Optimizing Association Retrieval Method based Recommender Systems using Multithreading

A Report on 7th Semester B. Tech Project pursued
under the guidance of
Mr. Prabhakar Sarma Neog
(Assistant Professor, Dept of CSE, NIT Silchar)

Members in Group:
Shival Gupta (11-1-5-002)
Vikas Singh (11-1-5-097)
Shishuvendra Tiwari (11-1-5-004)
Pramod Prasad (11-1-5-092)
Pabitra Pegu (11-1-5-091)
Manjil Brahma (11-1-5-065)

Class of 2015
Department of Computer Science and Engineering
National Institute of Technology Silchar – 788010

Table of Contents

Certificate

This is to certify that the Project Report entitled, “Optimization of Association Retrieval Method based Recommender Systems using Multithreading” submitted by **Shival Gupta, Vikas Singh, Shishuvendra Tiwari, Pramod Prasad, Pabitra Pegu, Manjil Brahma** to National Institute of Technology, Silchar, India, is a record of bonafide Project work carried out by him/her under my/our supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the Institute.

Mr Prabhakar Sarma Neog
Assistant Professor
Department of Computer Science and Engineering
National Institute of Technology Silchar

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Shival Gupta
11-1-5-002

Vikas Singh
11-1-5-097

Shishuvendra Tiwari
11-1-5-004

Pramod Prasad
11-1-5-092

Pabitra Pegu
11-1-5-091

Manjil Brahma
11-1-5-065

Abstract

Recommender systems are being widely applied in many fields, such as e-commerce etc, to provide products, services and information to potential customers. Collaborative filtering as the most successful approach, which recommends contents to the current customers mainly is based on the past transactions and feedback of the similar customer.

However, it is difficult to distinguish the similar interests between customers because the sparsity problem is caused by the insufficient number of the transactions and feedback data, which confined the usability of the collaborative filtering. This paper describes the Association Retrieval Technique as a method for developing Collaborative Filters. This method has the advantage of able to give highly accurate recommendations even when the user-item matrix is sparse beacuse it computes both direct as well as indirect similarity. But the problem with Association Retrieval Technique is that it has high time complexity because it involves matrix multiplications (O(n3)).

This paper aims to reduce the overall execution time of Association Retrieval Technique using multithreading. Multithreading involves executing various segments of a program in parallel. Its involves creating threads which is a separate path of execution. The paper aims to study effects of applying multithreading on various segments of the algorithm in terms of the time of execution.

1. Introduction and Literature Survey

1.1 Overview of Recommender Systems

Recommender systems or recommendation systems (sometimes replacing "system" with a synonym such as platform or engine) are a subclass of information filtering system that seek to predict the 'rating' or 'preference' that user would give to an item.

Recommender systems have become extremely common in recent years, and are applied in a variety of applications. The most popular ones are probably movies, music, news, books, research articles, search queries, social tags, and products in general. However, there are also recommender systems for experts, jokes, restaurants, financial services, life insurance, persons (online dating), and Twitter followers.

Recommender systems typically produce a list of recommendations in one of two ways - through collaborative or content-based filtering. Collaborative filtering approaches building a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users; then use that model to predict items (or ratings for items) that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties. These approaches are often combined (see Hybrid Recommender Systems).

The differences between collaborative and content-based filtering can be demonstrated by comparing two popular music recommender systems - Last.fm and Pandora Radio.

Pandora uses the properties of a song or artist (a subset of the 400 attributes provided by the Music Genome Project) in order to seed a "station" that plays music with similar properties. User feedback is used to refine the station's results, deemphasizing certain attributes when a user "dislikes" a particular song and emphasizing other attributes when a user "likes" a song. This is an example of a content-based approach.

Last.fm creates a "station" of recommended songs by observing what bands and individual tracks that the user has listened to on a regular basis and comparing those against the listening behavior of other users. Last.fm will play tracks that do not appear in the user's library, but are often played by

other users with similar interests. As this approach leverages the behavior of users, it is an example of a collaborative filtering technique.
Each type of system has its own strengths and weaknesses. In the above example, Last.fm requires a large amount of information on a user in order to make accurate recommendations. This is an example of the cold start problem, and is common in collaborative filtering systems. While Pandora needs very little information to get started, it is far more limited in scope (for example, it can only make recommendations that are similar to the original seed).

Recommender systems are a useful alternative to search algorithms since they help users discover items they might not have found by themselves. Interestingly enough, recommender systems are often implemented using search engines indexing non-traditional data.

Montaner provides the first overview of recommender systems, from an intelligent agents perspective. Adomavicius provides a new overview of recommender systems. Herlocker provides an additional overview of evaluation techniques for recommender systems, and Beel et al. discuss the problems of offline evaluations. They also provide a literature survey on research paper recommender systems.

Recommender system is an active research area in the data mining and machine learning areas.

1.2 Approaches of concept

1.2.1 Content based Filtering

Another common approach when designing recommender systems is content-based filtering. Content-based filtering methods are based on a description of the item and a profile of the user's preference. In a content-based recommender system, keywords are used to describe the items; beside, a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

To abstract the features of the items in the system, an item presentation algorithm is applied. A widely used algorithm is the tf–idf representation (also called vector space representation).

To create user profile, the system mostly focuses on two types of information: 1. A model of the user's preference. 2. A history of the user's interaction with the recommender system.

Basically, these methods use an item profile (i.e. a set of discrete attributes and features) characterizing the item within the system. The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. Simple approaches use the average values of the rated item vector while other sophisticated methods use machine learning techniques such as Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks in order to estimate the probability that the user is going to like the item.

Direct feedback from a user, usually in the form of a like or dislike button, can be used to assign higher or lower weights on the importance of certain attributes (using Rocchio Classification or other similar techniques).

A key issue with content-based filtering is whether the system is able to learn user preferences from user's actions regarding one content source and use them across other content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but it's much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing.

As previously detailed, Pandora Radio is a popular example of a content-based recommender system that plays music with similar characteristics to that of a song provided by the user as an initial seed. There are also a large number of content-based recommender systems aimed at providing movie recommendations, a few such examples include Rotten Tomatoes, Internet Movie Database, Jinni, Rovi Corporation, Jaman and See This Next.

1.2.2 Collaborative Filtering

Collaborative filtering (CF) is a technique used by some recommender systems. Collaborative filtering has two senses, a narrow one and a more general one. In general, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among

multiple agents, viewpoints, data sources, etc. Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc. The remainder of this discussion focuses on collaborative filtering for user data, although some of the methods and approaches may apply to the other major applications as well.

In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a person chosen randomly. For example, a collaborative filtering recommendation system for television tastes could make predictions about which television show a user should like given a partial list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user, but use information gleaned from many users. This differs from the simpler approach of giving an average (non-specific) score for each item of interest, for example based on its number of votes.

1.2.3 Types of Collaborative Filtering

Memory Based

This mechanism uses user rating data to compute similarity between users or items. This is used for making recommendations. This was the earlier mechanism and is used in many commercial systems. It is easy to implement and is effective. Typical examples of this mechanism are neighbourhood based CF and item-based/user-based top-N recommendations.

Model Based

Models are developed using data mining, machine learning algorithms to find patterns based on training data. These are used to make predictions for real data. There are many model-based CF algorithms. These include Bayesian networks, clustering models, latent semantic models such as singular value decomposition, probabilistic latent semantic analysis, Multiple Multiplicative Factor, Latent Dirichlet allocation and markov decision process based models.

This approach has a more holistic goal to uncover latent factors that explain observed ratings. Most of the models are based on creating a classification or clustering technique to identify the user based

on the test set. The number of the parameters can be reduced based on types of principal component analysis.

There are several advantages with this paradigm. It handles the sparsity better than memory based ones. This helps with scalability with large data sets. It improves the prediction performance. It gives an intuitive rationale for the recommendations.

The disadvantages with this approach are in the expensive model building. One needs to have a tradeoff between prediction performance and scalability. One can lose useful information due to reduction models. A number of models have difficulty explaining the predictions.

1.2.4 Challenges to Collaborative Filtering

Data sparsity

In practice, many commercial recommender systems are based on large datasets. As a result, the user-item matrix used for collaborative filtering could be extremely large and sparse, which brings about the challenges in the performances of the recommendation.

One typical problem caused by the data sparsity is the cold start problem. As collaborative filtering methods recommend items based on users' past preferences, new users will need to rate sufficient number of items to enable the system to capture their preferences accurately and thus provides reliable recommendations.

Similarly, new items also have the same problem. When new items are added to system, they need to be rated by substantial number of users before they could be recommended to users who have similar tastes with the ones rated them. The new item problem does not limit the content-based recommendation, because the recommendation of an item is based on its discrete set of descriptive qualities rather than its ratings.

Scalability

As the numbers of users and items grow, traditional CF algorithms will suffer serious scalability problems. For example, with tens of millions of customers O(M) and millions of items O(N), a CF algorithm with the complexity of n is already too large. As well, many systems need to react immediately to online requirements and make recommendations for all users regardless of their purchases and ratings history, which demands a higher scalability of a CF system. Large web companies such as Twitter use clusters of machines to scale recommendations for their millions of users, with most computations happening in very large memory machines.

Synonyms

Synonyms refers to the tendency of a number of the same or very similar items to have different names or entries. Most recommender systems are unable to discover this latent association and thus treat these products differently.

For example, the seemingly different items “children movie” and “children film” are actually referring to the same item. Indeed, the degree of variability in descriptive term usage is greater than commonly suspected. The prevalence of synonyms decreases the recommendation performance of CF systems. Topic Modeling (like the Latent Dirichlet Allocation technique) could solve this by grouping different words belonging to the same topic.

Grey sheep

Grey sheep refers to the users whose opinions do not consistently agree or disagree with any group of people and thus do not benefit from collaborative filtering. Black sheep are the opposite group whose idiosyncratic tastes make recommendations nearly impossible. Although this is a failure of the recommender system, non-electronic recommenders also have great problems in these cases, so black sheep is an acceptable failure.

Shilling attacks

In a recommendation system where everyone can give the ratings, people may give lots of positive ratings for their own items and negative ratings for their competitors. It is often necessary for the collaborative filtering systems to introduce precautions to discourage such kind of manipulations.

Diversity and the Long Tail

Collaborative filters are expected to increase diversity because they help us discover new products. Some algorithms, however, may unintentionally do the opposite. Because collaborative filters recommend products based on past sales or ratings, they cannot usually recommend products with limited historical data. This can create a rich-get-richer effect for popular products, akin to positive feedback. This bias toward popularity can prevent what are otherwise better consumer-product matches. A Wharton study details this phenomenon along with several ideas that may promote diversity and the "long tail."

2. Theory of Association Retrieval Method

2.1 Introduction and Sparsity Problem

One of the major problems faced by traditional collaborative filtering methods is called the sparsity problem. This process can be overcome to a certain admissible degree using a novel method known as association retrieval method.

Along with the rapidly development of the Internet, the number of the servers connected to Internet and the Webs on WWW show a trend of exponential growth. The rapidly development of the Internet present a mass of information to us at the same time, for example, there are tens of thousands movies in Netflix, millions of books in Amazon, more than 10 billion page collection in Del.icio.us, so much information, not to mention find some interesting contents, it is impossible that to gave all of information the once-over. The traditional search algorithm only presents the same ordered results to all of users; can not to provide different service to different users according to their different interests The information explosion reduced the use ratio of the information, this phenomenon is called information overload. Personalized recommendation, included personalized search, has been thought as one of the most effective tools to resolve the problem of informationoverload. Radically, the recommendation problem is to substitute user to evaluate the strange products, which include books, movies, CD, web and so on, it is a process from know to unknown.

Recommendation as a social process plays an important role in many applications for consumers, because it is overly expensive for every consumer to learn about all possible alternatives independently. Depending on the specific application setting, a consumer might be a buyer, an information seeker, or an organization searching for certain expertise.

