# DCF: A Dataflow Based Collaborative Filtering Training Algorithm

Xiangyu Ju[1], Quan Chen[1], Zhenning Wang[1], Minyi Guo[1], and Guang R. Gao[2]

[1] Shanghai Jiao Tong University, Shanghai 200240, China,
[2] University of Delaware, Newark 19716, USA
juxiangyu@sjtu.edu.cn

**Abstract.** Emerging recommender systems often adopt collaborative filtering technique to improve the recommending accuracy. Existing collaborative filtering techniques are implemented with either Alternating Least Square (ALS) algorithm or Gradient Descent (GD) algorithm. However, both of the two algorithms are not scalable because ALS suffers from high computational complexity and GD suffers from severe synchronization problem and tremendous data movement. To solve the above problem, we proposed a Dataflow-based Collaborative Filtering (DCF) algorithm. DCF exploits fine-grain asynchronous feature of dataflow model to minimize synchronization overhead, leverages mini-batch technique to ..., uses xxx techniques to .... By utilizing all the above techniques, DCF is able to significantly improve the performance of collaborative filtering. Our experiment on a cluster with one master node and ten slave modes show that DCF achieves 23x speedup over ALS on Spark and 18x speedup over gradient descent on Graphlab in public datasets.

**Keywords:** DCF; dataflow; collaborative filtering; gradient descent; asynchronous; fine-grain; parallel;

## 1 Introduction

With the immense growth of e-commerce and social network, recommender systems show increasing popularity and importance in recent years. To give more accurate and specific recommendations, collaborative filtering [3] algorithms are usually considered as effective solutions. Memory based and model based are two categories of collaborative filtering algorithms [7]. Existing studies [15] [6] show that model based algorithms have better prediction accuracy than memory based ones. Alternating least square (ALS) and gradient descent, two main training algorithms of model based collaborative filtering, are adopted by most big data platforms (e.g. Hadoop [1], Spark [18], Graphlab [17]) as the solution for collaborative filtering.

With increasing rating data from massive users, large scale data quantity and high data sparsity become the main bottlenecks of collaborative filtering algorithms. As the computation complexity of ALS increases with feature dimension quadratically, it shows low performance in large scale datasets [12]. Compared with ALS, gradient descent has a much lower computation complexity. Current studies [19] [8] [12] show that gradient descent outperforms ALS in large scale datasets. However, gradient descent also faces several challenges. **Firstly, network load becomes the bottleneck because of the high communication complexity. Secondly, individual operations of each data require**

**fine-grain task scheduling. Thirdly, gradient descent can not utilize computing resources well due to high data sparsity.**

In this case, we find that dataflow model can partially solve above challenges. Dataflow [9] is a parallel computation model which supports asynchronous fine-grain task scheduling, making it better for sparse dependencies. Fine-grain feature can naturally support individual data operations and asynchronization can fully utilize computing resources in high data sparsity scenario. However, applying dataflow model to gradient descent is not trivial. Fine-grain dependency checking overhead and frequent data movements will also become the bottlenecks of the algorithm.

To tackle above challenges, we propose a dataflow based collaborative filtering (DCF) algorithm to improve the performance of distributed recommender systems. While applying dataflow to gradient descent solves individual operation and data sparsity problems, we propose three optimizations to solve remaining challenges. Dummy edge technique exploits local data transmission in dataflow graph to reduce dependency checking overhead and unnecessary data movements. Multicasting technique avoids sending data to same node for reducing data movements. Mini-batch limits the dependency number of each feature vector to further reduce computation and communication complexity on algorithmic level. The contributions of our work are summarized as follows:

- We design DCF algorithm based on dataflow execution model. Mini-batch technique is applied to DCF algorithm for further reducing computation and communication complexity on algorithmic level
- We propose dummy edge technique to handle fine-grain overhead of data dependency checking and reduce unnecessary data movements by exploiting the data locality.
- We propose multicasting technique to further optimize data movements by reducing the quantity of data transferred by network.

