Distributed Implementation of Latent Rating Pattern Sharing based Cross-domain Recommender System Approach

Anil Kumar*, Vikas Kapur*, Apangshu Saha*,
Rajeev Gupta* and Arun Singh*
*Samsung Research and Development Institute Delhi
Sec.126, Noida, UP, India
e-mail: anil.k06@samsung.com

Santanu Chaudhury† and Sumeet Agarwal†
†Indian Institute of Technology
Hauz Khas, Delhi 110016, India

***Abstract*—Latent rating pattern sharing based approaches for cross-domain recommendations can alleviate the data sparsity problem by pulling the knowledge available from other domains and are faster in prediction; however since the prediction quality depends on number of chosen user and item class sizes for given data-set, the model training time becomes prohibitively large even for medium size data-sets. In this paper, we propose a MapReduce based distributed implementation of the cross domain recommendation algorithm. Our implementation has the capability to run on modern distributed computing frameworks, such as Hadoop and Twister, that utilize commodity machines. The experimental results show that the training time increases only linearly with user and item class sizes when compared to the exponential increase in case of its sequential counterpart.**

***Keywords*-Cross-domain recommendation System Scalability; Big Data, Flexible Mixture Model; Transfer Learning; MapReduce**

I. Introduction

In today's era of information technology, user preference information is widely available but scattered in various domains such as TV apps, books, movies and electronics gadgets. The joint analysis of these preferences can help uncover the key relationship among users, items and their features. It also gives the opportunity to utilize the knowledge gained in one domain to improve the recommendation quality in other domains where preference information is sparse; this process of knowledge transfer is classified as a cross-domain recommendation [1]. However in most cases, these domains are usually mutually exclusive. The users, items and their features in one domain may not be same in other domains. As a result, it is difficult to directly link users, items or features between different domains. One approach to tackle such cases is through implicit linkage established via sharing of common cluster level latent rating patterns [2], [3] and [4].

In [2], Li et al. constructed a codebook by co-clustering the user-item ratings of source domain and then the target domain is reconstructed using the codebook to fill the missing entries. This approach uses hard clustering that leads to performance degradations. This problem is addressed using Flexible Mixture Model (FMM) in Rating-Matrix Generative Model (RMGM) [3]. RMGM learns the shared latent rating patterns in terms of latent user and latent item class parameters. Thereafter, these shared latent rating patterns are used to transfer the knowledge learned from the source domain to the target domain. RMGM simultaneously improves prediction performance in sparse domains. Since rating information is aggregated and analyzed, training times become prohibitively large even for medium-size data-sets. In this paper, we implement a distributed implementation of RMGM algorithm, which can run on a cluster of commodity machines thus speeding up the training process without compromising prediction accuracy. The prediction process is also distributed to speed up the prediction in all involved domains.

The rest of the paper is organized as follows: Section 2 provides details of the RMGM algorithm; Section 3 presents our distributed implementation of RMGM algorithm using MapReduce; and Section 4 presents experimental evaluation of our distributed implementation on real-world data-sets. Finally, Section 5 concludes the paper.

II. PRELIMINARIES

The rating pattern sharing based cross-domain recommendation approach RMGM [3] groups users and items separately based on ratings given by users to different items. It allows users and items to belong to different classes simultaneously. The model assumes that there are two latent variables: user class Z_U and item class Z_I. Fig. 1 shows the graphical representation of FMM [5], RMGM extends FMM for grouping the users and items simultaneously. In this figure shaded variables user u, item i and corresponding rating r corresponds to the observed variables, while non-shaded variables Z_I and Z_U corresponds to item and user class latent variables. Table I provides the definition of variables used to describe the model.

The joint probability $P(i, u, r)$ of the user *u*, item *i* and rating *r* is given by equation (1) which requires the values $P(Z_U^k)$, $P(Z_I^l)$, $P(u|Z_U^k)$, $P(i|Z_I^l)$ and $P(j|Z_I^l, Z_U^k)$ model

Table I
DEFINITIONS OF SYMBOLS

Symbol	Description
K	Number of user classes
L	Number of item classes
T	Total number of ratings in all domains
M	Total number of users in all domains
N	Total number of items in all domains
R	Rating scale *1...R* with assumption that the rating scale is same in all domains
$P(Z_U^k)$	Prior probability for user class *k*: $(1 \leq k \leq K)$
$P(Z_I^l)$	Prior probability for item class *l*: $(1 \leq l \leq L)$
$P(u\|Z_U^k)$	Conditional probability of a user *u* given the user class Z_U^k :$(1 \leq u \leq M)$
$P(i\|Z_I^l)$	Conditional probability of a item *i* given the item class Z_I^k :$(1 \leq i \leq N)$
$P(r\|Z_I^l, Z_U^k)$	Conditional probability of a rating *r* given the item class Z_U^l and user class Z_U^k :$(1 \leq r \leq R)$

