International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 3 | ISSN : 2456-3307

Traffic-Aware Partition and Aggregation for Big Data Applications in Map-Reduce

Dinesh Kumar S., Siddique Ibrahim S. P. , Kirubakaran R

Department of Computer Science and Engineering, Kumarguru College of Technology, Coimbatore, TamilNadu, India

ABSTRACT

The Map Reduce programming model simplifies large-scale data processing on commodity cluster by exploiting parallel map tasks and reduces tasks. Map Reduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster .Although many efforts have been made to improve the performance of Map Reduce jobs, they ignore the network traffic generated in the shuffle phase, which plays a critical role in performance enhancement. Traditionally, a hash function is used to partition intermediate data among reduce tasks, which, however, is not traffic - efficient because network topology and data size associated with each key are not taken into consideration. The objective of this system is to reduce the network traffic cost for a map reduce job by designing a intermediate data partition scheme.

Keywords: Map Reduce, Hadoop, Stragglers, Partition

I. INTRODUCTION

Map Reduce has emerged as the most popular computing framework for big data processing due to its simple programming model and automatic management of parallel execution. MapReduce is an open source. Map Reduce is used for various big data applications, such as machine learning bioinformatics and cyber security. Map Reduce divides a computation into two main phases, namely map and reduce which in turn are carried out by several map tasks and reduce tasks, respectively. In the map phase, map tasks are launched in parallel to convert the original input splits into intermediate data in a form of key/value pairs. These key/value pairs are stored on local machine and organized into multiple data partitions, one per reduce task. In the reduce phase, each reduce task fetches its own share of data partitions from all map tasks to generate the final result.

There is a shuffle step between map and reduce phase. In this step, the data produced by the map phase are ordered, partitioned and transferred to the appropriate machines executing the reduce phase. The resulting network traffic pattern from all map tasks to all reduce tasks can cause a great volume of network traffic, imposing a serious constraint on the efficiency of data analytic applications.

We consider a toy example with two map tasks and two reduce tasks, where intermediate data of three keys K1, K2, and K3 are denoted by rectangle bars under each machine. If the hash function assigns data of K1 and K3 to reducer 1, and K2 to reducer 2, a large amount of traffic will go through the top switch. To tackle this problem occurred by the traffic-oblivious partition scheme, we take into account of both task locations and data size associated with each key in this paper. By assigning keys with larger data size to reduce tasks closer to map tasks, network traffic can be significantly reduced. In the same example above, if we assign K1 and K3 to reducer 2, and K2 to reducer 1, as shown in Fig. 1(b), the data transferred through the top switch will be significantly reduced.

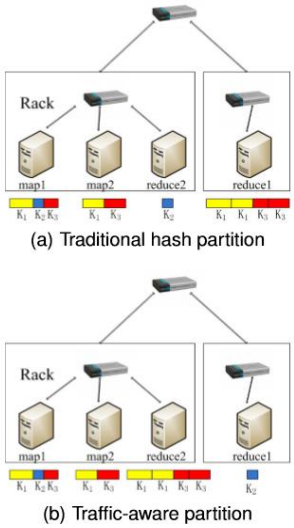


Figure 1. Two Map Reduce partition schemes

To further reduce network traffic within a MapReduce job, we consider to aggregate data with the same keys before sending them to remote reduce tasks. Although a similar function, called combine, has been already adopted by Hadoop, it operates immediately after a map task solely for its generated data, failing to exploit the data aggregation opportunities among multiple tasks on different machines. As an example shown in Fig. 2(a), in the traditional scheme, two map tasks individually send data of key K1 to the reduce task. If we aggregate the data of the same keys before sending them over the top switch, as shown in Fig. 2(b), the network traffic will be reduced.

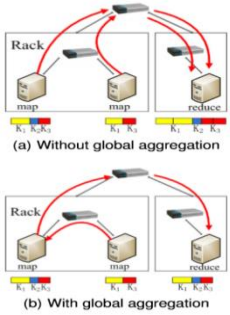


Figure 1. Two schemes of intermediate data transmission in the shuffle phase.

In this paper, we jointly consider data partition and aggregation for a MapReduce job with an objective that is to minimize the total network traffic. In particular, we propose a distributed algorithm for big data applications by decomposing the original large-scale problem into several sub problems that can be solved in parallel. Moreover, an online algorithm is designed to deal with the data partition and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost in both offline and online cases.

II. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM:

Existing problem of optimizing network usage in MapReduce scheduling in the reason that we are interested in network usage is twofold. Firstly, network utilization is a quantity of independent interest, as it is directly related to the throughput of the system. Note that the total amount of data processed in unit time is simply (CPU utilization) · (CPUcapacity) + (networkutilization)·(network capacity). CPU utilization will always be 1 as long as there are enough jobs in the map queue, but network utilization can be very sensitive to scheduling network utilization has been identified as a key component in optimization of MapReduce systems in several previous works.

Network usage could lead us to algorithms with smaller mean response time. We find the main motivation for this direction of our work in the results of the aforementioned overlap between map and shuffle phases, are shown to yield significantly better mean response time than Hadoop's fair scheduler. However, we observed that neither of these two algorithms explicitly attempted to optimize network usage, which suggested room for improvement. MapReduce has become one of the most popular frameworks for large-scale distributed computing, there exists a huge body of work regarding performance optimization of MapReduce.

For instance, researchers have tried to optimize MapReduce systems by efficiently detecting and eliminating the so-called "stragglers" providing better locality of data preventing starvation caused by large jobs analyzing the problem from a purely theoretical viewpoint of shuffle workload available at any given time is closely related to the output rate of the map phase, due to the inherent dependency between the map and shuffle phases. In particular, when the job that is being processed is 'map-heavy,' the available workload of the same job in the shuffle phase is upper-bounded by the output rate of the map phase. Therefore, poor scheduling of map tasks can have adverse effects on the throughput of the shuffle phase, causing the network to

be idle and the efficiency of the entire system to decrease.

2.1.1 DISADVANTAGES

Existing model, called the overlapping tandem queue model, is a job-level model for MapReduce where the map and shuffle phases of the MapReduce framework are modeled as two queues that are put in tandem. Since it is a job-level model, each job is represented by only the map size and the shuffle size simplification is justified by the introduction of two main assumptions. The first assumption states that each job consists of a large number of small-sized tasks, which allows us to represent the progress of each phase by real numbers.

2.2 PROPOSED SYSTEM

In this paper, we jointly consider data partition and aggregation for a MapReduce job with an objective that is to minimize the total network traffic. In particular, we propose a distributed algorithm for big data applications by decomposing the original large-scale problem into several sub problems that can be solved in parallel. Moreover, an online algorithm is designed to deal with the data partition and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost in both offline and online cases.

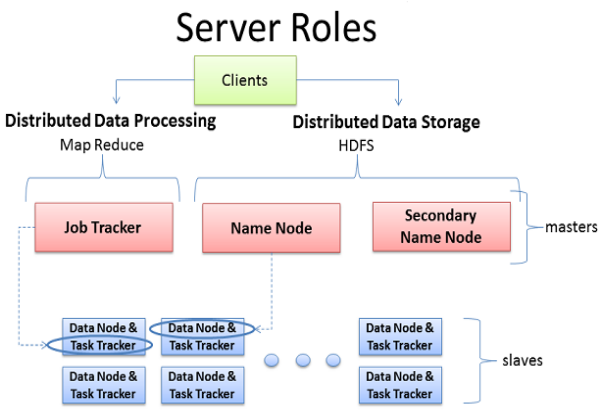
MapReduce resource allocation system, to enhance the performance of MapReduce jobs in the cloud by locating intermediate data to the local machines or close-by physical machines in this locality-awareness reduces network traffic in the shuffle phase generated in the cloud data center. However, little work has studied to optimize network performance of the shuffle process that generates large amounts of data traffic in MapReduce jobs. A critical factor to the network performance in the shuffle phase is the intermediate data partition. The default scheme adopted by Hadoop is hash-based partition that would yield unbalanced loads among reduce tasks due to its unawareness of the data size associated with each key.

