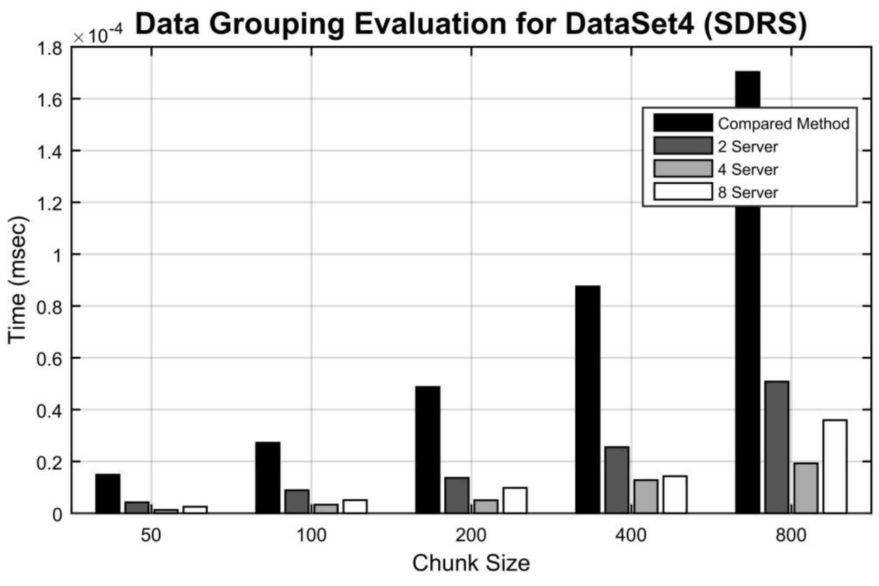

Figure 5.43: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for SDRS (Dataset4).

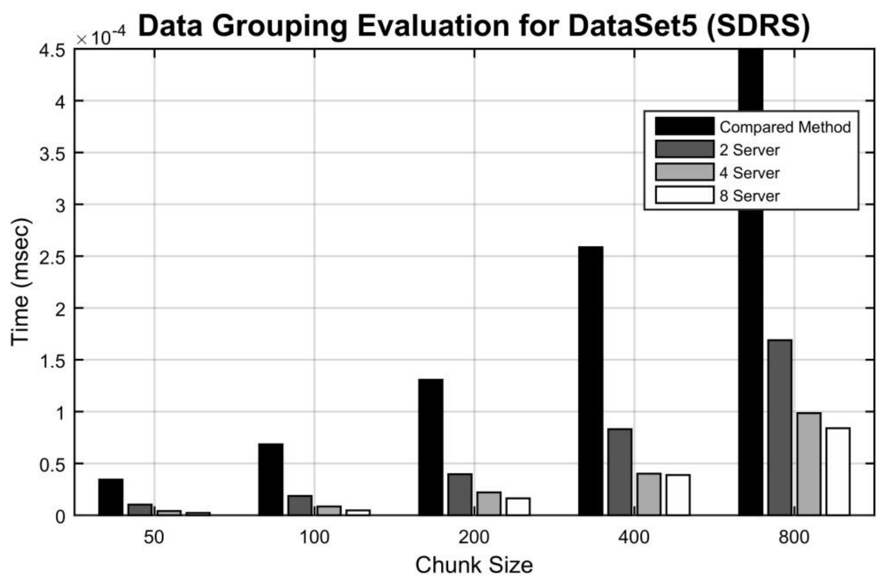

Figure 5.44: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for SDRS (Dataset5).

In all above experiments three factors affect the total processing speed. First is the total tweets processed by the recommender system, the second is the number of users which have those tweets that are being processed and the third factor which affects the recommendation processing time is the number of tweets for each users present in that chunk. In these experiments, it can be seen that with all chunk sizes (i.e. 50, 100, 200, 400, and 800) using all configurations of servers (i.e. 2, 4, and 8 servers) in our proposed method perform better than central which means the proposed solution also gets benefits of data grouping and is applicable for a small amount of data. It is clear with the increase in the number of tweets the proposed solutions become more superior.

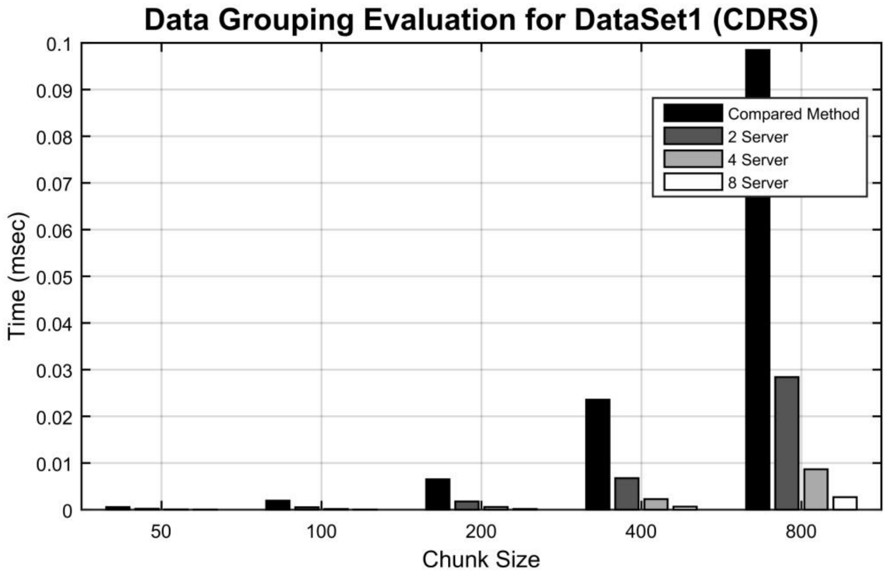

Figure 5.45: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for CDRS (Dataset1).