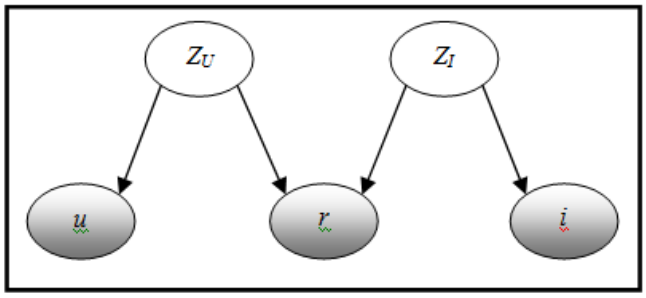

Figure 1. Graphical representation of FMM

parameters. These parameters are learned using training data-set.

$$P(i,u,r) = \sum_{Z_I^l, Z_U^k} P(Z_I^l)P(i|Z_I^l)P(Z_U^k)P(u|Z_U^k)P(r|Z_I^l, Z_U^k) \quad (1)$$

The overall rating of test user *u* for item *i* is predicted using equation (2).

$$Pred(u,i) = \sum_r r \frac{P(u,i,r)}{\sum_{r'} P(u,i,r')} \quad (2)$$

Learning Algorithm: All the model parameters $P(Z_U^k)$, $P(Z_I^l)$, $P(U_j|Z_U^k)$, $P(I_j|Z_I^l)$, $P(R_j|Z_I^l, Z_U^k)$ are estimated using Expectation Maximization (EM) [6] algorithm by maximizing the likelihood of the observed ratings. Following steps as described in [3] are performed iteratively until the solution converges:

1) **E-Step:** Calculate rating posterior probabilities using the equation (3), where *t* is the rating index in training data-set.

$$P(Z_U^k, Z_I^l|u_t, i_t, r_t) = \frac{P(u_t|Z_U^k)P(Z_U^k)P(i_t|Z_I^l)P(Z_I^l)P(u_t|Z_U^k)P(r_t|Z_U^k, Z_I^l)}{\sum_{k=,l=1}^{K,L} P(u_t|Z_U^k)P(Z_U^k)P(i_t|Z_I^l)P(Z_I^l)P(r_t|Z_I^l, Z_U^k)} \quad (3)$$

2) **M-Step:** Update the model parameters $P(Z_U^k)$, $P(Z_I^l)$, $P(u|Z_U^k)$, $P(i|Z_I^l,)$, $P(r|Z_U^k, Z_I^l)$ according to the equation (4)-(8) respectively.

$$P(Z_U^k) = \frac{\sum_{t=1}^{T}\sum_{l=1}^{L} P(Z_U^k, Z_I^l|u_t, i_t, r_t)}{T} \quad (4)$$

$$P(Z_I^l) = \frac{\sum_{t=1}^{T}\sum_{k=1}^{K} P(Z_U^k, Z_I^l|u_t, i_t, r_t)}{T} \quad (5)$$

$$P(u|Z_U^k) = \frac{\sum_{u_t \in u}^{T}\sum_{l=1}^{L} P(Z_U^k, Z_I^l|u_t, i_t, r_t)}{T \times P(Z_U^k)} \quad (6)$$

$$P(i|Z_I^l) = \frac{\sum_{i_t \in i}^{T}\sum_{k=1}^{K} P(Z_U^k, Z_I^l|u_t, i_t, r_t)}{T \times P(Z_I^l)} \quad (7)$$

$$P(r|Z_U^k, Z_I^l) = \frac{\sum_{r_t \in r}^{T} P(Z_U^k, Z_I^l|u_t, i_t, r_t)}{\sum_{t=1}^{T} P(Z_U^k, Z_I^l|u_t, i_t, r_t)} \quad (8)$$

Note that all these equations require the summation of various rating posterior probabilities $P(Z_U^k, Z_I^l|u_t, i_t, r_t)$ which provides opportunity to distribute computation and storage to make the training algorithm more scalable.

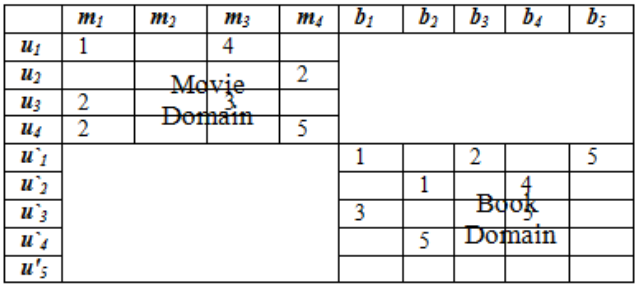

Figure 2. AGGREGATED DOMAINS

Cross-domain setting: For cross-domain recommendation multiple domains are aggregated diagonally into a single data-set as shown in Fig. 2. Users $u_1 \ldots u_4$ rated movies in movie domain and users $u'_1 \ldots u'_5$ rated books in books domain. Both rating matrices are aggregated, model variables are learned for given rating data-set and missing ratings are predicted using model parameters.

