Distributed Collaborative Filtering for Batch and Stream Processing-Based Recommendations

Kais Zaouali(✉), Mohamed Ramzi Haddad(✉), and Hajer Baazaoui Zghal(✉)

Riadi Laboratory, École Nationale des Sciences de l'Informatique,
University of Manouba, Manouba, Tunisia
zaouali.kais@gmail.com, haddad.medramzi@gmail.com,
hajer.baazaouizghal@riadi.rnu.tn

Abstract. Nowadays, user actions are tracked and recorded by multiple websites and e-commerce platforms, allowing them to better understand their preferences and support them with specific and accurate content suggestions. Researches have proposed several recommendation approaches and addressed several challenges such as data sparsity and cold start. However, the low-scalability problem remains a major challenge when handling large volumes of user actions data. This issue becomes more challenging when it comes to real-time applications. Such constraint requires a new class of low latency recommendation approaches capable of incrementally and continuously update their knowledge and models at scale as soon as data arrives. In this paper, we focus on the user-centered collaborative filtering as one of the most adopted recommendation approaches known for its lack of scalability. We propose two distributed and scalable implementations of collaborative filtering addressing the challenges and the requirements of batch offline and incremental online recommendation scenarios. Several experiments were conducted on a distributed environment using the MovieLens dataset in order to highlight the properties and the advantages of each variant.

Keywords: Distributed recommender systems
Collaborative filtering · Stream data processing
Batch data processing · Incremental learning

1 Introduction

Thanks to recommender systems, users are now assisted by personalized suggestions when searching for goods or content on the internet. Such systems help reduce the information overload problem and enable users to discover suitable choices and make better decisions. These suggestions are inferred by recommendation approaches based on the collected historical data describing users' actions and behaviors. In fact, historical data enable recommender systems and marketers to model users' consumption patterns and therefore better personalize and tailor their communications and offerings to each one based on their

H. Panetto et al. (Eds.): OTM 2018 Conferences, LNCS 11229, pp. 243–260, 2018.
https://doi.org/10.1007/978-3-030-02610-3_14

predicted future interests. Such smart systems are not only beneficial for users but also for service providers since they assist, support and implement their marketing strategy by inferring knowledge from the data they collect.

In the literature, various methods have been used in recommendation systems such as user-centered or item-centered collaborative filtering methods. User-centered collaborative filtering is based on the idea that a user can like items that have already been liked by similar users depending on their action history. The item-centered collaborative filtering is based on the idea that a user can like items similar to items he has previously liked. While collaborative filtering methods only use the users action history, other content-based methods have been used in recommendation systems to exploit the user demographics and the item characteristics. A third method family has been proposed in the literature and it is the hybrid method family that combines the two other method families.

In practice, the use of content-based methods requires user demographics (e.g. age, gender, country, etc.) or item characteristics (e.g. type, price, country of production, etc.) or both together. Since these data are not necessarily available on different recommendation systems, the use of collaborative filtering methods is becoming more and more popular. Thus, in this work we have chosen to focus on collaborative user-centered filtering and the problems that come with its use in current recommendation systems.

The major problems of collaborative filtering are cold-start, data sparsity and low scalability especially for the user-centered variant. Besides, the complexity of collaborative filtering and the fact that it requires processing all the historical data at once in a batch modes makes it resource heavy and leads to long processing times that become excessive with the growth of usage data. This explains the adopted strategy in production environments that resorts to discard all the learned models and reprocess all the available data periodically or when a given volume of usage data is collected. These collaborative filtering constraints makes it inadequate for real-time stream recommendations where suggestions must be provided with low latency in reaction to the latest observed behaviors. In this context, a stream based recommendation approach has the ability to better coordinate and synchronize the service/content provider's marketing and point of sale systems by ensuring they analyze and react to the same up-to-date view of each customer.

On the other hand, companies with common business purposes tend to share their users action history in order to better understand their preferences. Feeding a shared and cooperative recommendation system with user data from different sources further complicates the use of user-centered collaborative filtering techniques since many events (i.e. users actions) may occur from these sources and have to be shared and processed instantly.

In this work, we propose several variants and implementations of the user-centered collaborative filtering in order to make it able to handle the two recommendation scenarios that can face service providers, namely the online scenario for stream based real-time recommendation and the offline one based on a single

batch processing of historical data. All the proposed variants' implementations are distributed, resilient, fault tolerant and help reduce the processing time.

The first scenario addresses the continuous online recommendations based on consumption event streams that requires processing new incoming data with low latency in order to rapidly take into account the knowledge it carries. In order to achieve continuous stream recommendation, we resorted to making the collaborative filtering learning algorithm incremental and distributable. Indeed, distributed processing, using stateless procedures, increases the recommender system's horizontal scalability and ability to process more data simply by deploying new replicas of the computational nodes. Moreover, the incremental aspect of the proposed collaborative filtering variant avoids the need to reprocess all the historical data every time a new user action is observed. Therefore, new observations only require updating the part of the model or the subset of knowledge they affect.

The second scenario is related to offline, one time, batch recommendation use cases where large volumes of static data are available and at rest. In this context, we believe that batch recommendation algorithms are more adequate since they allow making several passes over the data which may lead to extract more knowledge when compared with incremental approaches making only one pass. Therefore, we propose a distributed implementation of user-centered collaborative filtering which is more suitable to one time offline learning and recommendation over big datasets.

We should point out that in this work we are careful to respect the privacy and the concerns of those to whom the data relate.