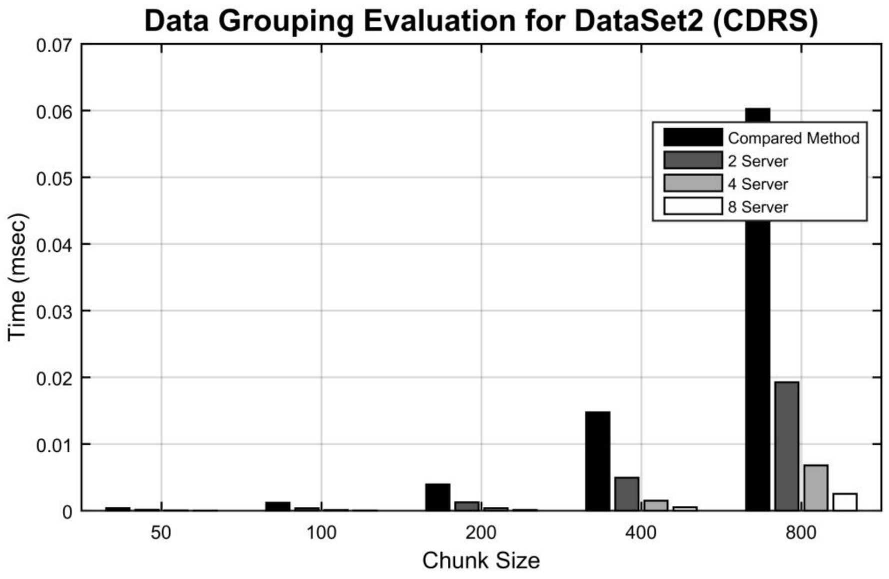

Figure 5.46: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for CDRS (Dataset2).

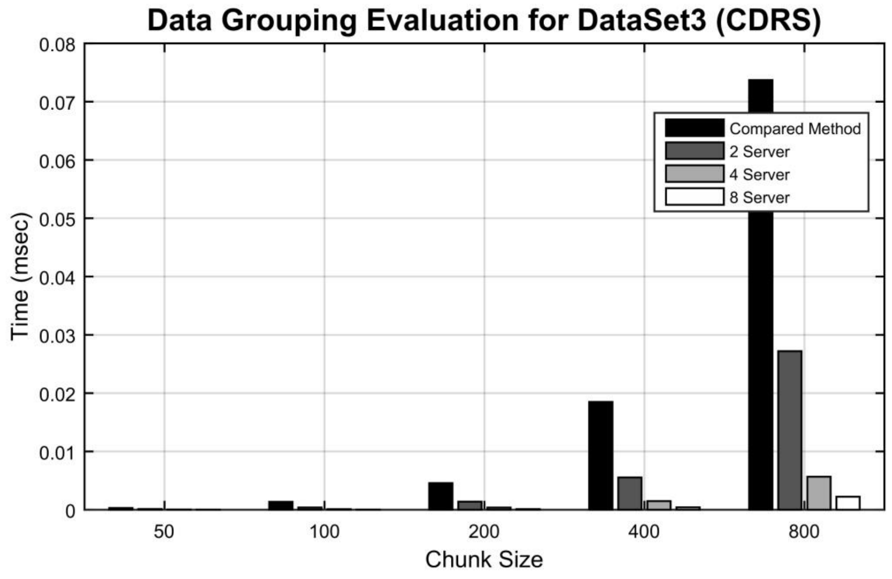

Figure 5.47: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for CDRS (Dataset3).

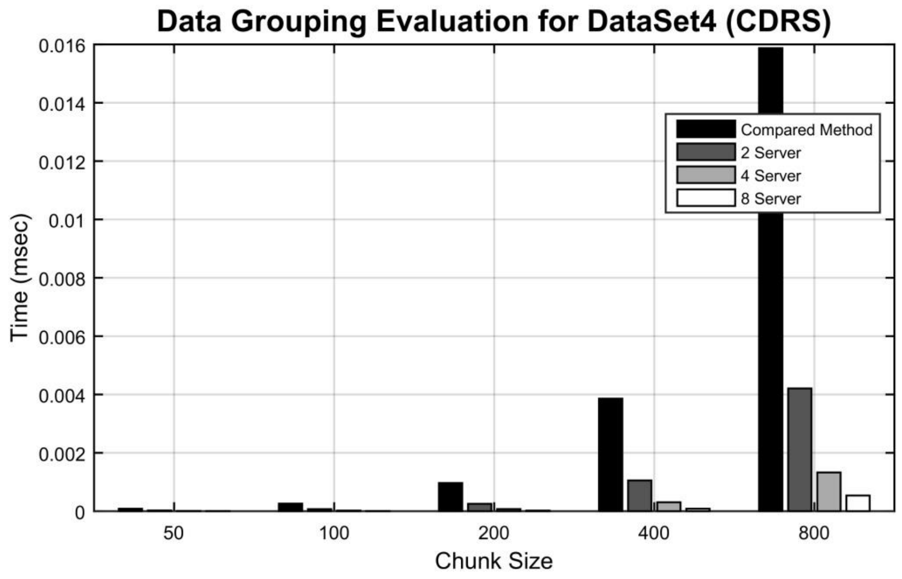

Figure 5.48: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for CDRS (Dataset4).