Li et al. [3] have demonstrated that the aggregation of multiple domains improve the recommendation quality in

the domain in which the data is sparse. The model was trained using EM which is an iterative process as discussed above and takes lot of time in training in case of big training data especially when user and item count is very large but rating matrix is sparse. All the model variables $P(Z_U^k)$, $P(Z_I^l)$, $P(u_t|Z_U^k)$, $P(i_t|Z_I^l,)$, $P(r_t|Z_U^k, Z_I^l)$ and posterior probabilities are tightly coupled and one depends on other in each iteration. Therefore, it impossible to distribute the iterations for scalability purpose but the distribution of computation and storage is possible within iterations.

We improved the algorithm in terms of scalability by distributing the EM computation through MapReduce paradigm. MapReduce is a distributed programming framework for developing applications to process vast amount computation and storage in a scalable and reliable manner using commodity machines [7]. The next section describes the distributed implementation of rating pattern sharing based cross-domain recommendation approach.

III. THE DISTRIBUTED IMPLEMETATION

The distributed implementation consists of a series of steps with one or more MR jobs chained together, each of which performs a specific operation. The complete model including initialization, training and prediction is distributed in various steps of implementation. The sequential implementation of the same algorithm is available at RMGM[1] and we use this sequential implementation to verify the results obtained in distributed implementation. The input, output and purpose of each step in distributed implementation is described as follows:

Step-1: *Initialize Rating Posterior and Rating Matrix:*

In this step rating posterior matrix is initialized for each rating record in training data-set. The initialization is carried out in following steps:

1) **Initialize Rating Posterior:** Fig. 3 depicts the flow for initializing the rating posterior matrix [user class *k*, item class *l*, *pIdx*, *pVal*]. User class and item class ids *k* and *l* are randomly generated for each observed rating record with index pIdx. The value *pVal* of posterior probability $P(Z_U^k, Z_I^l|u_t, i_t, r_t)$ is set to 1 initially indicating that each user, item and rating belongs to only single user and item class mixture but this value will get updated later in E-step as users and items may belong to multiple classes simultaneously. A MR job is executed which creates the sparse matrix representation of randomly created rating posterior matrix with user class id *k*as a **Key** and remaining part as a **Value** as shown Fig. 3. The rating posterior probabilities are distributed in sequence file as it will be used as an input to other MR jobs.

[1]https://sites.google.com/site/libin82cn/home/files/rmgm-em-codes.zip?attredirects=0