This paper is organized as follows. In Sect. 2, we present an overview of recommender systems and their underlying approaches. In Sect. 3, we present our proposed approach for scalable collaborative filtering with its two variants targeting (1) the online incremental stream processing and (2) the offline batch use cases. In Sect. 4, we evaluate our proposals and discuss the obtained results. Finally, in Sect. 5, we conclude the paper with some possible perspectives.

2 Related Works

Recommender systems are a class of personalization systems whose main objective is to predict users' interests towards the available informational content in the application domain. To achieve this goal, several approaches and methodologies were proposed in the literature [12,18]. Recommender systems based on collaborative filtering techniques use correlations in users' rating patterns in order to predict their interests. Collaborative filtering approaches are mainly classified into two categories: User-Centered Collaborative Filtering (UCCF) and Item-Centered Collaborative Filtering (ICCF).

UCCF is based on the idea that if two users have liked similar items in the past, they will probably have similar preferences and therefore look for the same items in the future.

UCCF first calculates the similarities between each pair of users through based on rating patterns in order to build neighborhood of similar users. From

there, the algorithm recommends to the current user the items that are appreciated by his neighborhood [12].

ICCF calculates the similarities between items based on their patterns. Then, items similar to the ones that the current user liked in the past will be recommended to him [15].

The main advantage of collaborative filtering techniques is that they do not require a pinpoint item description. Given that the recommendation process is based on user-item interactions, this technique makes it possible to apply in all application domains in order to recommend complex items without the need to describe or analyze them.

2.1 Scalable Stream Processing of Collaborative Filtering

Highly interactive online services (e.g. VOD websites such as netflix.com, MOD websites such as soundcloud.com) are increasingly popular on the Web. Such services can have a large amount of data characterizing consumers' behaviors since they interact continuously with these services and the items they provide. Since the process of neighborhood construction takes so long, it is then calculated periodically. If the recommendation system begins to analyze users' behaviors at a time t1 and finishes it at t2, the interactions that were observed during this period are not taken into consideration. The larger the amount of data to be analyzed, the greater the time between t1 and t2, and the more the newly collected data are ignored.

On the basis of this observation, several research studies have attempted to provide the necessary solutions to accommodate the continuous data flow problem. Recently, real-time recommendation systems are increasingly attracting researchers' attention [3,7].

Papagelis proposed an approach to solve the scalability problem [14]. His approach is based on incremental updates of the users' similarities, making it possible to calculate the similarities in an online application. Das discussed the problems issued from the large volumes of data and the accelerating content evolution [5]. He then proposed online templates in order to generate recommendations for users on Google News. Koren introduced a new neighborhood model with an improved accuracy [13]. The proposed approach takes into account recent actions. It is then more scalable than the previous method, without compromising recommendations relevance or other desired properties.

Diaz-Aviles and Chen provided users with a real-time thematic recommendation by analyzing social data flows [4,6]. However, the proposed systems have the inconvenience of not being able to support large volumes of data. StreamRec, proposed by Chandramouli, implements a recommendation model based on collaborative filtering in a scalable and incremental way allowing it to handle continuous data flows [2].

Huang was inspired by Papagelis [14] and StreamRec, to build TencentRec [11]. It is a general recommendation system that implements a series of algorithms in order to satisfy the different application requirements. The proposed approach takes into account various types of user action data, including implicit

and explicit actions as in StreamRec. Huang has used approximation techniques in this approach. In fact, even though Papagelis' method reduces the necessary computations for the similarities update, these calculations are still very expensive and cannot be done within a real-time delay.

2.2 Distributed Approaches for Batch Collaborative Filtering

Distributed collaborative filtering approaches started by the work of Tveit on UCCF [16]. The author used the Pearson correlation as a similarity measure whose computation was implemented in a peer-to-peer distributed architecture.

The UCCF has been implemented by Han [8,9] using distributed hash tables. Users were distributed over different computation nodes in a way that guarantees that each node handles users having in common at least one item rated similarly. This allowed to build local training sets and to calculate similarities between users belonging to the same computation node and then predict the missing ratings.

Later, more sophisticated approaches were proposed to associate traditional collaborative filtering techniques with other new ideas.

In order to calculate users' similarities, Xie [17] used a schema relying on distributed hash tables (DHT). The number of users considered in the similarity calculation process has been reduced by taking into account only those who have very similar preferences. Furthermore, in order to avoid rating biases, users' ratings have been normalized.

The approach proposed by Castagnos and Boyer in [1] made it possible to take into account the user implicit actions during the training step. Each user is characterized by a profile and an ID. Both allowed users to be categorized using Pearson correlation. In fact, for each user four lists of user ID have been defined: the most similar users, the ones with similarity exceeding the defined similarity threshold, the blacklisted users, and the users who added the active user in their profile.

2.3 Motivation and Objectives

Classic recommendation algorithms are complex and require a heavy processing footprint especially in the neighborhoods construction phase where each pair of items or users is processed in order to measure their similarity degree. Moreover, each time a user's attitude towards an item is observed (e.g. purchase or rating) the user's (resp. item's) neighborhood needs to be updated by recalculating the similarities with all the other users (resp. items).

Offline and batch execution often process large and complex static data and is more concerned with throughput than latency of individual components of the computation. This type of execution addresses the need to predict variables when it comes to making the prediction periodically on data batches.