We implement DCF and evaluate it with public rating datasets. The experiment results demonstrate that DCF shows a 23X speedup over ALS on Spark and 18X speedup over gradient descent on Graphlab with similar accuracy.

## 2 Background & Motivation

Model based collaborative filtering algorithms are adopted by most distributed recommender systems. As shown in Figure 1, Model based collaborative filtering algorithms convert a recommendation issue into a matrix factorization problem, transforming user-item-rating data into two separated matrixes which represent user features and item features.Predicting rates are made by the products of user feature vectors and item feature vectors. ALS and gradient descent are two main training algorithms for model based collaborative filtering.

### 2.1 ALS & Gradient Descent

ALS [20] converts the non-convex problem into a quadratic problem which can be optimally solved. According to the study of Gemulla et al. [12], the computation complexity of ALS is $O(k^3 + k^2(n_u + n_v) + kn_r)$ in one iteration, and it increases

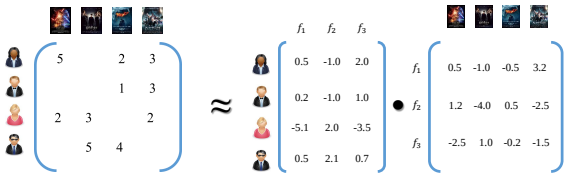


**Fig. 1.** Model Based Collaborative Filtering

with feature dimension $k$ quadratically because user number $n_u$ or item number $n_v$ is much larger than $k$. ALS shows low performance in large scale datasets because of the high computation complexity.

Gradient descent is an alternating training algorithm for model based collaborative filtering. According to Eq. 1 and Eq. 2, user feature vector $u_i$ or item feature vector $v_j$ can be updated by given ratings $r_{i,j}$ and dependent feature vectors where $\lambda$ represents the regularization parameter. The computation complexity of gradient descent is $O(k\bar{d}(n_u + n_v))$ in one iteration where $\bar{d}$ represents the average dependency number. Compared with ALS, gradient descent has a lower computation complexity. However, gradient descent also faces several challenges. Firstly, it requires fine-grain operations because each user feature vector $u_i$ or item feature vector $v_j$ can be updated individually and asynchronously. Secondly, high data sparsity causes load imbalance because dependency number of each feature vector varies in large scale. Thirdly, the communication complexity is $O(k\bar{d}(n_u + n_v))$ and it leads to high network load.

$$u_i = u_i - \alpha(\sum_{r_{i,j} \neq 0} (u_i^T v_j - r_{i,j})v_j + \lambda u_i) \tag{1}$$

$$v_j = v_j - \alpha(\sum_{r_{i,j} \neq 0} (u_i^T v_j - r_{i,j})u_i + \lambda v_j) \tag{2}$$

In order to develop high performance collaborative filtering algorithms, we select gradient descent as training algorithm for its low computation complexity. In addition, we need to find proper parallel model to solve its challenges. As a result, dataflow model is selected because it can partially solve the challenges of gradient descent.

### 2.2 Dataflow

Dataflow [9] is a data-driven computation model optimized for the execution of fine-grain parallel algorithms. Dataflow treats each computation as a separate task and the execution order is maintained by data dependency. Dataflow model can be represented as directed acyclic graph (DAG) where nodes represent computation and edges represent data dependencies.

The features of dataflow model can partially solve the challenges of gradient descent well. Fine-grain feature supports individual updating operations of each feature vector. Asynchronization supports sparse dependency well for fully utilizing computing resources. However, applying dataflow model to gradient descent is not straightforward because we need to solve the challenges of dataflow model as well. Fine-grain dependency checking overhead of each feature vector

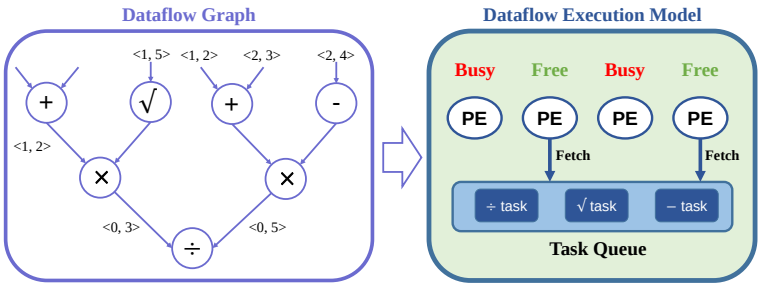


**Fig. 2.** Dataflow Execution Model

and frequent data movements are the main bottlenecks because of massive rating data.