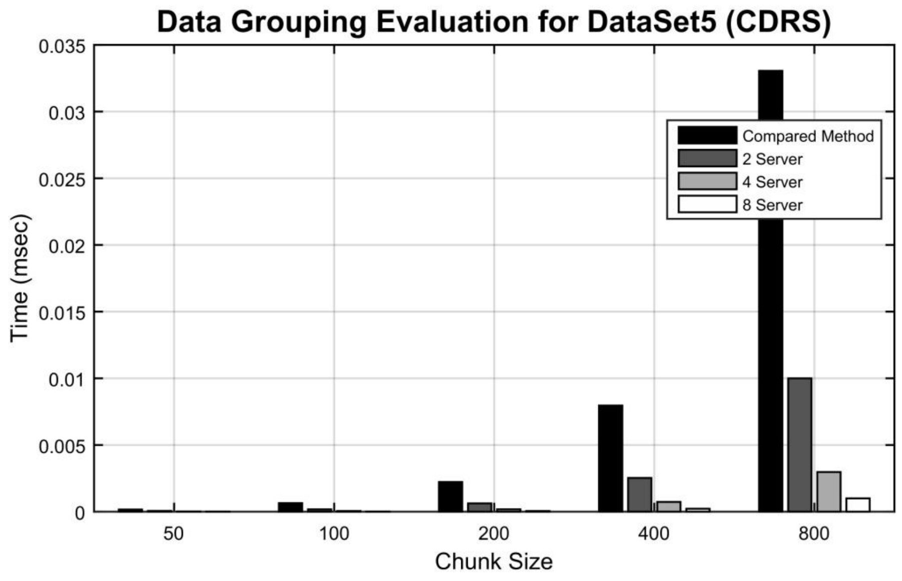

Figure 5.49: Shows the performance when 50, 100, 200, 400, and 800 tweet chunks are used for central, 2, 4, and 8 server configurations for CDRS (Dataset5).

In the above experiments for cluster-based distributed recommender system, it can be seen that when the chunk size is increased more processing time is required to process the chunk. Again, it is clear that the performance is improved by distributing the workload to 2, 4, and 8 servers, which are processing those distributed loads in parallel. Also, CDRS with 8 server configuration outperforms the other distributed configurations for all chunk size (data grouping) experiments. Finally, the comparison shows grouping data in the small chunk sizes less than 1,000 tweets reduces the processing time. However, it is clear the recommendation accuracy is reduced when number of tweets reduces. The graph shows our proposed method gains similar benefits from data grouping techniques in reducing processing time. But the advantage of our method is that it is useful for the recommendation techniques that need to use the large size of tweets to provide more accurate recommendations.

5.8. Discussion of Scalability Factor

The following graphs show for CDRS theoretical results confirmed very well by experimental results and using two servers (i.e. the scalability factor of two) makes the recommender system to be scalable for up to 25,000 tweets. For SDRS since the algorithms are faster than CDRS, even the central variation shows scalability for 25,000 tweets but still adding more servers reduces the running time significantly. Although message passing time increases by having more servers, these graphs show with having 2 to 8 servers for 25,000 of tweets and adding more servers provides more improvement for both SDRS and CDRS.

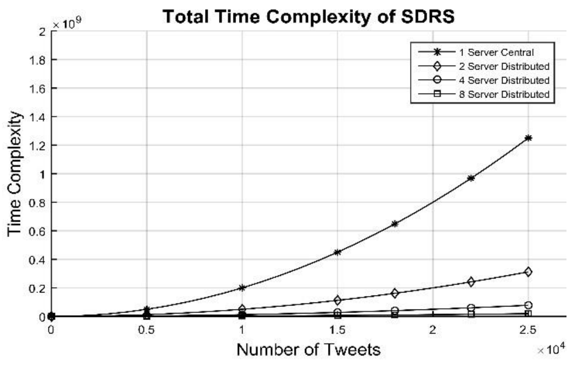

Figure 5.50: Shows the Numerical Analysis of total time using SDRS.

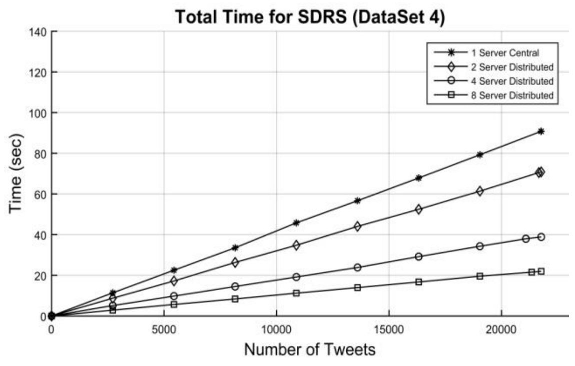

Figure 5.51: Shows the total running time using SDRS.

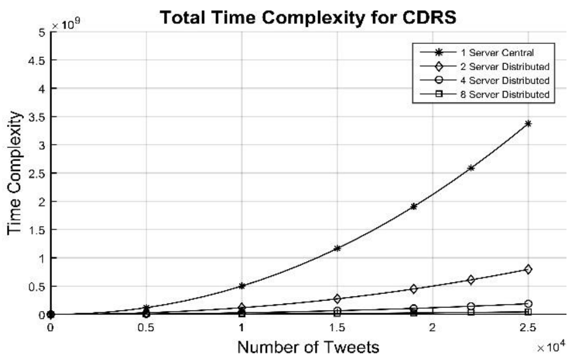

Figure 5.52: Shows the Numerical Analysis of total time using CDRS.

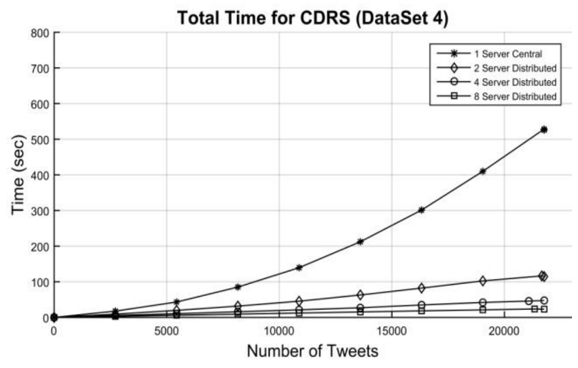

Figure 5.53: Shows the total running time using CDRS.

