International Journal of Computational Methods and Experimental Measurements
Vol. 12, No. 3, September, 2024, pp. 217-225
Journal homepage: http://iieta.org/journals/ijcmem

Parallel Memory-Based Collaborative Filtering for Distributed Big Data Environments

Pallavi Shree*, Somaraju Suvvari

Department of Computer Science, National Institute of Technology Patna, Patna 80005, India

Corresponding Author Email: pallavis.phd18.cs@nitp.ac.in

https://doi.org/10.18280/ijcmem.120303

Received: 25 June 2024
Revised: 9 September 2024
Accepted: 19 September 2024
Available online: 30 September 2024

Keywords:
memory-based, cosine similarity, euclidean distance, PySpark, parallel and distributed environment

ABSTRACT

The amount of information produced about any item or user has reached a very staggering level. Not only the volume of data, the velocity of data has reached an unprecedented magnitude. For any information retrieval or information processing system to work efficiently, it should be able to process massive amounts of data in real-time. Modern systems face a lot of challenges in managing data with high volume and velocity, especially when these systems are required to generate accurate predictions in a timely fashion. The most efficient way to ensure that modern information retrieval systems can adapt to the current volume and velocity of data is to implement them over a parallel and distributed environment. In this paper, we put forward a method for enhancing the scalability and performance of recommender systems in big data environments. By using the Euclidean distance to calculate the cosine similarity we introduce a technique which is efficient in parallelizing the algorithm for distributed environments. Thereby improving the computational efficiency and scalability of the recommender system. This enables such systems to manage large datasets with high accuracy and speed. With the help of parallel processing, our method can assist modern information retrieval systems keep up with the pace of ever-growing demands of data velocity and volume, ensuring real-time performance and robust scalability.

1. INTRODUCTION

There has been an incredible development and business surge in the online commerce industry. The economy fueled by this growth has evolved into a connected economy, and due to the rapid expansion of data, the network has now stepped into the age of big data. Users cannot correctly use the information made available by ever-growing e-commerce platforms, as the amount of commodity information has reached an inconveniently large scale. This has led to information overdose for the users. This means that incoming information is above and beyond the processing capacity of recipients, users, and systems alike. Due to this astronomical data growth, parallel and distributed systems of recommendation are becoming increasingly important. One of the key benefits of such systems being distributed and parallel is their ability to process large datasets more quickly. Real-time recommendations to users have to be the main aim for recommender systems, such as those used by streaming services and e-commerce websites. Another benefit of parallel and distributed recommender systems is their ability to handle larger datasets. This is important for recommender systems, considering the number of parameters that are factored in while providing recommendations, including but not limited to user likings, item features, and information related to users and items. Additionally, parallelized distributed recommender systems are scalable, resilient, and cost-effective. This means they can be deployed on large-scale systems, handle failures, and be deployed on commodity hardware. We present an implementation of a memory-based collaborative filtering algorithm in a parallel and distributed environment in this paper. Our implementation uses several techniques to improve performance, including:

(1) Partitioning the user-item preference matrix across multiple processors.

(2) Using a more efficient, parallelizable version of the cosine similarity formula.

Our experimental results show that our parallel implementation of a memory-based collaborative filtering algorithm can significantly improve performance over a serial execution. The paper has been divided into six sections. In the second section, we discuss the basic terminologies used in recommender systems. The third section of the paper discusses the similar efforts in the field. Our methodology of parallel implementation of a memory-based collaborative filtering algorithm is laid out in the fourth section. All the experimental results of the proposed method have been presented using graphical and tabular data in the fifth section. We finally conclude the paper in the final and sixth sections.

2. BACKGROUND

Before we jump into recommender systems, the most imperative step should be to understand different categories of data processing systems referred to as Information Retrieval systems. Information retrieval (IR) can be defined as the process of using a source of data and extracting information pertinent to an inquiry done by, any user or any other system e.g., a movie based on genre from a streaming platform, a journal from a repository based on a subject, or results produced by the search engine based on a question asked by the user [1]. One the types of IR is recommender systems. Hence, we can define a recommender system (RS) as a subcategory of an information filtering system that calculates the most accurate rating a user would provide for an item [2]. RS as a software solution has its roots in the most basic human tendency of asking for suggestions or recommendations before trying out any new experience or object or even for making friends. The information provided by these systems helps the users make decisions like purchasing an item, renting a movie, etc. The recommendations presented are designed to assist individuals in making informed decisions across a range of contexts. This means that the primary objective of these systems is to provide personalized recommendations which is the major difference between recommender systems and information retrieval search engines [2]. Recommender systems have emerged as essential tools in electronic commerce, providing effective solutions for online users grappling with information overdose. Their significance lies in their ability to sift through vast amounts of data, enabling users to make informed decisions. These systems have become pivotal in addressing the challenges posed by the overwhelming volume of information on online platforms. Hence, numerous methods for generating recommendations have been put forward. Companies like Netflix, Amazon, Facebook, etc. have successfully applied and gained from these methods for recommending books, movies and friends. Any RS has two main objects: “Items” and “Users”. The topic or object for which the suggestions are generated is generally called “Item”. Normally, RS is meant to recommend a specific type of item like movies, books, songs, restaurants, etc. Such systems are mainly aimed at individuals (referred to as “User” in RS) who are relatively new in a certain domain, like people looking for hotel suggestions before visiting a new place. Interaction between the items and users is called “Transaction”. Transactions give us data about items, users and preferences. The recorded preferences of users act as an input to the RS. These inputs can be collected implicitly or explicitly. Explicit feedback [3] shows the direct preference of users for an item. Explicit ratings are mostly on a numerical scale like a range of 1 rating for worst and 5 being the best, like otherwise dislike, etc. Implicit feedback is extracted from user actions like the amount of time the user was on any given page, clicks performed by users on websites, whether the user purchased the item or watched the video, etc. Based on the way recommendations are generated, RS can be classified as given in Figure 1.