In summary, we propose DCF algorithm with optimizations to solve the challenges of gradient descent and dataflow model.

## 3 DCF: Dataflow Based Collaborative Filtering Algorithm

In this section, we give a gradient descent algorithm based on dataflow execution model. Firstly, we demonstrate the working mechanism of dataflow execution model. Secondly, we transform gradient descent into dataflow graph and explain the working process of it. Thirdly, we propose optimizations to solve the challenges of dataflow model and gradient descent. Finally, we give a detailed DCF algorithm by applying proposed optimizations.

### 3.1 Dataflow Execution Model

In order to design dataflow algorithm, we introduce the working process of dataflow execution model. As shown in Figure 2, we give an example to explain dataflow execution model explicitly. The figure shows a snapshot of a simple computing progress which consists of several basic operator tasks. Dataflow execution model schedules and processes tasks generated by dataflow graph. Dataflow graph is a directed acyclic graph which edges represent data dependencies and nodes represent functions. The basic data structure transferred in dataflow graph is key-value pair. A task is composed of data from input edges with same key and function of the node. Once dependent data of one node is satisfied, corresponding tasks will be generated and pushed to task queue, such as $-$ task, $\sqrt{}$ task and $\div$ task. As the input data of $+$ node have different keys, the task will be not generated until dependent data with same key arrive. Processing elements (PE) will fetch new tasks in task queue until tasks are finished.

### 3.2 DCF Dataflow Graph

In order to design dataflow based gradient descent algorithm, we need to transform the algorithm into dataflow graph expression. As shown in Figure 3, we construct dataflow graph of gradient descent according to Eq. 1 and Eq. 2. Node $R$ transforms the input data into the triplet form $< userid, itemid, rating >$,

and sends user id and item id information to $InitU$ and $InitV$ for initializing feature vectors. Meanwhile, node $R$ sends rating data to $U_i$ and $V_i$ for gradient computing, where $U_i$ and $V_i$ represent the $i$th iteration of user node and item node. After initializing feature vectors, $InitU$ and $InitV$ send feature vectors to $U_1$ and $V_1$. In each iteration $i$, $U_i$ and $V_i$ are used to compute the gradients and update feature vectors until convergence. After the gradient computing tasks are finished in last iteration $n$ , $U_n$ and $V_n$ output the trained feature vectors.

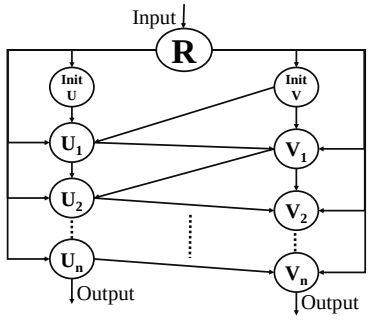


**Fig. 3.** Dataflow Graph of Gradient Descent

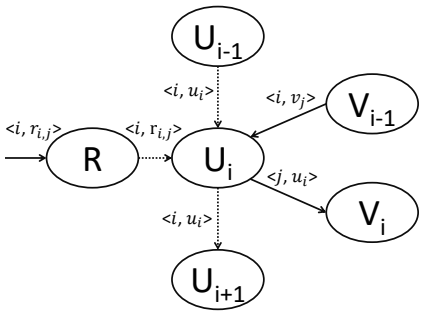


**Fig. 4.** Dummy Edge of User Node

By using dataflow model to implement gradient descent, data sparsity and fine-grain operations can be easily handled. The updating task of each feature vector is scheduled and executed individually. Fine-grain feature naturally supports the updating operation of gradient descent. In addition, the computation load of each feature vector varies in large scale because of the sparse data dependency. Asynchronization feature avoids unnecessary barrier time for waiting hot spot feature vectors in one iteration. As a result, computing resources can be utilized better. In addition, DCF shows the same convergency performance with original gradient descent algorithms because the deterministic feature of dataflow does not change the algorithm execution logic.