For the datasets which are greater than 25,000 tweets, gathering real data is very time consuming because the datasets are collected in different time frames for the specific twitter users and their followers, therefore capturing different types of tweets from different sets of users during these time frames is very time consuming. However, in theory the numerical analysis of larger datasets for example for 5,000,000 tweets shown below indicates the proposed method for providing scalability is still effective but the scalability factor should be different. The following graphs determine the scalability factor for both SDRS and CDRS with 5,000,000 tweets based on numerical analysis of time complexities. It is clear using 2 servers' causes non-linear relation whereas the scalability factor of 4 (using at least 4 servers) makes the systems more scalable.

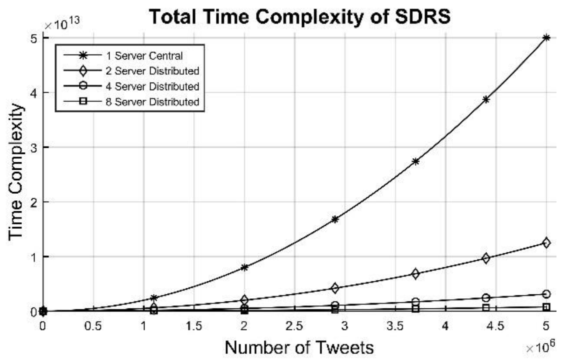

Figure 5.54: Shows the Numerical Analysis of total time for 5,000,000 tweets using SDRS.

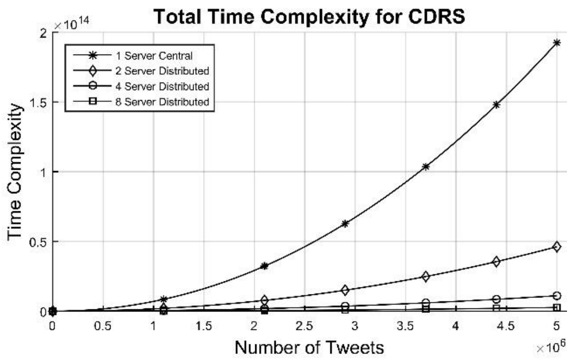

Figure 5.55: Shows the Numerical Analysis of total time for 5,000,000 tweets using CDRS.

Please note that the practical experiments with the smaller traces that is used, show that the running times of the recommender systems even with 2 servers are measured by seconds and using larger number of servers is preferred to have faster processing time. It means in practice for processing of hundreds of thousands of the tweets more servers than the scalability factor (that shows the scalable rate of growth) should be used in building online distributed recommender systems. Thus, in this thesis the scalability factor and the achieved improvement (shown as percentage of running time reduction) by using more servers is also discussed. Both Figure 5.4 and Figure 5.5 show significant improvement in running time speed will be achieved by adding more servers for processing of millions of tweets.

5.9. Summary

In this chapter, we have shown the performance results on five different datasets. The results for scalability and performance improvement are achieved by using both variations of distributed configurations. With regards to scalability for the algorithms with the time complexity of $O(N^2)$ the results for agent-based simulation confirmed the theoretical result which is using at least two servers for all tested datasets makes them linear and scalable. In SDRS, because disk latency contributes more than algorithms running time, for the lower amount of datasets SDRS is scalable even for central configuration because the distribution of data already gains lots of reduction in total running time of SDRS. However, CDRS clearly shows that using minimum two servers makes the system to be scalable for all datasets with regards to improvement. We found, for smaller datasets with the amount of less than 10,000 tweets the central variation of SDRS is scalable and using such small dataset for both systems, adding eight servers instead of four servers improves the running time marginally. For all of our used traces when using eight server distributed configuration for SDRS, performance improvement is in the range of 75.82% to 86.1%. While for CDRS the performance improvement result using eight servers is almost 93% to 98.35%. We also evaluated the accuracy for both SDRS and CDRS distributed recommender systems. In summary, using at least two servers makes both systems to be scalable, which is achieved by using distributed database and parallelization of algorithms. In the last part of this chapter, we used a data grouping technique by reducing the number of tweets which is used in similar works to reduce processing time. It is clear that when the less number of tweets is used by a recommender system, the accuracy will be reduced. In the conducted experiments we examined how our proposed technique performed against data grouping technique when the number of tweets is reduced assuming that we don’t want high accuracy. The results show our proposed technique gained similar benefits for small tweets and is superior for larger number of tweets. However, the reason for proposing to distribute and parallelization of data is to have maximum possible accuracy by processing more tweets. That is why in our proposed technique all tweets were used in each dataset and the proposed technique showed significant improvement in reducing the processing time with having high accuracy.

Chapter 6

Conclusions and Future Works

In this thesis, a novel framework of a distributed recommender system for social networks is proposed and effects of using multiple servers on scalability are discussed. First, in Chapter 1 the problem of having a scalable recommender system is formalized. Then in Chapter 2, a comprehensive discussion on prominent works on a social network has been presented. Second, in Chapter 3 we have discussed the proposed distributed recommender system in terms of algorithms and mathematical models, mainly with regards to scalability and distributed processing effectiveness. Also in Chapter 3 we have presented the theoretical results of time complexity to capture the scalability behaviours of the designed algorithms. Third, in Chapter 3 we have explained the models of two variations of proposed distributed recommender systems referred to as similarity-based distributed recommender system (i.e. SDRS) and cluster-based distributed recommender system (i.e. CDRS) for building their prototypes used in simulation. In Chapter 4, we have implemented CDRS and SDRS prototypes by using multi-agent based system. Fourth, in Chapter 4 a whole framework for providing multi-agent based simulation for testing the prototypes is discussed and developed. Finally, in Chapter 5 we have used CDRS and SDRS prototypes in several experiments with disk and memory based data and then compared them to their central recommender system counterparts. Running time and number of tweets are used as the main factors for finding performance metrics of the scalability factor and running time improvement of the proposed distributed recommender systems.