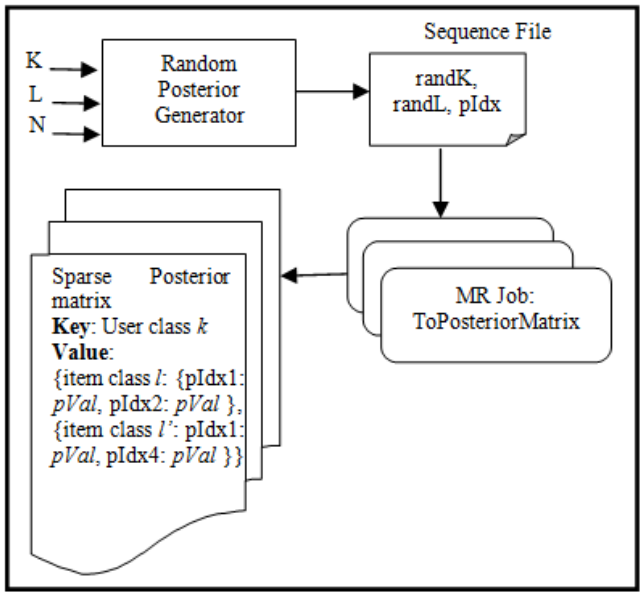

Figure 3. Initializing Rating Posterior

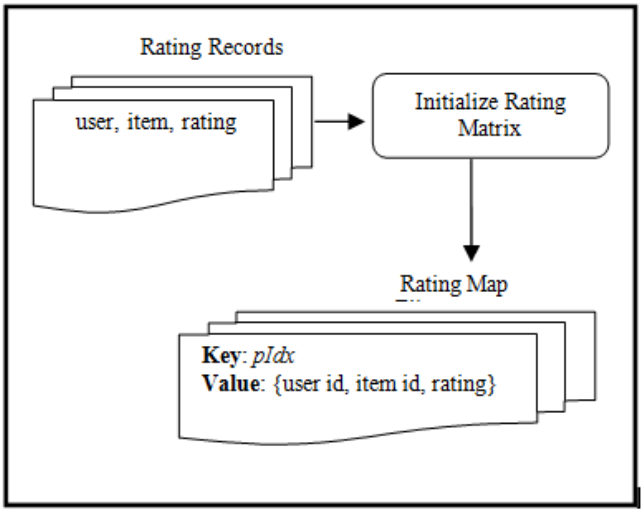

Figure 4. Initializing Rating Matrix

2) **Initializing Rating Matrix:** Fig. 4 depicts the flow for initializing the rating matrix. Rating matrix map file is created corresponding to the rating matrix. This map file contains the mapping of posterior index with rating record as shown in Fig. 3. This map file is used in step 2 to access rating record information by posterior index *pIdx*. This step also creates sequence file containing map file data information which is used later as a MR job input.

Step-2: *EM Iterations*

EM steps are executed iteratively to learn the values of $P(Z_U^k)$, $P(Z_I^l)$, $P(u|Z_U^k)$, $P(i|Z_I^l,)$, $P(r|Z_U^k, Z_I^l)$ model

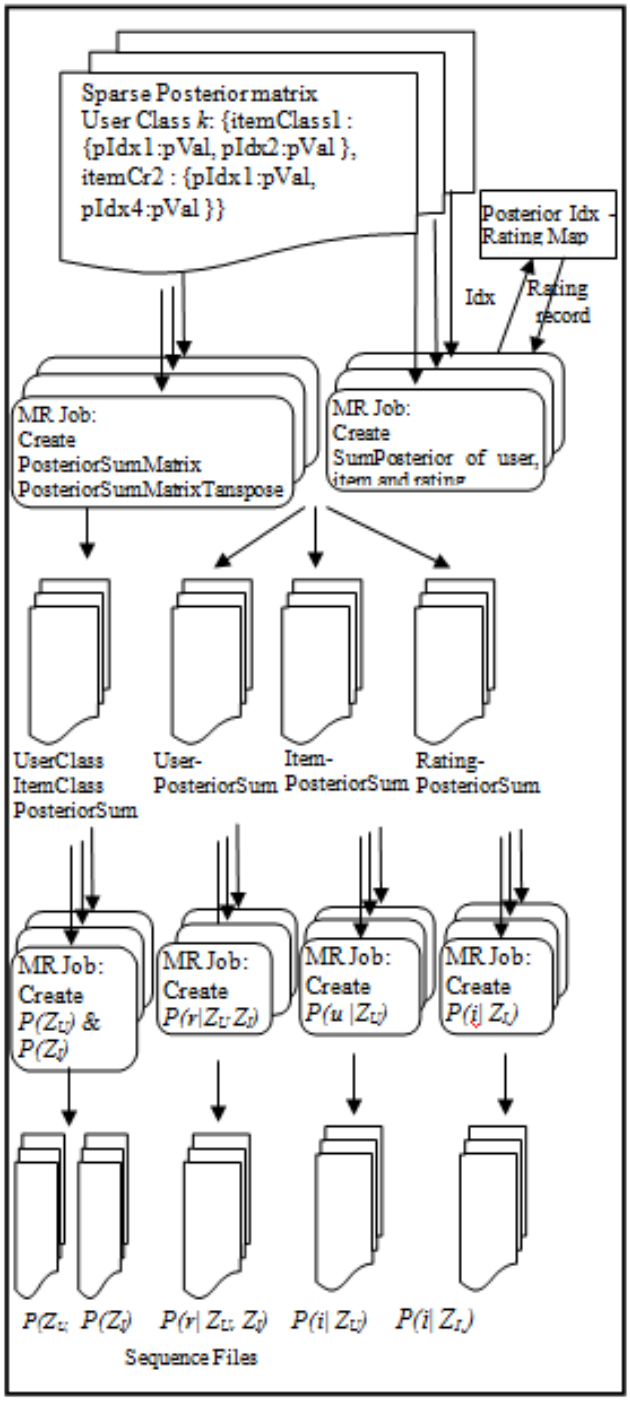

Figure 5. M-step

parameters. These steps are repeated again and again until the algorithm converges.

M-step: Fig. 5 shows the MR operations performed in M-step to update the model parameters from sparse posterior matrix and rating matrix:

1) Creating $P(Z_U^k)$ using matrix equation (4) and $P(Z_I^l)$ matrix using equation (5) matrices. In this step four MR jobs are executed:
   a) Create PosteriorSumMatrix Job
      Input: Sparse representation of posterior matrix (created in Step 1).