Though dataflow model supports gradient descent well, high communication complexity and dependency checking overhead are still challenges can not be solved. Hence, we propose three optimizations to solve these challenges.

### 3.3 Optimizations

**Dummy Edge** We propose dummy edge technique (dashed line in the dataflow graph) to avoid fine-grain overhead of dependency checking and reduce local data movements. As shown in Figure 4, we take the user node $U_i$ as an example to explain dummy edge. The data transferred by dummy edge can be accessed locally because the key of the data does not change. As local data are always ready to be accessed, dependency checking of dummy edge can be avoided.

The reconstructed dataflow graph with dummy edge is shown in Figure 5. Similar with $R$ in Figure 3, $R_U$ and $R_V$ transforming rating data into triplet format, are indexed by user id and item id respectively, in order to be locally accessed by $U$ and $V$. As for gradient computing nodes $U$ and $V$, two input edges

are replaced by two dummy edges , and the other input edge remains unchanged. Data movements can be reduced because only feature vectors from normal edge should be transferred by network. There is no dependency checking operation of every node in reconstructed dataflow graph because there is only one normal input edge of them.

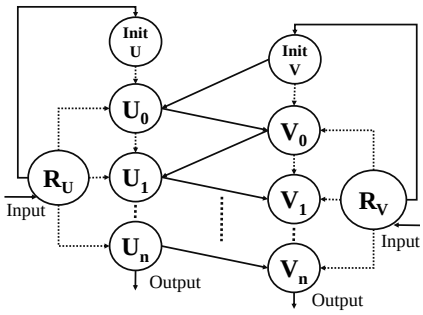


**Fig. 5.** Dataflow Graph of Gradient Descent Using Dummy Edge

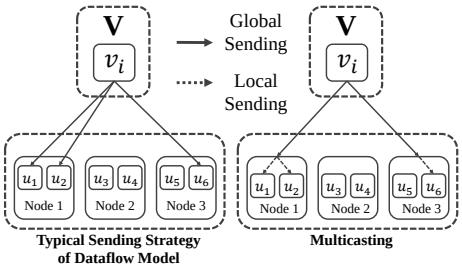


**Fig. 6.** Sending Strategy of Naive Dataflow Model

**Multicasting** With dummy edge, data movements can be reduced by avoiding sending local data. However, the quantity of data transferred by normal edges is much more than that by two dummy edges. Hence, we propose multicasting strategy to further optimize the data movements. As shown in Figure 6, global sending means sending data to different nodes in the cluster through network, while local sending means sending data in the node without network. In the scenario of sending data from $V$ to $U$, typical sending strategy of dataflow model requires data to be sent to the location of dependent users by global sending directly. In this process, key-value pairs with the same value may be sent to different locations on the same node. For example, the key-value pairs sent to $u_1$ and $u_2$ on node 1 has the same value $v_i$. Repetitive data sent to the same node will increase the network load of the algorithm. As for multicasting strategy, key-value pairs of same feature vectors are sent to the nodes which have dependent feature vectors by global sending. For example, $v_i$ depends on user $u_1$, $u_2$ and $u_6$, and there is no dependent data on node 2. Key-value pairs of $v_i$ are sent to node 1 and 3 first by global sending. Then, each node sends the key-value pairs to dependent feature vectors by local sending. $u_1$ and $u_2$ get the key-value pair from node 1 and $u_6$ gets the key-value pair from node 3.

Compared with original the communication complexity described in Section 2.1, the complexity by using multicasting is $O(ck(n_u + n_v))$ where $c$ represents the node number of cluster. The complexity is reduced by $\bar{d}/c$ times where average dependency number $\bar{d}$ increases with rating data linearly. In conclusion, multicasting can significantly reduce data movements in large scale dataset.