Until 1990s, personalized recommendation research as an independent concept be advanced. It rapidly development origin from the web2.0‘s maturity, which make the user become a participant from browser. In an actually recommender system, there are tens of thousands, or even more than one millions products need to be recommended, for instance, Amazon, eBay, Youtube, etc, also there are huge users. Accurate and high-performance recommender system can mine the potential propensity to consume of the user and provide personalized services for users. In the increasingly fierce competitive environment, personalized recommendation system is not just business marketing means, more importantly, it can improve the user‘s loyalty and prevent the loss of users.

A recommender system is compose of three parts: action recorder module collect the user‘s information, model analysis module analyze the user‘s preference and recommendation algorithm module, thereinto, the recommendation algorithm module is the most core part of the recommendation system. At present, recommendation algorithm mainly includes collaborative filtering algorithm, content-based algorithm, the bipartite relationship graph recommendation algorithm based on user-product and hybrid recommendation algorithm. This project focus on the sparsity and precision problem, compute the similarly matrix through the relatively distance between the user‘s rating and use the association retrieval technology to realize a new collaborative filtering approach. The remainder of the project is organized as follows. Further section of this report surveys existing work on collaborative filtering and discusses the sparsity problem in detail. It introduces associative retrieval and summarizes our associative retrieval-based approach to dealing with the sparsity problem and improve the quality of the recommendation. Also presents an experimental study and the experimental data analysis. Finally concludes the article by summarizing our research contributions and pointing out future directions.

Sparsity Problem

In collaborative filtering systems, users or consumers are typically represented by the items they have purchased or rated. For instance, in an online cinema have 3 million movies; each consumer is represented by a Boolean feature vector of 3 million elements. The value for each element is determined by whether this consumer has viewed the corresponding movie in the past time. Typically the value of 1 to 5 indicates that such a view had occurred and 0 indicates that no such event has occurred. When multiple consumers are concerned, a matrix composed of all vectors representing these consumers can be used to capture past view events. We call this matrix the consumer–product interaction matrix. In this article, we use C to denote the set of consumers and I to represent the set of items. We represent the consumer–product interaction matrix by a $|C|\times|I|$ matrix $R = (r_{ij})$, such that

$$r_{ij} = \begin{cases} k & k = 1...5, \text{if user } i \text{ rated item } j \\ 0, & \text{otherwise} \end{cases}$$

In many large-scale applications, both the number of items and the number of consumers are large. In such cases, even when many events have been recorded, the consumer–product interaction matrix can still be extremely sparse, that is, there are very few elements in R whose value is not 0.

This problem, commonly referred to as the sparsity problem, has a major negative impact on the effectiveness of a collaborative filtering approach. Because of sparsity, it is highly probable that the similarity (or correlation) between two given users is zero, rendering collaborative filtering useless. Even for pairs of users that are positively correlated, such correlation measures may not be reliable.

The cold-start problem further illustrates the importance of addressing the sparsity problem. The coldstart problem refers to the situation in which a new user or item has just entered the system. Collaborative filtering cannot generate useful recommendations for the new user because of the lack of sufficient previous ratings or purchases. Similarly, when a new item enters the system, it is unlikely that collaborative filtering systems will recommend it to many users because very few users have yet rated or purchased this item. Conceptually, the cold-start problem can be viewed as a special instance of the sparsity problem, where most elements in certain rows or columns of the consumer–product interaction matrix A are 0.

Many researchers have attempted to alleviate the sparsity problem. The author proposed an itembased approach to addressing both the scalability and sparsity problems. Another proposed approach, dimensionality reduction, aims to reduce the dimensionality of the consumer–product interaction matrix directly. A simple strategy to reduce the dimensionality is to form clusters of items or users and then use these clusters as basic units in the prediction.

More advanced techniques can be applied to achieve dimensionality reduction. Essentially, dimensionality reduction approaches deal with the sparsity problem by generating a denser user-item interaction matrix that considers only the most relevant users and items. Predictions are then made using this reduced matrix.

Empirical studies indicate that dimensionality reduction can improve recommendation quality significantly in some applications, but performs poorly in others, the potentially useful information might be lost during this reduction process. Researchers have also attempted to combine collaborative filtering with content-based recommendation approaches to alleviate the sparsity problem. In addition to user-item interactions, such techniques also consider similarities between items derived from their content, which allow them to make more accurate predictions. However, the hybrid approach requires additional information regarding the products and a metric to compute meaningful similarities among them. In practice, such product information may be difficult or expensive to acquire and a related similarity metric may not be readily available.

Another category of methods consider the data as a bipartite graph where nodes represent the users

and items, and an edge (i, j) exists between a user i and an item j if i has rated j. Moreover, edge (i, j) is given a weight corresponding to the rating given by i to j. These methods then derive global similarities between users or items using graph theoretic measures. For instance, one such method computes similarities between two users as the average commute time between their respective nodes in a random-walk of the graph. Other graph theoretic measures were also investigated, such as the minimal hop distance between nodes of the graph, and the spread activation of the nodes in the graph. The main drawback of these approaches is that there is often no good interpretation of the similarity measures in the context of the prediction problem.

Our research focuses on developing a computational approach to exploring transitive between users to address the sparsity problem and improving the accurate in the context of collaborative filtering.

2.2 Collaborative Filtering based on Association Retrieval

2.2.1 Finding relationship between users by association

Firstly, we supposed that C = {c_1, c_2, c_3) represent a user set which includes 3 users I = {i_1, i_2, i_3, i_4} represents a movie set which includes 4 movies, represent a user‘s rating matrix which includes elements R = |C|x|I| represent a user ratings matrix which includes 12 elements.

$$R=\begin{bmatrix}0 & 3 & 0 & 4\\ 0 & 2 & 3 & 5\\ 4 & 0 & 3 & 0\end{bmatrix}$$

The rows represent the user, the columns represents the movie, for example, the first row represents the user c_1 viewed the movies i_2 and i_4, the rating is 3 and 4 respectively

$$B=\begin{bmatrix}0 & 1 & 0 & 1\\ 0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\end{bmatrix}$$

From the second line in the matrix B, we can know that the user c_2 viewed the movie i_2, i_3 and i_4. It is easy to find that the user c_1 and c_2 viewed the movie i_2 and i_4 from matrix R and B. According to similarity theory, we can ascertain that the user c_1 is similarity with the user $c_{2,}$ so the movie i_3 can be recommended to the user c_1 through the user c2, but the movie i_1 cannot be recommended to c_1 forever. However, the above example only has 4 movies. At present, the online movie provider more than millions movies, the ―dark information will appear if only through the direct similarity users to recommend, some of movies will cannot be recommended to some of users, the requirements of the user cannot be satisfied.

According to the association retrieval theory, users as a set of nodes, the products as a set of nodes, we use the bipartite graph to express the matrix B, as shown in Fig1.

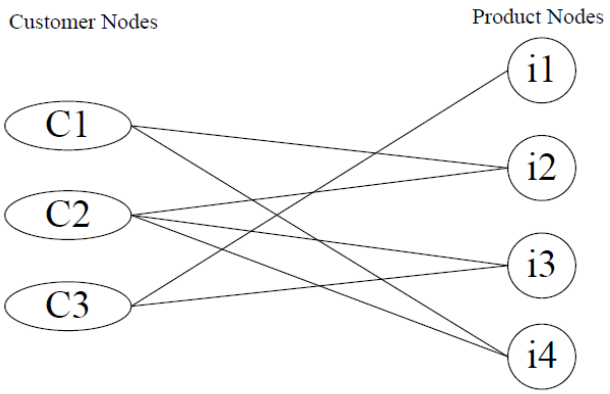


Figure 1. Transitive associations in collaborative filtering.

Accordering to Fig 1, the length of the association path is assumed to be 3, there are c_1-i_2-c_2-i_3 and c_1-i_4-c_2-i_3 two paths, the movie i_3 is recommended to the user c_1, but there is not a path whose length is 3 between i_1 and c_1, so i_1 will not be recommended to the user c_1. If the length of the path is extended to 5, we can find that the movie i_1 can be recommended to the user c_1 through the path c_1-i_2-c_2-i_3-c_3-i_1 and c_1-i_4-c_2-i_3-c_3-i_1.

2.2.2 Computing the direct similarity matrix.

In the computing of the direct similarity matrix, we do not use the Pearson-correlation and cosine of the angle. Through the research we find that whatever the user rating is high or low after the user viewed the movie, to some extent, which express some of similarity between users both on the personal preferences and the preference of ratings. For example, in the matrix R, the user c_1 and c_2 rated i_2 and i_4, the rating value of the c_1 is 3 and 4, the rating value of the c_2 is 2 and 5, we can use formula (3) to compute the rating similarity degree between c_1 and c_2 for the same movie $sim\_2_{(12)}$ = 0.8 and $sim\_4_{(12)}$ = 0.8.

$$sim_{K(ij)} = \begin{cases} \frac{max(R)-abs(r_{ik}-r_{jk})}{max(R)}, if\ r_{ik}\ and\ r_{jk}\ not\ equal\ 0, \\ 0 \qquad\qquad\qquad\qquad Otherwise \end{cases}$$

max is the maximum value function; abs is the absolute value function; R represents the value set of the rating, such as R={0,1,2,3,4,5}; r_{ik} , the value of the user i rate product k. Formula (4) was used to compute the user similarity a_{ij} between i and j after get the rating similarity degree.

$$a_{ij} = \begin{cases} \frac{\sum_{k=1}^{m} sim\_K_{(ij)}}{m}, & if\ i\ not\ equal\ j, \\ 1 \qquad , & if\ i\ equal\ j \end{cases}$$

Note that m, the number of the products. We use the rating matrix R as an example, the user similarity a_{ij} = (0.8+0.8)/4 = 0.4 , according to this method, we can get the user similarity matrix A_{sim} as follows:

$$A_{similarity} = \begin{bmatrix} 1 & 0.4 & 0 \\ 0.4 & 1 & 0.25 \\ 0 & 0.25 & 1 \end{bmatrix}$$

2.2.3 Computing Recommendation Matrix

We use the data the section 3.2 provided to recommend for the user c1. When M=3, we can find that c1 has two recommendation path c_1-i_2-c_2-i_3 and c_1-i_4-c_2-i3 from the data; the similarity between c_1 and c_2 is 0.4 from the similarity matrix in section 3.3, the weight of the path is 0.4; so we get the correlation degree of the i_3 is 0.4 x 2 = 0.8; Because c_1 and c_2 have the highest similarity, the rating value of the c_2 for i_3 is 3, so the recommendation value is 0.8 x 3 = 2.4 . When M=5, there are two recommendation path c_1-i_2-c_2-i_3-c_3-i_1 and c_1-i_4-c_2-i_3-c_3-i_1, the weight is a_{12} x a_{23} = 0.4 x 0.25 = 0.1 , the value of the correlation degree is 0.1 x 2 = 0.2 , the rating value of the c_3 for i_1 is 4, so the recommendation value is 0.2 x 4 = 0.8.

The recommendation matrix Matrix_R was defined in

$$Matrix\_R^M = \begin{cases} R, & M = 1, \\ B \cdot B^T \cdot A_{sim} \cdot Matrix\_R^{M-2}, & M = 3,5,7 \ldots \end{cases}$$

Note that R, the rating matrix, is the similarity matrix, B is the marked matrix. Using the data, we

get the recommendation matrix Matrix_R^3 and Matrix_R^5 through formula above when M=3 and M=5.

$$\text{Matrix\_}R^3 = \begin{bmatrix} 0.0000 & 7.6000 & \mathbf{2.4000} & 12.0000 \\ 1.0000 & 8.4000 & 9.7500 & 18.2000 \\ 8.0000 & \mathbf{0.5000} & 6.7500 & \mathbf{1.2500} \end{bmatrix}$$

$$\text{Matrix\_}R^5 = \begin{bmatrix} \mathbf{0.8000} & 21.9200 & 12.6000 & 38.5600 \\ 5.0000 & 31.4050 & 32.8575 & 64.5125 \\ 16.2500 & 3.1000 & 15.9375 & 7.0500 \end{bmatrix}$$

2.2.4 Algorithm

Collaborative algorithm based on association retrieval.

