Scientific Journal of Impact Factor (SJIF): 5.71
e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406
International Journal of Advance Engineering and Research Development
Volume 5, Issue 03, March -2018

ON TRAFFIC-AWARE PARTITION AND AGGREGATION IN MAP REDUCE FOR BIG DATA APPLICATIONS

Kirubakaran.S[1], Indhu.G[2], Indhuja.S[3], Indhumathi.S[4], John Veslin.M[5]

[1]*Associate Professor, Info Institute of Engineering, Kovilpalayam, Coimbatore- 641 107*
[2,3,4,5]*UG graduate, Info Institute of Engineering, Kovilpalayam, Coimbatore- 641 107*

Abstract---*Map Reduce 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. We jointly consider data partition and aggregation for a Map Reduce 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 subproblems 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.*

Keywords---*Map Reduce, Hadoop, Data aggregation, Dynamic manner, Online algorithm, Distributed algorithm.*

IINTRODUCTION

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. Map Reduce and its open source implementation Hadoop have been adopted by leading companies, such as Yahoo!, Google and Facebook, for various big data applications, such as machine learning bioinformatics and cybersecurity. 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.

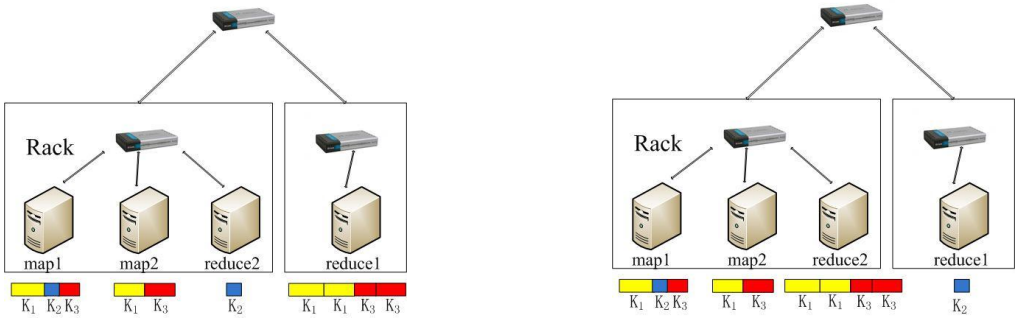
In this step, the data produced by the map phase are ordered, partitioned and transferred to the appropriate machines executing the reduce phase. The resulting network traffic pattern from all map tasks to all reduce tasks can cause a great volume of network traffic, imposing a serious constraint on the efficiency of data analytic applications. For example, with tens of thousands of machines, data shuffling accounts for 58.6% of the cross-pod traffic and amounts to over 200 petabytes in total in the analysis of SCOPE jobs. For shuffle-heavy Map Reduce tasks, the high traffic could incur considerable performance overhead up to 30-40 % as shown in default, intermediate data are shuffled according to a hash function in Hadoop, which would lead to large network traffic because it ignores network topology and data size associated with each key.

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 incurred 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.

To further reduce network traffic within a Map Reduce 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.

In this paper, we jointly consider data partition and aggregation for a Map Reduce 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 subproblems 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.



(a) Traditional hash partition (b) Traffic-aware Partiton

Fig. 1.Two Map Reduce partition schemes.

II RELATED WORK

Most existing work focuses on Map Reduce performance improvement by optimizing its data transmission. Blanca et al have investigated the question of whether optimizing network usage can lead to better system performance and found that high network utilization and low network congestion should be achieved simul-taneously for a job with good performance. Palanisamyet al have presented Purlieus, a Map Reduce re-source allocation system, to enhance the performance of Map Reduce jobs in the cloud by locating intermediate data to the local machines or close-by physical ma-chines. 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 Map Reduce 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, Ibrahim et al. [20] 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. Meanwhile, Liya et al. [21] have designed an algorithm to schedule operations based on the key distribution of intermediate key/value pairs to improve the load balance. Lars et al. [22] have proposed and eval-uated two effective load balancing approaches to data skew handling for Map Reduce-based entity resolution. Unfortunately, all above work focuses on load balance at reduce tasks, ignoring the network traffic during the shuffle phase.