Real-time recommendation systems are more demanding in terms of latency. They process data streams whose calculation is executed using elementary data

or a smallish window of recent data. This calculation is relatively simpler than that of batch data and must be carried out quasi-instantaneously.

To make predictions even simpler, one solution would be to carry out the calculations incrementally in order to reuse the results of the previously carried out calculations and infer the new predictions using fewer computation resources.

3 An Approach for Distributed Collaborative Filtering Supporting Stream and Batch Recommendations

Our proposal includes two implementations of the user-centered collaborative filtering algorithm. First, we detail the approach for processing batches or streams of unary or binary data using incremental calculations. This ensures that the last data collected is taken into account when generating recommendations, which is very well adapted to the real-time context that is found in several applications. Then, we address application domains relying on multi-valued data such as ratings to model users' attitudes by a distributed variant for offline recommendation on big data at rest, which do not plan to handle more user data before the end of the current recommendation process.

3.1 Distributed Incremental Collaborative Filtering for Stream Recommendation (DCFS)

The proposed approach targets use cases where users' behaviors are modeled using unary or binary variables. It is based on a distributed and incremental modeling and prediction of pair of users or items correlation, independently of the other pairs.

A recommendation system based on collaborative filtering has two main steps: users' similarity computation (or the items' one), and ratings' prediction. Since the computations' complexity lies in the similarity step, we have then focused on alleviate these computations in our approach, and for which we have chosen the Jaccard measure. This choice was made by taking into account how to incrementally upgrade the different similarity measures. In fact, the Jaccard measure has the fewest variables to update when it comes to making calculations incrementally, so this is the best measure for our approach.

The Jaccard measure (cf. Eq. 1) is based on calculating two correlations. The first correlation concerns the shared actions (i.e. set of actions made by both users) and the second concerns the union of actions (i.e. set of all actions performed by at least one of the two users). In the case of user-centered collaborative filtering, it can be reduced to merely calculating the number of common items for a pair of users, and the number of items present in their actions history.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

The Fig. 1 details the workflow of our proposal which contains two steps. It starts by generating users' correlation tuples in the first step and goes then ahead to calculate similarity degrees in a second step.

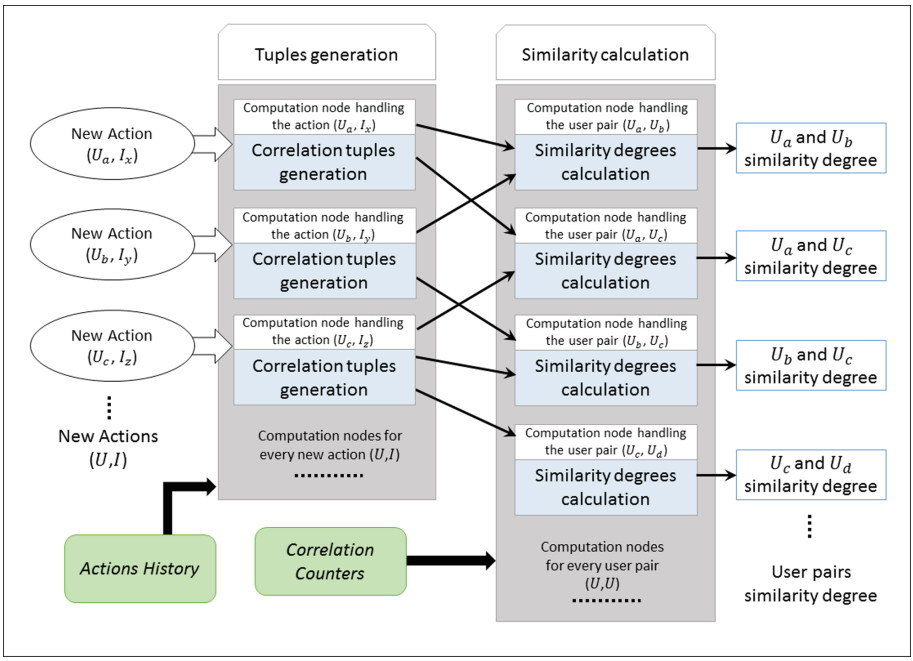

Fig. 1. Distributed incremental collaborative filtering

The calculation of the two correlation counters (intersection counter IC, and union counter UC) for a pair of users can be done in an incremental manner, and independently of the counters of the other pairs. This consists in calculating, for each new action between a user u_x and an item i_i, the correlation between u_x and any other user u_y with respect to i_i: if u_y has a recorded action for i_i, the intersection counter is incremented, otherwise the union counter is incremented.

These increments are carried out by generating correlation tuples in the form (User pair, Intersection counter, Union counter). For example, if the action (u_x, i_i) arrives to the recommendation system, the following correlation tuples will be generated:

- For each user u_y with the same action as u_x towards i_i, we generate the tuple $((u_x, u_y), 1, 0)$ which consists in incrementing the intersection counter.
- For any user u_y that does not have the same action as u_x towards the item i_i, we generate the tuple $((u_x, u_y), 0, 1)$ which consists in incrementing the union counter.

Then, for each user pair, we join their correlation tuples with their intersection and union counters. The correlation tuples of a user pair will allow updating the counters and subsequently infer the new similarity value between the two users:

$$Sim(A, B) = \frac{IC_{A,B}}{UC_{A,B}} \tag{2}$$

The following table (Table 1) illustrates an example of a user-item action matrix, where "x" denotes the user's action towards the item. Our proposal does not need to record this user-item matrix, it is only mentioned to illustrate the example.