Input：user rating matrix R , the length of path M

Output：Recommendation matrix

Step 1: Matrix B = Matrix R, If r_{ij} not equal 0 then b_{ij} = 1 for each r_{ij} .

Step 2: Set the iteration variable N=1.

Step 3: Original recommendation matrix Matrix_R^N = R

Step 4: Compute the direct similarity matrix A_{sim} according to formulae given earlier.

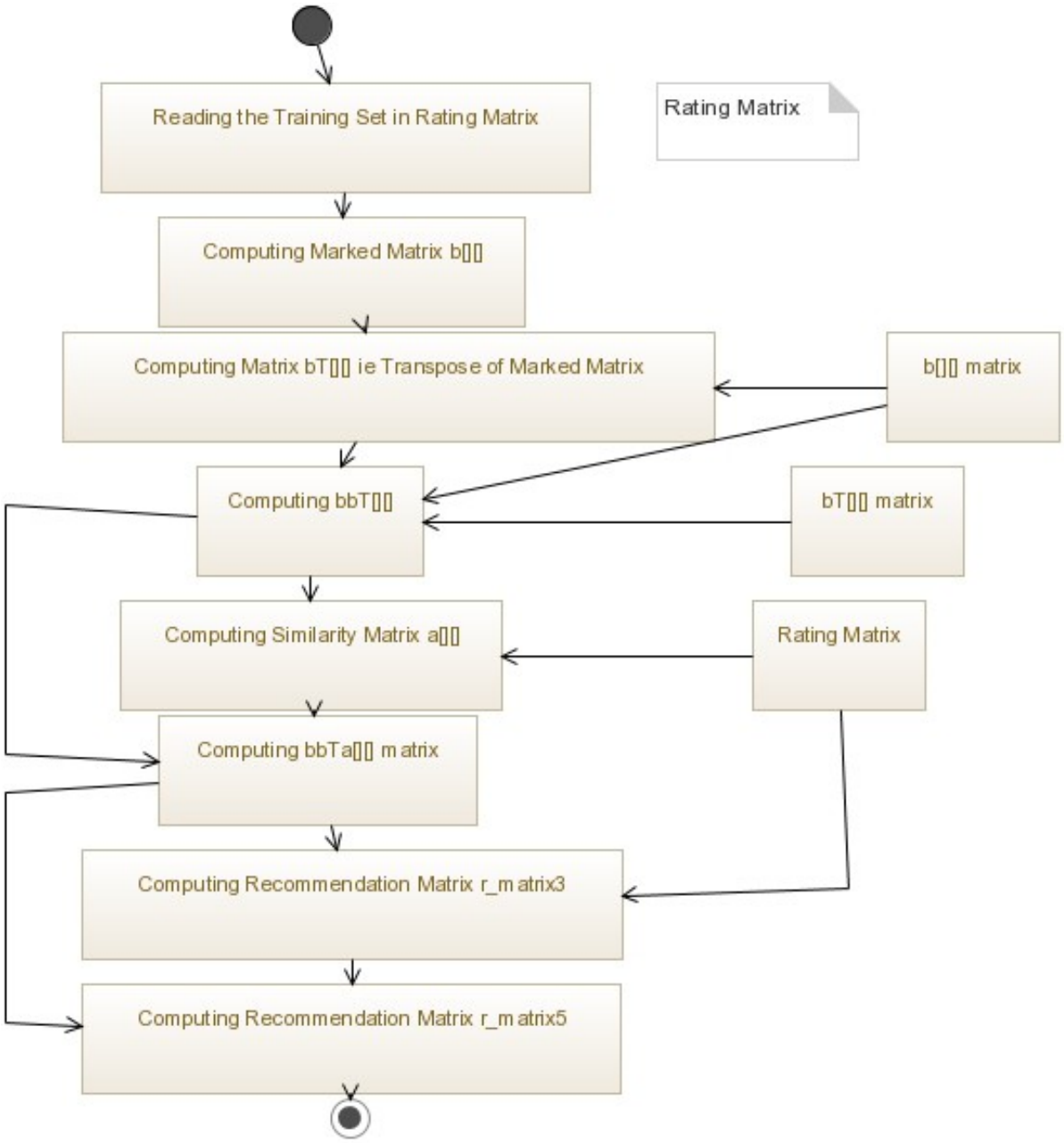
Step 5: Compute the transpose B^T

Step 6: Compute the matrix Matrix_R^M according to using formula in the last section.

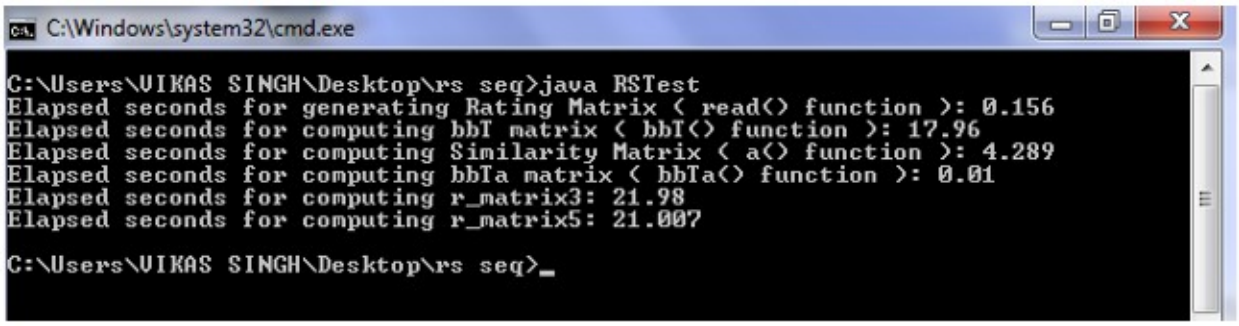
Step 7: If N+2 less than M then N=N+2, goto Step 3 until N larger than M.

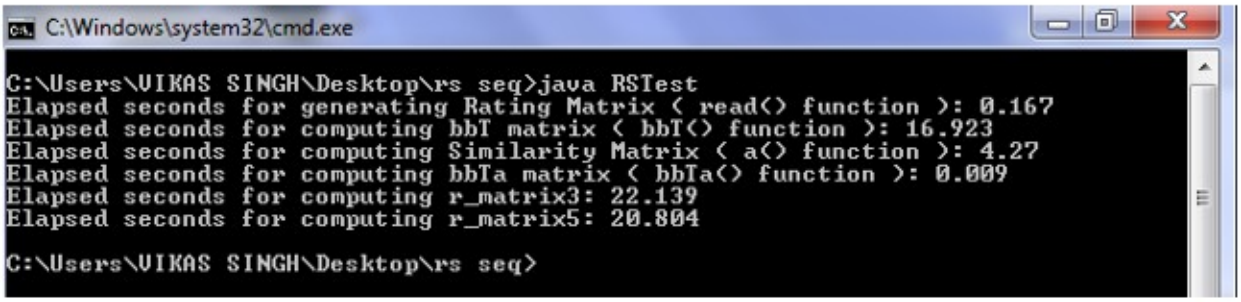
3. Analyzing CFAR algorithm (non-multithreaded) with real data

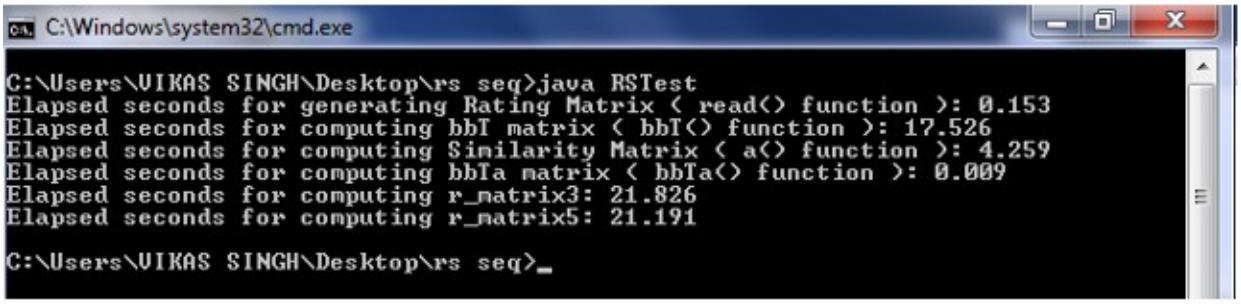
3.1 Activity Diagram of program

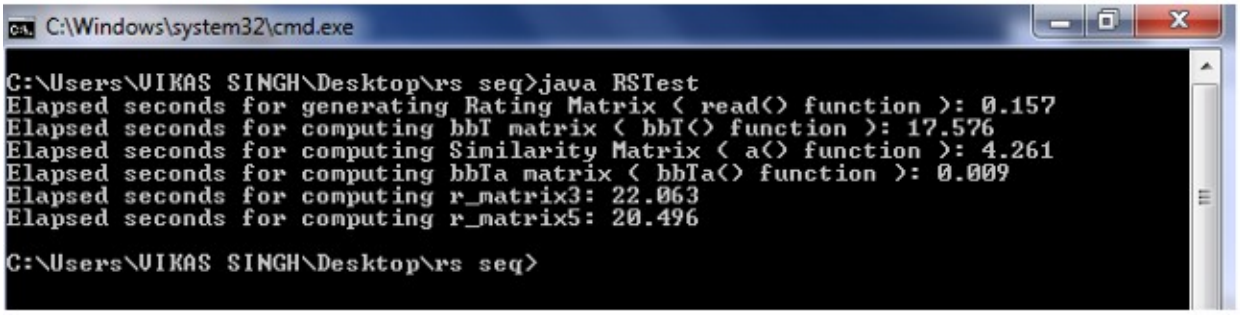


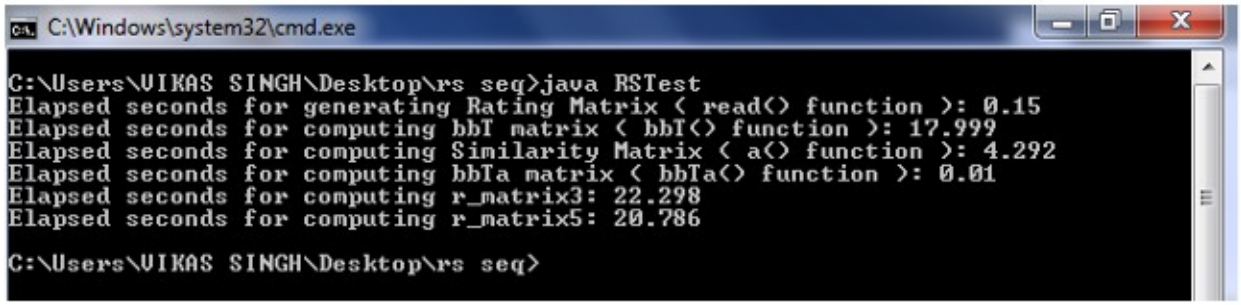
3.2 Outputs


C:\Windows\system32\cmd.exe
C:\Users\VIKAS SINGH\Desktop\rs seq>java RSTest
Elapsed seconds for generating Rating Matrix (read() function): 0.156
Elapsed seconds for computing bbT matrix (bbT() function): 17.96
Elapsed seconds for computing Similarity Matrix (a() function): 4.289
Elapsed seconds for computing bbTa matrix (bbTa() function): 0.01
Elapsed seconds for computing r_matrix3: 21.98
Elapsed seconds for computing r_matrix5: 21.007
C:\Users\VIKAS SINGH\Desktop\rs seq>