Also in Chapter 5, to demonstrate the effectiveness of the proposed distributed recommender system used in the social network environment, we have examined five different simple text based algorithms such as text cleaning, TF, TFIDF, cosine similarity and k-mean clustering algorithms. These algorithms are deployed on different agents for conducting multi-agent based simulation for finding the effectiveness of the distributed recommender system for social networks. Also in Chapter 5, we have used mean absolute error to find the accuracy of the proposed distributed recommender system and found that accuracy is not affected in the proposed distributed recommender systems. To test accuracy, in several experiments the proposed distributed recommender system generated lists of recommendations to the followees (to follow similar users), ranked based on their previous tweets and tweets of their own followees. These recommendation lists have been checked against the real selections of same users and based on that mean absolute error has been calculated to measure accuracy of the system.

The scalability factor is defined as the minimum number of servers that makes the system scalable. This is illustrated by the experiments shown in Chapter 5. The experiments are conducted on four different configurations, namely central, 2 server, 4 server, and 8 server configurations, which indicated that using at least two servers the proposed distributed system architecture become scalable, which confirmed the results achieved in

Chapter 3. For example, CDRS when running in the distributed environment with records (tweets) stored on the disk has achieved a reduction in running time in the range of 67% to 87% by using 2 servers for all the tested datasets respectively. In Chapter 3, theoretical results shows that by using 2 servers the system becomes scalable and can provide up to 75% reduction in running time.

In summary the multi-agent based simulation results achieved in Chapter 5 are in line with theoretical results achieved in Chapter 3. The results of both similarity and cluster-based distributed recommender systems have better performance of processing times compared to their central counterparts. For cluster-based recommender system the proposed solution is more promising in terms of efficiency. By using only two servers for up to 25,000 tweets used in each tested dataset the system become scalable and the running time is reduced by at least 67% in the CDRS. The proposed distributed recommender system can also be easily deployed in the cloud environment. It is a straightforward task to assign each distributed component to different servers running in the cloud environment.

The major contributions of the research are:

1. Development and implementation of a novel distributed recommender system based on multi-agent based system in two prototypes, referred to as SDRS and CDRS which significantly reduce the running time of the central system. Theses prototypes can be easily turned into real systems in future work.

2. Development of a framework for multi-agent based simulation including following components:
   2.1. The Twitter data crawler module based on multi-agent system, which is able to generate user network graph or links, and can download the required information from the website.
   2.2. Development of the Twitter simulation module, also based on a multi-agent based system. This agent based system is capable of tweeting "tweets" of the users under test. The tweets are first read from the stored corpus downloaded by the crawler. This environment can be used as a test bed for performing experiments where a researcher wants to mimic real time tweeting environment.

3. Performing simulation and determining the scalability factor which has confirmed theoretical results and indicated that by using a minimum of two servers in common recommender systems becomes scalable for processing up to 25,000 tweets.

In the future, we would like to continue our work on the distributed recommender system and its performance improvement by following new directions stated below:

1. To expand the role of the agents, we would like to develop more agents for the proposed distributed recommender system. These new agents will be capable of using different environments such as mobile based environment to provide recommendation to the users who are using cell phones or other mobile devices.

New agents for other text based algorithm in different configuration will be developed. By using MAS, each agent with different algorithms (stored as its behaviour) is able to connect with other agents.

2. We can also introduce a followee deletion pattern of the user. This may be useful for finding better recommendations, which can also improve overall performance of the recommendation process. If we consider followee deletion, the effect of adding wrong followee can be reduced on the recommendations.

3. It will be interesting to investigate our achieved results in other massive data processing systems such as gaming, etc. The agent based approach used in this research can be extended to build a distributed system that requires scalability to improve efficiency for using them in social networking websites.

4. Finally, we plan to design a graphical user interface for the system that regular users of the system will be capable to configure the agents and to generate other variations of the recommender systems similar to CDRS and SDRS without having the deep knowledge of agent programming.

Appendix I

List of mysql Tables

Table Name: **userTweets**

Field	**Data Type**
referenceUser	varchar (100)
tweetID	bigint(40) PK
created_at	Datetime
userID	varchar(45)
screen_name	varchar(45)
text	varchar(500)

Table Name: **userfollowers**

Field	**Data Type**
referenceUser	varchar(45)
cursorposition	varchar(60)
followerlist	varchar(500)
followername	varchar(45) PK
addedon	datetime

Table Name: **userfollowees**

Field	**Data Type**
referenceUser	varchar(45)
cursorposition	varchar(60)
followeelist	varchar(500)
followeename	varchar(45) PK
addedon	datetime

Table Name: **restrictedUser**

Field	**Data Type**
referenceUser	varchar(45)
followerorfollowee	varchar(45)
restrictedUser	varchar(45) PK
addedon	datetime

Appendix II

Prediction Accuracy for 2, 4, and 8 Server Configurations using CDRS

Table: Shows the Prediction Accuracy for 2 server configuration using CDRS for five datasets.

MAE	**Prediction Accuracy of CDRS**				
top k	**DataSet1**	**DataSet2**	**DataSet3**	**DataSet4**	**DataSet5**
3	0.10	0.93	0.76	1.00	0.86
10	0.00	0.71	0.65	1.00	0.86
20	0.00	0.47	0.64	0.00	0.86
30	0.00	0.28	0.45	0.00	0.79
40	0.00	0.00	0.14	0.00	0.54
50	0.00	0.00	0.01	0.00	0.00

Table: Shows the Prediction Accuracy for 4 server configuration using CDRS for five datasets.

MAE	**Prediction Accuracy of CDRS**				
top k	**DataSet1**	**DataSet2**	**DataSet3**	**DataSet4**	**DataSet5**
3	0.72	0.04	0.76	1.00	0.84
10	0.29	0.00	0.68	1.00	0.84
20	0.17	0.00	0.62	0.18	0.83
30	0.07	0.00	0.61	0.18	0.74
40	0.00	0.00	0.52	0.18	0.48
50	0.00	0.00	0.46	0.18	0.00

Table: Shows the Prediction Accuracy for 8 server configuration using CDRS for five datasets.