In addition to data partition, many efforts have been made on local aggregation, in-mapper combining and in-network aggregation to reduce network traffic withinMap Reduce jobs. Condie et al. [23] have introduced a combiner function that reduces the amount of data to be shuffled and merged to reduce tasks. Lin and Dyerhave proposed an in-mapper combining scheme by exploiting the fact that mappers can preserve state across the processing of multiple input key/value pairs and de-fer 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. Costa et al. [25] have proposed a Map Reduce-like system to decrease the traffic by pushing aggregation from the edge into the network. However, it can be only applied to the network topology with servers directly linked to other servers, which is of limited practical use.Different from existing work, we investigate network traffic reduction within Map Reduce jobs by jointly ex-ploiting traffic-aware intermediate data partition and data aggregation among multiple map tasks.

(a) Without global aggregation (b) With global aggregation

Fig. 2. Two schemes of intermediate data transmission in the shuffle phase.

III PRELIMINARIES

This section presents the related concepts of Storm and OrientStream, and gives the problem definition and description.

3.1SYSTEM MODEL:

Map Reduce is a programming model based on two primitives: map function and reduce function. The for-mer processes key/value pairs hk; vi and produces a set of intermediate key/value pairs hk^0; v^0i. Intermedi-ate key/value pairs are merged and sorted based on the intermediate key k^0 and provided as input to the reduce function. A Map Reduce job is executed over a distributed system composed of a master and a set of workers. The input is divided into chunks that are assigned to map tasks. The master schedules map tasks in the workers by taking into account of data locality. The output of the map tasks is divided into as many partitions as the number of reducers for the job. Entries with the same intermediate key should be assigned to the same partition to guarantee the correctness of the execution. All the intermediate key/value pairs of a given partition are sorted and sent to the worker with the corresponding reduce task to be executed. Default scheduling of reduce tasks does not take any data local-ity constraint into consideration. As a result, the amount of data that has to be transferred through the network in the shuffle process may be significant.

In this paper, we consider a typical Map Reduce job on a large cluster consisting of a set N of machines. We let d_{xy} denote the distance between two machines x and y, which represents the cost of delivering a unit data. When the job is executed, two types of tasks, i.e., map and reduce, are created. The sets of map and reduce tasks are denoted by M and R, respectively, which are already placed on machines. The input data are divided into independent chunks that are processed by map tasks in parallel. The generated intermediate results in forms of key/value pairs may be shuffled and sorted by the framework, and then are fetched by reduce tasks to produce final results. We let P denote the set of keys contained in the intermediate results, and m^p_i denotethe data volume of key/value pairs with key p 2 P generated by mapper i 2 M.

A set of aggregators are available to the intermediate results before they are sent to reducers. These aggregators can be placed on any machine, and one is enough for data aggregation on each machine if adopted. The data reduction ratio of an aggregator is denoted by , which can be obtained via profiling before job execution.The cost of delivering a certain amount of traffic over a network link is evaluated by the product of data size and link distance. Our objective in this paper is to minimize the total network traffic cost of a Map Reduce job by jointly considering aggregator placement and intermediate data partition.

3.2 PROBLEM FORMULATION

In this section, we formulate the network traffic mini-mization problem. To facilitate our analysis, we construct an auxiliary graph with a three-layer structure as shown in Fig. 3. The given placement of mappers and reducers applies in the map layer and the reduce layer, respec-tively. In the aggregation layer, we create a potential aggregator at each machine, which can aggregate data from all mappers. Since a single potential aggregator is sufficient at each machine, we also use N to denote all potential aggregators. In addition, we create a shadownode for each mapper on its residential machine. In contrast with potential aggregators, each shadow node can receive data only from its corresponding mapper in the same machine. It mimics the process that the generated intermediate results will be delivered to a reduce directly without going through any aggregator. All nodes in the aggregation layers are maintained in set A. Finally, the output data of aggregation layer are sent to the reduce layer. Each edge (u; v) in the auxiliary graph is associated with a weight $d_{(u)(v)}$, where (u) denotes the machine containing node u in the auxiliary graph.