      Output: Sparse user-item class matrix containing posterior sum.
   b) Create PosteriorSumMatrixTanspose Job
      Input: Sparse user-item class matrix containing posterior sum.
      Output: Sparse item-user class matrix containing posterior sum.
   c) Create $P(Z_U^k)$
      Input: Sparse user-item class matrix containing posterior sum.
      Output: $P(Z_U^k)$ matrix
   d) Create $P(Z_I^l)$ Job
      Input: Sparse item-user class matrix containing posterior sum.
      Output: $P(Z_I^l)$ matrix

2) Creating $P(u, Z_{Uk})$ (using (5)), $P(i|Z_{Il})$, (using equation (6))) and $P(r|Z_U^k, Z_I^l)$ (using (7)) matrices In this step four MR jobs are executed:
   a) Create SumPosterior matrix of user, item and rating
      Input: Sparse representation of posterior matrix (created in Step1-a)
      Output: User Posterior Sum Matrix, Item Posterior Sum Matrix and Rating Posterior Sum Matrix
      Description: Each Mapper reads a row of input posterior matrix, it then queries the Rating map file (created in Step1-b), to get the rating record corresponding to posterior index. It then writes multiple outputs corresponding to each user, item and rating. Reducer accumulates the posterior sum corresponding to each user, item and rating.
   b) Create $P(i|Z_U^k)$ Job
      Input: User Posterior Sum Matrix, Global Posterior Sum Matrix (Created in Step 2: M-step–a-i)
      Output: Sparse $P(j|Z_U^k)$ Matrix
   c) Create $P(I_j|Z_I)$ Job
      Input: Item Posterior Sum Matrix, Global Posterior Sum Matrix (Created in Step 2: M-step–a-i)
      Output: Sparse $P(i|Z_I^l)$ Matrix
   d) Create $P(r|Z_U^k, Z_I^l)$ Job
      Input: Rating Posterior Sum Matrix, Global Posterior Sum Matrix (Created in Step 2: M-step–a-i)
      Output: Sparse $P(r|Z_U^k, Z_I^l)$ Matrix

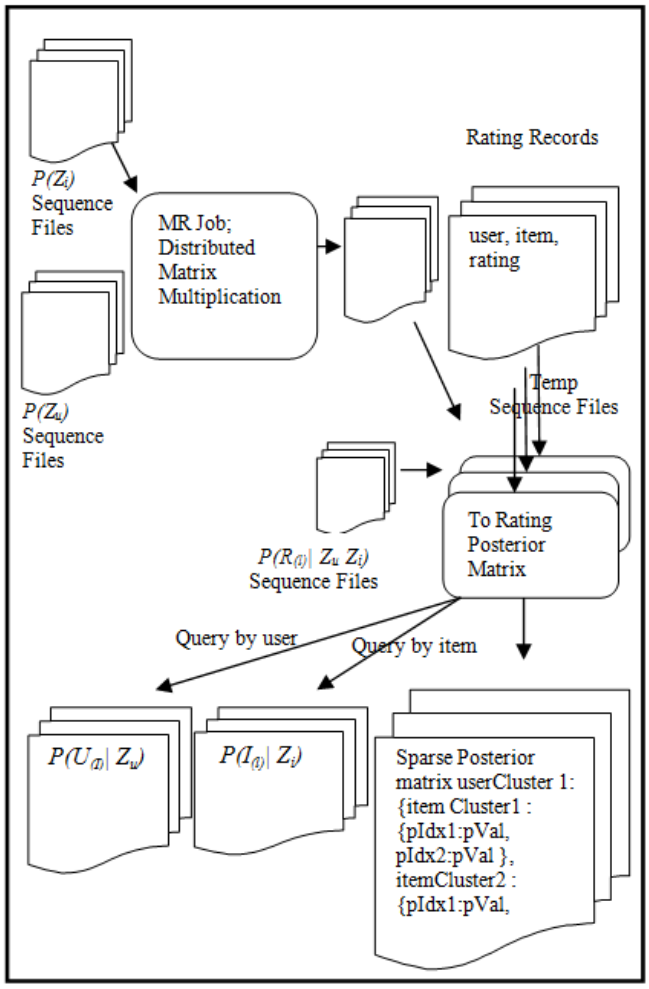

Figure 6. E- step

E-step: Fig. 6 describes the MR operation for E-steps to update the rating posterior matrix:

1) Creating intermediate matrix Temp from $P(Z_U^k)$ and $P(Z_I^l)$ matrices. In this step two MR jobs are executed:
   a) Transpose $P(Z_U^k)$ Matrix Job