C:\Windows\system32\cmd.exe
C:\Users\VIKAS SINGH\Desktop\rs seq>java RSTest
Elapsed seconds for generating Rating Matrix (read() function): 0.167
Elapsed seconds for computing bbT matrix (bbT() function): 16.923
Elapsed seconds for computing Similarity Matrix (a() function): 4.27
Elapsed seconds for computing bbTa matrix (bbTa() function): 0.009
Elapsed seconds for computing r_matrix3: 22.139
Elapsed seconds for computing r_matrix5: 20.804
C:\Users\VIKAS SINGH\Desktop\rs seq>


C:\Windows\system32\cmd.exe
C:\Users\VIKAS SINGH\Desktop\rs seq>java RSTest
Elapsed seconds for generating Rating Matrix (read() function): 0.153
Elapsed seconds for computing bbT matrix (bbT() function): 17.526
Elapsed seconds for computing Similarity Matrix (a() function): 4.259
Elapsed seconds for computing bbTa matrix (bbTa() function): 0.009
Elapsed seconds for computing r_matrix3: 21.826
Elapsed seconds for computing r_matrix5: 21.191
C:\Users\VIKAS SINGH\Desktop\rs seq>


C:\Windows\system32\cmd.exe
C:\Users\VIKAS SINGH\Desktop\rs seq>java RSTest
Elapsed seconds for generating Rating Matrix (read() function): 0.157
Elapsed seconds for computing bbT matrix (bbT() function): 17.576
Elapsed seconds for computing Similarity Matrix (a() function): 4.261
Elapsed seconds for computing bbTa matrix (bbTa() function): 0.009
Elapsed seconds for computing r_matrix3: 22.063
Elapsed seconds for computing r_matrix5: 20.496
C:\Users\VIKAS SINGH\Desktop\rs seq>


C:\Windows\system32\cmd.exe
C:\Users\VIKAS SINGH\Desktop\rs seq>java RSTest
Elapsed seconds for generating Rating Matrix (read() function): 0.15
Elapsed seconds for computing bbT matrix (bbT() function): 17.999
Elapsed seconds for computing Similarity Matrix (a() function): 4.292
Elapsed seconds for computing bbTa matrix (bbTa() function): 0.01
Elapsed seconds for computing r_matrix3: 22.298
Elapsed seconds for computing r_matrix5: 20.786
C:\Users\VIKAS SINGH\Desktop\rs seq>

Source code can be found in Appendix I. 5 runs of the same program were done.

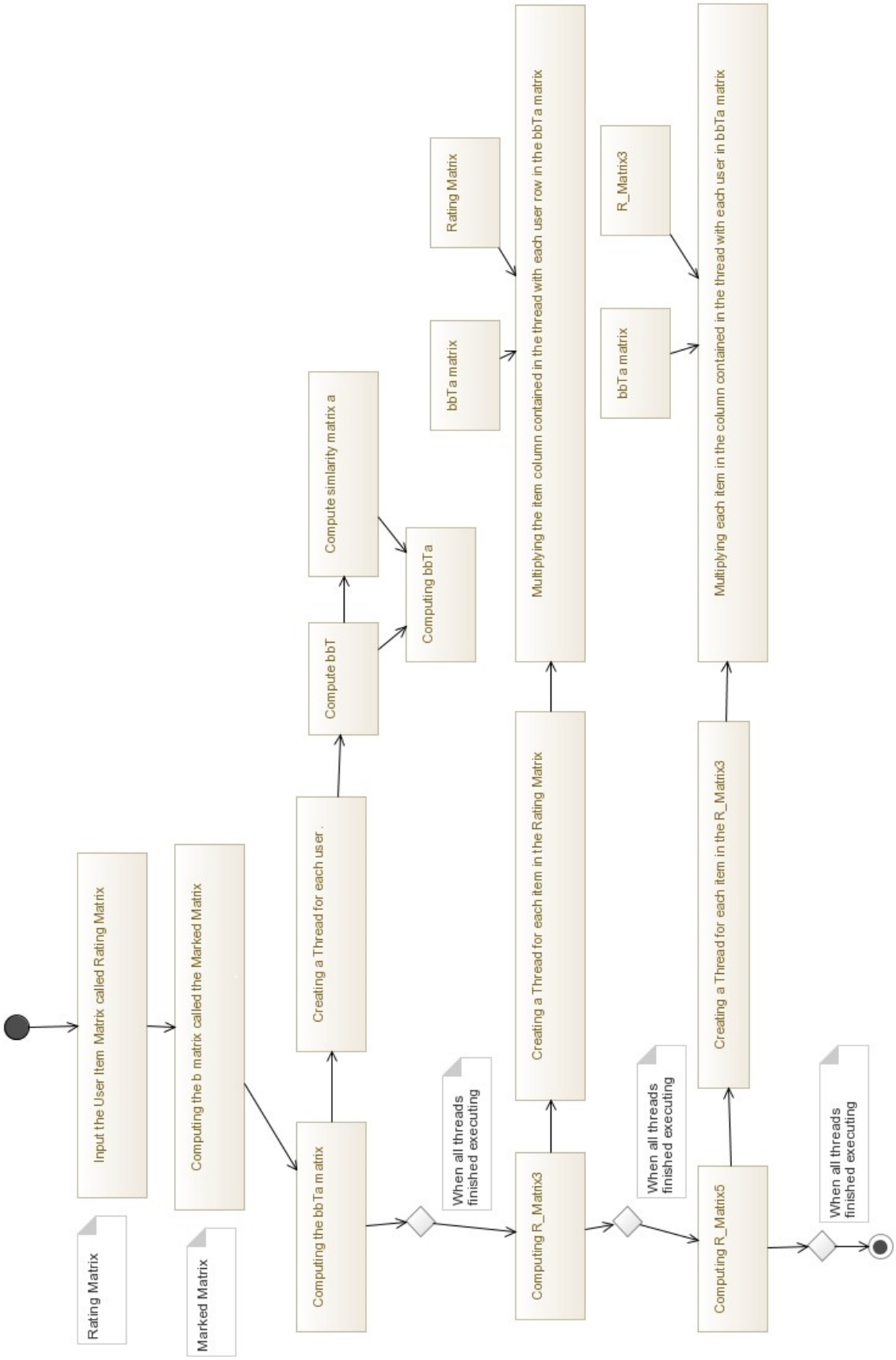
3.3 Analysis

Using the outputs above, the average time taken is calculated.

	read()	bbT()	a()	bbTa()	r_matrix3	r_matrix5	Total
Run 1	0.156	17.960	4.289	0.010	21.980	21.007	65.402
Run 2	0.167	16.293	4.270	0.009	22.139	20.804	63.682
Run 3	0.153	17.526	4.259	0.009	21.926	21.191	65.064
Run 4	0.157	17.576	4.261	0.009	22.063	20.496	64.562
Run 5	0.150	17.999	4.292	0.010	22.298	20.786	65.535
Mean	**0.156**	**17.596**	**4.274**	**0.009**	**22.061**	**20.856**	**64.849**

4. Optimization of CFAR algorithm with multithreading

4.1 Activity diagram



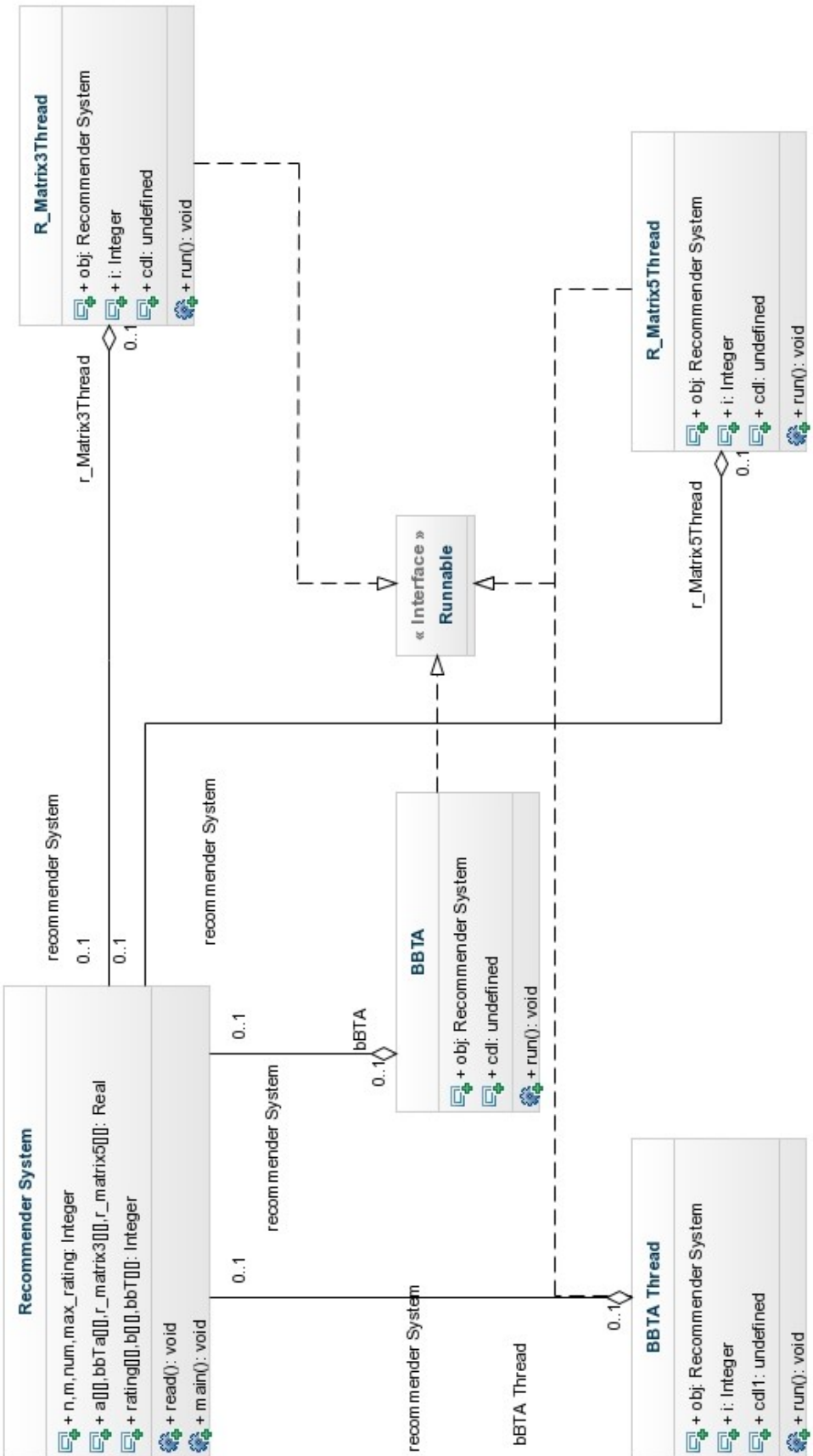
Overview

Since this algorithm requires multiplication of matrices, we have tried to reduce the time taken for the algorithm to run by using multithreading. The concept of multithreading is applied to remove the sequential nature of the program and add a parallel approach. This approach, by using multiple threads, enables our program to run more efficiently and take astronomically lesser than time than it's earlier representation.