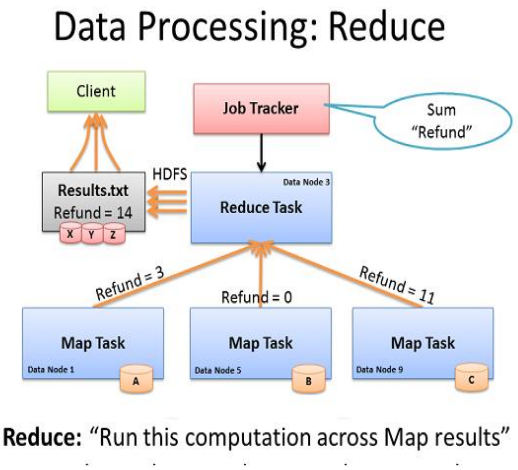
We have developed a fairness-aware key partition approach that keeps track of the distribution of intermediate keys' frequencies, and guarantees a fair distribution among reduce tasks. have introduced a combiner function that reduces the amount of data to be shuffled and merged to reduce tasks an in-mapper combining scheme by exploiting the fact that mappers can preserve state across the processing of multiple input key/value pairs and defer emission of intermediate data until all input records have been processed. Both proposals are constrained to a single map task, ignoring the data aggregation opportunities from multiple map tasks a MapReduce-like system to decrease the traffic by pushing aggregation from the edge into the network.

2.2.1 ADVANTAGES

- Our proposed distributed algorithm and the optimal solution obtained by solving the MILP formulation. Due to the high computational complexity of the MILP formulation, we consider small-scale problem instances with 10 keys in this set of simulations.
- Our distributed algorithm is very close to the optimal solution. Although network traffic cost increases as the number of keys grows for all algorithms, the performance enhancement of our proposed algorithms to the other two schemes becomes larger.
- Our distributed algorithm with the other two schemes a default simulation setting with a number of parameters, and then study the performance by changing one parameter while fixing others. We consider a MapReduce job with 100 keys and other parameters are the same above. the network traffic cost shows as an increasing function of number of keys from 1 to 100 under all algorithms.

III. ARCHITECTURE DIAGRAM





3.1 ONLINE EXTENSION OF HRA AND HNA

In this section, we conduct extensive simulations to evaluate the performance of our proposed distributed algorithm DA. We compare DA with HNA, which is the default method in Hadoop. To our best knowledge, we are the first to propose the aggregator placement algorithm, and compared with the HRA that focuses on a random aggregator placement. All simulation results are averaged over 30 random instances.

HNA: Hash-based partition with No Aggregation. It exploits the traditional hash partitioning for the intermediate data, which are transferred to reducers without going through aggregators. It is the default method in Hadoop.

HRA: Hash-based partition with Random Aggregation. It adds a random aggregator placement algorithm based on the traditional Hadoop. Through randomly placing aggregators in the shuffle phase, it aims to reducing the network traffic cost in the comparison of traditional method in Hadoop.

Our proposed distributed algorithm and the optimal solution obtained by solving the MILP formulation. Due to the high computational complexity of the MILP formulation, we consider small-scale problem instances with 10 keys in this set of simulations. Each key associated with random data size within [1-50]. There are 20 mappers, and 2 reducers on a cluster of 20 machines. The parameter α is set to 0.5. The distance between any two machines is randomly chosen within [1-60]. The performance of our distributed algorithm is very close to the optimal solution. Although network traffic cost increases as the number of keys grows for all algorithms, the performance enhancement of our proposed algorithms to the other two schemes becomes larger. When the number of keys is set to 10, the default algorithm HNA has a cost of 5.0 × 104 while optimal solution is only 2.7×104 , with 46% traffic reduction.

3.2 ALGORITHM

3.2.1 DISTRIBUTED ALGORITHM

The problem above can be solved by highly efficient approximation algorithms, e.g., branch-and-bound, and fast off-the-shelf solvers, e.g., CPLEX, for moderate-sized input. An additional challenge arises in dealing with the MapReduce job for big data. In such a job, there are hundreds or even thousands of keys, each of which is associated with a set of variables (e.g., x p ij and y p k) and constraints in our formulation, leading to a large-scale optimization problem that is hardly handled by existing algorithms and solvers in practice.

Algorithm 1 Distributed Algorithm

1: set $t = 1$, and $\nu_j^p (j \in A, p \in P)$ to arbitrary nonnegative values;
2: **for** $t < T$ **do**
3: distributively solve the subproblem **SUB_DP** and **SUB_AP** on multiple machines in a parallel manner;
4: update the values of ν_j^p with the gradient method (15), and send the results to all subproblems;
5: set $t = t + 1$;
6: **end for**

3.2.2 ONLINE ALGORITHM

We take the data size m p i and data aggregation ratio αj as input of our algorithms. In order to get their values, we need to wait all mappers to finish before starting reduce tasks, or conduct estimation via profiling on a small set of data. In practice, map and reduce tasks may partially overlap in execution to increase system throughput, and it is difficult to estimate system parameters at a high accuracy for big data applications. These motivate us to design an online algorithm to dynamically adjust data partition and aggregation during the execution of map and reduce tasks.

