Distributed Scalable Collaborative Filtering Algorithm

Ankur Narang, Abhinav Srivastava, and Naga Praveen Kumar Katta

IBM India Research Laboratory, New Delhi
{annarang,abhin122,nagapkat}@in.ibm.com

Abstract. Collaborative filtering (CF) based recommender systems have gained wide popularity in Internet companies like Amazon, Netflix, Google News, and others. These systems make automatic predictions about the interests of a user by inferring from information about like-minded users. Real-time CF on highly sparse massive datasets, while achieving a high prediction accuracy, is a computationally challenging problem. In this paper, we present a novel design for soft real-time (less than 10 *sec.*) distributed co-clustering based Collaborative Filtering algorithm. Our distributed algorithm has been optimized for multi-core cluster architectures using pipelined parallelism, computation communication overlap and communication optimizations. Theoretical parallel time complexity analysis of our algorithm proves the efficacy of our approach. Using the Netflix dataset (100M ratings), we demonstrate the performance and scalability of our algorithm on 1024-node Blue Gene/P system. Our distributed algorithm (implemented using OpenMP with MPI) delivered training time of around $6s$ on the full Netflix dataset and prediction time of $2.5s$ on $1.4M$ ratings ($1.78\mu s$ per rating prediction). Our training time is around $20\times$ (more than one order of magnitude) better than the best known parallel training time, along with high accuracy (0.87 ± 0.02 RMSE). To the best of our knowledge, this is the best known parallel performance for collaborative filtering on Netflix data at such high accuracy and also the first such implementation on multi-core cluster architectures such as Blue Gene/P.

1 Introduction

Collaborative filtering (CF) is a subfield of machine learning that aims at creating algorithms to predict user preferences based on past user behavior in purchasing or rating of items [13], [15]. Here, the input is a set of known item preferences per user, typically in the form of a user-item ratings matrix. This $(user * item)$ ratings matrix is typically very sparse. The Collaborative Filtering problem is to find the unknown preferences of a user for a specific item, i.e. an unknown entry in the ratings matrix, using the underlying collaborative behavior of the user-item preferences. Collaborative Filtering based recommender systems are very important in e-commerce applications. They help people find more easily, items that they would like to purchase [16]. This enhances the user experience which typically leads to improvements in sales and revenue. Such systems are also increasingly important in dealing with information overload since they can lead users to information that others like them have found useful. With massive data rates in telecom, finance and other industries, there is a strong need to deliver soft real-time training for CF as it will lead to further increase in customer experience and revenue generation. Hence, soft real-time CF (with less than 10 *sec.*) based recommender systems are very useful.

Typical approaches for CF include matrix factorization based techniques, correlation based techniques , co-clustering based techniques and concept decomposition based techniques. Matrix factorization [17] and correlation [5] based techniques are computationally expensive hence cannot deliver soft real-time CF. Further, in matrix factorization based approaches, updates to the input ratings matrix leads to non-local changes which leads to higher computational cost for online CF. Concept Decomposition based technique [1] perform spherical k-means followed by least-squares based approximation of the original matrix. This work presents only sequential performance of 13.5 minutes for training of the full Netflix dataset which is far from being considered soft real-time. Co-clustering based techniques [8], [6] have better scalability but have not been optimized to deliver high throughput on massive data sets. [6] presented dataflow parallelism based co-clustering implementation which did not scale beyond 8 cores due to cache miss and in-memory lookup overheads. Moreover, CF over highly sparse data sets leads to lower compute utilization. Further, for large scale distributed / cluster environment (256 nodes and beyond), communication cost can dominate the overall performance and the communication cost becomes worse with increasing size of the cluster, leading to performance degradation. Thus, high computational demand, low parallel efficiency (due to cache overheads and low compute utilization) and high communication cost are the key challenges to achieving high throughput distributed Collaborative Filtering on highly sparse data sets.

In order to optimize the parallel performance, achieve high parallel efficiency and give near real time guarantees, we optimized our distributed algorithm using pipelined parallelism, compute communication overlap and communication optimizations (including topology mapping, steiner node for communication time reduction) for massively parallel multi-core cluster architectures such as Blue Gene/P [1]. In order to maintain high parallel efficiency, our algorithm makes compute vs. communication tradeoffs at various phases of the algorithm. Analytical parallel time complexity analysis proves the scalability provided by our performance optimizations as compared to the naive MPI based approach that has been used in all prior implementations. We evaluated our parallel CF algorithm on the prestigious Netflix Prize data set [3]. Netflix provides around $100M$ ratings (on a scale from 1 to 5 integral stars) along with their dates from 480189 randomly-chosen, anonymous subscribers on 17770 movie titles. On this dataset, we test the hybrid(MPI+OMP) parallel version of our optimized algorithm. We demonstrate around $20\times$ CF performance (including training time) over the full Netflix dataset as compared to the best prior parallel approaches.

This paper makes the following key contributions:

– We present the design of a novel distributed co-clustering based Collaborative Filtering algorithm for soft real-time (less than 10 *sec.*) performance over highly sparse massive data sets on multi-core cluster architectures. Our algorithm involves performance optimizations such as pipelined parallelism, computation communication overlap and communication optimizations (including topology mapping and steiner nodes for communication cost reduction).
– Analytical parallel time complexity analysis, theoretically establishes the improvement in performance and scalability using our algorithm.