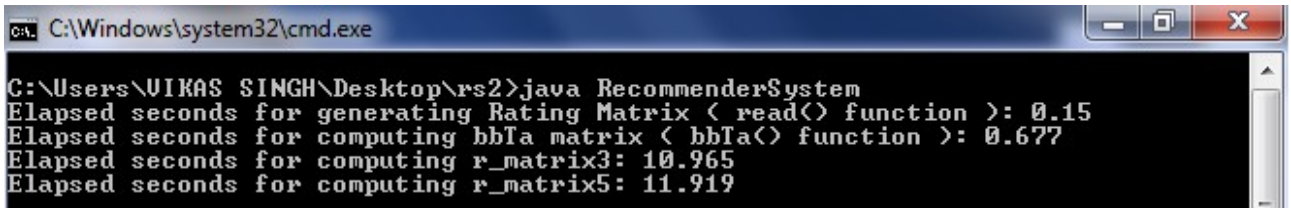
The revised activity diagram with threads is given below.

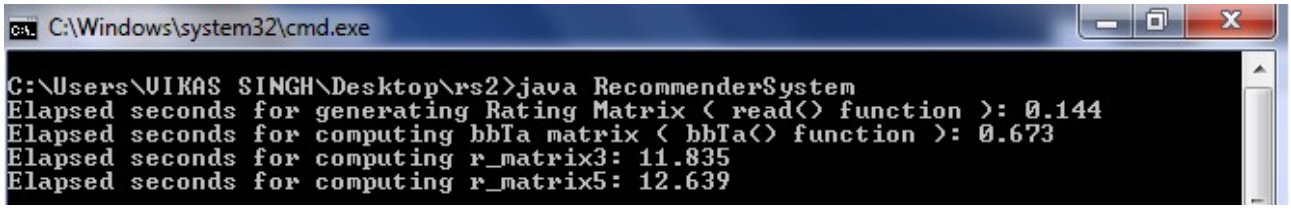
Class Diagram

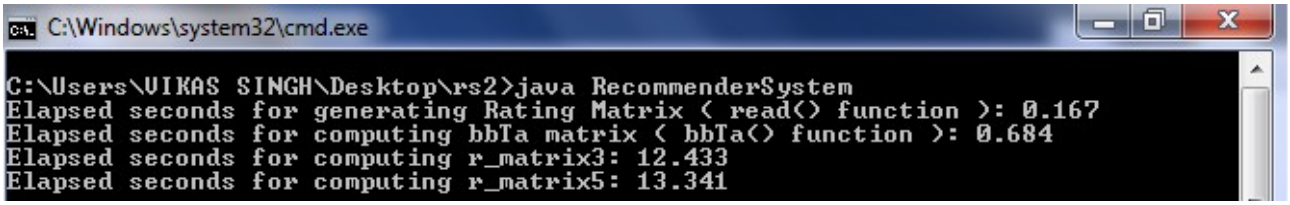
Continued on next page

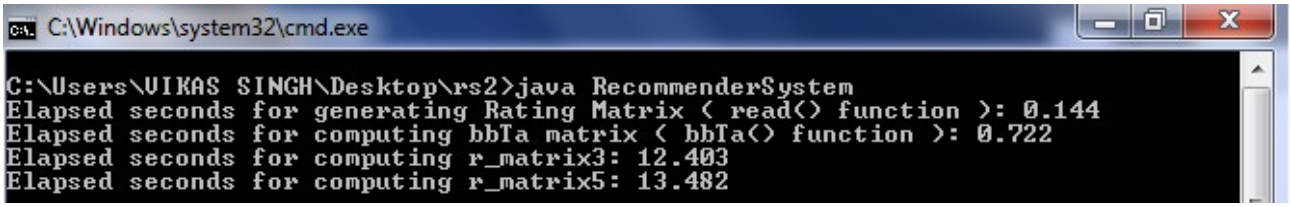

R_Matrix3Thread
+ obj: Recommender System
+ i: Integer
+ cdl: undefined
+ run(): void
r_Matrix3Thread
0..1
0..1
recommender System
0..1
recommender System
« Interface »
Runnable
R_Matrix5Thread
+ obj: Recommender System
+ i: Integer
+ cdl: undefined
+ run(): void
r_Matrix5Thread
0..1
Recommender System
+ n,m,num,max_rating: Integer
+ a[][],bbTa[][],r_matrix3[][],r_matrix5[][]: Real
+ rating[][],b[][],bbT[][]: Integer
+ read(): void
+ main(): void
0..1
recommender System
bBTA
0..1
BBTA
+ obj: Recommender System
+ cdl: undefined
+ run(): void
0..1
recommender System
bBTA Thread
BBTA Thread
+ obj: Recommender System
+ i: Integer
+ cdl1: undefined
+ run(): void

4.2 Outputs









4.3 Analysis

	read()	bbTa()	r_matrix3	r_matrix5	Total
Run 1	0.150	0.677	10.965	11.919	23.710
Run 2	0.144	0.673	11.835	12.639	25.291
Run 3	0.146	0.676	12.140	13.116	26.078
Run 4	0.167	0.684	12.433	13.341	26.625
Run 5	0.144	0.722	12.403	13.482	26.751
Mean	**0.150**	**0.686**	**11.955**	**12.899**	**25.691**

5. Addressing New-Element Drawback of CFAR algorithm

5.1 Overview

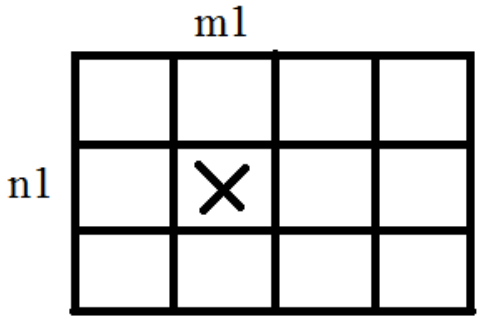
Whenever a new element is added to the training set, the whole values of R Matrix, B Matrix and A matrix are changed. For this reason, everytime the new recommendation values have to be calculated on every new addition. This causes the algorithm to run all over again on the data set and yield results. We define this as the "New-Element Drawback" of the CFAR algorithm.

To address this drawback, we have used a novel method to get rid of the need to calculate each and everytime the values for the whole data set. Instead, we utilize this new approach that we have discovered as an output of our sincere research in the field of recommender systems and specially the CFAR algorithm.

5.2 Our approach

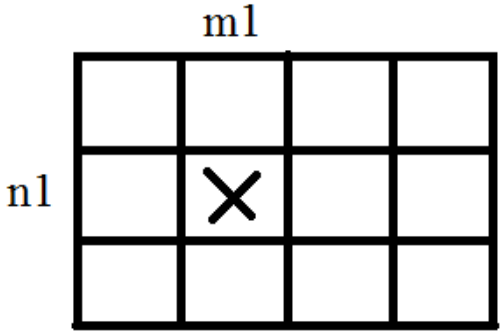
Tackling the change in R Matrix:

Suppose a new rating arrives for (n_1, m_1) pair then only one value will change in the R Matrix.



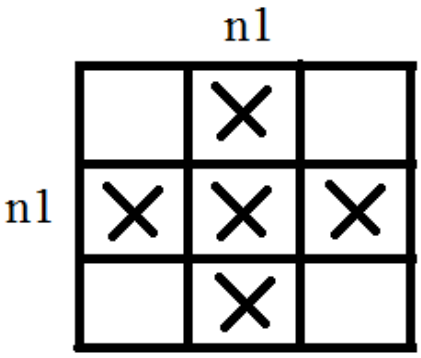
Tackling the change in B Matrix:

Here also only one value will be modified corresponding to (n_1, m_1). It will be changed to 1 which was earlier 0.



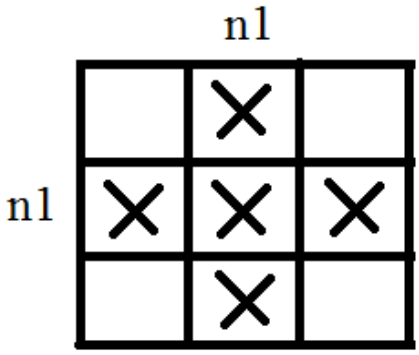
Tackling the change in BB^T Matrix:

Here the row and column corresponding to the n1th user would be modified as the path length for n1th user with respect to other users will only change. Rest will remain unchanged.



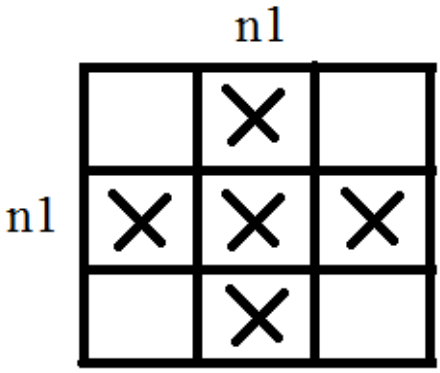
Tackling the change in A Matrix:

Here the row and column corresponding to the n1th user would be modified as the similarity for n1th user with respect to other users will only change. Rest will remain unchanged.



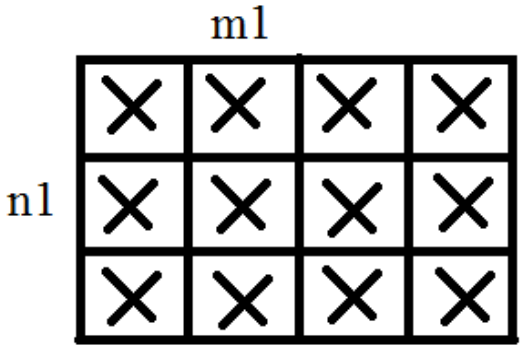
Tackling the change in BB^TA Matrix:

Since it is the scalar multiplication of BB^T and A, here also the row and column corresponding to n1th user will only change with respect to others.



Tackling the change in R3 Matrix:

Here the entire matrix would be modified but for each row of BBTA matrix, when multiplied with a column of R Matrix, only 1 singleton multiplication would change, except for the n1th row. Rest would remain constant. For the n1th row since all the values in this row are modified, this row's multiplication with other columns of R matrix has to be recomputed.



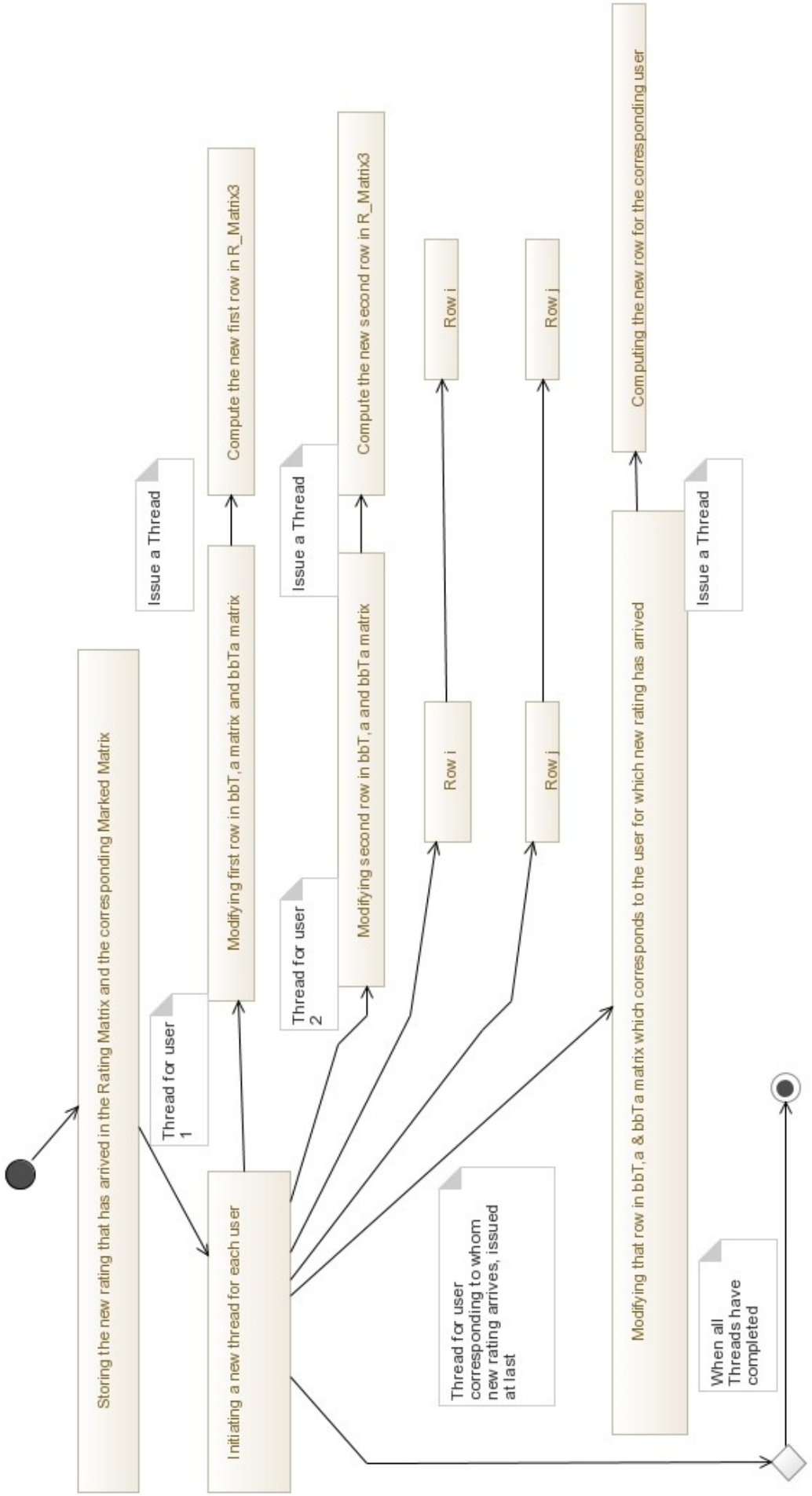
Taking advantage of these very singular changes in light of a single addition, we have improved the algorithm significantly.