Algorithm 2 Online Algorithm

1: $t = 1$ and $\hat{t} = 1$;
2: solve the OPT_ONE_SHOT problem for $t = 1$;
3: **while** $t \leq T$ **do**
4: **if** $\sum_{\tau=\hat{t}}^{t} \sum_{p \in P} C_t^p(\tau) > \gamma C_M(\hat{t})$ **then**
5: solve the following optimization problem:

$$\min \sum_{p \in P} C^p(t)$$

subject to:$(1) - (7), (9),$ and $(10),$ for time slot t.

6: **if** the solution indicates a migration event **then**
7: conduct migration according to the new solution;
8: $\hat{t} = t$;
9: update $C_M(\hat{t})$;
10: **end if**
11: **end if**
12: $t = t + 1$;
13: **end while**

3.3 MODULES

- **SERVER CLIENTS**
- **DITRIBUTED DATA**
- **SHEDULING TASK**
- **NETWORK TRAFFIC TRACES**
- **MAPREDUCE TASK**

3.4 MODULE DESCRIPTION

3.4.1 SERVER CLIENTS

Client-server computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client also shares any of its resources; Clients therefore initiate communication sessions with servers which await (listen to) incoming requests.

3.4.2 DITRIBUTED DATA

We develop a distributed algorithm to solve the problem on multiple machines in a parallel manner. Our basic idea is to decompose the original large-scale problem into several distributive solvable sub problems that are coordinated by a high-level master problem. We jointly consider data partition and aggregation for a MapReduce job with an objective that is to minimize the total network traffic. In particular, we propose a distributed algorithm for big data applications by decomposing the original large-scale problem into several sub problems that can be solved in parallel. Moreover, an online algorithm is designed to deal with the data partition and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost in both offline and online cases.

4.4.3 SHEDULING TASK

MapReduce divides a computation into two main phases, namely map and reduce which in turn are carried out by several map tasks and reduce tasks, respectively. In the map phase, map tasks are launched in parallel to convert the original input splits into intermediate data in a form of key/value pairs. These key/value pairs are stored on local machine and organized into multiple data partitions, one per reduce task. In the reduce phase, each reduce task fetches its own share of data partitions from all map tasks to generate the final result. There is a shuffle step between map and reduce phase. In this step, the data produced by the map phase are ordered, partitioned and transferred to the appropriate machines executing the reduce phase. The resulting network traffic pattern from all map tasks to all reduce tasks can cause a great volume of network traffic, imposing a serious constraint on the efficiency of data analytic applications.

3.4.4 NETWORK TRAFFIC TRACES

Network traffic within a MapReduce job, we consider to aggregate data with the same keys before sending them to remote reduce tasks. Although a similar function, called combiner has been already adopted by Hadoop, it operates immediately after a map task solely for its generated data, failing to exploit the data aggregation opportunities among multiple tasks on different machines. As an example shown in Fig. 2(a), in the traditional scheme, two map tasks individually send data of key K1 to the reduce task. If we aggregate the data of the same keys before sending them over the top switch, as shown in Fig. 2(b), the network traffic will be reduced. We tested the real network traffic cost in Hadoop using the real data source from latest dumps files in Wikimedia (http://dumps.wikimedia.org/enwiki/latest/). In the meantime, we executed our distributed algorithm using the same data source for comparison. Since our distributed algorithm is based on a known aggregation ratio _, we have done some experiments to evaluate it in Hadoop environment.