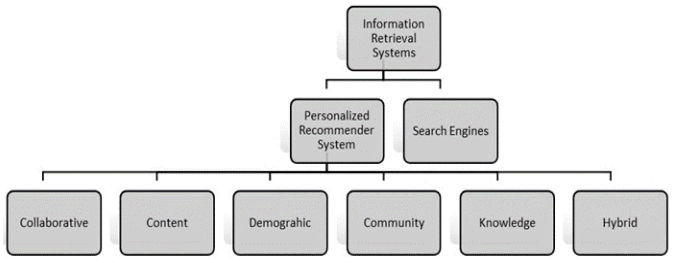


Figure 1. Type of recommendation system

(1) Collaborative filtering: Collaborative filtering [4] recommender systems leverage the preferences and behaviours of other users to suggest items or content to a particular user. By analysing the choices and interactions of a diverse user base, these systems identify patterns and correlations, allowing them to generate personalized recommendations. This approach helps users navigate the abundance of available options, making their online experience more tailored and relevant. Through sophisticated algorithms and data analysis, recommender systems enhance user engagement and satisfaction by presenting them with choices aligned with their interests and preferences. For instance, a recommendation of a film for a viewer can be grounded on the explicit or implicit feedback given by various other viewers who have watched the movie.

(2) Content-based: Content-based [5] recommender systems use the previous interactions of the uses with the system and item properties of the items under consideration. Such systems might recommend movies based on the genre of the previously watched movies of the users or a book recommender tool might suggest books of the same author whose other books have already been read by the same user.

(3) Demographic: These systems use the date of birth, sex of the user, and their current geographical location to generate suggestions for the user. For instance, this type of recommender would recommend the product to any user on the same lines as the products that users of the same age and gender have purchased.

(4) Knowledge-based: Industry or domain-specific knowledge is used by such systems to provide useful suggestions to the user. The best example is a system suggesting a recipe to the user, considering the dietary restrictions and the ingredients they have on hand.

(5) Community-based: Community-based recommender systems recommend items to a user relying on the predilections of user clusters having similar features. For instance, this type of recommender would use all views of all users of the same online forum to which another user belongs and suggest a movie rated highly by the users of the forum.

(6) Hybrid filtering: These recommender systems are built using a combination of systems mentioned above.

2.1 Content-based filtering algorithms

Content-based filtering (CBF) generates a feature set of items and preference or behaviour profiles for users based on additional information about user demography, online behavior, their friend network, and the properties of items used by customers. CBF is classified as content-based when it provides recommendations grounded in the content details of items. Conversely, when these recommendations rely on the contextual information of users, CBF is termed context-based. In most situations, extracting relevant information about users or items becomes challenging due to a lack of information or information overload. This limits the performance and application of CBF.

2.2 Collaborative filtering algorithms

The majority of the recommender systems are based on Collaborative filtering algorithms. It can be defined as a method that generates suggestions, i.e., filters the information related to the choices of any person by gathering information from a large quantity of other people, i.e., collaborative filtering [6]. Breese et al. [7] categorized CF techniques into systems which are based on memory while another type of system is built on models.

2.2.1 Model-based CF techniques

In the model-based approach, the system generates parameters to model the behavior of users and features of items, which enables it to make suggestions using the created parameters. Filtering techniques are collaborative in nature and build learning models based on machines for predicting ratings that an item might get from the user. These models are trained on a dataset of user ratings and item features. Once the models are trained, they can forecast the most probable rating an item would get from a specific user, even in scenarios where the user-item interaction would not have occurred earlier and no rating data is recorded for this user-item pair. As the main of such techniques is to predict rating, probability of purchase, etc., these systems are most commonly configured as supervised learning problems.

2.2.2 Memory-based CF techniques

Memory-based collaborative filtering techniques are relatively simple to implement and can be very effective for generating personalized recommendations. Memory or neighbourhood-based CF are implemented by calculating distance or similarity metrics. In memory-based CF, recommendations are based on similarities among users [8] or items [9].

(1) User-item Collaborative Filtering: items used or purchased or rated by users similar to us. Such systems first find users similar to the user under consideration and then generate recommendations for users based on their purchase or rating history.

(2) Item-item Collaborative Filtering: based on the segregation that suggests that users who show interest in specific items are more likely to be interested in these items. Here, we first find similarities among a bunch of items and recommend items that are most similar to items already rated by that user.

A detailed comparison in terms of the various characteristics of both these methods can be found in Table 1.

Table 1. Comparison of memory and model-based CF

Characteristic	**Model-Based Collaborative Filtering**	**Memory-Based Collaborative Filtering**
Simplicity	More complex	Simpler
Interpretability	Less interpretable	More interpretable
Flexibility	Less flexible	More flexible
Cold-start handling	Worse	Better
Scalability	More scalable for large datasets (once models are trained)	Less scalable for large datasets
Accuracy	More accurate, especially for sparse datasets	Less accurate, especially for sparse datasets
Explainability	Less explainable	More explainable

2.3 Challenges of the recommender system

Recommender systems are complex algorithms that use data to predict what users will like. They are used in various applications, like online shopping, streaming services, and social media. However, recommender systems also face several challenges, including:

Lack of data: Recommender systems need data to learn user preferences and make accurate recommendations. However, it is quite possible that there is not enough data for the users with very specific interests and this makes it very difficult to make useful and correct recommendations.