**Mini-batch** Mini-batch [10] was firstly proposed to improve the performance by limiting batch size of each iteration in deep learning scenarios . Hence, we apply mini-batch technique in collaborative filter for reducing computation and

communication complexities. While the batch size of each feature vector is different in collaborative filtering, we select the minimal number between dependency number $d$ and mini-batch size $b$ as the final dependency number for each feature vector. In large rating scenario, the mini-batch $b$ is always set as the final dependency number because real dependency number $d$ is larger than mini-batch size $b$. The computation and communication complexities of one iteration are $O(kb(n_u + n_v))$, and the complexities are reduced by $\bar{d}/b$ times where average dependency number $\bar{d}$ increases with rating number linearly. As a result, mini-batch technique can significantly reduce the computation and communication in large scale datasets.

We evaluate all optimizations in Section 4.2 and the experiments results shows that these optimizations can significantly improve the performance of our algorithm.

### 3.4 DCF Algorithm

By Applying dummy edge, multicasting and mini-batch optimizations, we get the finalized algorithm. In order to design a dataflow based algorithm, we need to construct a dataflow graph and give detailed algorithms for each node in the graph. The finalized dataflow graph is shown in Figure 5. The key nodes are gradient computing nodes ($U$ and $V$), while other nodes are used to preprocess data. As the algorithm logic is same for $U$ and $V$, we give the detailed algorithm of user node $U$.

**Algorithm 1** Execution Process of DCF User Node $U$

Receive $< i, v_j >$ from upstream node
Get $u_i$ from user data, get $r_{i,j}$ from rating data
**if** $d_{u_i}$ for $u_i$ is not initialized **then**
    Get $n_{u_i}$ from rating data
    $d_{u_i} = min(b, n_{u_i})$
    $u_{sum} = 0$
**if** $d_{v_j}$ for $v_j$ is not initialized **then**
    $d_{v_j} = b$
**if** $d_{u_i} > 0$ **then**
    Get $r_{i,j}$ from rating data
    $u_{sum} = u_{sum} + (u_i^T v_j - r_{i,j})v_j$
    $d_{u_i} = d_{u_i} - 1$
**if** $d_{u_i} = 0$ **then**
    $u_i = u_i - \alpha(u_{sum} + \lambda u_i)$
    **for** all $v_k$ of $u_i$ **do**
        **if** $d_{v_k} > 0$ **then**
            Send $< k, u_i >$ to dependent $v_k$ using multicasting
            $d_{v_k} = d_{v_k} - 1$

According to dummy edge technique, user vector $u_i$ and rating $r_{i,j}$ are obtained by querying with user id $i$ and item id $j$ after receiving the key-value

pair $< i, v_j >$ in lines 1-2. In lines 3-6, temporary gradient $u_{sum}$ and dependency counter $d_{u_i}$ are initialized to record the intermediate result of gradient computing, instead of checking redundant dependency. In line 5 and 7-8, mini-batch strategy is applied to initialize the dependency counter $d_{u_i}$ and the sending counter $d_{v_j}$ for reducing computation and communication complexities. If $d_{u_i} > 0$, gradient will be added to the temporary gradient $u_{sum}$ in lines 9-12. In lines 13-18, if the item dependencies for $u_i$ are satisfied ($d_{u_i} = 0$) and ), $u_i$ will be updated by temporary gradient $u_{sum}$ and old feature vector $u_i$. For every dependent items $v_k$ of $u_i$, if the sending dependency is satisfied ($d_{v_k} > 0$), the updated $u_i$ will be sent to the item $v_k$ by multicasting.

## 4 Experiments

We implement DCF with Java and evaluate its performance on public datasets under both multicore and distributed platforms. Spark ALS [18] and Graphlab [17] gradient descent (Graphlab GD) are two most widely used distributed collaborative filtering algorithms, so we select them as the contrasts of DCF. We evaluate the performance of optimizations (dummy edge, multicasting and mini-batch) separately. Scalability of DCF over contrast algorithm are assessed at the last of this section. In all the experiments, we use 1 master + 10 slaves cluster, and the configuration of each node is shown in Table 1. As for parameter setting, default iteration is 10 and mini-batch size is 40.