      Input: Sparse $P(Z_U^k)$ matrix
      Output: Sparse transposed $P(Z_U^k)$ matrix.
   b) Matrix Multiplication Job
      Input: Sparse transposed $P(Z_U^k)$ matrix and sparse $P(Z_I^l)$ matrix.
      Output: Sparse Temp matrix. Generated as a result of distributed multiplication of transposed $P(Z_U^k)$ and $P(Z_I^k)$ matrices.
2) Creating rating posterior matrix from Temp, $P(u|Z_U^k), P(i|Z_I^l), P(r|Z_U^k, Z_I^l)$ matrices. In this step one MR jobs is executed:
   a) To Rating Posterior Matrix Job

      Input: Rating sequence file (Created in Step1-b), Temp and $P(r|Z_U^k, Z_I^l)$ (Loaded in Job setup), $P(u|Z_U^k)$ and $P(i)|Z_I^k)$ (Map file queried per userid or itemid)
      Output: Sparse rating posterior matrix

Step 3: *Creating intermediate matrices for prediction*

Fig. 7 demonstrates the MR jobs for creating intermediate matrices: user membership, item-membership matrices using five learned model parameters for predicting the ratings.

1) Create Intermediate matrix from $P(r|Z_U^k, Z_I^l)$ matrix In this step one MR jobs is executed:
2) Create Intermediate Matrix Job
   Input: $P(r|Z_U^k, Z_I^l)$ matrix (Created: M-step)
   Output: Sparse Intermediate matrix
3) Create user membership from $P(u|Z_U^k), P(Z_U^k)$ matrices. In this step one MR jobs Create UserMembership Job is executed:

   Create UserMembership Job
   Input: $P(U_j|Z_U)$ matrix (Created M-step) and $P(Z_U)$ matrix (Created in M-step)
   Output: Sparse UserMembership matrix
4) Create item membership from $P(i|Z_I^l)$, $P(Z_I^l)$ matrices. In this step one MR job Create Item Membership Job is executed:
   Input: $P(i|Z_I^l)$ matrix (Created in M-step) and $P(Z_I^l)$ matrix (Created in M-step)
   Output: Sparse ItemMembership matrix

Step 4: *Generating recommendations using equation* (1) *and* (8)

Fig. 8 depicts the flow for generating recommendations. The user-membership matrix, intermediate matrix and item-membership matrix are multiplied through distributed MR jobs to produce user-item prediction matrix. Finally Filter MR job is executed which takes user-item prediction matrix and original rating matrix (Created in Step 1-b) as input and filter the contents already consumed by users from user-item prediction matrix.

IV. EXPERIMENTS

In this section we examine how our proposed distributed implementation behaves on real world data-set and compare its performance with the sequential version of RMGM.

Experimental Setup: Experiments were performed using seven nodes, each on a separate commodity machine in the Hadoop cluster. Each machine was equipped with Intel Core i3-2100 CPU@ 3.10 GHz, 64 bit. 8 GB RAM. The total capacity of HDFS was 2.42 TB and the HDFS block size was 67 MB. Both versions of the RMGM algorithm were