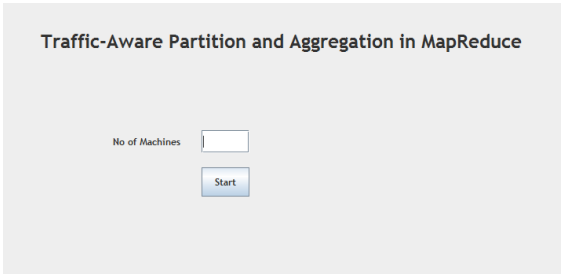
3.4.5 MAPREDUCE TASK

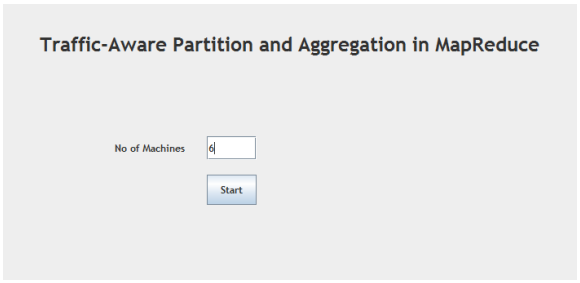
We focus on MapReduce performance improvement by optimizing its data transmission optimizing network usage can lead to better system performance and found that high network utilization and low network congestion should be achieved simultaneously for a job with good performance. MapReduce resource allocation system, to enhance the performance of MapReduce jobs in the cloud by locating intermediate data to the local machines or close-by physical machines locality-awareness reduces network traffic in the shuffle phase generated in the cloud data center. However, little work has studied to optimize network performance of the shuffle process that generates large amounts of data traffic in MapReduce jobs. A critical factor to the network performance in the shuffle phase is the intermediate data partition. The default scheme adopted by Hadoop is hash-based partition that would yield unbalanced loads among reduce tasks due to its unawareness of the data size associated with each key.

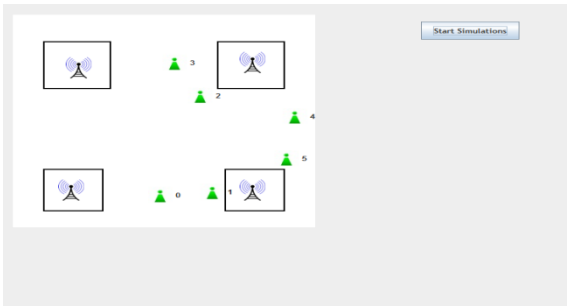
To overcome this shortcoming, we have developed a fairness-aware key partition approach that keeps track of the distribution of intermediate keys' frequencies, and guarantees a fair distribution among reduce tasks. In addition to data partition, many efforts have been made on local aggregation, in-mapper combining and in-network aggregation to reduce network traffic within MapReduce jobs. have introduced a combiner function that reduces the amount of data to be shuffled and merged to reduce tasks an in-mapper combining scheme by exploiting the fact that mappers can preserve state across the processing of multiple input key/value pairs and defer emission of intermediate data until all input records have been processed. Both proposals are constrained to a single map task, ignoring the data aggregation opportunities from multiple map tasks have proposed a MapReduce-like system to decrease the traffic by pushing aggregation from the edge into the network.