Cold start problem: It is a phenomenon faced by a recommendation system when a new user or item enters the system and recommendations have to be generated for such users or items. Being new to the system there is no associated data for such users or items. This makes it very hard to provide recommendations when no data is available for the user's interests or the item's popularity. Thus, making it very challenging while generating recommendations.

Scalability: The most critical feature of any Recommender system should be its ability to scale up to the ever-increasing volume of data being generated by users and items. As most of the recommender systems run on very complex algorithms which are computationally demanding, scalability becomes one of the biggest challenges that should be considered while designing any recommender systems.

Sparsity: As the majority of the users do not interact with the majority of the items, the user-item matrix in a recommender system is often very sparse. This means data is not available for most of the user-item pairs. The absence of data makes it very difficult to make useful recommendations and understand the preferential pattern of the user or the popularity of items.

Bias: The methods or even data used to generate recommendations can be the source of an inherent bias towards certain items or users. Once the bias is present in the recommendations there is a very high chance of the same items being recommended to the majority of users and not taking into account the actual preferences of the user.

Privacy: There is an automatic concern relating to the collection and storage of data for making more accurate recommendations. This data can be transactional data or implicit data like browsing history and purchase history. This is not only a security concern but also raises ethical concerns as to what is the extent to which we should collect data without infringing the privacy of any user.

Apart from the challenges discussed above, there are numerous other challenges faced by any recommender system. It should be flexible enough to cater to the ever-evolving preferences of the user and also consider the new items which are regularly added to the inventory. It also should be robust to handle attacks such as shilling attacks. A shilling attack is an attack where the system is flooded by fake user profiles and their review of items which can either promote or paint a bad review for any item. Even with such challenges, the recommender system is a very useful tool which not only helps users identify the most suitable items for them but also discovers new content and products which they would normally not try.

3. PREVIOUS WORK

With the increasing ease of accessibility to the internet and a large number of online and connected devices, the majority of the applications running on such devices have become data-centric. Data is now being generated at a very tremendous rate. Applications like search engines, social media platforms, content streaming and sharing platforms have data and intelligent usage at their core. They are processing data from a few gigabytes to several terabytes or even petabytes. Google for example is processing around twenty petabytes of data daily [10]. There have been various reviews of different recommender system techniques and applications. Lu et al. [11] provided a comprehensive survey of real-world recommender system applications and categorizes the definite necessities for recommendation methods in each application field. The author has also systematically reviewed recommender systems (online software) by considering four aspects:

(1) Recommendation methods: This includes the different algorithms recommender systems use to generate recommendations, such as collaborative filtering, content-based filtering, and knowledge-based filtering.

(2) Recommender system software refers to specific applications that implement recommender systems, such as BizSeeker.

(3) Real-world application domains: This refers to the different areas in which recommender systems are used, such as e-business, e-learning, and entertainment.

(4) Application platforms: This refers to the different devices and platforms on which recommender systems are available, such as mobile phones, TVs, and websites.

Chen et al. [12] provided a clear and concise overview of CF-based recommender systems, covering rudimentary ideas, different CF algorithms, and assessment metrics. They also discuss traditional CF methods' challenges, such as cold start, data sparsity and scalability. The authors introduce the hybrid CF methods based on social networks, which have shown promising results in addressing the challenges of traditional CF methods. This work discusses a wide range of memory and model-based techniques, including enhanced similarity measures, memory-based trust-aware CF, model-based social matrix factorization-based CF, and dimensionality reduction techniques.

Collaborative filtering algorithm is one of the most deployed personalized recommendation approaches especially in commercial recommendation systems [13, 14]. Scalability is a major concern for collaborative filtering. This has also been pointed out by Mishra et al. [15] who consolidated the research problems in Recommendation Systems, scalability is one of the most challenging problems to be solved. Bobadilla [16] also studied the cold start problem present in all recommender systems alongside similarity metrics tailored for this problem. Authors have also dwelled on providing a survey of social filtering focusing on trust, reputation and credibility.

One of the approaches used in addressing such a problem is the reduction of data size [17]. This is done by either reducing the number of users by randomly sampling customers or by not considering users who have made fewer purchases. Items can also be reduced by selecting certain specific categories of items [18]. This approach of addressing scalability issues does not work as recommendation quality worsens significantly.

The segmentation method [19] has also been used to tackle scaling issues where users are segmented into groups of similar customers. After segments are generated, the similarity between users and the vector which summarizes each segment is calculated. Cluster models are efficient as compared to the data size reduction approach.

Most recommendation algorithms have tackled the scalability issue by moving the computationally heavy part of running any model into an offline phase. The same has been performed in Amazon.com recommendations [20] where it generates a similar item table and finds items similar to the items purchased by the user in offline mode. Part of the recommendation, which is only on a real-time basis, is listing the most similar items for that particular user. The real-time approach does not depend on the total number of items but only on the purchases made by that user, making item-item CF a highly scalable recommendation algorithm.