MAE	**Prediction Accuracy of CDRS**				
top k	**DataSet1**	**DataSet2**	**DataSet3**	**DataSet4**	**DataSet5**
3	0.88	0.46	0.77	1.00	0.84
10	0.57	0.10	0.65	0.96	0.83
20	0.31	0.00	0.60	0.11	0.81
30	0.04	0.00	0.54	0.11	0.74
40	0.00	0.00	0.45	0.11	0.68
50	0.00	0.00	0.39	0.11	0.23

References

1. Miller, B.N., Konstan, J.A., and Riedl, J., *PocketLens: Toward a personal recommender system.* ACM Trans. Inf. Syst., 2004. 22(3), pp. 437-476.

2. Cho, Y.H., Kim, J.K., and Kim, S.H., *A personalized recommender system based on web usage mining and decision tree induction.* Expert Systems with Applications, 2002. 23(3), pp. 329-342.

3. Ahmed, L. and Abhari, A. *Agent-Based Simulation of Twitter for Building Effective Recommender System*. In *17th Communications and Networking Simulation Symposium (CNS14) of SCS/ACM*. Tampa, Florida, Society for Computer Simulation International, 2014, pp. 266-272.

4. Ahmed, L. and Abhari, A. *Distributed Recommender System for Online Processing of Big Social Data*. In *Spring Simulation Multiconference (SpringSim'15 Poster Track), SCS/ACM*. Alexandria, VA, USA., 2015, pp. 699-700.

5. Ahmed, L. and Abhari, A., *A multi-agent-based simulator for a transmission control protocol/internet protocol network.* SIMULATION, 2014. 90(5), pp. 511-521.

6. Adomavicius, G. and Tuzhilin, A., *Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions.* Knowledge and Data Engineering, IEEE Transactions on, 2005. 17(6), pp. 734-749.

7. Candillier, L., Meyer, F., and Boullé, M., *Comparing State-of-the-Art Collaborative Filtering Systems*, In *Machine Learning and Data Mining in Pattern Recognition*, P. Perner, Editor 2007, Springer Berlin Heidelberg, pp. 548-562.

8. Mooney, R.J. and Roy, L. *Content-based book recommending using learning for text categorization*. In *Proceedings of the fifth ACM conference on Digital libraries*. San Antonio, Texas, USA, ACM, 2000, pp. 195-204.

9. Amazon, http://www.amazon.com, Last visited: Jan. 8, 2016.

10. Goldberg, D., et al., *Using collaborative filtering to weave an information tapestry.* Communications of The ACM, 1992. 35(12), pp. 61-70.

11. Park, D.H., et al., *A literature review and classification of recommender systems research.* Expert Systems with Applications, 2012. 39(11), pp. 10059-10072.

12. Bobadilla, J., Serradilla, F., and Bernal, J., *A new collaborative filtering metric that improves the behavior of recommender systems.* Knowledge-Based Systems, 2010. 23(6), pp. 520-528.

13. Luo, X., Xia, Y., and Zhu, Q., *Incremental Collaborative Filtering recommender based on Regularized Matrix Factorization.* Knowledge-Based Systems, 2012. 27, pp. 271-280.

14. Bobadilla, J. and Serradilla, F. *The effect of sparsity on collaborative filtering metrics*. In *Proceedings of the Twentieth Australasian Conference on Australasian Database - Volume 92*. Wellington, New Zealand, Australian Computer Society, Inc., 2009, pp. 9-18.

15. Pazzani, M., *A Framework for Collaborative, Content-Based and Demographic Filtering.* Artificial Intelligence Review, 1999. 13(5-6), pp. 393-408.

16. Krulwich, B., *Lifestyle finder: Intelligent user profiling using large-scale demographic data.* AI Magazine, 1997. 18(2), pp. 37-45.

17. Burke, R., *Hybrid Recommender Systems: Survey and Experiments.* User Modeling and User-Adapted Interaction, 2002. 12(4), pp. 331-370.

18. Porcel, C., et al., *A hybrid recommender system for the selective dissemination of research resources in a Technology Transfer Office.* Information Sciences, 2012. 184(1), pp. 1-19.

19. Vozalis, M.G. and Margaritis, K.G., *Using SVD and demographic data for the enhancement of generalized Collaborative Filtering.* Information Sciences, 2007. 177(15), pp. 3017-3037.

20. Barragáns-Martínez, A.B., et al., *A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition.* Information Sciences, 2010. 180(22), pp. 4290-4311.

21. Al-Shamri, M.Y.H. and Bharadwaj, K.K., *Fuzzy-genetic approach to recommender systems based on a novel hybrid user model.* Expert Syst. Appl., 2008. 35(3), pp. 1386-1399.

22. Jarvis, R.A. and Patrick, E.A., *Clustering using a similarity measure based on shared near neighbors.* Computers, IEEE Transactions on, 1973. 100(11), pp. 1025-1034.

23. Ortega, F., et al., *Improving collaborative filtering-based recommender systems results using Pareto dominance.* Information Sciences, 2013. 239, pp. 50-61.

24. Jae Yoon, C., et al. *An effective similarity metric for application traffic classification*. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*. 2010, pp. 286-292.

25. Bobadilla, J., Ortega, F., and Hernando, A., *A collaborative filtering similarity measure based on singularities.* Information Processing & Management, 2012. 48(2), pp. 204-217.

26. Bobadilla, J., et al., *Improving collaborative filtering recommender system results and performance using genetic algorithms.* Knowledge-Based Systems, 2011. 24(8), pp. 1310-1316.

27. Yuan, W., et al., *Improved trust-aware recommender system using small-worldness of trust networks.* Knowledge-Based Systems, 2010. 23(3), pp. 232-238.