[1] www.research.ibm.com/bluegene

– We demonstrate soft real-time distributed CF using the Netflix Prize dataset on a 1024-node Blue Gene/P system. We achieved a training time of around $6s$ with the full Netflix dataset and prediction time of $2.5s$ on $1.4M$ ratings with RMSE (Root Mean Square Error) of 0.87 ± 0.02. This is around $20\times$ (more than one order of magnitude) better than the best known parallel training time [6] along with high accuracy. To the best of our knowledge, this is the highest known distributed performance at such high accuracy. Our algorithm also demonstrates high scalability for large number of nodes on MPP architectures.

2 Related Work

Typical CF techniques are based on correlation criteria [5] and matrix factorization [17]. The correlation-based techniques use similarity measures such as Pearson correlation and cosine similarity to determine a neighborhood of like-minded users for each user and then predict the users rating for a product as a weighted average of ratings of the neighbors. Correlation-based techniques are computationally very expensive as the correlation between every pair of users needs to be computed during the training phase. Further, they have much reduced coverage since they cannot detect item synonymy. The matrix factorization approaches include Singular Value Decomposition (SVD [14]) and Non-Negative Matrix Factorization (NNMF) based [17] filtering techniques. They predict the unknown ratings based on a low rank approximation of the original ratings matrix. The missing values in the original matrix are filled using average values of the rows or columns. Unlike correlation-based methods, the matrix factorization techniques treat the users and items symmetrically and hence, handle item synonymy and sparsity in a better fashion. However, the training component of these techniques is computationally intensive, which makes them impractical to have frequent re-training. Incremental versions of SVD based on folding-in and exact rank-1 updates [4] partially alleviate this problem. But, since the effects of small updates are not localized, the update operations are not very efficient.

[8] studies a special case of the weighted Bregman co-clustering algorithm. The co-clustering problem is formulated as a matrix approximation problem with non-uniform weights on the input matrix elements. Both the users and the items are clustered so that item synonymy ceases to be a problem. As in the case of SVD and NNMF, the co-clustering algorithm also optimizes the approximation error of a low parameter reconstruction of the ratings matrix. However, unlike SVD and NNMF, the effects of changes in the ratings matrix are localized which makes it possible to have efficient incremental updates. [8] presents parallel algorithm design based on co-clustering. It compares the performance of the algorithm against matrix factorization and correlation based approaches on the MovieLens[2] and BookCrossing dataset [19] (269392 explicit rating(1-10) from 47034 users on 133438 books). We consider soft real-time (around 10 *sec.*) CF framework using hierarchical parallel co-clustering optimized for multi-core clusters using pipelined parallelism and computation communication overlap. We deliver scalable performance over 100M ratings of the Netflix data using 1024 nodes

[2] http://www.grouplens.org/data/. 100K ratings(1-5) 943 users, 1682 movies.

of Blue Gene/P with 4 cores at each node. [6] uses a dataflow parallelism based framework (in Java) to study performance vs. accuracy trade-offs of co-clustering based CF. However, it doesn't consider re-training time for incremental input changes. Further, the parallel implementation does not scale well beyond 8 cores due to cache miss and in-memory lookup overheads. We demonstrate parallel scalable performance on 1024 nodes of Blue Gene/P and $20\times$ better training time and better prediction time along with high prediction accuracy (0.87 ± 0.02 RMSE).

[18] presents a parallel algorithm based on Alternating-Least-Squares with Weighted-λ-Regularization (ALS-WR) for the Netflix Prize dataset. Their solution, using parallel Matlab on a Linux cluster, takes 2.5 hrs for training (30 ALS iterations) and with RMSE value around 0.9 on 1000 hidden features. We address the matrix approximation problem using a novel distributed co-clustering algorithm that incorporates performance optimizations to achieve highly scalable performance with the record training time of $5.9s$ on the full Netflix dataset and high accuracy. [10] studies IO scalable co-clustering by mapping a significant fraction of computations performed by the Bregman co-clustering algorithm to an on-line analytical processing (OLAP) engine. [12] studies the scalability of basic MPI based implementation of co-clustering. We deliver more than one order of magnitude higher performance compared to this work, by performing communication optimizations for multi-core cluster based MPPs such as Blue Gene/P. [1] presents results of collaborative filtering using *Concept decomposition* based approach. Concept decomposition is a matrix approximation scheme that solves a least-squares problem after clustering. It has been empirically established [7] that the approximation power (when measured using the Frobenius norm) of concept decompositions is comparable to the best possible approximations by truncated SVDs [9]. However, [1] presents the results of a sequential concept decomposition based algorithm that takes 13.5mins. training time for the full Netflix data, which is very high when looking at soft real-time performance. We achieve around $138\times$ better performance using an optimized distributed algorithm designed for multi-core cluster architectures.