However, moving the calculation steps, which consume a maximum amount of time to the offline phase and saving intermediate results for the online phase helped in scaling as per dataset. Still, the offline phase is a step which consumes a large amount of time and a tremendous number of resources. Then researchers started using a parallel data processing method such as Map-Reduce over a distributed environment to implement collaborating filtering. Varanasi [21] implemented user-based collaborative filtering over Map-Reduce in a Hadoop environment where Jaccard distance was the similarity measure being calculated. The experiments in this approach do not include the effort for pre-processing as the performance measurement metric. Only the running time and data size are considered. It uses 6 MapReduce jobs.

Bobadilla et al. [22] identified the limitations of traditional similarity metrics, such as Pearson correlation, which are not well-suited for discrete data, and proposes a new metric that addresses these limitations by combining numerical and nonnumerical information. Three of the most widely used practical datasets were used by the author to evaluate the new metric and prove its much better performance than the traditional metrics regarding accuracy, coverage, and precision/recall.

Varanasi [23] implemented an item-item CF using Map-Reduce with multiple similarity measures namely Jaccard Similarity, Tanimoto Similarity, Cosine Similarity and Pearson Coefficient. The results show that as the authors increased the number of nodes execution time decreased. However, even with a 6-node cluster, the time consumed is well above 4 hours and reaches around 16 hours for certain datasets. This work uses 7 MapReduce jobs for implementation.

When parallel implementation of the basic recommendation algorithms [24] used Pearson correlation, adjusted cosine similarity and alternating least squares models on a platform like TensorFlow. The results pointed out that the adjusted cosine similarity neighbourhood approach provided the best accuracy, whereas the alternating least square method gave the lowest accuracy. The offline computation phase of adjusted cosine similarity on the other hand took around 8 hours in execution.

In another Map-Reduce-based approach used in "Scalable Recommender System over MapReduce" [25], item-item and user-user collaborative are directly implemented without changing the approach to calculate similarity. Here, they focus on the accuracy of the RS, not on the efficiency of the RS. It used 4 maps and 3 reduce jobs. A lot of contributions and work has been done where the Hadoop MapReduce framework has been used to process the calculation of collaborative filtering in a parallel manner [26], but there seems to be a lack of focus on the serial processing required in executing a MapReduce job. Hence it is imperative that fewer full scans and sequential access should be assured while executing MapReduce jobs as it is paramount for maintaining the superior efficacy of parallel processing because such jobs require disk operations on the data nodes for getting input data and writing back the processed information.

The author has put forward a new method for aggregating recommendations from multiple algorithms in paper [27]. The method, called Collaborative Rank Aggregation (CRA), uses a metaheuristic algorithm to find weights for each algorithm's ranking, such that the aggregated ranking is more accurate than any of the individual rankings. But this method requires a training set to tune the weights of the individual algorithms. The CRA method may not be able to improve the accuracy of recommendations if the individual algorithms are not accurate and also may not be able to enhance the accuracy of suggestions for all users.

Dahdouh et al. [28] used Spark as processing system, recommendations are made to around 1218 learners from a list of more than 150 courses. The work has been done by using 3 node cluster and a dataset of 5000 transactions where the execution time is 55 seconds. Sun et al. [29] have proposed SACF model learns a similarity matrix that embeds features which are both related and unrelated to sequence, which is more informative for personalized e-government recommendations. SACF uses matrix factorization to learn the similarity matrix, which can effectively calculate the similarity between a pair of users having no items rated by both of them. SACF reduces the complexity of computing user similarity from quadratic to linear, making it more efficient for large-scale e-government recommendation tasks. It is evaluated on a live e-governance database and shows significant improvement over the cutting-edge methods.

All the works discussed above have used direct implementations of existing algorithms. This may improve the efficiency to a certain extent, but to completely parallelize any algorithms, we might have to use specific implementations of algorithms which are more feasible for parallel and distributed processing. In the next section, we discuss our approach for using a different version of existing cosine similarity in addition to parallel and distributed methods of processing.

4. PROPOSED WORK

The most practical implementation of a memory-based CF is calculating the distance metric like cosine similarity [19], Pearson correlation [30] and Jaccard coefficient. We have focused on cosine similarity. It measures the similarity of two items, A and B, by measuring the cosine of the angle between the two vectors. The original formula for the cosine similarity is as given in Eq. (1):

$$\text{Similarity}(A,B) = \text{Cos}(A,B) = \frac{|A \cdot B|}{\| A \| * \| B \|} \tag{1}$$

In this paper, item-item CF is implemented by using cosine similarity in parallelly in a parallel manner. For this parallel implementation, the proposed calculation of cosine similarity is given in Eq. (2):

$$\text{Similarity}(A,B) = \frac{|A|^2 + |B|^2 - C^2}{2 * |A| * |B|} \tag{2}$$

where, A and B are item vector and C is the Euclidean distance between A and B.

When applying cosine similarity in item-item CF each vector corresponds to an item and vector dimension corresponds to users who have rated the item.

The following algorithm [19] provides an approach by calculating the similarity between a single item and all related items.

Algorithm Iterative approach to find likeness among any item and remaining associated items

```
1: Loop every item Ix
2: Loop every User U who rated Ix
3: Loop every item Iy rated by user U
4: Save values when a user rated Ix and Iy
5: Loop every item Iy
6: Calculate the similarity between Ix and Iy
```

The computation described above is the extremely time intensive. To improve the efficiency, it requires reducing the problem into manageable proportions. Number of users and items range in millions and become unmanageable. The approach taken in our work is to perform independent calculations in a parallelized and distributed manner. To achieve parallelization, the following steps are required.