Table 1. User-item action matrix

	I_1	I_2	I_3
U_1	-	-	x
U_2	-	x	x
U_3	x	x	-
U_4	x	-	x

At this level, the user u_1 shares the following counters with the users u_2, u_3 and u_4:

- $IC_{1,2} = 1$; $UC_{1,2} = 2$
- $IC_{1,3} = 0$; $UC_{1,3} = 3$
- $IC_{1,4} = 1$; $UC_{1,4} = 2$

If, for example, a new action (u_1, i_1) arrives to the recommendation system, the following correlation tuples will be generated:

- Intersection tuples: Users with the same action as u_1 towards i_1 are u_3 and u_4, therefore we generate the tuples ((u_1, u_3), 1, 0) and ((u_1, u_4), 1, 0).
- Union tuples: Only the user u_2 does not have the same action as u_1 towards i_1, therefore we generate the tuple ((u_1, u_2), 0, 1).

These generated tuples make it possible to update the correlation counters of u_1 with the other users, which will become: $UC_{1,2} = 3$; $IC_{1,3} = 1$; $IC_{1,4} = 2$.

Thus, the similarity degrees between u_1 and other users can be inferred: $Sim_{1,2} = 0,33$; $Sim_{1,3} = 0,33$; $Sim_{1,4} = 1$.

Note that our proposal allows processing static data in a batch mode, with one pass computation, by generating the tuples of correlation for all the actions at the same time. Then, tuples are grouped by user pairs allowing to calculate the sum of the increments' values in order to obtain two final counters for each user, and subsequently deducing the degree of similarity.

In both cases of use, our proposal offers a constant complexity for the similarity calculation after the arrival of a new user action and makes it possible to use the results of the previous calculations in order to perform the future ones.

3.2 Distributed Collaborative Filtering for Batch Recommendation (DCFB)

The incremental variant of our approach used a low complexity similarity measure to better fit the real-time recommendation requirements. Outside the real-time application context, other complex measures can be implemented since

they may outperform the Jaccard similarity measure by taking into account all the available data points every time two users need to be compared. Moreover, classic similarity measures makes it possible to handle all types of variables modeling user's actions and attitudes since they aren't restricted to binary or unary variables.

Batch processing allows accessing all the data at hand when a recommendation is needed. This offline processing provides the ability to manage multi-valued static data and calculate the exact degrees of similarity for collaborative filtering when using classic non-incremental measures. In this context, we propose a solution for batch recommendation by incorporating the most used similarity measurements and rating estimators.

The used similarity measures are the Cosine similarity (cf. Eq. 3) and the Pearson correlation (cf. Eq. 4).

$$sim_{x,y} = \frac{\overrightarrow{R_x} \cdot \overrightarrow{R_y}}{||\overrightarrow{R_x}||_2 \times ||\overrightarrow{R_y}||_2} = \frac{\sum_{i \in I} r_{x,i} \times r_{y,i}}{\sqrt{\sum_{i \in I} r_{x,i}^2} \times \sqrt{\sum_{i \in I} r_{y,i}^2}} \tag{3}$$

$$sim_{x,y} = \frac{\sum_{i \in I_x \cap I_y} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_x \cap I_y} (r_{x,i} - \bar{r}_x)^2} \times \sqrt{\sum_{i \in I_x \cap I_y} (r_{y,i} - \bar{r}_y)^2}} \tag{4}$$

Concerning the rating estimators, the following three estimators were adopted:

- The similarity-weighted average (SW) which predicts the user rating for an item based on the user's neighborhood ratings for this item.

$$r_{x,i} = \frac{\sum_{y \in U} sim_{x,y} \times r_{y,i}}{\sum_{y \in U} |sim_{x,y}|} \tag{5}$$

- The average based on the centered ratings (CR) which is based on biased ratings in order to avoid the differences in user rating behavior.

$$r_{x,i} = \bar{r}_x + \frac{\sum_{y \in U} sim_{x,y} \times (r_{y,i} - \bar{r}_y)}{\sum_{y \in U} |sim_{x,y}|} \tag{6}$$

- The average based on the standard deviation (SD) thereby compensating for users differing in ratings spread as well as mean ratings.

$$r_{x,i} = \bar{r}_x + \sigma_x \frac{\sum_{y \in U} sim_{x,y} \times (r_{y,i} - \bar{r}_y)/\sigma_y}{\sum_{y \in U} |sim_{x,y}|} \tag{7}$$

The general architecture of our proposal is detailed in Fig. 2. The first step allows preparing the necessary statistics for the similarities calculation and ratings predictions. Afterwards, the similarity between each pair of users is calculated according to the chosen similarity measure using the previously calculated statistics. The third step consists on predicting the missing ratings for each user based on his similarity degrees with the rest of the users and using a rating estimator. Finally, after predicting all missing ratings for a given user, a list of the best items is recommended.

In the following, we detail all the steps required to generate recommendations while highlighting their distributed algorithms.