28. Kwon, K., Cho, J., and Park, Y., *Multidimensional credibility model for neighbor selection in collaborative recommendation.* Expert Systems with Applications, 2009. 36(3, Part 2), pp. 7114-7122.

29. Jeong, B., Lee, J., and Cho, H., *User credit-based collaborative filtering.* Expert Systems with Applications, 2009. 36(3, Part 2), pp. 7309-7312.

30. Bobadilla, J., et al., *Recommender systems survey*. Knowledge-Based Systems, 2013. 46, pp. 109-132.

31. Antunes, P., et al., *Structuring dimensions for collaborative systems evaluation.* ACM Comput. Surv., 2012. 44(2), pp. 1-28.

32. Bengio, Y. and Grandvalet, Y., *No Unbiased Estimator of the Variance of K-Fold Cross-Validation.* J. Mach. Learn. Res., 2004. 5, pp. 1089-1105.

33. Aggarwal, C.C. and Reddy, C.K., *Data clustering: algorithms and applications* 2014, CRC Press.

34. Manning, C.D., Raghavan, P., and Schütze, H., *Introduction to Information Retrieval*. Computational Linguistics. Vol. 35. 2009, MIT Press, 307-309.

35. Salton, G., Wong, A., and Yang, C.S., *A vector space model for information retrieval.* Journal of the American Society for Information Science, 1975. 18(11), pp. 613-620.

36. Buckley, C., Singhal, A., and Mitra, M. *New retrieval approaches using SMART: TREC 4*. In *4th Text Retrieval conference (TREC-4)*. Gaithersburg, 1996.

37. Sebastiani, F., *Machine learning in automated text categorization.* ACM Comput. Surv., 2002. 34(1), pp. 1-47.

38. Jones, K.S. and Willett, P., *Readings in Information Retrieval, chapter 3,* 1997, Morgan Kaufmann Publishers, San Francisco, CA, pp. 305-312.

39. Salton, G. and Buckley, C., *Term-weighting approaches in automatic text retrieval.* Information Processing and Management, 1988. 24(5), pp. 513-523.

40. Jones, K.S., *A Statistical Interpretation of Term Specificity and its Application in Retrieval*. Journal of Documentation, 1972. 28(1), pp. 11-21.

41. Salton, G., *Automatic Information Organization and Retrieval*. 2nd ed. Computer Science Series 1968, McGraw-Hill.

42. Zipf, G., *Selective Studies and the Principle of Relative Frequencies in Language*, 1932, MIT Press.

43. Reed, J.W., et al. *TF-ICF: A New Term Weighting Scheme for Clustering Dynamic Data Streams*. In *Machine Learning and Applications, 2006. ICMLA '06. 5th International Conference on*. 2006, pp. 258-263.

44. Wordpress, https://wordpress.com, Last visited: Jan. 8, 2016.

45. Twitter, "Year in Review: Tweets per second." https://twitter.com, Last visited: Jan. 8, 2016.

46. FaceBook, https://www.facebook.com, Last visited: Jan. 8, 2016.

47. YouTube, https://www.youtube.com, Last visited: Jan. 8, 2016.

48. Flickr, https://www.flickr.com, Last visited: Jan. 8, 2016.

49. Digg, http://digg.com, Last visited: Jan. 8, 2016.

50. Delicious, https://delicious.com, Last visited: Jan. 8, 2016.

51. Guy, I., et al. *Social media recommendation based on people and tags*. In *Proceedings of the 33rd international ACM SIGIR conference on Research and*

development in information retrieval. Geneva, Switzerland, ACM, 2010, pp. 194-201.

52. Furl, http://www.furl.com, Last visited: Jan. 8, 2016.

53. CiteULike, http://www.citeulike.org, Last visited: Jan. 8, 2016.

54. Pinterest, https://www.pinterest.com, Last visited: Jan. 8, 2016.

55. Zhou, X., et al., *The state-of-the-art in personalized recommender systems for social networking.* Artificial Intelligence Review, 2012. 37(2), pp. 119-132.

56. Zhang, Y. and Pennacchiotti, M. *Recommending branded products from social media*. In *Proceedings of the 7th ACM conference on Recommender systems*. Hong Kong, China, ACM, 2013, pp. 77-84.

57. Sinha, R. and Swearingen, K., *Comparing Recommendations Made by Online Systems and Friends.* In Proceedings of the DELOS-NSF workshop on personalization and recommender systems in digital libraries, 2001.

58. Fabio Bellifemine, Giovanni Caire, and Greenwood, D., *Developing multi-agent systems with JADE* 2007, Chichester, West Sussex, John Wiley & Sons Ltd.

59. Kowalczyk, W. and Vlassis, N., *Newscast EM*, In *Advances in neural information processing systems 17*, Lawrence K. Saul, Yair Weiss, and L. Bottou, Editors. 2005, MIT Press, Cambridge, pp. 713-720.

60. Paskin, M., Guestrin, C., and McFadden, J. *A robust architecture for distributed inference in sensor networks*. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*. 2005, pp. 55-62.

61. Lesser, V., Ortiz, C.L., and Tambe, M., eds. *Distributed Sensor Networks: A Multiagent Perspective*. 2003, Kluwer, Dodrecht.

62. Drumond, L. and Girardi, R., *A multi-agent legal recommender system.* Artificial Intelligence and Law, 2008. 16(2), pp. 175 - 207.

63. Moin, S., Muhamamd, A., and Martinez-Enriquez, A.M. *Agent based mobile recommender system*. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2014 11th International Conference on*. 2014, pp. 1-6.

64. Chau, D.H., et al. *Parallel crawling for online social networks*. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 1283-1284.

65. George, T. and Merugu, S. *A scalable collaborative filtering framework based on co-clustering*. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 1-4.

66. Bedi, P., et al., *MARST: Multi-Agent Recommender System for e-Tourism Using Reputation Based Collaborative Filtering*, In *Databases in Networked Information Systems*, A. Madaan, S. Kikuchi, and S. Bhalla, Editors. 2014, Springer International Publishing, pp. 189-201.