Step 1: Load and partition data

```
1: Read item ID, user id & rating from csv source
2: Partition data with item ID column
```

Once we have partitioned data, we can carry on with further transformations. Given below are transformations applied.

Step 2: Consolidate dimensions of each item vector

```
1: Mapper 2: - Map data into key-value pair
2: Input: - Partition data from pre-processing step.
3: Output: - (Key(item id), value (user id, rating))
4: Reducer 1: - Consolidate all rating for each item
5: Input: - (Key(item id), value (user id, rating))
6: Output: - (Key(item id), value (Magnitude of item vector,((user 1, rating), (user N, rating))))
```

Step 3: Generate item pair for similarity calculation

```
1: Mapper 3: - Generate item pair as key and value as pair of magnitude of vector and user & rating pair
2: Input: - (Key(item id), value (Magnitude of item vector, ((user 1, rating)(user N, rating))))
3: Output: - (Key(item item), value (Magnitude of item I vector, ((user 1, rating), (user N, rating))), (Magnitude of item J vector, ((user 1, rating), (user N, rating))
```

Step 4: Calculate cosine similarity for each item pair

```
1: Mapper 4: - Use the formula in the Eq. (9) to calculate the cosine similarity.
2: Input: - (Key(itemi,itemj), value (Magnitude of item vector, ((user 1, rating), (user N, rating))))
3: Output: - (Key(itemi,itemj), value (Cosine similarity of item pairs))
4: Reducer2: - Produce a single file with item pair and corresponding cosine similarity
```

Our approach has multiple facets, which makes it more efficient.

1. The magnitude of item vectors is calculated in a parallelized manner.
2. Use of Euclidean distance for calculating cosine similarity.
3. Calculation of similarity in a distributed Spark cluster.

The Eq. (1) is slower because it computes the sum of products whereas Eq. (2) calculates sum of square distances. Multiplication is an expensive operation compared to subtraction and square. The Eq. (2) does not require the computation of the product, and is therefore faster. We break down and discuss each component of both formulae in the next paragraph.

When we calculate the square root of the sum of the squares of each corresponding element of any vector, we can say that we have calculated the norm of that particular vector is defined as the square root of the sum of the squares of its elements. For example, let us consider a vector A, the norm can be calculated by Eq. (3).

$$\| A \| = \sqrt{\sum A^2} \tag{3}$$

We need to partition vector A into smaller blocks and then compute the norm of each block in parallel when the norm has to be calculated in a distributed environment. Then, all the intermediate norm of each block is summed up to achieve the final norm of the vector. Given, two vectors A and B, the dot product can be calculated by computing the sum of the products of their corresponding elements. This is also called a scalar product of the two vectors which is calculated using Eq. (4).

$$A \cdot B = \sum A * B \tag{4}$$

We need to partition the vectors A and B into smaller blocks and then compute the dot product of each block in parallel in a distributed environment. Then we can achieve the final dot product of the two vectors by adding up the dot products of the blocks. The Euclidean distance between two vectors is defined as the square root of the sum of the squares of the differences of their corresponding elements. In other words, for two vectors A and B, the Euclidean distance is given by: This can also be computed in a distributed environment by partitioning vectors A and B into smaller blocks and computing the parallel Euclidean distance between each block. Once the Euclidean distance between each block is computed, the overall Euclidean distance between the vectors can be computed by summing the Euclidean distances between the blocks.

In the proposed Eq. (1), we have precalculated the magnitude of the item vector, so this calculation does not contribute to execution time when calculating similarity in the final step. As the data is partitioned based on items, all the dimensions corresponding to users rating the same item are present in a single partition. This ensures minimum shuffle between the partitions.

Table 2. Comparison of number of mapper and reducers

Paper	Mapper	Reducer
Varanasi [21]	6	6
Varanasi [23]	7	7
Wang and Yao [25]	4	3
Proposed Method	4	2

It can be understood from the Table 2 that number of MapReduce jobs is very important. Our work uses optimum number of mappers and reducers. Hence, we get the improved results.

4.1 Roles of distributed system in recommendation

Apart from the calculation changes. We also made sure to use of distributed and parallel computing as the two main weapons to fight the challenges and enhance the performance of these recommendation engines.

These programming paradigms have a two-pronged approach, where the computation work is spread or distributed across multiple computers whereas parallel processing utilized the multiple cores of each machine and the workload is further distributed in multiple cores of each machine. Furthermore, these systems can be upgraded by using a large number of commodity hardware and by scaling parallelly thereby reducing the cost of expensive vertical upgrades. This collection of computing resources makes it possible to handle large datasets and enables recommender systems to generate real-time recommendations.

There are a bunch of advantages provided by the use of distributed and parallel computation:

Scalability: A distributed system can easily scale to match the growing rate of data, users and items. As such systems scale up horizontally which means adding commodity hardware instead of expensive servers, it is much cheaper and becomes more viable for the future too. The proposed work uses the user-item interaction matrix which is partitioned over the distributed environment. The system can distribute the new workload across nodes when new data and users are added. This ensures that the system can easily handle larger datasets. If required we can just add more inexpensive hardware instead of high-end servers. Thus, horizontal scaling ensures scalability in a much cheaper manner than vertical scaling.