3 Background and Notation

In this paper, we deal with partitional co-clustering where all the rows and columns are partitioned into disjoint row and column clusters respectively. We consider a general framework for addressing this problem that considerably expands the scope and applicability of the co-clustering methodology. As part of this generalization, we view partitional co-clustering as a lossy data compression problem [2] where, given a specified number of rows and column clusters, one attempts to retain as much information as possible about the original data matrix in terms of statistics based on the co-clustering [11]. The main idea is that a reconstruction based on co-clustering should result in the same set of user-specified statistics as the original matrix.

A $k * l$ partitional co-clustering is defined as a pair of functions:
$\rho : 1, \ldots, m \longmapsto 1, ..., k$; and, $\gamma : 1, \ldots, n \longmapsto 1, ..., l$. Let $\hat{U}$ and $\hat{V}$ be random variables that take values in $1, ..., k$ and $1, ..., l$ such that $\hat{U} = \rho(U)$ and $\hat{V} = \gamma(V)$. Let, $\hat{Z} = [\hat{z}_{uv}] \in S^{m \times n}$ be an approximation of the data matrix Z such that $\hat{Z}$ *depends only upon* a given co-clustering (ρ, γ) and certain summary statistics derived from

co-clustering. Let $\hat{Z}$ be a (U,V)-measurable random variable that takes values in this approximate matrix $\hat{Z}$ following w, i.e., $p(\hat{Z}(U,V) = \hat{z}_{uv}) = w_{uv}$. Then, the goodness of the underlying co-clustering can be measured in terms of the expected distortion between Z and $\hat{Z}$, that is,

$$E[d_\phi(Z,\hat{Z})] = \sum_{u=1}^{m}\sum_{v=1}^{n} w_{uv} d_\phi(z_{uv},\hat{z}_{uv}) = d_{\Phi_w}(Z,\hat{Z}) \tag{1}$$

where $\Phi_w : S^{m\times n} \longmapsto \mathbb{R}$ is a is a separable convex function induced on the matrices such that the Bregman divergence ($d_\Phi()$) between any pair of matrices is the weighted sum of the element-wise Bregman divergences corresponding to the convex function ϕ. From the matrix approximation viewpoint, the above quantity is simply the weighted element-wise distortion between the given matrix Z and the approximation $\hat{Z}$. The co-clustering problem is then to find (ρ,γ) such that (1) is minimized.

Now we consider two important convex functions that satisfy the Bregman divergence criteria and are hence studied in this paper.

(I-Divergence) : Given $z \in \mathbb{R}_+$, let $\phi(z) = zlogz - z$. For $z1, z2 \in \mathbb{R}$, $d_\phi(z_1,z_2) = z_1 log(z_1/z_2) - (z_1 - z_2)$.

(Squared Euclidean distance) : Given $z \in \mathbb{R}$, let $\phi(z) = z^2$. For $z1, z2 \in \mathbb{R}$, $d_\phi(z_1,z_2) = (z_1 - z_2)^2$.

Given a co-clustering (ρ,γ), Modha et al. discuss six co-clustering bases where each co-clustering basis preserves certain summary statistics on the original matrix. It also proves that the possible co-clustering bases ($C1 \ldots C6$) form a hierarchical order in the number of cluster summary statistics they preserve. The co-clustering basis $C6$ preserves all the summaries preserved by the other co-clustering bases and hence is considered the most general among the bases. In this paper we discuss the partitioning co-cluster algorithms for the basis $C6$. For co-clustering basis $C6$ and Euclidean-divergence objective, the matrix approximation is given by: $\hat{A}_{ij} = A^{COC}_{gh} + (A^{CC}_{ih} - A^{RC}_{gj})$, where, $A^{RC}_{gj} = \frac{S^{RC}_{gj}}{W^{RC}_{gj}} = \frac{\sum_{i'|\rho(i')=g} A_{i'j}}{\sum_{i'|\rho(i')=g} W_{i'j}}$; $A^{CC}_{ih} = \frac{S^{CC}_{ih}}{W^{CC}_{ih}} = \frac{\sum_{j'|\gamma(j')=h} A_{ij'}}{\sum_{j'|\gamma(j')=h} W_{ij'}}$;

Algorithm 1. Sequential Static Training via Co-Clustering

```
Input: Ratings Matrix $A$, Non-zeros matrix $W$, No. of row clusters $l$, No. of column clusters $k$.
Output: Locally optimal co-clustering ($\rho$,$\gamma$) and averages $A^{COC}$,$A^{RC}$,$A^{CC}$,$A^{R}$ and $A^{C}$.
Method:
1. Randomly initialize ($\rho$,$\gamma$)
while RMSE value is converging do
    2a. Compute averages $A^{COC}$,$A^{RC}_{gj}$,$A^{CC}_{ih}$,$A^{R}$ and $A^{C}$.
    2b. Update row cluster assignments
    $\rho(i) = \mathbf{argmin}_{1\le g\le k} \sum_{j=1}^{n} W_{ij} d_\phi(A_{ij},\hat{A}_{ij}), 1 \le i \le m$
    2c. Update column cluster assignments
    $\gamma(i) = \mathbf{argmin}_{1\le h\le l} \sum_{i=1}^{m} W_{ij} d_\phi(A_{ij},\hat{A}_{ij}), 1 \le j \le n$
end
```