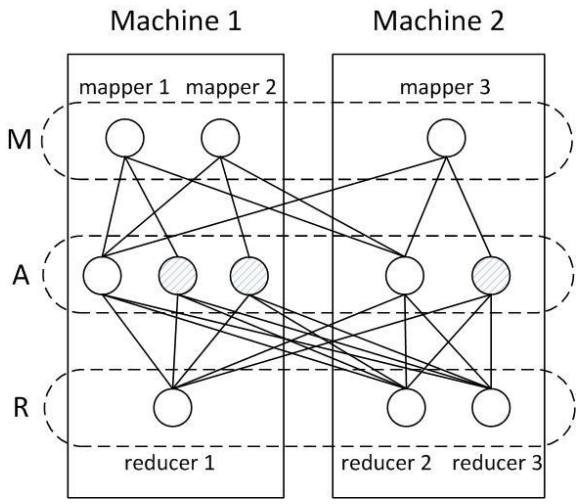


Fig. 3.Three-layer model for the network traffic minimization problem.

IV SYSTEM ANALYSIS

4.1 EXISTING SYSTEM:

Existing problem of optimizing network usage in Map Reduce 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)·(CPU capacity)+ (network utilization)·(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 Map Reduce 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. Map Reduce has become one of the most popular frameworks for large-scale distributed computing, there exists a huge body of work regarding performance optimization of Map Reduce.

For instance, researchers have tried to optimize Map Reduce 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.

4.1.1 DIS-ADVANTAGES:

Existing model, called the overlapping tandem queue model, is a job-level model for Map Reduce where the map and shuffle phases of the Map Reduce 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.

The job-level model offers two big disadvantages over the more complicated task-level models.

Firstly, it gives rise to algorithms that are much simpler than those of task-level models, which enhances chances of being deployed in an actual system.

Secondly, the number of jobs in a system is often smaller than the number of tasks by several orders of magnitude, making the problem computationally much less strenuous note that there are still some questions to be studied regarding the general applicability of the additional assumptions of the job-level model, which are interesting research questions in their own light

4.2 PROPOSED SYSTEM:

In this paper, we jointly consider data partition and aggregation for a Map Reduce 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 subproblems 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.

Map Reduce resource allocation system, to enhance the performance of Map Reduce 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 Map Reduce 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 Map Reduce-like system to decrease the traffic by pushing aggregation from the edge into the network.

4.2.1 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 Map Reduce 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.

V ALGORITHM

5.1 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 Map Reduce 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**

5.2 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:$(1) - (7), (9), \text{and } (10), \text{for time slot } t.$

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**

VI CONCLUSION AND FUTURE WORK

In this paper, we study the joint optimization of intermediate data partition and aggregation in Map Reduce 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 Map Reduce 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.

VII REFERENCES

[1] J. Dean and S. Ghemawat, "Map Reduce: 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 Map Reduce 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 Map Reduce 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 Map Reduce 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.

[8] J. Wang, D. Crawl, I. Altintas, K. Tzoumas, and V. Markl, "Comparison of distributed data-parallelization patterns for big data analysis: A bioinformatics case study," in Proceedings of the Fourth International Workshop on Data Intensive Computing in the Clouds (DataCloud), 2013.

[9] R. Liao, Y. Zhang, J. Guan, and S. Zhou, "Cloudnmf: A Map Reduce implementation of nonnegative matrix factorization for largescale biological datasets," Genomics, proteomics & bioinformatics, vol. 12, no. 1, pp. 48–51, 2014.

