www.ijiarec.com

ISSN:2348-2079

Volume-6 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Network Traffic Separation and Aggregation in Mapreduce for Bigdata Applications

[1]Mrs. K.E.Eswari, M.C.A., M.Phil., M.E.,Associate Professor/MCA
[2]Ms.P.Sandhiya, Final MCA,
Department of MCA , Nandha Engineering College(Autonomous), Erode-52.
E-mail ID:eswari.eswaramoorthy@nandhaengg.org,sandhiya17496@gmail.com.

Abstract: ***The 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. In this paper, we study to reduce network traffic cost for a MapReduce job by designing a novel intermediate data partition scheme. Furthermore, we jointly consider the aggregator placement problem, where each aggregator can reduce merged traffic from multiple map tasks. A decomposition-based distributed algorithm is proposed to deal with the large-scale optimization problem for big data application.***

Index term: MapReduce, Network traffic, Network topology.

I.INTRODUCTION

MapReduce[1][2][3] 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 and its open source implementation Hadoop[4][5] have been adopted by leading companies, such as Yahoo!, Google and Facebook, for various big data applications.

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.

For shuffle-heavy MapReduce tasks, the high traffic could incur considerable performance overhead up to 30-40 % .By 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.

As shown in Fig.1(a), 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 MapReduce job, we consider to aggregate data with the same keys before sending them to remote reduce tasks. 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),

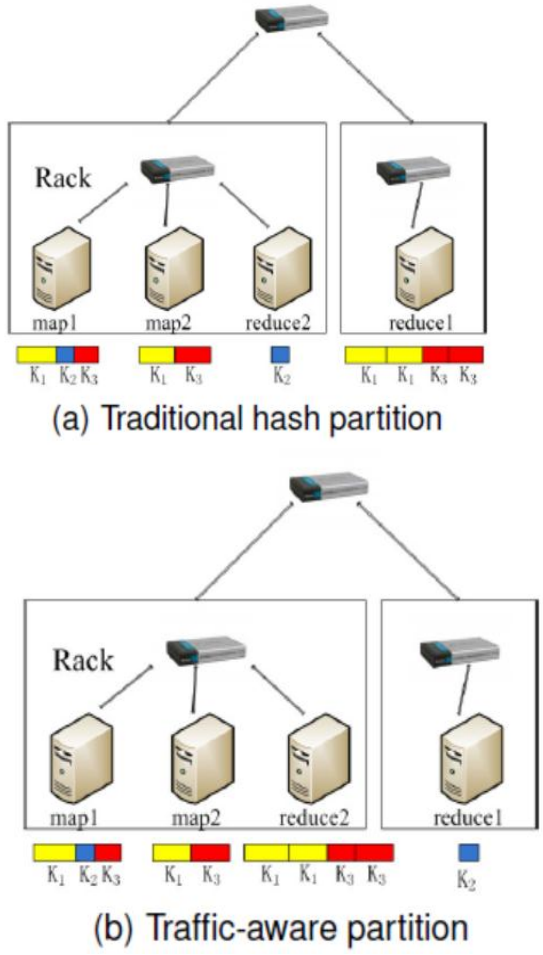

Fig. 1. Two MapReduce partition schemes.

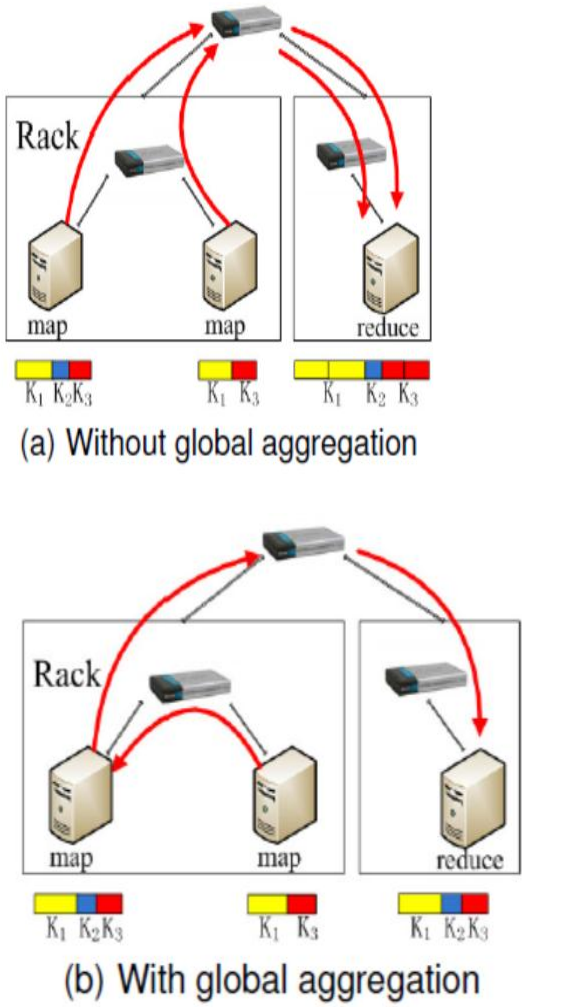

Fig. 2. Two schemes of intermediate data transmission in the shuffle phase.

II. RELATED WORK

Most existing work focuses on MapReduce 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 simultaneously for a job with good performance.

Palanisamy et al. have presented Purlieus, a MapReduce 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.

Lin and Dyer have 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 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. Costa et al 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.

III. SYSTEM MODEL

MapReduce is a programming model based on two primitives: map function and reduce function. The past processes key/value pairs hk; vi and produces a set of intermediate key/value pairs hk0; v0i. Intermediate key/value pairs are merged and sorted based on the intermediate key k0 and provided as input to the reduce function. In this paper, we consider a typical MapReduce job on a large cluster consisting of a set N of machines. We let dxy 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. We let P denote the set of keys contained in the intermediate results, and mp i denote the 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.