**Table 1.** Configuration of Experiment Machine

| Operating System | CPU | Core Number | Memory |
|---|---|---|---|
| CentOS 6.7 | Intel(R) Xeon(R) E5-2630 | 32 | 128G |

Movielens, Netflix and Yahoo Music are the three datasets used in the experiments. Detailed information of the datasets are shown in Table 2. We use 80% data for training and 20% data for testing in each dataset.

**Table 2.** The Statistical Information of Each Dataset

| **Dataset** | Movielens | Netflix | Yahoo Music |
|---|---|---|---|
| user number | 260,000 | 480,190 | 1,000,990 |
| item number | 40,000 | 17,770 | 624,961 |
| rating | 24,000,000 | 99,072,112 | 252,800,275 |

### 4.1 Comparing DCF with Other Distributed Algorithms

**Performance:** Figure 7 shows the results of DCF comparing with Spark ALS and Graphlab GD under public datasets. The trends of three algorithms are the same under different datasets. DCF shows the best performance among three algorithms. The speedup of DCF over Graphlab ranges from 5.4 to 18.01 and the average speedup is 9.76. Compared with Graphlab GD, the results demonstrate that DCF significantly improves the performance of gradient descent on

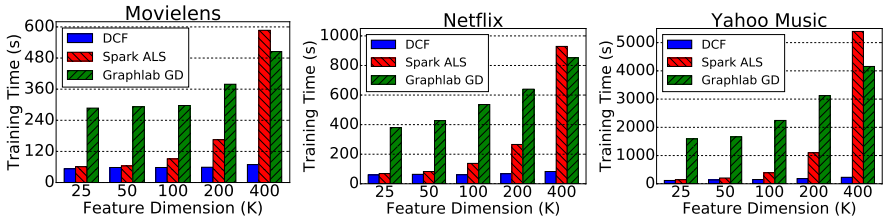


**Fig. 7.** Comparisons of DCF With Other Distributed Algorithms Under Public Datasets

distributed platform by applying dataflow with optimizations. The speedup of DCF over ALS ranges from 1.15 to 23.01 and the average speedup is 4.6. As $k$ increases, the training time increases linearly with DCF but quadratically with ALS. When $k = 400$, DCF can achieve 14.34 speedup over ALS on average. The results prove the computation complexity analysis in Section 2, and show that DCF outperforms ALS due to algorithm selection and the design.

As rating size is different in three datasets, DCF shows better performance with increasing rating size. In Movielens, Netflix and Yahoo Music, DCF has an average speedup of 3.0, 3.9, 6.9 over ALS, and 5.81, 8.29, 15.11 over Graphlab GD. In three datasets, movielens has the smallest ratings and Yahoo Music has the largest ratings. DCF shows the best performance in the largest dataset.

**Accuracy:** We also evaluate the accuracy of three algorithms on Netflix dataset. Root mean square error (RMSE) [3] is the general metrics for model based collaborative filtering algorithms. With smaller RMSE, the algorithms show better convergency performance. To get RMSE at 0.95, DCF, ALS and Graphlab GD converge with iterations of 9, 6 and 9 respectively. According above data of iteration time, DCF achieves the same accuracy with significantly improvements in performance. In addition, DCF shows the same convergency performance with other distributed gradient descent algorithms.

### 4.2 Optimizations Evaluation

In this subsection, we evaluate the optimizations of DCF algorithm in Netflix dataset. As shown in left part of Figure 8, there are four experiment algorithms used to evaluate dummy edge and multicasting. DCF-Naive represents the algorithm without using proposed optimizations and mini-batch technique is not applied to four experiment algorithms. DCF-Naive with multicasting has a speedup of 2.04 over DCF-Naive. When $k$ increases, it shows better performance, and the speedup goes up to 3.87. The results prove that multicasting can significantly improve the performance by reducing data movements. The speedup of DCF-Naive with dummy edge over DCF-Naive is not distinct. With multicasting, dummy edge can improve the performance by 30% on average. However, dummy edge improve the performance less than 5% without multicasting. This difference in two comparisons indicates that data movements is the primary challenge in this algorithm. In conclusion, dummy edge and multicasting can significantly improve the performance.