Faster Training: With the use of parallel processing the algorithms itself can be parallelized. This expedites the calculation of values like similarities. Thus, in turn improving the speed of the training of algorithms many folds, enabling them to learn from extensive datasets quickly. This use of parallel processing confirms that similarities and recommendations are always updated with the latest user-item interactions. Our implementation uses parallel processing as each node has 4 CPU cores. This speed up the calculations of the similarities as larger calculations are broken down into smaller tasks. This can be easily ascertained using the speedup measure of the results.

Real-time Recommendations: When distributed and parallel data processing is combined, the prospect of real-time recommendations becomes a possibility. This ensures that all recommendations are always updated with the most recent trends of user preferences and item popularity. In the proposed solution Apache Spark has been used as the engine which provides in-memory processing. This further compliment the new formula by providing near real-time calculations, so that similarity values can be always recalculated if there is any change in user preferences and popularity of the items.

Complex Models: Such systems also allow the researchers to use more sophisticated methods like deep learning-based recommendation models. These models can emulate the behavior of the users and user-item relations much more efficiently. In our approach, due to the use of Apache Spark which supports a large number of data science and ML libraries. We can easily build further complex models.

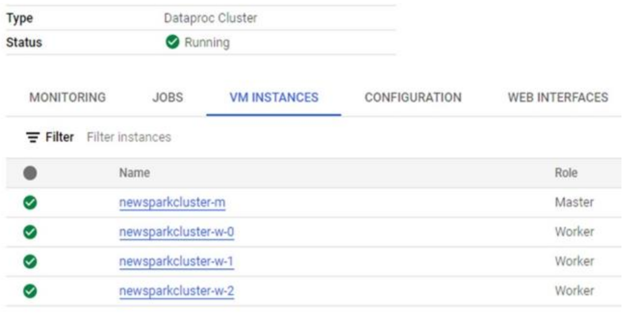
These implementation techniques have shown a good improvement in the execution time of cosine similarity; the results are discussed in detail in the next section.

5. EXPERIMENTS AND RESULTS

5.1 Experiments setup

The setup used in the experiment is the Google Dataproc Spark cluster. It has a 4-node cluster setup on the Google Cloud platform with 8GB of RAM and 4 cores for each node. The cluster has 1 master and 3 slaves. Apache Spark is the processing engine for executing the code.

Dataproc is a platform managed by Google Cloud platform. It provides Hadoop and Spark services. It is a very useful tool for batch processing, machine learning and stream processing. It is very user-friendly as it lets users create clusters and manage them using the Google Cloud platform dashboard. Figure 2 gives a screengrab of the VM instance list of the Google Cloud platform.



Type	Dataproc Cluster
Status	✓ Running

●	Name	Role
✓	newsparkcluster-m	Master
✓	newsparkcluster-w-0	Worker
✓	newsparkcluster-w-1	Worker
✓	newsparkcluster-w-2	Worker

Figure 2. Cluster setup in Dataproc

5.2 Datasets

Dataset used in the experiment is called MovieLens. It contains ratings given by a number of users from the MovieLens website. This data has been aggregated by the MovieLens website over a large amount of time. The dataset used in our work has 1000209 ratings provided by 6040 unique users for 3706 movies.

5.3 Measures of performance

The most common measures for determining the performance of a parallel system are as follows:

(1) Execution Time: It is the most basic and intuitive measure which tracks the time taken between submission of a job for similarity calculation and job completion.

(2) Speedup: This is a ratio of the execution time of an application on a single core and the execution time when the same application is executed using parallel computation [14]. It signifies the improvement in the execution time when using parallel computation. It is given in Eq. (5).

$$S(n) = \frac{T(n)}{T(n)} \tag{5}$$

here, the time of execution with one processor is T (1), and the execution time with n processors is T(n).

(3) Efficiency: The percentage of time during which a machine is effectively utilized in parallel computing. It is also calculated by dividing the speedup by the number of processors [28].

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{n * T(n)} \tag{6}$$

In the formula given above speedup is denoted as S(n), whereas the time of execution with one processor is T (1), and the execution time with n processors is T(n).

5.4 Results

Experiments were run with different data volumes for proposed cosine similarity Eq. (2) and existing cosine similarity Eq. (1).

(1) **Execution Time:** For calculating this measure, we executed the proposed cosine and original cosine both on the 4-node cluster and recorded execution time for 5k, 10k, 20k, 50k, 100k, 200k and 500k, 1M and 2M number of transactions. Execution time was recorded for different volumes and different number of partitions. The results are captured in the Table 3.

Table 3. Comparison of execution time

Data Volume	No. of Users	Partitions	Proposed Cosine Similarity[1]	Cosine Similarity[1]
		2	25.22	32.06
5001	2645	4	16.81	21.37
		6	20.17	25.64
		2	26.31	55.89
10000	3722	4	17.54	37.26
		6	21.05	44.71
		2	31.79	104.54
20000	4680	4	21.19	69.69
		6	25.4	83.5
		2	74.01	290.97
50000	5637	4	49.34	193.98
		6	59.21	232
		2	246.80	762.95
100000	5966	4	164.5	508.63
		6	197.4	610.46
		2	1123	2308
200000	6037	4	748.96	1539.43
		6	898	1847.32
		2	5245.4	9759
500000	6040	4	3503	6506
		6	4203	7807
		2	6834	19665
1000000	6040	4	4556	13110
		6	5467	15732
		2	15990	48375
2000000	20000	4	10660	32250
		6	12792	38700

Note: [1]unit of measurement of execution time is seconds.