Important Note –

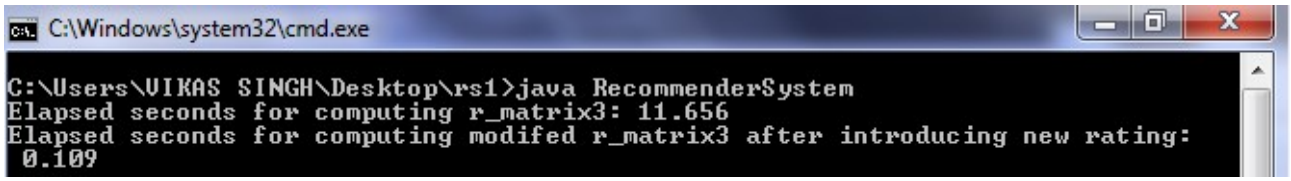
1. In our approach, the multiplication operations are being replaced by additions and subtractions since multiplication is more resource costly than addition and subtraction. This causes an additional boost to the performance of CFAR Algorithm.
2. We are not calculating R5 because for most practical applications, R3 is sufficient to generate a large of number of values to give a top-N recommendation.

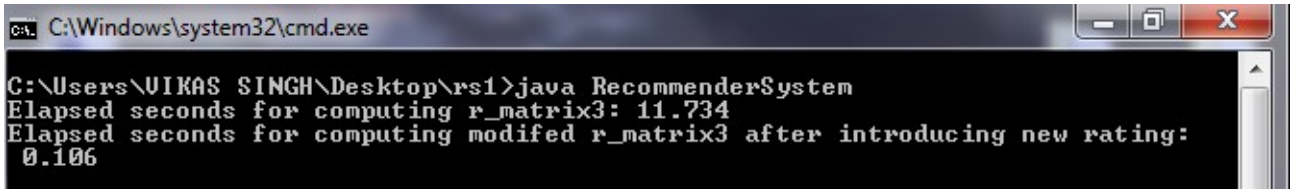
5.3 Activity Diagram

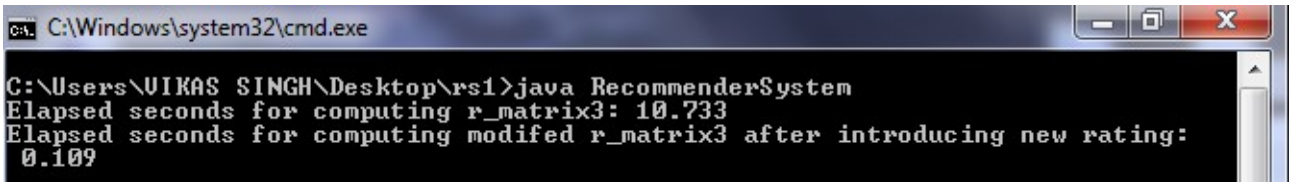
Continued on next page

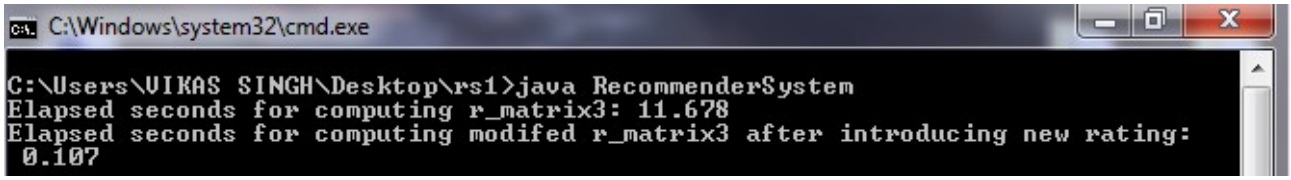

Storing the new rating that has arrived in the Rating Matrix and the corresponding Marked Matrix
Initiating a new thread for each user
Thread for user 1
Issue a Thread
Modifying first row in bbT,a matrix and bbTa matrix
Compute the new first row in R_Matrix3
Thread for user 2
Issue a Thread
Modifying second row in bbT,a and bbTa matrix
Compute the new second row in R_Matrix3
Row i
Row j
Row i
Row j
Thread for user corresponding to whom new rating arrives, issued at last
Modifying that row in bbT,a & bbTa matrix which corresponds to the user for which new rating has arrived
Computing the new row for the corresponding user
Issue a Thread
When all Threads have completed

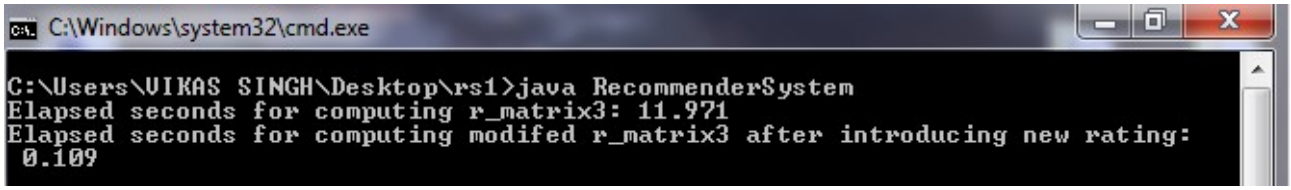
5.4 Outputs











5.5 Analysis

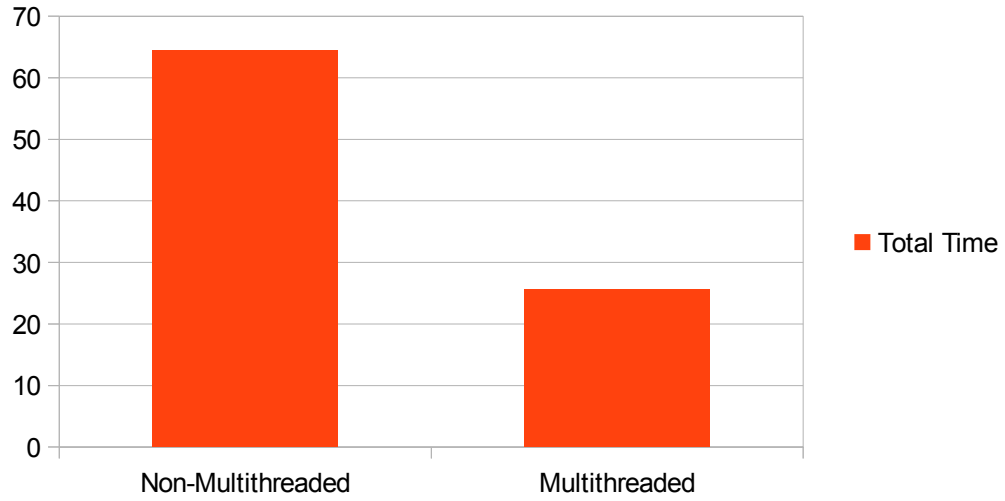
	r_matrix3 (normal)	r_matrix3 (optimized)
Run 1	11.656	0.109
Run 2	11.734	0.106
Run 3	10.733	0.109
Run 4	11.678	0.107
Run 5	11.971	0.109
Mean	**11.554**	**0.108**

6. Results and Conclusion

6.1 Result

6.1.1 Comparison between CFAR (non-multithreaded) and CFAR (multithreaded)

Percentage improvement in execution time for CFAR (multithreaded) wrt CFAR (non-multithreaded)

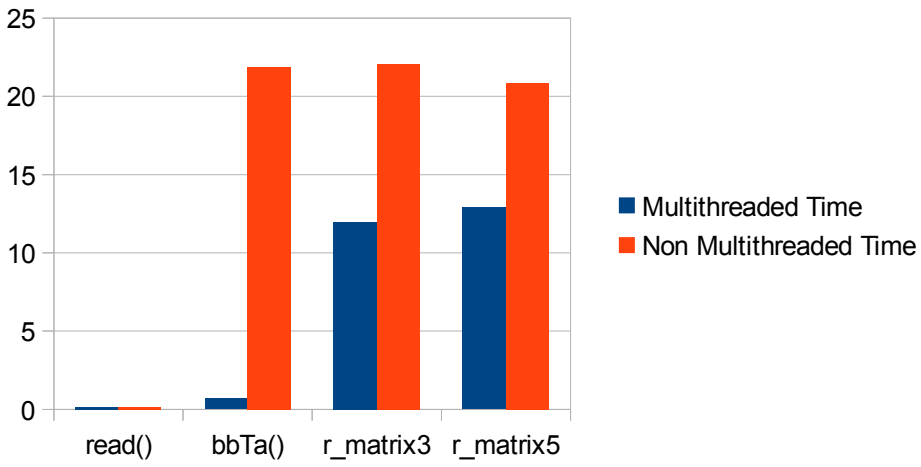


Percentage improvement = (Non-Multithreaded Time – Multithreaded time)/ Non – Multithreaded Time) * 100

= (64.849 – 25. 691) / 64.849 * 100

= 60.383%

6.1.2 Comparison between individual parameters



Here read() denotes total input reading time.

BbTa() denotes computing the matrix by multiplying bbT and similarity matrix.

R_matrix3 denotes the recommendation matrix where the path length is 3.

R_matrix5 denotes the recommendation matrix where the path length is 5.

Percentage Change in execution time of read():

Since both methods use same algorithm, no significant change is observed.