$A_{gh}^{COC} = \frac{S_{gh}^{COC}}{W_{gh}^{COC}} = \frac{\sum_{i'|\rho(i')=g}\sum_{j'|\gamma(j')=h} A_{i'j'}}{\sum_{i'|\rho(i')=g}\sum_{j'|\gamma(j')=h} W_{i'j'}}$. The sequential update algorithm for the basis $C6$ is as shown in Algorithm 1 where the approximation matrix $\hat{A}$ for various co-clustering bases can be obtained from [2]. For Euclidean divergence, Step 2b. and 2c. of Algorithm 1 use $d_\phi(A_{ij}, \hat{A}_{ij}) = (A_{ij} - \hat{A}_{ij})^2$. For I-divergence, Step 2b. and 2c. of Algorithm 1 use $d_\phi(A_{ij}, \hat{A}_{ij}) = A_{ij} * \log(\hat{A}_{ij}/A_{ij}) - A_{ij} + \hat{A}_{ij}$

In the above sequential algorithm (Algorithm 1), we notice two important steps - a) Calculating the matrix averages, and, b) updating the row and column cluster assignments. Further, given the matrix averages, row and column cluster updates can be done independently, and row updates themselves can be done in parallel.

4 Optimized Distributed Co-clustering Algorithm

For multi-core cluster architectures, one can utilize the available intra-node parallelism along with inter-node parallelism to get highly scalable distributed co-clustering algorithm. Let, c be the number of cores (threads) per node in the distributed architecture, referred to as $T_1 \dots T_c$. These cores (threads) per node can be used to obtain computation communication overlap as well pipelining across the iterations in the distributed algorithm. This can significantly reduce the communication bottlenecks of the algorithm. Algorithm 2 presents the distributed algorithm with these performance optimizations. The *while loop* executes iterations until the RMSE value converges to within a given error bound. Within each iteration the following steps (Step5..Step10) get executed. In **Step 5.**, threads $(T_2 \dots T_c)$ compute the partial contribution to row-cluster averages, A_{gj}^{RC}; while simultaneously, thread T_1, performs *MPI_Allgather* to get the column-cluster membership (γ). Thus, (intra-iteration) computation communication overlap is achieved which leads to improved performance. Similarly, computation communication overlap is achieved in the following steps. In **Step 6.**, threads $(T_2 \dots T_c)$ compute the partial contribution to column-cluster averages, A_{ih}^{CC}; while simultaneously, thread T_1, performs *MPI_Allreduce* to compute the row-cluster averages A_{gj}^{RC}. In **Step 7.**, threads $(T_2 \dots T_c)$ compute the partial contribution to co-cluster averages, A_{gh}^{COC}; while simultaneously, thread T_1, performs *MPI_Allreduce* to compute the column-cluster averages A_{ih}^{CC}. In **Step 8.**, threads $(T_2 \dots T_c)$ compute the partial $\hat{A}_{ij}$ values using A_{ih}^{CC} and A_{gj}^{RC}; while simultaneously, thread T_1, performs *MPI_Allreduce* to compute the co-cluster averages A_{gh}^{COC}. In **Step 9.**, all threads $(T_1 \dots T_c)$ in a node, compute final row-cluster memberships for all the rows that are owned by that node. In **Step 10.**, threads $(T_2 \dots T_c)$ compute final column-cluster memberships while simultaneously, thread T_1, performs *MPI_Allgather* to get the row-cluster memberships from all other nodes.

In order to reduce the communication cost, the nodes are divided into groups, each group having the same number of nodes. A small constant number of nodes in each group act as *Steiner nodes* and help in inter-group communication. So, each communication step in Algorithm 2 is broken into two phases: (a) Intra-group communication, followed by (b) Inter-group communication using the Steiner nodes. Since, the communication group sizes are significantly reduced by using this grouping strategy, the communication cost goes down thus improving the scalability of the distributed algorithm. Further, to ensure non-overlap, across any two groups, of their intra-group communication, we use topology mapping to map each group onto a plane in the 3D Torus Interconnect architecture of Blue Gene/P. This leads to further decrease in the communication time.

5 Parallel Time Complexity Analysis

In this section, we establish theoretically, the performance and scalability advantage of our optimized distributed algorithm. Refer notation given in Table 5.

The distributed algorithm described in section 2 takes a certain number of iterations, say I. In each iteration the rows are assigned to row clusters and columns are assigned to column clusters. Each iteration has multiple steps. In *Step 5.*, the thread T_1 of all nodes communicate using all-gather operation to aggregate column to column-cluster mapping information. This communication time is given by: $O(S_0 + (n/B_0) * log(P_0))$. Simultaneously, threads $T_2 \ldots T_c$ of each node compute partial contributions of each node towards A^{RC}_{gj}. This computation time is $O(mn/(P_0.c))$. The overall time for *Step 5.* is given by $\max(O(S_0 + (n/B_0) * log(P_0)), mn/(P_0.c))$. Assuming, that compute time dominates, the time complexity for *Step 5.* can be approximated by $O(mn/(P_0{*}c))$.