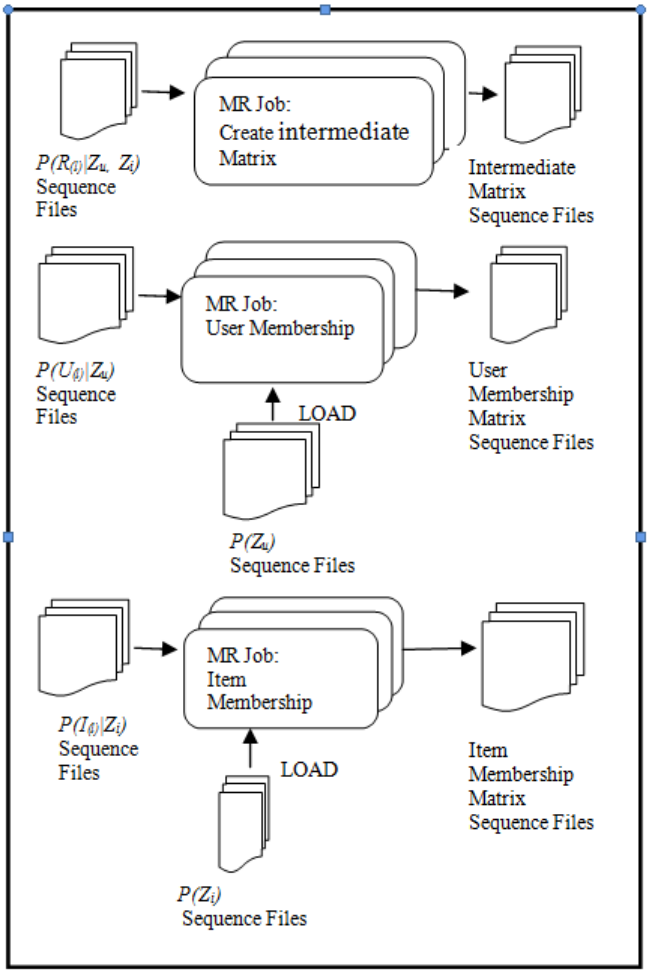

Figure 7. Creating intermediate, user-membership, item-membership matrices

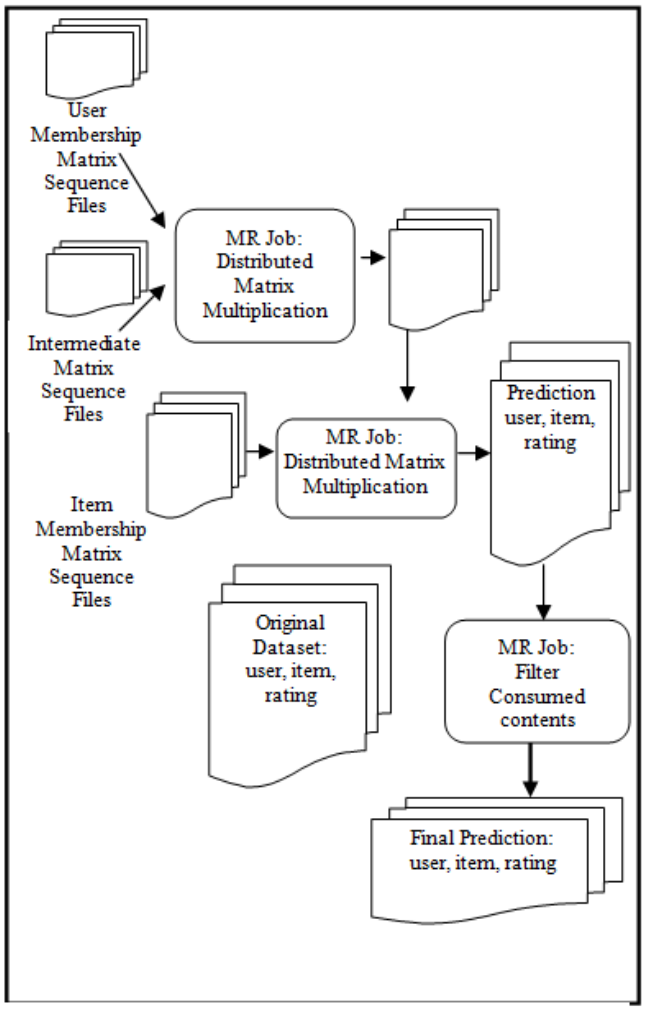

Figure 8. Generating recommendations using intermediate, user-membership, item-membership matrices

implemented in Java with the use of efficient data structures for time and memory usage.

Data-set: Experiments were conducted on a combined data-set containing movies and books domain data. The movie domain data-set is obtained from the Movielens[2]and book domain data-set, Book-crossing[3]. The movie data-set comprises 100,000 ratings on scale 1-5 provided by 943 users on 1682 movies while Book Crossing data-set comprises 67,552 ratings on scale 1-10 provided by 1395 users on 10279 books. The book crossing data-set was converted from rating scale 1-10 to scale 1-5 for the experiment. Therefore total of 167,552 ratings were used for training and testing. 20 percent data was randomly chosen for testing and remaining data was used for training.

Experimental Results

Fig. 9 shows the run time in distributed and sequential implementation for first two iterations of the EM algorithm with different user and item class sizes on same cross domain data-set. The algorithm converges on different iteration for different user and item class sizes, so we choose to use time for first two iterations to demonstrate the training time saved with the proposed distributed implementation. For smaller class sizes such as 50 for both user and item class size, sequential implementation performed slightly better in terms of training time because of I/O overhead in distributed implementation on Hadoop. But the prediction accuracy in terms of Root Mean Square Error (RMSE) was only 0.9919 after convergence as depicted in Fig. 10. As we increased the class sizes(where both user and item class sizes were 110), the RMSE improved to 0.9227 and training time for two iterations also reduced to 6.32 hours, while the sequential version took 122.54 hours. The experiment with user and item class sizes 110 could not run on sequential version due to memory limitation.