Percentage Change in execution time of bbTa():

= (21.879-0.686)/ 21.879 * 100

= 96.864%

Percentage Change in computation time of r_matrix3 :

= (22.061-11.955)/22.061 * 100

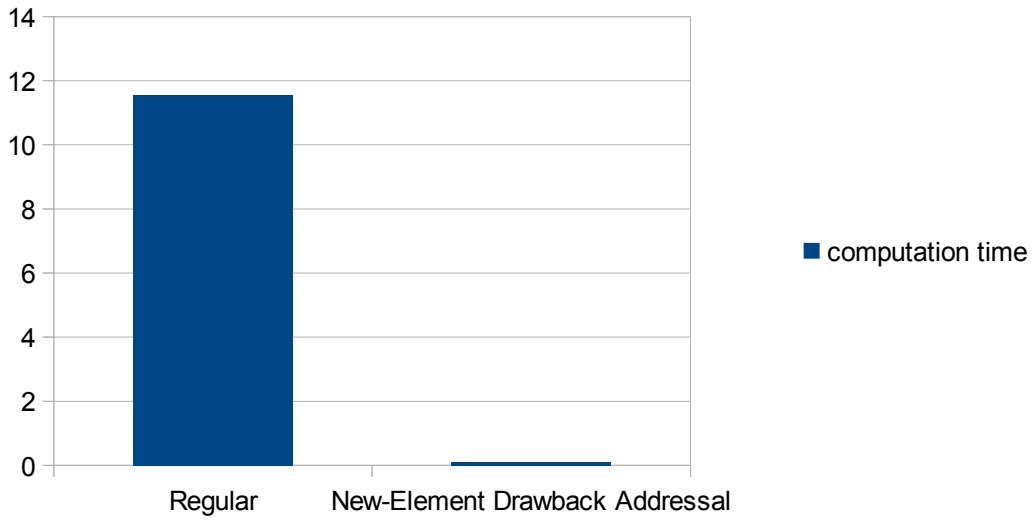
= 45.809%

Percentage Change in computation time of r_matrix5 :

= (20.856 – 12.899) / 20.856

= 38.152%

6.1.3 Comparison between regular r_matrix3 computation and r_matrix3 computation with new-element drawback addressal



Percentage change in computation time :

= (11.554-0.108)/11.554 * 100

= 99.065%

6.2 Conclusion

We can clearly see from our results that the improvements made on the CFAR algorithm by our research and implementation has wide ranging significant improvements in execution and computation time.

CFAR algorithm can be used industrially in high capacity servers where users and items would range in order of millions. Here such significant improvements will very efficiently cut down cost, time, energy and finances.

Acknowldegements

This project couldn't have been done without the presence of Mr Prabhakar Sarma Neog, our respected project guide. He has been with us at each step of our journey. Motivating us when we were short of effort and helping us like careful mentor who has stake in their students' progress.

Also we would like to thank our department, Computer Science and Engineering, NIT Silchar for giving us the opportunity to investigate and research such a wonderful and amusing application of recommender system.

It was a great learning experience. We can assume that everyone in the team learned a lot.

This project couldn't have been completed without a sincere team effort.

References

1. Solving the Sparsity Problem in Recommender Systems Using Association Retrieval. Yibo Chen.ChanLe Wu. Ming Xie. Xiajon Wou. http://ojs.academypublisher.com/index.php/jcp/article/download/jcp060918961902/3499
2. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering . Z Huang. dl.acm.org/citation.cfm?id=96377
3. ItemBased Collaborative Filtering Recommendation Algorithms.Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl files.grouplens.org/papers/www10_sarwar.pdf
4. Collaborative Filtering Recommender Systems. Michael D. Ekstrand, John T. Riedl and Joseph A. Konstanfiles.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf
5. Recommender Systems – en.wikipedia.org/wiki/Recommender_system
6. Collaborative Filtering - en.wikipedia.org/wiki/Collaborative_filtering

Appendix 1

Source code of CFAR Algorithm (non-multithreaded) with real data

```
import java.io.*;
import java.util.*;
class RSTest
{
int n,m;                        // Dimensions of User-Item matrix
int num;                        // Number of ratings in User-Item Matrix
int max_rating=5;
int rating[][];                 // Containing User-Item matrix
int b[][];                      // Marked matrix
int bT[][];                     // Transpose of Marked Matrix
int bbT[][];                    // Marked matrix and its transpose multiplied matrix
float a[][];                    // Similarity Matrix
float bbTa[][];                 // Association Matrix
```

```
float r_matrix3[][];                    // Recommendation Matrix where path length=3
float r_matrix5[][];                    // Recommendation Matrix where path length=5
      public void read()throws IOException          // Reading Training Set
      {
            FileReader fr=new FileReader("LargeDataset.txt");
            BufferedReader br=new BufferedReader(fr);
            String str=br.readLine();
            StringTokenizer st=new StringTokenizer(str,"  ");
            n=Integer.parseInt(st.nextToken());
            m=Integer.parseInt(st.nextToken());
            rating=new int[n][m];
            b=new int[n][m];
            bT=new int[m][n];
            bbT=new int[n][n];
            a=new float[n][n];
            bbTa=new float[n][n];
            r_matrix3=new float[n][m];
            r_matrix5=new float[n][m];
            str=br.readLine();
            num=Integer.parseInt(str);
            int n=1;
            while( n <= num )
            {
                  str=br.readLine();
                  st=new StringTokenizer(str,"     ");
                  int i=Integer.parseInt(st.nextToken());
                  int j=Integer.parseInt(st.nextToken());
                  int k=Integer.parseInt(st.nextToken());
                  rating[i-1][j-1]=k;
                  b[i-1][j-1]=bT[j-1][i-1]=1;
                  n++;
            }
            fr.close();
      }                                             // Calculating bbT matrix
      public void bbT()
      {
      for(int i=0; i<n ; i++)
            {
                  for(int j=0; j<n ; j++)
                  {
                        for(int k=0 ; k<m ; k++)
                              bbT[i][j] += b[i][k]*bT[k][j];
                  }
            }
      }
```

```
	public void a()                          // Calculating Similarity Matrix
	{
		for( int i=0 ; i<n ; i++)
		{
			for(int j=0 ; j<n ; j++)
			{
				float sum=0;
				for(int k=0 ; k < m ; k++ )
				{
					if( b[i][k] == 1 && b[j][k] == 1 )
						sum += ( max_rating - Math.abs( rating[i][k] -
rating[j][k] ) ) / (float)max_rating ;
				}
				if( i == j )
					a[i][j]=1;
				else
					a[i][j]= sum/m;
			}
		}

	}
	public void bbTa()                      // Calculating Association Matrix
	{
		for( int i=0 ; i<n ; i++)
		{
			for(int j=0 ; j<n ; j++)
				bbTa[i][j]=bbT[i][j]*a[i][j];
		}
	}
	public void r_Matrix3()    // Calculating Recommendation Matrix where path length=3
	{
		for(int i=0 ; i<n ;i++)
		{
			for(int j=0 ; j<m ; j++)
			{
				for(int k=0 ; k<n ;k++)
					r_matrix3[i][j] += bbTa[i][k]*rating[k][j];
			}
		}
	}
	public void r_Matrix5()throws IOException // Calculating Recommendation Matrix
where path                                          //length=5
	{
		for(int i=0 ; i<n ;i++)
		{
			for(int j=0 ; j<m ; j++)
```

```
                {
                        for(int k=0 ; k<n ;k++)
                                r_matrix5[i][j] += bbTa[i][k]*r_matrix3[k][j];
                }
        }
    }
    public static void main(String ar[])throws Exception
    {

        RSTest obj=new RSTest();
        long lStartTime = new Date().getTime();

        obj.read();

        long lEndTime = new Date().getTime();
        long difference = lEndTime - lStartTime;
        System.out.println("Elapsed seconds for generating Rating Matrix ( read()
function ): " + difference/1000.0);

        lStartTime = new Date().getTime();

        obj.bbT();

        lEndTime = new Date().getTime();
        difference = lEndTime - lStartTime;
        System.out.println("Elapsed seconds for computing bbT matrix ( bbT()
function ): " + difference/1000.0);

        lStartTime = new Date().getTime();

        obj.a();

        lEndTime = new Date().getTime();
        difference = lEndTime - lStartTime;
        System.out.println("Elapsed seconds for computing Similarity Matrix ( a()
function ): " + difference/1000.0);

        lStartTime = new Date().getTime();

        obj.bbTa();

        lEndTime = new Date().getTime();
        difference = lEndTime - lStartTime;
        System.out.println("Elapsed seconds for computing bbTa matrix ( bbTa()
function ): " + difference/1000.0);
```

```
            lStartTime = new Date().getTime();

            obj.r_Matrix3();


            lEndTime = new Date().getTime();
            difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing r_matrix3: " +
difference/1000.0);

            lStartTime = new Date().getTime();


            obj.r_Matrix5();

            lEndTime = new Date().getTime();
            difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing r_matrix5: " +
difference/1000.0);
      }
}
```

Appendix II

Source code of CFAR Algorithm (multithreaded) with real data

```
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.math.*;
class RecommenderSystem
{
int n,m;
int num;
int max_rating=5;
int rating[][];
int b[][];
int bbT[][];
float a[][];
float bbTa[][];
```

```
float r_matrix3[][];
float r_matrix5[][];
	public void read()throws IOException
	{
		FileReader fr=new FileReader("LargeDataset.txt");
		BufferedReader br=new BufferedReader(fr);
		String str=br.readLine();
		StringTokenizer st=new StringTokenizer(str,"  ");
		n=Integer.parseInt(st.nextToken());
		m=Integer.parseInt(st.nextToken());
		rating=new int[n][m];
		b=new int[n][m];
		bbT=new int[n][n];
		a=new float[n][n];
		bbTa=new float[n][n];
		r_matrix3=new float[n][m];
		r_matrix5=new float[n][m];
		str=br.readLine();
		num=Integer.parseInt(str);
		int n=1;
		while( n <= num )
		{
			str=br.readLine();
			st=new StringTokenizer(str,"     ");
			int i=Integer.parseInt(st.nextToken());
			int j=Integer.parseInt(st.nextToken());
			int k=Integer.parseInt(st.nextToken());
			rating[i-1][j-1]=k;
			b[i-1][j-1]=1;
			n++;
		}
		fr.close();
	}
	public static void main(String ar[])throws Exception
	{

		RecommenderSystem obj=new RecommenderSystem();

		long lStartTime = new Date().getTime();

		obj.read();

		long lEndTime = new Date().getTime();
		long difference = lEndTime - lStartTime;
		System.out.println("Elapsed seconds for generating Rating Matrix ( read()
function ): " + difference/1000.0);
```

```
            lStartTime = new Date().getTime();

            CountDownLatch cdl=new CountDownLatch(1);
            new BBTA(obj,cdl);
            cdl.await();


            lEndTime = new Date().getTime();
            difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing bbTa matrix ( bbTa()
function ): " + difference/1000.0);

            lStartTime = new Date().getTime();


            cdl=new CountDownLatch(obj.m);

            for(int i=0 ; i < obj.m ; i++)
                  new R_Matrix3Thread(obj,i,cdl);

            cdl.await();

            lEndTime = new Date().getTime();
            difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing r_matrix3: " +
difference/1000.0);

            lStartTime = new Date().getTime();

            cdl=new CountDownLatch(obj.m);
            for(int i=0 ; i < obj.m ; i++)
                  new R_Matrix5Thread(obj,i,cdl);
            cdl.await();

            lEndTime = new Date().getTime();
            difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing r_matrix5: " +
difference/1000.0);
      }
}
class BBTA implements Runnable
{
RecommenderSystem obj;
CountDownLatch cdl;
      public BBTA(RecommenderSystem obj,CountDownLatch cdl)
```

```
	{
		this.obj=obj;
		this.cdl=cdl;
		new Thread(this).start();
	}
	public void run()
	{
		CountDownLatch cdl1=new CountDownLatch(obj.n);
		for( int i=0; i < obj.n ;i++)
		{
			new BBTAThread(obj,i,cdl1);
		}
		try
		{
			cdl1.await();
		}
		catch(InterruptedException e)
		{}
		cdl.countDown();
	}
}
class BBTAThread implements Runnable
{
RecommenderSystem obj;
int i;
CountDownLatch cdl1;
	public BBTAThread(RecommenderSystem obj,int i,CountDownLatch cdl1)
	{
		this.obj=obj;
		this.i=i;
		this.cdl1=cdl1;
		Thread t=new Thread(this);
		t.start();
	}
	public void run()
	{
		for(int j=0 ; j <= i ; j++ )
		{
			obj.bbT[i][j]=0;
			float sum=0;
			for(int k=0 ; k < obj.m ; k++ )
			{
				obj.bbT[i][j] +=obj.b[i][k]*obj.b[j][k];
				if( obj.b[i][k] == 1 && obj.b[j][k] == 1 )
					sum+= ( obj.max_rating - Math.abs( obj.rating[i]
[k] - obj.rating[j][k] ) ) / (float)obj.max_rating ;
```

```
                }
                if( i == j )
                        obj.a[i][j]=1;
                else
                        obj.a[i][j]= sum/obj.m;
                obj.bbTa[i][j]=obj.bbTa[j][i]=obj.bbT[i][j] * obj.a[i][j];
        }
        cdl1.countDown();
    }
}
class R_Matrix3Thread implements Runnable
{
RecommenderSystem obj;
int i;
CountDownLatch cdl;
    public R_Matrix3Thread(RecommenderSystem obj,int i,CountDownLatch cdl)
    {
        this.obj=obj;
        this.i=i;
        this.cdl=cdl;
        new Thread(this).start();
    }
    public void run()
    {
        for(int j=0 ; j < obj.n ; j++ )
        {
                obj.r_matrix3[j][i]=0;
                for(int k=0 ; k < obj.n ; k++ )
                        obj.r_matrix3[j][i] +=obj.bbTa[j][k] * obj.rating[k][i];
        }
        cdl.countDown();
    }
}
class R_Matrix5Thread implements Runnable
{
RecommenderSystem obj;
int i;
CountDownLatch cdl;
    public R_Matrix5Thread(RecommenderSystem obj,int i,CountDownLatch cdl)
    {
        this.obj=obj;
        this.i=i;
        this.cdl=cdl;
        new Thread(this).start();
    }
    public void run()
```

```
        {
                for(int j=0 ; j < obj.n ; j++ )
                {
                        obj.r_matrix5[j][i]=0;
                        for(int k=0 ; k < obj.n ; k++ )
                                obj.r_matrix5[j][i] +=obj.bbTa[j][k] * obj.r_matrix3[k][i];
                }
                cdl.countDown();
        }
}
```