Results clearly state that the proposed implementation of cosine similarity is much more efficient than the original implementation. As the data volume increases, the execution time increases for both approaches significantly. The difference between execution time is less when small volumes of data. However, for 2M rows of data, the execution time is more than 50 % less in the proposed solution.

Another observation that can be made is that increasing the number of partitions improves execution time for both methods. It can also be seen that 4 partitions are ideal for the dataset as using 2 partitions reduces the performance. However, increasing the number of partitions to 6 also causes the execution time to increase due to overheads. Further, this approach is also able to scale according to the increasing number of users. Figure 3 showcases that execution time growth for the proposed solution does increase exponentially with the increasing volume.

The authors [28] have used a 3-node cluster and worked on a maximum of 5000 transactions. So, we also set up one more 3-node cluster to compare with the results provided by the user. The results for execution time have been noted in Table 4 which also states that it is faster than the similar items calculated in the previous work.

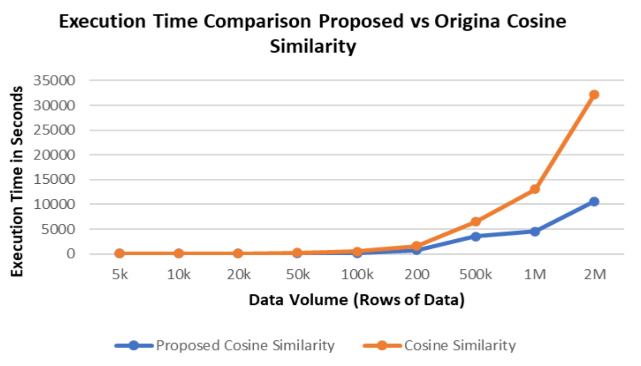


Figure 3. Execution time comparison

Table 4. Comparison of execution time 5k rows for 3 node cluster

Data Volume	Proposed Cosine Similarity[1]	FP-Growth Algorithm [28][1]
5001	22.25	50

Note: [1]unit of measurement of execution time in seconds.

(2) **Speed Up:** For calculating this measure, we executed data once on a single core of a CPU with memory (RAM) of 8 GB and then using a CPU with quad cores without any changes in the configuration of RAM for both proposed cosine similarity and original cosine similarity.

Table 5 shows that parallel implementation of any algorithm considerably speeds up the algorithm. However, our approach speeds up the similarity calculation 3 times against 2 times when using the original cosine formula.

Table 5. Comparison of speed up measure

Algorithm	T(1) Seconds	T(4) Seconds	S(4)=T(1)/T(4)
Proposed Cosine Similarity	55.64	16.81	3.31
Cosine similarity	47	21.37	2.20

(3) Efficiency is based on the speedup measure calculated above. Using the values in the Table 2.

Table 6. Comparison of efficiency measure

Algorithm	S(p), where p=4	p=4	E(4)=S(4)/4
Proposed Cosine Similarity	3.31	4	0.82
Cosine similarity	2.20	4	0.55

The results shown above in Table 6 prove that our approach has a better percentage of time during which a machine is effectively utilized in parallel computing. Parallel processing also ensures more efficient use of utilization of resources of each node of the cluster. This means that computational resources are more efficiently utilized in performing all the calculations and processing large volumes of data.

6. CONCLUSIONS

All the experimental results provided above showcase that the proposed method provides improved scalability and performance. These features are critical for a recommendation system to be considered useful in a real-world scenario. Given below are a few important considerations showcasing the usefulness of this approach:

(1) Improved execution time: In the above results it is clear that execution time is nearly reduced to half of that of the original formula. For example, in Table 3 for 2000000 rows of data execution time is 10000 seconds as compared to that of 32000 seconds in the original cosine formula.

(2) Better parallelization: The greater speedup factor noted in Table 5 also showcases that our implementation is very much suitable for parallel execution as increasing the number of cores reduces the execution time by a factor of more than 3 whereas the execution time in the traditional approach only improved by a factor of 2.

(3) Ability to handle more users: As the volume of the data is increased thereby increasing the number of users the results in Table 3 again show that the proposed method can manage increment in the data without a proportion increase in execution time.

(4) Better resource utilization: The proposed work showcases an improved efficiency of 0.82 compared to 0.55 of the original approach. This means that our approach utilizes the available resources more effectively than the current approach.

There are many papers which have focused on the parallel implementation of collaborative filtering. However, the focus has always been on using the direct implementations of existing algorithms. We have proved with the experiments that even adjusting the derivation of cosine similarity can tremendously improve execution time. Further, we can also safely state that the formula used in this paper has a better speedup. The use of the new formula also utilizes the resources much more efficiently. All these parameters remain consistently in favour of using the new formula.

Though the use of a suitable formula did improve the efficiency of the memory-based collaborative filtering, there were no changes done to improve the accuracy. We plan to focus next on improving the accuracy of such a system. This would provide us with more efficient and accurate novel approaches to make sense of the ever-growing data.