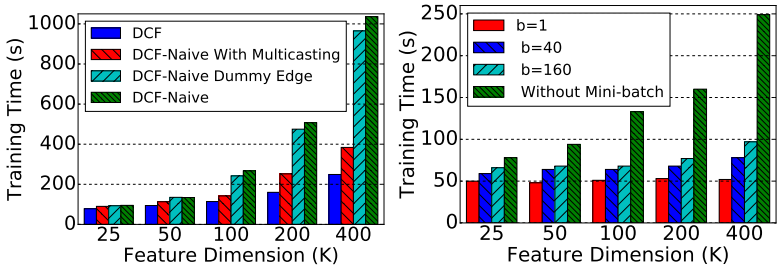


**Fig. 8.** Optimizations Evaluation of DCF

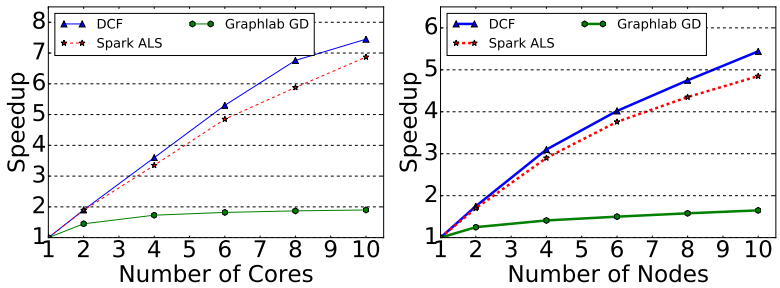


**Fig. 9.** Speedup Comparisons of Three Algorithms in Multicore System and Distributed System

As shown in the right part of Figure 8, we also evaluate mini-batch technique of DCF. $b = 1$ is a special case to set the dependency number to minimal value. Compared $b = 40$ with not using mini-batch situation, the speedup of using mini-batch strategy ranges from 1.32 to 3.19, and the average speedup is 2.08. The results demonstrate that mini-batch technique can improve the performance by optimizing computation and communication complexities. With decreasing $b$, mini-batch technique does not show obvious improvement when $b$ is less than 40. $b = 40$ situation takes 31.02% longer training time than $b = 1$, which can be considered as a similar performance with $b = 1$. However, the convergence speed of $b = 1$ is much lower than other situations. As a result, $b = 40$ is the best situation balancing the performance and the convergency.

### 4.3 Scalability

In the last subsection, we evaluate the scalability of DCF in comparison with Spark ALS and Graphlab GD. The evaluation of the three algorithms is conducted on multicore system and distributed system. Yahoo Music has the largest rating size, which leads to high computation and communication load, so we select it as the dataset for this experiment. As shown in left part of Figure 9, DCF has a near linear scalability when the core number increases, and achieves a speedup of 7.45 on 10 cores. ALS has similar scalability with DCF and achieve a speedup of 6.87 on 10 cores. Graphlab GD has the worst scalability and only achieve a speedup of 1.9 on 10 cores.

In distributed system, we evaluate the scalability on 10 nodes. As shown in right part of Figure 9, DCF achieves a speedup of 5.44 on 10 nodes, and ALS

has similar speedup of 4.85. Graphlab GD also shows the worst scalability on nodes and only achieves a speedup of 1.65 on 10 nodes.

In conclusion, DCF shows the best scalability in three algorithms and has superior performance in both multicore system and distributed system.

## 5 Related Work

In recent years, parallel researches on collaborative filtering are developing parallel algorithms for ALS and gradient descent. Mahout, the machine learning library for Hadoop [1], implemented model based algorithms with MapReduce [11] computation model. Spark [18] implemented ALS [13] and utilized its Dataframe [4] data structure to improve the performance. However, high computation complexity of ALS becomes the bottleneck of these systems.