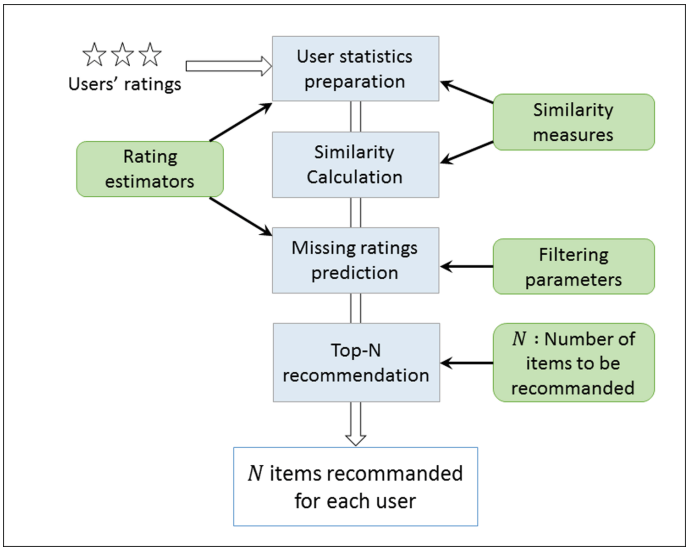

Fig. 2. Distributed collaborative filtering architecture

Users Statistics Preparation: The statistics that need to be computed for each user differ according to the rating estimator and the chosen measure of similarity. They can be the user mean rating $\bar{r}_i$, the Euclidean norm $||\overrightarrow{R_i}||_2$ or the user ratings' standard deviation σ_i. Table 2 summarizes the statistics that need to be calculated according to the used measures of similarity and rating estimators. The calculation is done by grouping ratings per user and then inferring the statistics of each user.

Similarity Calculation: This step focuses on calculating the similarity for each pair of users. First, users' ratings are grouped by items (i.e. ratings concerning the same item are redirected to the same calculation node). Then, for each item, correlation tuples are generated for each pair of users who have rated the item. The correlation tuples are defined as (User pair, T1, T2, T3) tuple.

Table 2. User statistics to be calculated

	Pearson correlation	Cosine similarity
Similarity-weighted average	User mean rating	Euclidean norm
Average based on the centered ratings	User mean rating	User mean rating Euclidean norm
Average based on the standard deviation	User mean rating Standard deviation	User mean rating Euclidean norm Standard deviation

Similarity measures are based on formulas that are of the form of $a/(b \times c)$. The three terms T1 (cf. Eq. 8), T2 (cf. Eq. 9) and T3 (cf. Eq. 10) calculate respectively the elementary values of a, b and c produced from a single item. These terms allow calculating the similarity degree between each pair of users (cf. Eq. 11).

$$T1 = \begin{cases} r_{x,z} \times r_{y,z} & if\ Cosine \\ (r_{x,z} - \bar{r}_x) \times (r_{y,z} - \bar{r}_y) & if\ Pearson \end{cases} \tag{8}$$

$$T2 = \begin{cases} null & if\ Cosine \\ (r_{x,z} - \bar{r}_x)^2 & if\ Pearson \end{cases} \tag{9}$$

$$T3 = \begin{cases} null & if\ Cosine \\ (r_{y,z} - \bar{r}_y)^2 & if\ Pearson \end{cases} \tag{10}$$

$$sim_{x,y} = \begin{cases} \frac{\sum T1}{||\vec{R_x}||_2 \times ||\vec{R_y}||_2} & if\ Cosine \\ \\ \frac{\sum T1}{\sqrt{\sum T2} \times \sqrt{\sum T3}} & if\ Pearson \end{cases} \tag{11}$$

The following user-item ratings matrix (Table 3) is used to illustrate examples for each component of our proposal for batch data:

Table 3. User-item ratings matrix

	I_x	I_y	I_z	I_w
U_a	$r_{a,x}$	$r_{a,y}$	-	-
U_b	$r_{b,x}$	$r_{b,y}$	$r_{b,z}$	-
U_c	$r_{c,x}$	-	$r_{c,z}$	-
U_d	-	$r_{d,y}$	$r_{d,z}$	$r_{d,w}$

The similarity calculation process for this user-item ratings matrix is described within the Fig. 3.

Missing Ratings Prediction: The third step predicts the ratings that users have not yet assigned. A filtering can be carried out in order to preserve only the most similar users for the active user. This filtering is performed either by pruning the degrees of similarity which are less than a similarity threshold t or by fixing the neighborhood size at n.

The similarity degrees of a same user are then grouped together to generate correlation tuples. For each item that the active user has not rated, and