In *Step 6.*, the thread T_1 of all nodes communicate using all-reduce operation to compute the row-cluster averages A^{RC}_{gj}. This communication time is given by: $O(S_0 + (mn/B_0) * log(P_0))$. Simultaneously, threads $T_2 \ldots T_c$ of each node compute partial contributions of each node towards A^{CC}_{ih}. This computation time is $O(mn/(P_0.c))$. Thus, the overall time for *Step 6.* is given by $\max(O(S_0 + (mn/B_0) * log(P_0)), mn/(P_0.c))$. Assuming, that the communication time dominates, the time complexity for *Step 6.* can be approximated by $O(S_0 + (mn/B_0) * log(P_0))$. Similarly, the time complexity for *Step 7.* can be approximated by $O(S_0 + (mn/B_0) * log(P_0))$.

In *Step 8.*, the thread T_1 of all nodes communicate using all-reduce operation to compute the co-cluster averages A^{COC}_{gh}. This communication time is given by: $O(S_0 + (kl/B_0) * log(P_0))$. Simultaneously, threads $T_2 \ldots T_c$ of each node compute partial values for assignment of each row (and column) to k possible row-clusters (and l possible column-clusters). This computation time is $O(mns * (k + l)/(P_0.c))$. Thus, the overall time for *Step 8.* is given by $\max(O(S_0 + (kl/B_0) * log(P_0)), mns * (k + l)/(P_0.c))$. Assuming that the compute time dominates, the time complexity for *Step 8.* can be approximated by $O(mns * (k + l)/(P_0.c))$. In a similar fashion, the compute time for *Step 9.* is $O(mns * (k + l)/(P_0 * c))$. Assuming that the compute time dominates *Step 10.*, its time complexity can be approximated by $O(mns * (k + l)/(P_0 * c))$.

Thus, the overall time complexity for the hybrid distributed co-clustering algorithm , per iteration, is given by:

$$\begin{aligned} T_h(m, n, P_0) = O((mn/P_0 * c) + S_0 + (mn/B_0) * log(P_0) + \\ mns * (k + l)/(P_0 * c)) \end{aligned} \quad (2)$$

One can observe, that an MPI only (algorithm referred to as *base* algorithm), which does not have computation communication overlap, has run time around $c\times$ higher as compared to the hybrid Algorithm 2. This is so, because the hybrid algorithm achieves effective overlap between computation and communication in most of the steps of the algorithm and utilizes c available cores per node to get higher performance, while the

Algorithm 2. Distributed (Hybrid - MPI+OMP) Static Training via Co-Clustering

```
Input: Ratings Matrix $(A)$, Non-zeros matrix $(W)$, No. of row clusters $(l)$, No. of column clusters $(k)$.
Output: Locally optimal co-clustering ($\rho$,$\gamma$) and averages $A^{COC}_{gh}$,$A^{RC}_{gj}$,$A^{CC}_{ih}$.
Data Distribution: (Each node has total $c$ threads - $\{T_1 \dots T_c\}$)
1. Each node $p$ gets $m_p = m/P_0$ rows and $n_p = n/P_0$ columns.
2. Further, threads $T_1 \dots T_c$ of each node $p$, each get $m_{p'}$ rows (i.e, a $m_{p'} \times n$ submatrix) and $n_{p'}$ columns (i.e, a $m \times n_{p'}$ submatrix, where $m'_p = \frac{m_p}{T_c}$ and $n'_p = \frac{n_p}{T_c}$.

Method:
1. Each $T_i, i \in [1..c]$ : Randomly initialize ($\rho^p_i$,$\gamma^p_i$)
2. $T_1$ : Gather all the row and column sums/weights $S^R_i$, $S^C_j$, $W^R_i$, $W^C_j$ $\forall i,j$ from the other nodes using MPI_Allgather.
3. $T_1 \dots T_c$ : Calculate all row and column averages $A^R_i = \frac{S^R_i}{W^R_i}$ and $A^C_j$.
(Note that Step 2 and Step 3 can be executed in parallel)
4. $T_1$: Gather the global Row-cluster membership ($\rho$) by concatenating ($\rho^p$) using MPI_Allgather.

while RMSE value has not converged, Each thread in a node does the following do
    5. $T_2 \dots T_c$ : Calculate the partial contributions to Row-Cluster Averages $A^{RC}_{gj}$
    $T_1$ : Gather the global Column-cluster membership ($\gamma$) by concatenating ($\gamma^p$) using MPI_Allgather.

    6. $T_2 \dots T_c$ : Calculate the partial contribution to Column-Cluster Averages $A^{CC}_{ih}$
    $T_1$: Do MPI_AllReduce to compute the global row-cluster averages $A^{RC}_{gj}$
    7. $T_2 \dots T_c$ : Calculate the contribution of the local rows and columns to the co-cluster sums/weights i.e, $S^{COC}_p$ and $W^{COC}_p$.

    $T_1$: Do MPI_AllReduce to compute the global col-cluster averages $A^{RC}_{ih}$
    8. $T_1$:Do an MPI_AllReduce on above contributions and get the global co-cluster sums/weights $S^{COC}$, $W^{COC}$ and calculate $A^{COC}$.
    $T_2 \dots T_c$ : Partially compute $\hat{A}^R(i,j,g)$, $\hat{A}^C(i,j,h)$ the local row cluster and column cluster assignment steps for each choice of assignment $g$,$h$

    9. $T_1 \dots T_c$ : Update all the local row cluster assignments $\rho^p$ by first updating $\hat{A}^R(i,j,g)$ with the co-cluster averages to generate $\hat{A}_{ij}$
    $\rho^p(i) = \underset{1 \le g \le k}{\textbf{argmin}} \sum_{j=1}^{n} w_{ij} d_\phi(A_{ij}, \hat{A}_{ij})$, $i$ : rows owned by node $p$

    10. $T_0$ : Gather the global Row-cluster membership ($\rho$) by concatenating ($\rho^p$) using MPI_Allgather.
    $T_1 \dots T_3$: Update all the local column cluster assignments $\gamma^p$ by first updating $\hat{A}^C(i,j,h)$ with the cocluster averages to generate $\hat{A}_{ij}$

    $\gamma^p(i) = \underset{1 \le h \le l}{\textbf{argmin}} \sum_{i=1}^{m} w_{ij} d_\phi(A_{ij}, \hat{A}_{ij})$, $j$ : cols owned by node $p$
end
```