Appendix III

Source code of New-Element Drawback Addresal

```
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.math.*;
class RecommenderSystem
{
int n,m;
int num;
int max_rating=5;
int rating[][];
int b[][];
int bbT[][];
float a[][];
float bbTa[][];
float r_matrix3[][];
float r_matrix5[][];
```

```
public void read()throws IOException
{
	FileReader fr=new FileReader("LargeDataset.txt");
	BufferedReader br=new BufferedReader(fr);
	String str=br.readLine();
	StringTokenizer st=new StringTokenizer(str,"  ");
	n=Integer.parseInt(st.nextToken());
	m=Integer.parseInt(st.nextToken());
	rating=new int[n][m];
	b=new int[n][m];
	bbT=new int[n][n];
	a=new float[n][n];
	bbTa=new float[n][n];
	r_matrix3=new float[n][m];
	r_matrix5=new float[n][m];
	str=br.readLine();
	num=Integer.parseInt(str);
	int n=1;
	while( n <= num )
	{
		str=br.readLine();
		st=new StringTokenizer(str,"     ");
		int i=Integer.parseInt(st.nextToken());
		int j=Integer.parseInt(st.nextToken());
		int k=Integer.parseInt(st.nextToken());
		rating[i-1][j-1]=k;
		b[i-1][j-1]=1;
		n++;
	}
	fr.close();
}
public static void main(String ar[])throws Exception
{
	RecommenderSystem obj=new RecommenderSystem();
	obj.read();

	CountDownLatch cdl=new CountDownLatch(1);
	new BBTA(obj,cdl);
	cdl.await();

	long lStartTime = new Date().getTime();

	cdl=new CountDownLatch(obj.m);

	for(int i=0 ; i < obj.m ; i++)
		new R_Matrix3Thread(obj,i,cdl);
```

```
            cdl.await();

            long lEndTime = new Date().getTime();
            long difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing r_matrix3: " +
difference/1000.0);

            // Computing when a new rating arrives.

            lStartTime = new Date().getTime();
            int n1=11,m1=202,r=3;
            obj.rating[n1][m1]=r;
            obj.b[n1][m1]=1;
            NewRating obj1=new NewRating(obj,n1,m1,r);


            obj1.execute();

            lEndTime = new Date().getTime();
            difference = lEndTime - lStartTime;
            System.out.println("Elapsed seconds for computing modifed r_matrix3 after
introducing new rating:    " + difference/1000.0);
      }
}
class BBTA implements Runnable
{
RecommenderSystem obj;
CountDownLatch cdl;
      public BBTA(RecommenderSystem obj,CountDownLatch cdl)
      {
            this.obj=obj;
            this.cdl=cdl;
            new Thread(this).start();
      }
      public void run()
      {
            CountDownLatch cdl1=new CountDownLatch(obj.n);
            for( int i=0; i < obj.n ;i++)
            {
                  new BBTAThread(obj,i,cdl1);
            }
            try
            {
                  cdl1.await();
            }
```

```
            catch(InterruptedException e)
            {}
            cdl.countDown();
      }
}
class BBTAThread implements Runnable
{
RecommenderSystem obj;
int i;
CountDownLatch cdl1;
      public BBTAThread(RecommenderSystem obj,int i,CountDownLatch cdl1)
      {
            this.obj=obj;
            this.i=i;
            this.cdl1=cdl1;
            Thread t=new Thread(this);
            t.start();
      }
      public void run()
      {
            for(int j=0 ; j <= i ; j++ )
            {
                  obj.bbT[i][j]=0;
                  float sum=0;
                  for(int k=0 ; k < obj.m ; k++ )
                  {
                        obj.bbT[i][j] +=obj.b[i][k]*obj.b[j][k];
                        if( obj.b[i][k] == 1 && obj.b[j][k] == 1 )
                                    sum+= ( obj.max_rating - Math.abs( obj.rating[i]
[k] - obj.rating[j][k] ) ) / (float)obj.max_rating ;
                  }
                  if( i == j )
                        obj.a[i][j]=1;
                  else
                        obj.a[i][j]= sum/obj.m;
                  obj.bbTa[i][j]=obj.bbTa[j][i]=obj.bbT[i][j] * obj.a[i][j];
            }
            cdl1.countDown();
      }
}
class R_Matrix3Thread implements Runnable
{
RecommenderSystem obj;
int i;
CountDownLatch cdl;
      public R_Matrix3Thread(RecommenderSystem obj,int i,CountDownLatch cdl)
```

```
	{
		this.obj=obj;
		this.i=i;
		this.cdl=cdl;
		new Thread(this).start();
	}
	public void run()
	{
		for(int j=0 ; j < obj.n ; j++ )
		{
			obj.r_matrix3[j][i]=0;
			for(int k=0 ; k < obj.n ; k++ )
				obj.r_matrix3[j][i] +=obj.bbTa[j][k] * obj.rating[k][i];
		}
		cdl.countDown();
	}
}
class R_Matrix5Thread implements Runnable
{
RecommenderSystem obj;
int i;
CountDownLatch cdl;
	public R_Matrix5Thread(RecommenderSystem obj,int i,CountDownLatch cdl)
	{
		this.obj=obj;
		this.i=i;
		this.cdl=cdl;
		new Thread(this).start();
	}
	public void run()
	{
		for(int j=0 ; j < obj.n ; j++ )
		{
			obj.r_matrix5[j][i]=0;
			for(int k=0 ; k < obj.n ; k++ )
				obj.r_matrix5[j][i] +=obj.bbTa[j][k] * obj.r_matrix3[k][i];
		}
		cdl.countDown();
	}
}
class NewRating
{
RecommenderSystem obj;
int n1,m1,r;
	public NewRating(RecommenderSystem obj,int n1,int m1,int r)
	{
```

```
            this.obj=obj;
            this.n1=n1;
            this.m1=m1;
            this.r=r;
      }
      public void execute()throws Exception
      {
            CountDownLatch cdl=new CountDownLatch(obj.n);
            for(int i=0 ; i<n1 ; i++)
            {
                  if( obj.b[i][m1] == 1 )
                  {
                        obj.bbT[n1][i] += obj.b[i][m1];
                        obj.a[n1][i] =( obj.a[n1][i] * obj.m + ( obj.max_rating -
Math.abs( obj.rating[n1][m1] - obj.rating[i][m1] ) ) / (float)obj.max_rating ) / obj.m;
                  }
                  float old=obj.bbTa[n1][i];
                  obj.bbTa[n1][i]=obj.bbTa[i][n1]=obj.bbT[n1][i] * obj.a[n1][i];
                  new R_Matrix3ThreadNew1(obj, i, n1,m1,old,cdl);
            }
            for(int i=n1+1 ; i<obj.n ; i++)
            {
                  if( obj.b[i][m1] == 1 )
                  {
                        obj.bbT[i][n1] += obj.b[i][m1];
                        obj.a[i][n1] =( obj.a[i][n1] * obj.m + ( obj.max_rating -
Math.abs( obj.rating[n1][m1] - obj.rating[i][m1] ) ) / (float)obj.max_rating ) / obj.m;
                  }
                  float old=obj.bbTa[i][n1];
                  obj.bbTa[i][n1]=obj.bbTa[n1][i]=obj.bbT[i][n1] * obj.a[i][n1];
                  new R_Matrix3ThreadNew1(obj, i, n1,m1,old,cdl);
            }
            obj.bbT[n1][n1] += 1;
            obj.bbTa[n1][n1]=obj.bbT[n1][n1];
            new R_Matrix3ThreadNew2(obj , n1,cdl);
            cdl.await();
      }
}
class R_Matrix3ThreadNew1 implements Runnable
{
RecommenderSystem obj;
int i,n1,m1;
float old_BBTA_value;
CountDownLatch cdl;
      public R_Matrix3ThreadNew1(RecommenderSystem obj,int i,int n1,int m1,float
old_BBTA_value,CountDownLatch cdl)
```

```
	{
		this.obj=obj;
		this.i=i;
		this.n1=n1;
		this.m1=m1;
		this.old_BBTA_value=old_BBTA_value;
		this.cdl=cdl;
		new Thread(this).start();
	}
	public void run()
	{
		for( int j=0 ; j<m1 ; j++)
		{
			obj.r_matrix3[i][j] -= old_BBTA_value * obj.rating[n1][j];
			obj.r_matrix3[i][j] += obj.bbTa[i][n1] * obj.rating[n1][j];
		}
		obj.r_matrix3[i][m1] += obj.bbTa[i][n1] * obj.rating[n1][m1];
		for( int j=m1+1 ; j<obj.m ; j++)
		{
			obj.r_matrix3[i][j] -= old_BBTA_value * obj.rating[n1][j];
			obj.r_matrix3[i][j] += obj.bbTa[i][n1] * obj.rating[n1][j];
		}
		cdl.countDown();
	}
}
class R_Matrix3ThreadNew2 implements Runnable
{
RecommenderSystem obj;
int n1;
CountDownLatch cdl;
	public R_Matrix3ThreadNew2(RecommenderSystem obj,int n1,CountDownLatch cdl)
	{
		this.obj=obj;
		this.n1=n1;
		this.cdl=cdl;
		new Thread(this).start();
	}
	public void run()
	{
		for( int j=0 ; j<obj.m ; j++)
		{
			obj.r_matrix3[n1][j]=0;
			for(int k=0 ; k<obj.n ; k++)
				obj.r_matrix3[n1][j] += obj.bbTa[n1][k] * obj.rating[k][j];
		}
		cdl.countDown();
```

```
        }
}
```