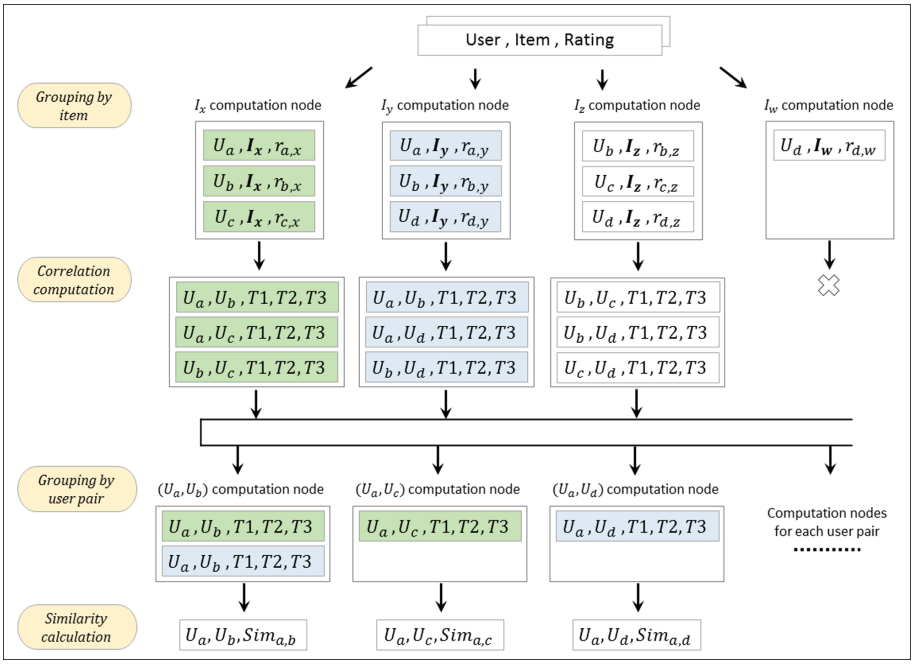

Fig. 3. Similarity degrees calculation

for each similar user who has rated the item, a correlation tuple is generated gathering data about the second user rating and the similarity degree between both users.

These tuples are defined as (Active user, Item not rated, Term T4 (cf. Eq. 12), Term T5 (cf. Eq. 13)) tuple. They allow predicting the missing user ratings through Eq. 14.

The Fig. 4 details the missing ratings prediction process for each user, based on similarity degrees already calculated.

$$T4 = \begin{cases} sim_{x,y} \times r_{y,z} & if\ SW \\ sim_{x,y} \times (r_{y,z} - \bar{r}_y) & if\ CR \\ sim_{x,y} \times \frac{r_{y,z} - \bar{r}_y}{\sigma_y} & if\ SD \end{cases} \tag{12}$$

$$T5 = |sim_{x,y}| \tag{13}$$

$$r_{x,z} = \begin{cases} \frac{\sum T4}{\sum T5} & if\ SW \\ \bar{r}_x + \frac{\sum T4}{\sum T5} & if\ CR \\ \bar{r}_x + (\sigma_x \times \frac{\sum T4}{\sum T5}) & if\ SD \end{cases} \tag{14}$$

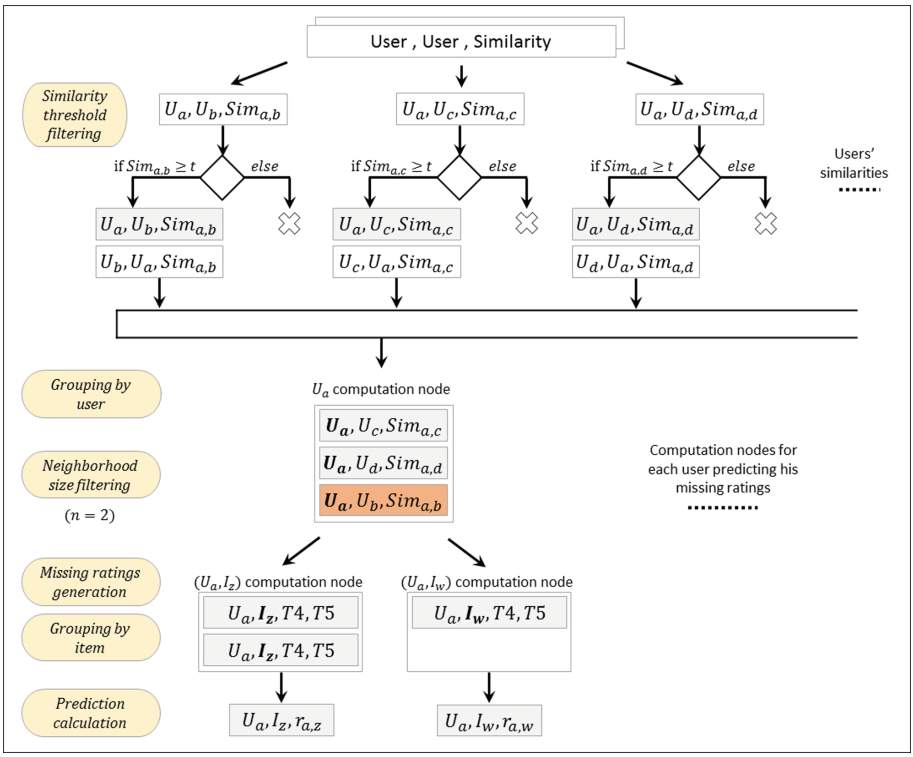

Fig. 4. Missing ratings prediction

4 Experimentation

In this section, we present the experimentation of our approach, detail the targeted application domain, and discuss the obtained results for the different variants that were evaluated.

4.1 Experimentation Setup

In this work, we adopted the MovieLens 1 m dataset [10] since it is the closest to our requirements with regard to the available variables. In fact, this dataset includes ratings assigned by a set of users to the movies they have watched. The dataset used in the following experiments includes 3.900 movies, 6.040 users and 1.000.209 ratings (1 to 5).

In order to evaluate our approaches relevance and performances, we computed the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) as quality measures and the Mean Execution Time (MET) as the evaluation metric of efficiency. The experiments were carried out on a cluster consisting of 16 processing machines, each having one core of 2.4 GHz.

$$MAE = \sqrt{\frac{1}{S_{Test}} \sum_{(u,i) \in S_T est} |\hat{r}_{u,i} - r_{u,i}|} \tag{15}$$

$$RMSE = \sqrt{\frac{1}{S_{Test}} \sum_{(u,i) \in S_T est} (\hat{r}_{u,i} - r_{u,i})^2} \tag{16}$$

4.2 Implemented Prototype

Our approach has been implemented using the Flink distribution platform. It is a distributed execution engine based on the MapReduce paradigm. Furthermore, it allows building execution plans that are much more complex than the simple combination of both map and reduce operations. Flink is also distinguished from other distributed execution engines by its ability to handle continuous data flows in a transparent and controllable way.