Table 1. Notation

Symbol	Definition
P_0	Total number of nodes for computation
c	Number of threads (cores) per node
(m, n)	Number of rows and columns in the input matrix
s	Sparsity factor of the matrix
(k, l)	Number of row and column clusters respectively
(m/k)	Average number of rows per row cluster
n/l	Average number of columns per column cluster
B_0	Interconnect Bandwidth for AllReduce / Allgather operation
S_0	Setup cost for AllReduce / Allgather operation

MPI only (base) algorithm performs all the computation and communication steps sequentially. Further, the load-imbalance across the nodes is reduced, by factor c, in the hybrid algorithm as compared to the base algorithm. Assuming, γ as the load-imbalance factor for the base algorithm, the training time for the base algorithm gets magnified by the factor, $(1 + \gamma)$. For the same data distribution, the hybrid algorithm will have load imbalance of $(1 + \gamma/c)$ and hence a lower magnification factor in its training time. Thus, in the best case, this leads to $O(c^2 * \frac{(1+\gamma)}{(c+\gamma)})$ performance gain of hybrid vs. base algorithm.

5.1 Optimum Thread Distribution

In a general case, one can optimize the communication by providing more than one thread for communication. We study this general communication optimization technique in this section and determine the optimum number of threads to achieve best performance.

Let r be the number of threads (cores) that are devoted to computation per step, while the remaining $(c - r)$ threads (cores), perform communication per step. When, multiple threads are used for communication, we assume that it takes x steps to complete one communication task across all nodes. In this case, the time complexity of the hybrid distributed co-clustering algorithm is given by:

$$\begin{aligned} T_h(m, n, r, P_0) = O((mn/P_0 * r) + (S_0 + (mn/B_0) * log(P_0)) * (2x/(c - r)) + \\ 3mns * (k + l)/(P_0 * r)) \end{aligned} \quad (3)$$

Differentiating the above expression for $T_h(m, n, r, P_0)$ with respect to r, and setting it to zero, we can determine the optimum number of threads to be used for computation per node. We get the following quadratic equation to determine the optimum r:

$$2xP_0 * (S_0 + mn/B_0 * \log(P_0)) * r^2 = (mn + 3mns * (k + l)) * (c^2 + r^2 - 2cr) \quad (4)$$

Solving, the optimum value of r is given by:

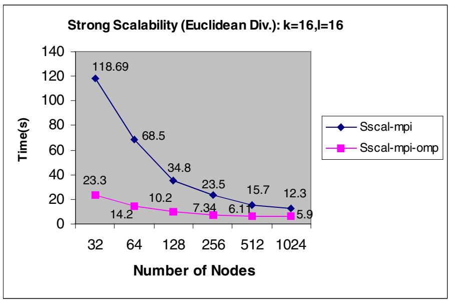
$$r^* = \frac{\sqrt(4c^2Z^2 + 8c^2xP_0Y) - 2cZ}{2 * (2xP_0Y - Z)}, \text{where,}$$
$$Y = S_0 + mn/B_0 * \log(P_0), \text{and, } Z = mn + 3mns(k + l) \quad (5)$$