In order to achieve higher performance, many studies focus on parallel gradient descent algorithm for its low computation complexity feature. Graphlab [17], a distributed asynchronous system, provided asynchronous gradient descent. Due to massive data movements between nodes, graphlab shows low performance in large dataset because of the network bottleneck. In recent years, parameter server architecture were proposed to support distributed gradient descent. STRADS [14] in Petuum was proposed to schedule model parallel machine learning algorithms on distributed platform. Parameter Server [16] proposed bounded synchronous technique and several network optimizations to support gradient descent. As for these parameter sever architecture based systems, the task scheduling level is on data partition but not on key-value pair. Tensorflow [2], another dataflow inspired system, only adopted pipelining feature of dataflow model to support gradient descent. As for parallel collaborative filtering studies [8] [19] [5], They mainly focused on multicore system but not distributed system.

## 6 Conclusion

Dataflow model is a natural parallel model which can exploit maximum parallelism of algorithms by analyzing data dependency to asynchronously execute tasks in runtime execution. At present, recommender systems are facing challenges of the increasing rating size and low performance of current big data systems. We design and implement dataflow based collaborative filtering (DCF) algorithm to improve the performance. We propose three optimizations (dummy edge, multicasting and mini-bach) to further improve our design by avoiding fine-grain overhead and reducing computation and data movements. The experiment results illustrate that DCF can significantly improve the performance on distributed system.

## References


1. Apache hadoop project. http://hadoop.apache.org/ (2017)
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. Savannah, Georgia, USA (2016)

3. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering 17(6), 734–749 (2005)
4. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., et al.: Spark sql: Relational data processing in spark. In: SIGMOD. pp. 1383–1394. ACM (2015)
5. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Euro-Par 23, 187–198 (Feb 2011)
6. Bobadilla, J., Ortega, F., Hernando, A., Gutiérrez, A.: Recommender systems survey. Knowledge-based systems 46, 109–132 (2013)
7. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. pp. 43–52. Morgan Kaufmann Publishers Inc. (1998)
8. Chin, W.S., Zhuang, Y., Juan, Y.C., Lin, C.J.: A fast parallel stochastic gradient method for matrix factorization in shared memory systems. ACM Transactions on Intelligent Systems and Technology (TIST) 6(1), 2 (2015)
9. Culler, D.E.: Dataflow architectures. Tech. rep., DTIC Document (1986)
10. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: Advances in neural information processing systems. pp. 1223–1231 (2012)
11. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM 51(1), 107–113 (Jan 2008)
12. Gemulla, R., Nijkamp, E., Haas, P.J., Sismanis, Y.: Large-scale matrix factorization with distributed stochastic gradient descent. In: SIGKDD. pp. 69–77. ACM (2011)
13. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: ICDM. pp. 263–272. Ieee (2008)
14. Kim, J.K., Ho, Q., Lee, S., Zheng, X., Dai, W., Gibson, G.A., Xing, E.P.: Strads: a distributed framework for scheduled model parallel machine learning. In: Proceedings of the Eleventh European Conference on Computer Systems. p. 5. ACM (2016)
15. Koren, Y., Bell, R., Volinsky, C., et al.: Matrix factorization techniques for recommender systems. Computer 42(8), 30–37 (2009)
16. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., Su, B.Y.: Scaling distributed machine learning with the parameter server. In: OSDI. vol. 14, pp. 583–598 (2014)
17. Low, Y., Gonzalez, J.E., Kyrola, A., Bickson, D., Guestrin, C.E., Hellerstein, J.: Graphlab: A new framework for parallel machine learning. arXiv preprint arXiv:1408.2041 (2014)
18. Meng, X., Bradley, J., Yuvaz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: Machine learning in apache spark. JMLR 17(34), 1–7 (2016)
19. Oh, J., Han, W.S., Yu, H., Jiang, X.: Fast and robust parallel sgd matrix factorization. In: SIGKDD. pp. 865–874. ACM (2015)
20. Takane, Y., Young, F.W., De Leeuw, J.: Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. Psychometrika 42(1), 7–67 (1977)