It's worth mentioning that any framework that is able to apply processing on a pipeline or data stream in a MapReduce way can be used to implement our proposal (e.g. samza, kafka streams, spark).

4.3 Incremental Collaborative Filtering Evaluation

We compared our incremental variant DCFS to the traditional non-distributed UCCF approach (using Jaccard measure) and DCFB (using Cosine-SW combination). We have experimented different numbers of processing machines which we used to draw the curve shown in Fig. 5. We have to mention that the results in this figure are for the similarity step only. In addition, we have to specify that for the DCFB and the non-distributed Jaccard the similarity of each pair of users was calculated at once (i.e. 18.237.780 similarities were calculated for the 6.040 users), whereas for the DCFS the similarities are calculated progressively as in a real-time recommendation scenario where at the arrival of each new user action the similarities concerned by the new action will be updated (i.e. 6.036.667.952 similarities were calculated or updated for the 6.040 users).

In practice, this means that the results of the DCFS concern the calculations made during the arrival of all the actions of the users, whereas the results of the two other approaches concern a single similarity calculation operation that have been made in order to maintain the accuracy of the similarity values. Then the DCFS calculates 331 times more similarity values compared to the other two approaches. So, to be able to compare these approaches we must divide the DCFS computation times by 331 to obtain the computation times which concern the same number of similarity values calculated by the DFCB or the non-distributed Jaccard.

The mean execution time of our approach is much better than the traditional approach, even when using a single processing machine. Indeed, this is due to the incremental aspect of our approach, which preserves the already performed calculations results and used them to infer the new calculations ones.

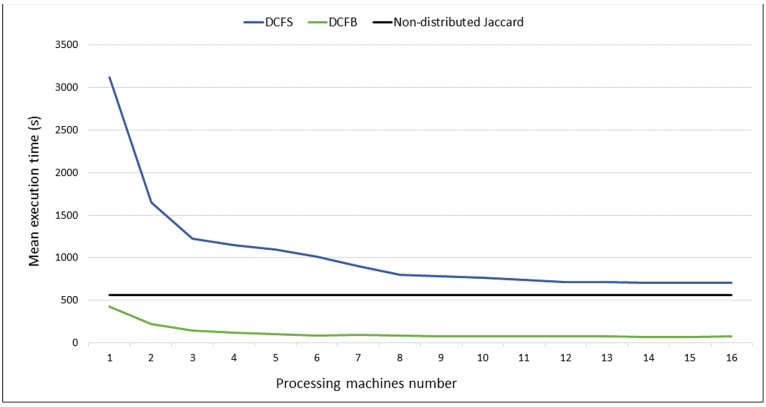

Fig. 5. DCFS mean execution time evaluation

By increasing the number of processing machines, the mean execution time of our approach decreases which validates the motivation behind distributing computations. However, this improvement is not as important as the one coming from the incremental calculations.

4.4 Batch Collaborative Filtering Evaluation

The experiments of the batch variant took into account the six combinations of the similarity measures (Cosine and Pearson) and the rating estimators (SW, CR and SD). These experiments were performed using 16 processing machines and taking into account all users without pruning non-similar users. The Table 4 enumerates MET (in seconds), MAE and RMSE results of these combinations.

The obtained results show that the CR estimator has the poorest prediction quality independently of the similarity measure. The best MAE and RMSE values were obtained respectively by the Cosine-SD and the Pearson-SD combinations. The mean execution time results led to the conclusion that the use of the Pearson correlation measure allows to have better performances than those involving the Cosine similarity. Concerning the rating estimators, SW offers the shortest execution time, followed by SD then CR. Thus, the most efficient combination was the Pearson-SW.

Table 4. DCFB and traditional UCCF evaluation

	P-SW	P-CR	P-SD	C-SW	C-CR	C-SD	UCCF (P-SW)
MAE	0.745	0.811	0.744	0.748	0.787	0.742	0.745
RMSE	0.910	0.998	0.909	0.913	0.971	0.911	0.910
MET	305	314	306	313	318	308	1063

The Pearson-SW combination of our batch variant has the same MAE and RMSE value as the UCCF (Pearson-SW) since it adopts the same algorithm, only distributing its calculations on different nodes with no approximation. In addition, our approach has a shorter mean execution time than the traditional approach. These results confirm that our approach has improved the execution time without information loss or decrease in prediction accuracy.

We also compared the performance of our batch variant with the traditional one in terms of mean execution time using different numbers of processing machines (cf. Fig. 6).

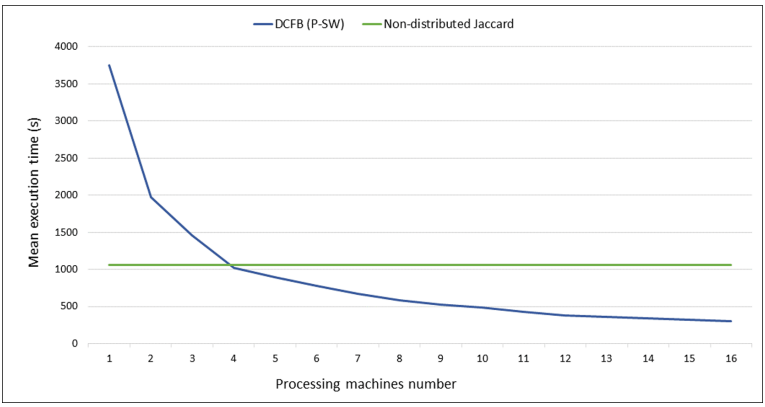

Fig. 6. DCFB mean execution time evaluation

We used in this experiment the Pearson measure and the similarity-weighted average rating estimator. From the obtained results, we can find that starting from the use of 4 processing machines, our approach provides a shorter mean execution time than the traditional approach.