6 Results and Analysis

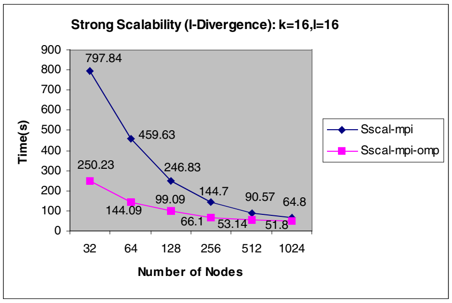
The hybrid distributed algorithm was implemented using MPI and OpenMP, while the base distributed algorithm was implemented using only MPI. The Netflix Prize dataset was used to evaluate and compare the performance and scalability of these distributed co-clustering algorithms. The experiments were performed on the Blue gene/P (MPP) architecture. Each node in Blue Gene/P is a quad-core chip with frequency of 850 MHz having 2 GB of DRAM and 32 KB of L1 cache per core. Blue Gene/P has the following major interconnects:(a) 3D-Torus interconnect which provides 3.4 Gbps per link on each of the 12 links per node (total 5.1 GBps per node), and, (b) Collective Network that provides 6.8 Gbps per link. MPI was used across the nodes for communication, while within each node OpenMP was used to parallelize the computation and communication amongst the four cores. For all the experiments, we obtained RMSE in the range 0.87 ± 0.02 on the data. Below, k refers to the number of row clusters generated while l refers to the number of column clusters generated. Netflix data was used for evaluation of the distributed algorithms. For Netflix, the number of rows, m, is around $480K$; the number of columns, n, is $17,770$, and the sparsity factor, s is around 85. We present the strong, weak and data scalability analysis of the training phase for both Euclidean divergence and I-divergence based co-clustering.

6.1 Strong Scalability

For strong scalability, we used the full Netflix data for each experiment, while increasing the number of nodes, from 64 to 1024. Fig. $1(a)$ illustrates that the hybrid algorithm (for Euclidean divergence) has consistently better performance over the base algorithm: $5.1\times$ better than the base when $P_0 = 32$ and $2.1\times$ better at $P_0 = 1024$. Here, the hybrid algorithm has more than $(c = 4)\times$ better performance than the base algorithm due to reduction in load-imbalance as explained in Section 5. In the hybrid algorithm, as the number of nodes increases from 32 to 1024, the compute time decreases by $26\times$ while the communication time remains almost the same, this leads to overall $4\times$ decrease in total training time with $32\times$ increase in the number of nodes(P_0). Fig. $1(b)$ illustrates the performance gain of the hybrid algorithm over the base algorithm for I-divergence. Here, the performance gain of hybrid vs base decreases from $3.2\times$ for $P_0 = 32$ nodes to $1.25\times$ for $P_0 = 1024$ nodes. By using more efficient load balancing techniques, the performance of the hybrid (MPI+OpenMP) algorithm can be improved further. Moreover, by using the optimum number of cores for communication using the formula specified in the Section 5.1, one can get better overall performance. Further, for I-divergence, the gain for the hybrid algorithm from the decrease in inter-node load-imbalance is offset by the loss from intra-node load-imbalance amongst the threads. Hence, in case of I-divergence the gain of the hybrid algorithm over the base algorithm is not as large as in the Euclidean divergence.



(a)



(b)

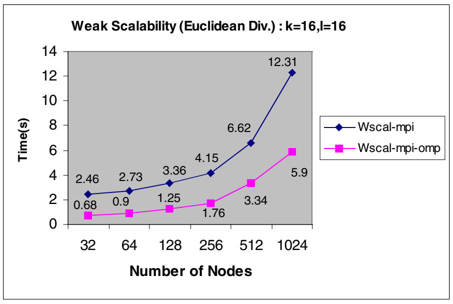
Fig. 1. Strong Scalability: (a) Euclidean divergence. (b) I-divergence

6.2 Weak Scalability

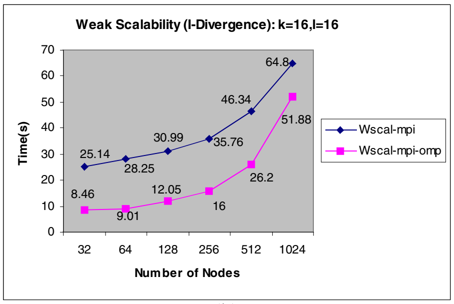
Fig. $2(a)$ displays the weak scalability for Euclidean distance based co-clustering as the number of nodes (P_0) increases from 32 to 1024 and the training data increases from 3.125% to 100% of the full Netflix dataset (with $k = 16$, $l = 16$). Here, the hybrid algorithm performs consistently better compared to the base algorithm: $3.61\times$ better at $P_0 = 32$ and $2.1\times$ better at $P_0 = 1024$. The total time for the hybrid algorithm increases by $8.67\times$ as the number of nodes increase from 32 to 1024. This is due to the compute time increase by $2.91\times$ and also increase in load imbalance. Fig. $2(b)$ illustrates the weak scalability of the hybrid algorithm for I-divergence: with $32\times$ increase in the data and number of nodes, the training time only increases by $6.13\times$. Further, the hybrid algorithm performs consistently better than the base algorithm.

6.3 Data Scalability

Fig. $3(a)$ displays the data scalability for Euclidean distance based co-clustering as the training data increases from 6.25% to 100% of the full Netflix dataset, while $P_0 = 1024$.