REFERENCES

[1] Bellogín, A., Said, A. (2019). Information retrieval and recommender systems. Data Science in Practice, 79-96. https://doi.org/10.1007/978-3-319-97556-6_5

[2] Shapira, B., Rokach, L., Ricci, F. (2022). Recommender Systems Handbook. Springer, Boston, MA, USA. https://doi.org/10.1007/978-0-387-85820-3

[3] Amatriain, X., Pujol, J.M., Oliver, N. (2009). I like it... i like it not: Evaluating user ratings noise in recommender systems. In International Conference on User Modeling, Adaptation, and Personalization, pp. 247-258. https://doi.org/10.1007/978-3-642-02247-0_24

[4] Koren, Y., Bell, R. (2015) Advances in collaborative filtering. In: Ricci, F., Rokach, L. and Shapira, B., Eds., Recommender Systems Handbook, Springer, Boston, 77-118. https://doi.org/10.1007/978-1-4899-7637-6_3

[5] Rendle, S. (2012). Factorization machines with libfm. ACM Transactions on Intelligent Systems and Technology, 3(3): 1-22. https://doi.org/10.1145/2168752 2168771

[6] Wikipedia. Collaborative filtering. https://en.wikipedia.org/w/index.php?title=Collaborative filtering& oldid=1190204039.

[7] Breese, J.S., Heckerman, D., Kadie, C. (2013). Empirical analysis of predictive algorithms for collaborative filtering. arXiv preprint arXiv:1301.7363. https://doi.org/10.48550/arXiv.1301.7363

[8] Herlocker, J., Konstan, J., Borchers, A., Riedl, J. (2017). An algorithmic framework for performing collaborative filtering. ACM SIGIR Forum, 8(51): 227-234. https://doi.org/10.1145/3130348.3130372

[9] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web, pp. 285-295. https://doi.org/10.1145/371920.372071

[10] Dean, J., Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1): 107-113. https://doi.org/10.1145/1327452.1327492

[11] Lu, J., Wu, D., Mao, M., Wang, W., Zhang, G. (2015). Recommender system application developments: A survey. Decision Support Systems, 74: 12-32. https://doi.org/ 10.1016/j.dss.2015.03.008

[12] Chen, R., Hua, Q., Chang, Y.S., Wang, B., Zhang, L., Kong, X. (2018). A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks. IEEE Access, 6: 64301-64320. https://doi.org/10.1109/ACCESS.2018.2877208

[13] Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, 22(1): 5-53. https://doi.org/10.1145/963770.963772

[14] Adomavicius, G., Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6): 734-749. https://doi.org/ 10.1109/TKDE.2005.99

[15] Mishra, N., Chaturvedi, S., Vij, A., Tripathi, S. (2021). Research problems in recommender systems. Journal of Physics: Conference Series, 1(1717): 012002. https://doi.org/10.1088/1742-6596/1717/1/012002

[16] Bobadilla, J., Ortega, F., Hernando, A., Gutiérrez, A. (2013). Recommender systems survey. Knowledge-Based Systems, 46: 109-132. https://doi.org/10.1016/j.knosys.2013.03.012

[17] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2000). Analysis of recommendation algorithms for e-commerce. In Proceedings of the 2nd ACM Conference on Electronic Commerce, pp. 158-167. https://doi.org/10.1145/352871. 352887

[18] Goldberg, K., Roeder, T., Gupta, D., Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval, 4: 133-151. https://doi.org/10.1023/A:1011419012209

[19] Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S. (2007). Collaborative filtering recommender systems. In the Adaptive Web: Methods and Strategies of Web Personalization, pp. 291-324. https://doi.org/10.1007/978-3-540-72079-9_9

[20] Linden, G., Smith, B., York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, 7(1): 76-80. https://doi.org/10.1109/MIC. 2003.1167344

[21] Varanasi, S. (2015). User-based recommendation algorithm on Hadoop cluster. Master's thesis, Auburn University.

[22] Bobadilla, J., Serradilla, F., Bernal, J. (2010). A new collaborative filtering metric that improves the behavior of recommender systems. Knowledge-Based Systems, 23(6): 520-528. https://doi.org/10.1016/j.knosys.2010.03.009

[23] Varanasi, C.P. (2015). Item-based recommendation algorithm using hadoop. Master's thesis, Auburn University.

[24] Siomos, T. (2019). Parallel implementation of basic recommendation algorithms. https://repository.ihu.edu.gr//xmlui/handle/11544/29406.

[25] Wang, Z., Yao, S. (2020). Scalable recommender system over MapReduce. https://api.semanticscholar.org/CorpusID:270555497.

[26] Verma, J.P., Patel, B., Patel, A. (2015). Big data analysis: Recommendation system with Hadoop framework. In 2015 IEEE International Conference on Computational Intelligence & Communication Technology, pp. 92-97. https://doi.org/10.1109/CICT.2015.86

[27] Bałchanowski, M., Boryczka, U. (2022). Collaborative rank aggregation in recommendation systems. Procedia Computer Science, 207: 2213-2222. https://doi.org/10.1016/j.procs.2022.09.281

[28] Dahdouh, K., Dakkak, A., Oughdir, L., Ibriz, A. (2019). Large-scale e-learning recommender system based on Spark and Hadoop. Journal of Big Data, 6(1): 1-23. https://doi.org/10.1186/s40537-019-0169-4

[29] Sun, N., Luo, Q., Ran, L., Jia, P. (2023). Similarity matrix enhanced collaborative filtering for e-government recommendation. Data & Knowledge Engineering, 145: 102179. https://doi.org/10.1016/j.datak.2023.102179

[30] Sheugh, L., Alizadeh, S.H. (2015). A note on pearson correlation coefficient as a metric of similarity in recommender system. In 2015 AI & Robotics (IRANOPEN), pp. 1-6. https://doi.org/10.1109/RIOS.2015.7270736

NOMENCLATURE

RS	Recommendation System
CF	Collaborative Filtering