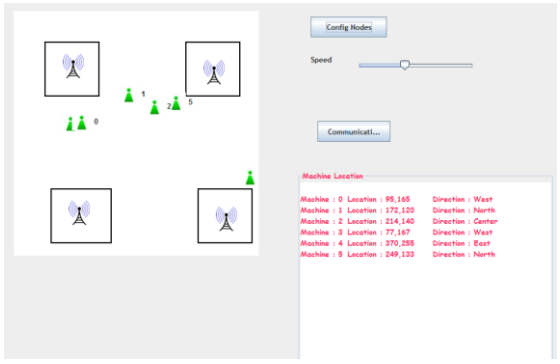
IV. APPENDIX

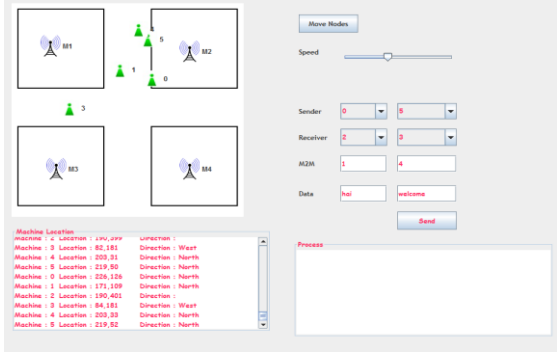
4.1 SAMPLE SCREEN SHOTS

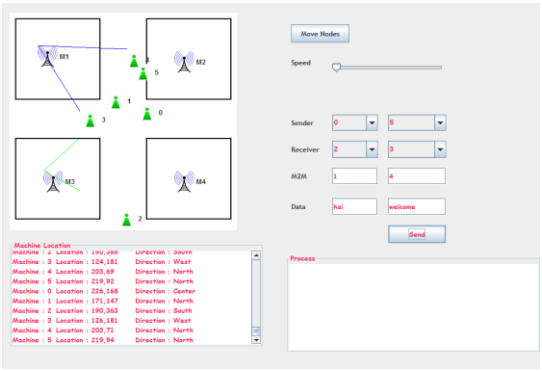


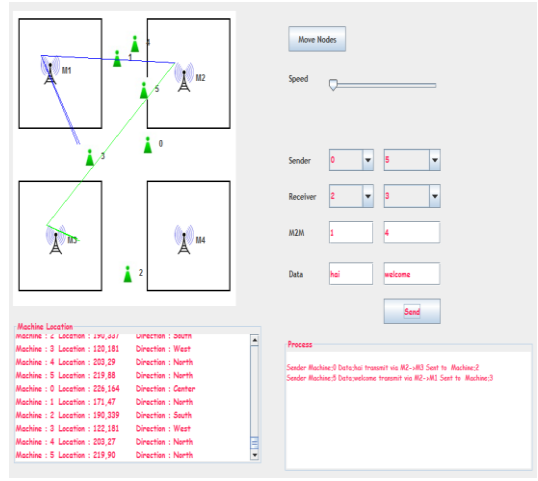


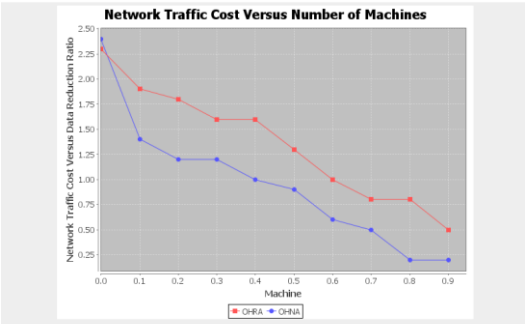


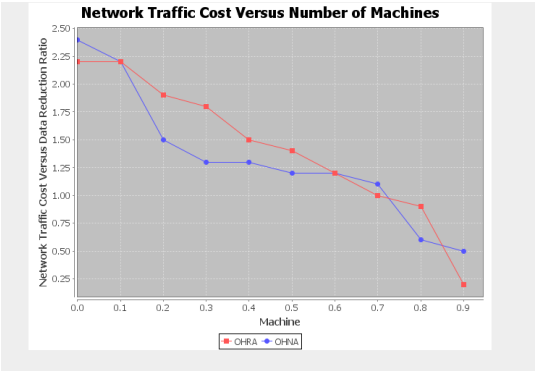












V. CONCLUSION

In this paper, we study the joint optimization of intermediate data partition and aggregation in MapReduce to minimize network traffic cost for big data applications. We propose a three-layer model for this problem and formulate it as a mixed-integer nonlinear problem, which is then transferred into a linear form that can be solved by mathematical tools. To deal with the large-scale formulation due to big data, we design a distributed algorithm to solve the problem on multiple machines. Furthermore, we extend our algorithm to handle the MapReduce job in an online manner when some system parameters are not given. Finally, we conduct extensive simulations to evaluate our proposed algorithm under both offline cases and online cases. The simulation results demonstrate that our proposals can effectively reduce network traffic cost under various network settings.

VI. REFERENCES

[1]. J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[2]. W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in INFOCOM, 2013 Proceedings IEEE. IEEE, 2013, pp. 1609–1617.

[3]. F. Chen, M. Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in INFOCOM, 2012 Proceedings IEEE. IEEE, 2012, pp. 1143–1151.

[4]. Y. Wang, W. Wang, C. Ma, and D. Meng, "Zput: A speedy data uploading approach for the hadoop distributed file system," in Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE, 2013, pp. 1–5.

[5]. S. Chen and S. W. Schlosser, "Map-reduce meets wider varieties of applications," Intel Research Pittsburgh, Tech. Rep. IRP-TR-08-05, 2008.

[6]. J. Rosen, N. Polyzotis, V. Borkar, Y. Bu, M. J. Carey, M. Weimer, T. Condie, and R. Ramakrishnan, "Iterative mapreduce for large scale machine learning," arXiv preprint arXiv:1303.3517, 2013.

[7]. S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber, "Presto: distributed machine learning and graph processing with sparse matrices," in Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013, pp. 197– 210.