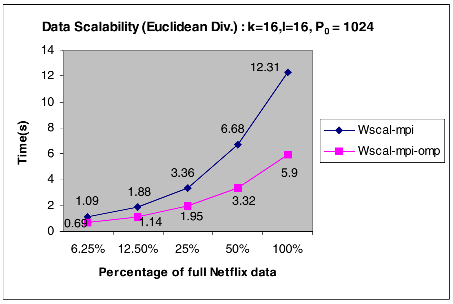
(a)



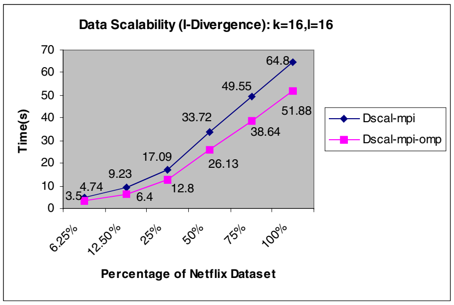
(b)

Fig. 2. Weak Scalability: (a) Euclidean divergence. (b) I-divergence

The training time for the hybrid algorithm increases by $8.55\times$ with $16\times$ increase in data, while that for the base algorithm increases by $11.3\times$. Thus, the hybrid algorithm shows better than linear data scalability and also better data scalability as compared to the base algorithm. The hybrid algorithm also performs better than the base by $1.58\times$ at $P_0 = 32$ and $2.1\times$ better at $P_0 = 1024$. Fig. $3(b)$ illustrates the data scalability for the hybrid algorithm with I-divergence as the training time increases only by $14.8\times$ with $16\times$ increase in data, while the number of nodes is kept constant at $P_0 = 1024$.



(a)



(b)

Fig. 3. Data Scalability: (a)Euclidean divergence. (b) I-divergence

7 Conclusions and Future Work

Real-time collaborative filtering with high prediction accuracy is a computationally challenging problem. We have presented the design of a novel distributed co-clustering based Collaborative Filtering algorithm. Our algorithm demonstrates soft real-time (less than 10 *sec.*) performance over highly sparse massive data sets. Using pipelined parallelism and compute communication overlap optimizations our hybrid (MPI+OpenMP) algorithm outperforms all known prior results for CF while maintaining high accuracy. Theoretical time complexity analysis proves the scalability of our algorithm. We demonstrated soft real-time parallel CF using the Netflix Prize dataset on Blue Gene/P architecture. We delivered the best known training time of around $6s$ for the full Netflix dataset and the best known prediction of $1.78us$ per prediction (rating) for $1.4M$ ratings with high prediction accuracy (RMSE value of 0.87 ± 0.02). This training time is $20\times$ (more than one order of magnitude) better than the best known parallel training time. We also demonstrated strong, weak and data scalability for multi-core cluster architectures. In future, we intend to investigate performance analysis using queuing theoretic models for large scale systems.

References

1. Ampazis, N.: Collaborative filtering via concept decomposition on the netflix dataset. In: ECAI, pp. 143–175 (2008)
2. Banerjee, A., Dhillon, I., Ghosh, J., Merugu, S., Modha, D.S.: A generalized maximum entropy approach to bregman co-clustering and matrix approximation. Journal of Machine Learning Research 8(1), 1919–1986 (2007)

3. Bennett, J., Lanning, S.: The netflix prize. In: KDD-Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2007)
4. Brand, M.: Fast online svd revisions for lightweight recommender systems. In: SIAM International Conference on Data Mining, pp. 37–48 (2003)
5. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Fourteenth International Conference on Uncertainty in Artificial Intelligence, pp. 43–52 (1998)
6. Daruru, S., Marin, N.M., Walker, M., Ghosh, J.: Pervasive parallelism in data mining: dataflow solution to co-clustering large and sparse netflix data. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1115–1124 (2009)
7. Dhillon, I.S., Modha, D.S.: Concept decompositions for large sparse text data using clustering. In: Machine Learning, pp. 143–175 (1999)
8. George, T., Merugu, S.: A scalable collaborative filtering framework based on co-clustering. In: Fifth International Conference on Data Mining, pp. 625–628 (2005)
9. Golub, G.H., Loan, C.F.V.: Matrix computations. The Johns Hopkins University Press, Baltimore (1996)
10. Hsu, K.-W., Banerjee, A., Srivastava, J.: I/o scalable bregman co-clustering. In: Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (2008)
11. Mallela, I.D.S., Modha, D.: Information-theoretic co-clustering. In: Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining, pp. 89–98 (2003)
12. Kwon, B., Cho, H.: Scalable co-clustering algorithms. In: Hsu, C.-H., Yang, L.T., Park, J.H., Yeo, S.-S. (eds.) ICA3PP 2010. LNCS, vol. 6081, pp. 32–43. Springer, Heidelberg (2010)
13. Resnick, P., Varian, H.R.: Recommender systems - introduction to special section. Comm. ACM 40(3), 56–58 (1997)
14. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Application of dimensionality reduction in recommender systems: a case study. In: WebKDD Workshop (2000)
15. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: ACM Conference on Electronic Commerce, pp. 158–167 (2000)
16. Schafer, J.B., Konstan, J.A., Riedi, J.: Recommender systems in e-commerce. In: ACM Conference on Electronic Commerce, pp. 158–166 (1999)
17. Srebro, N., Jaakkola, T.: Weighted low rank approximation. In: Twentieth International Conference on Machine Learning, pp. 720–728 (2003)
18. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large scale parallel collaborative filtering for the netflix prize. In: Fourth International Conference on Algorithmic Aspects in Information and Management, pp. 337–348 (2008)
19. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: Fourteenth International World Wide Web Conference (2005)