67. Joshi, M. and Belsare, N. *BlogHarvest: Blog Mining and Search Framework*. In *Proc. of the Int'l Conf. on Management of Data COMAD*. 2006.

68. Ferreira, R., et al. *RetriBlog: a framework for creating blog crawlers*. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. Trento, Italy, ACM, 2012, pp. 696-701.

69. Golbeck, J. and Hendler, J., *Inferring binary trust relationships in Web-based social networks.* ACM Trans. Internet Technol., 2006. 6(4), pp. 497-529.

70. Quijano-Sanchez, L., et al., *Social factors in group recommender systems.* ACM Trans. Intell. Syst. Technol., 2013. 4(1), pp. 1-30.

71. Ma, H., King, I., and Lyu, M.R. *Learning to recommend with social trust ensemble*. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. Boston, MA, USA, ACM, 2009, pp. 203-210.

72. Massa, P. and Avesani, P. *Trust-aware recommender systems*. In *Proceedings of the 2007 ACM conference on Recommender systems*. Minneapolis, MN, USA, ACM, 2007, pp. 17-24.

73. Yuan, W., et al. *Efficient routing on finding recommenders for trust-aware recommender systems*. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*. Kuala Lumpur, Malaysia, ACM, 2012, pp. 1-6.

74. Ziegler, C.-N. and Golbeck, J., *Investigating Correlations of Trust and Interest Similarity - Do Birds of a Feather Really Flock Together?* Decis Support Systems, 2005, pp. 1-34.

75. Kim, Y., Park, Y., and Shim, K. *DIGTOBI: a recommendation system for Digg articles using probabilistic modeling*. In *Proceedings of the 22nd international conference on World Wide Web*. Rio de Janeiro, Brazil, International World Wide Web Conferences Steering Committee, 2013, pp. 691-702.

76. Hofmann, T. *Probabilistic latent semantic indexing*. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. Berkeley, California, USA, ACM, 1999, pp. 50-57.

77. Pazzani, M. and Billsus, D., *Learning and Revising User Profiles: The Identification of Interesting Web Sites.* Machine Learning, 1997. 27(3), pp. 313-331.

78. NetFlix, https://www.netflix.com, Last visited: Jan. 8, 2016.

79. Technorati, http://technorati.com, Last visited: Jan. 8, 2016.

80. Heymann, P., Koutrika, G., and Garcia-Molina, H. *Can social bookmarking improve web search?* In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. Palo Alto, California, USA, ACM, 2008, pp. 195-206.

81. Halpin, H., Robu, V., and Shepherd, H. *The complex dynamics of collaborative tagging*. In *Proceedings of the 16th international conference on World Wide Web*. Banff, Alberta, Canada, ACM, 2007, pp. 211-220.

82. Liang, H., et al. *Personalized Recommender Systems Integrating Social Tags and Item Taxonomy*. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*. IEEE Computer Society, 2009, pp. 540-547.

83. Guy, I., et al. *Harvesting with SONAR: the value of aggregating social network information*. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Florence, Italy, ACM, 2008, pp. 1017-1026.

84. Ronen, I., et al. *Social networks and discovery in the enterprise (SaND)*. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. Boston, MA, USA, ACM, 2009, pp. 836-836.

85. Carmel, D., et al. *Personalized social search based on the user's social network*. In *Proceedings of the 18th ACM conference on Information and knowledge management*. Hong Kong, China, ACM, 2009, pp. 1227-1236.

86. Kwak, H., et al. *What is Twitter, a social network or a news media?* In *Proceedings of the 19th international conference on World wide web*. Raleigh, North Carolina, USA, ACM, 2010, pp. 591-600.

87. Hannon, J., Bennett, M., and Smyth, B. *Recommending twitter users to follow using content and collaborative filtering approaches*. In *Proceedings of the fourth ACM conference on Recommender systems*. Barcelona, Spain, ACM, 2010, pp. 199-206.

88. Erra, U., et al., *Approximate TF–IDF based on topic extraction from massive message stream using the GPU.* Information Sciences, 2015. 292, pp. 143-161.

89. Petrović, S., Osborne, M., and Lavrenko, V. *Streaming first story detection with application to twitter*. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 181-189.

90. Indyk, P. and Motwani, R. *Approximate nearest neighbors: towards removing the curse of dimensionality*. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604-613.

91. Kamath, K.Y. and Caverlee, J. *Content-based crowd retrieval on the real-time web*. In *Proceedings of the 21st ACM international conference on information and knowledge management*. ACM, 2012, pp. 195-204.

92. Salton, G. and McGill, M.J., *Introduction to Modern Information Retrieval* 1983, New York, McGraw Hill Book Co.

93. Singhal, A., Buckley, C., and Mitra, M., *Pivoted document length normalization.* In proceedings 19th annual international ACM Special Interest Group on Information Retrieval conference on Research and Development in Information Retrieval, SIGIR'96, New York, NY,USA, 1996, pp. 21–29.

94. Tian Xia and Chai, Y., *An Improvement to TF-IDF - Term Distribution based Term Weight Algorithm.* Journal of Software, 2011. 6(3).

95. Manning, C., Raghavan, P., and Schütze, H., *Introduction to Information Retrieval* 2008, Cambridge University Press.

96. Java Agent Development Framework, http://jade.tilab.com, Last visited: Jan. 8, 2016.

97. Niazi, M.A., Hussain, A., and Kolberg, M. *Verification and Validation of Agent Based Simulations using the VOMAS (Virtual Overlay Multi-agent System) approach*. In *Proceedings of the Third Workshop on Multi-Agent Systems and Simulation '09 (MASS '09)*. Torino, Italy, 2009.

98. Segrouchni, A.E.F., et al., eds. *Multi-Agent Programming Languages, Tools, and Applications*. 2009, Springer, New York.