[2]http://grouplens.org/datasets/movielens/

[3]http://www.informatik.uni-freiburg.de/cziegler/BX/

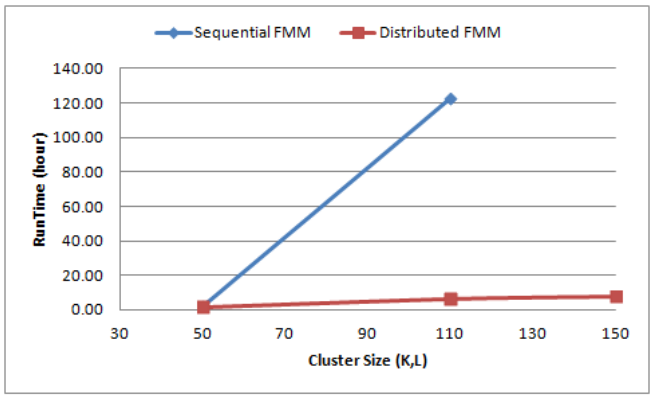

Figure 9. Sequential/Distributed run time vs user and item class sizes for for first two iterations

Figure 10. RMSE vs user and item class sizes (Optimal RMSE 0.92 is achieved with user and item class sizes 110

As we can see in Fig. 9, the distributed version started performing better with increased user and item class sizes. At the same time the completion time increased linearly when compared to the exponential increase with the sequential version. This happened because MR jobs are distributed with class ids, it means bigger is the class size, the more is the distribution of storage and computation which results in lesser run time.

Fig. 11 shows the run time for various number of nodes in the Hadoop cluster number of node in Hadoop cluster with [110, 110] user and item class size. As we can observe, the performance improvement is proportional to the number of nodes in the cluster but the run time doesnt improve much after 4 nodes.

Proof of correctness: The distributed implementation partitioned the data and performed the computation on individual parts before aggregating them into final result. The Rating Posterior matrix is updated in E-step in both sequential and distributed implementation. In one experiment we stated with same configuration: ratings dataset, number of user and item cluster sizes and Rating Posterior matrix. After each

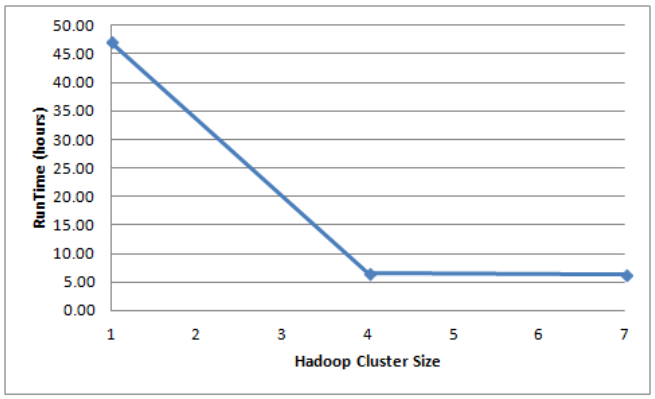

Figure 11. Run time vs Hadoop Cluster size (Run time for first two iterations with user and item class sizes as 110

iteration updated Rating Posterior matrix was same in both sequential and distributed implementation. This confirms that the distributed did not lose any information.

V. Conclusion

The proposed distributed approach solved the problem of scalability over sequential approach by distributing the RMGM into a series of steps with one or more MapReduce jobs chained together, each of which performs a specific operation. The computation as well as storage is distributed as map and reduce phases of MapReduce. The approach used in sequential implementation was not scalable because with the increase in user and item class sizes, its computation time was growing exponentially even with medium size data-set. Therefore the proposed distributed implementation can act as scalable algorithm for latent rating pattern sharing based cross-domain recommendations where multiple domain data leading to large data-set is involved in learning.

References

[1] I. Fernndez-Tobas, I. Cantador and M. Ka, "Cross-domain recommender systems: A survey of the state of the art.," in 2nd Spanish Conference on Information Retrieval (CERI 2012), 2012.

[2] B. Li, Q. Yang, and X. Xue, Can Movies and Books Collaborate? Cross-Domain Collaborative Filtering for Sparsity Reduction, Proc. 21st International Joint Conf. Artificial Intelligence, July 2009.

[3] B. Li, Q. Yang, and X. Xue, Transfer Learning for Collaborative Filtering via a Rating-Matrix Generative Model, Proc. 26th International Conf. Machine Learning, June 2009.

[4] S. Gao, H. Luo, D. Chen, S. Li, P. Gallinari and J. Guo, "Cross-Domain Recommendation via Cluster-Level Latent Factor Model.," in ECML/PKDD, 2013.

[5] S. Lu and J. Rang, "Flexible Mixture Model for Collaborative Filtering," in 20th Intl Conf. Machine Learning, 2003.

[6] D. A., L. N. and R. D., "Maximum Likelihood from incomplete data via the EM algorithm," Journal of the Royal Statistical Society, vol. B, no. 39, pp. 1-38, 1977.

[7] J. Dean and S. Ghemawa, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 106-113, 2008.