5 Conclusion and Future Works

In this work, we proposed two distributed variants of the collaborative filtering approach for batch and stream processing-based recommendations. Our offline approach resorts to processing distribution in order to manage the large volume of batch data whereas the online one relies on an incremental processing of continuous data streams for online real-time recommendation. The experiments show that the proposed approaches improve the processing time in both scenarios and preserve recommendation quality.

In future works, we will focus on proposing new approximate similarity measures for multi-valued variables which are more adapted to incremental modeling and recommendation. Such measures should improve recommender systems performances without compromising the exactitude of their results or the real-time low latency constraints.

References

1. Castagnos, S., Boyer, A.: Personalized communities in a distributed recommender system. In: Amati, G., Carpineto, C., Romano, G. (eds.) ECIR 2007. LNCS, vol. 4425, pp. 343–355. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71496-5_32. http://dl.acm.org/citation.cfm?id=1763653.1763695
2. Chandramouli, B., Levandoski, J.J., Eldawy, A., Mokbel, M.F.: StreamRec: a real-time recommender system. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, pp. 1243–1246. ACM, New York (2011). http://doi.acm.org/10.1145/1989323.1989465
3. Chang, S., et al.: Streaming recommender systems. In: Proceedings of the 26th International Conference on World Wide Web, WWW 2017, International World Wide Web Conferences Steering Committee, pp. 381–389. Republic and Canton of Geneva, Switzerland (2017). https://doi.org/10.1145/3038912.3052627
4. Chen, C., Yin, H., Yao, J., Cui, B.: TeRec: a temporal recommender system over tweet stream. Proc. VLDB Endow. **6**(12), 1254–1257 (2013). https://doi.org/10.14778/2536274.2536289
5. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 271–280. ACM, New York (2007). http://doi.acm.org/10.1145/1242572.1242610
6. Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., Nejdl, W.: Real-time top-N recommendation in social streams. In: Proceedings of the Sixth ACM Conference on Recommender Systems, RecSys 2012, pp. 59–66. ACM, New York (2012). http://doi.acm.org/10.1145/2365952.2365968
7. Domann, J., Lommatzsch, A.: A highly available real-time news recommender based on apache spark. In: Jones, J.F. (ed.) CLEF 2017. LNCS, vol. 10456, pp. 161–172. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65813-1_17
8. Han, P., Xie, B., Yang, F., Shen, R.: A scalable P2P recommender system based on distributed collaborative filtering. Expert. Syst. Appl. **27**(2), 203–210 (2004). https://doi.org/10.1016/j.eswa.2004.01.003
9. Han, P., Xie, B., Yang, F., Wang, J., Shen, R.: A novel distributed collaborative filtering algorithm and its implementation on P2P overlay network. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 106–115. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24775-3_13
10. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. **5**(4), 19:1–19:19 (2015). http://doi.acm.org/10.1145/2827872
11. Huang, Y., Cui, B., Zhang, W., Jiang, J., Xu, Y.: TencentRec: real-time stream recommendation in practice. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD 2015, pp. 227–238. ACM, New York (2015). http://doi.acm.org/10.1145/2723372.2742785
12. Kluver, D., Ekstrand, M.D., Konstan, J.A.: Rating-based collaborative filtering: algorithms and evaluation. In: Brusilovsky, P., He, D. (eds.) Social Information Access. LNCS, vol. 10100, pp. 344–390. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90092-6_10
13. Koren, Y.: Factor in the neighbors: Scalable and accurate collaborative filtering. ACM Trans. Knowl. Discov. Data **4**(1), 1:1–1:24 (2010). http://doi.acm.org/10.1145/1644873.1644874

14. Papagelis, M., Rousidis, I., Plexousakis, D., Theoharopoulos, E.: Incremental collaborative filtering for highly-scalable recommendation algorithms. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS (LNAI), vol. 3488, pp. 553–561. Springer, Heidelberg (2005). https://doi.org/10.1007/11425274_57
15. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, WWW 2001, pp. 285–295. ACM, New York (2001). http://doi.acm.org/10.1145/371920.372071
16. Tveit, A.: Peer-to-peer based recommendations for mobile commerce. In: Proceedings of the 1st International Workshop on Mobile Commerce, WMC 2001, pp. 26–29. ACM, New York (2001). http://doi.acm.org/10.1145/381461.381466
17. Xie, B., Han, P., Yang, F., Shen, R.M., Zeng, H.J., Chen, Z.: DCFLA: a distributed collaborative-filtering neighbor-locating algorithm. Inf. Sci. **177**(6), 1349–1363 (2007). https://doi.org/10.1016/j.ins.2006.09.005
18. Zanitti, M., Kosta, S., Sørensen, J.: A user-centric diversity by design recommender system for the movie application domain. In: Companion Proceedings of the The Web Conference 2018, pp. 1381–1389, WWW 2018. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland (2018). https://doi.org/10.1145/3184558.3191580