Notations	Description
N	a set of physical machines
d_{xy}	distance between two machines x and y
M	a set of map tasks in map layer
R	a set of reduce tasks in reduce layer
A	a set of nodes in aggregation layer
P	a set of intermediate keys
A_i	a set of neighbors of mapper $i \in M$
δ	maximum number of aggregators
m_i^p	data volume of key $p \in P$ generated by mapper $i \in M$
$\phi(u)$	the machine containing node u
x_{ij}^p	binary variable denoting whether mapper $i \in M$ sends data of key $p \in P$ to node $j \in A$
f_{ij}^p	traffic for key $p \in P$ from mapper $i \in M$ to node $j \in A$
I_j^p	input data of key $p \in P$ on node $j \in A$

IV. PROBLEM FORMULATION

In this section, we formulate the network traffic minimization 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, respectively Since a single potential aggregator is sufficient at each machine, we also use N to denote all potential aggregators. In addition, we create a shadow,

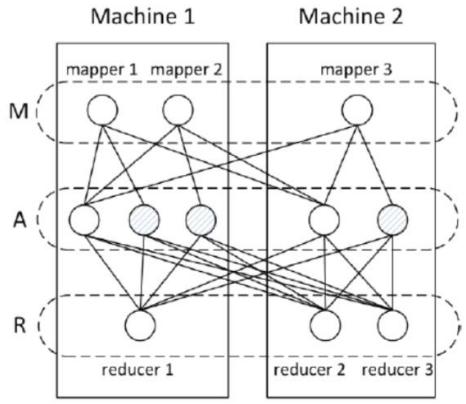

Fig. 3. Three-layer model for the network traffic minimization problem.

node 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. 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.

To formulate the traffic minimization problem, we first consider the data forwarding between the map layer and the aggregation layer. We define a binary variable

$$x_{ij}^p = \begin{cases} 1, \text{ if mapper } i \in M \text{ sends data of key } p \in P \\ \quad \text{to node } j \in A; \\ 0, \text{ otherwise.} \end{cases}$$

Since all data generated in the map layer should be sent to nodes in the aggregation layer,we have the following constraint for x^p_{ij};

$$\sum_{j \in \mathcal{A}_i} x_{ij}^p = 1, \forall i \in M, p \in P, \quad (1)$$

Where A_i denotes the set of neighbors of mapper i in the aggregation layer. we let f^p_{ij} denote the traffic from mapper $i \in M$ to node $j \in A$, wich can be calculated by:

$$f_{ij}^p = x_{ij}^p m_i^p, \forall i \in M, j \in A_i, p \in P. \quad (2)$$

The input data of node $j \in A$ can be calculated by summing up all incoming traffic, i.e.,

$$I_j^p = \sum_{i \in M_j} f_{ij}^p, \forall j \in A, p \in P, \quad (3)$$

$$O_j^p = \alpha_j I_j^p, \forall j \in A, p \in P, \quad (4)$$

where $\alpha_j = \alpha$ if node j is a potential aggregator.Otherwise, i.e., node j is a shadow node, we have $\alpha_j = 1$. We further define a binary variable z_j for aggregator placement, i.e.,

$$z_j = \begin{cases} 1, \text{ if a potential aggregator } j \in N \text{ is activated} \\ \quad \text{for data aggregation,} \\ 0, \text{ otherwise.} \end{cases}$$

Since the total number of aggregators is constrained by δ, we have:

$$\sum_{j \in N} z_j \leq \delta. \quad (5)$$

The relationship among x^p_{ij} and z_j can be represented by:

$$x_{ij}^p \leq z_j, \forall j \in N, i \in M_j, p \in P. \quad (6)$$

In other words, if a potential aggregator $j \in N$ is not activated for data aggregation, i.e., $z_j = 0$, no data should be forwarded to it, i.e., $x^p_{ij} = 0$.

Finally, we define a binary variable y_k^p to describe intermediate data partition at reducers, i.e.,

$$y_k^p = \begin{cases} 1, \text{ if data of key } p \in P \text{ are processed by} \\ \quad \text{reducer } k \in R, \\ 0, \text{ otherwise.} \end{cases}$$

Since the intermediate data with the same key will be processed by a single reducer, we have the constraint:

$$\sum_{k \in R} y_k^p = 1, \forall p \in P. \quad (7)$$

The network traffic from node $j \in A$ to reducer $k \in R$ can be calculated by:

$$g_{jk}^p = O_j y_k^p, \forall j \in A, k \in R, p \in P. \quad (8)$$

With the objective to minimize the total cost of net-work traffic within the MapReduce job, the problem can be formulated as:

$$\min \sum_{p \in P} \left(\sum_{i \in M} \sum_{j \in A_i} f_{ij}^p d_{ij} + \sum_{j \in A} \sum_{k \in R} g_{jk}^p d_{jk} \right)$$
$$\text{subject to:} \quad (1) - (8).$$

Note that the formulation above is a mixed-integer nonlinear programming (MINLP) problem. By applying linearization technique, we transfer it to a mixed-integer linear programming (MILP) that can be solved by ex-isting mathematical tools. Specifically, we replace the nonlinear constraint (8) with the following linear ones:

$$0 \leq g_{jk}^p \leq O_j^p, \forall j \in \mathcal{A}, k \in R, p \in P, \quad (9)$$
$$O_j^p - (1 - y_k^p)\bar{O}_j^p \leq g_{jk}^p \leq \bar{O}_j^p, \forall j \in \mathcal{A}, k \in R, p \in P, \quad (10)$$

where constant $\bar{O}_j^p = \alpha_j \sum_{i \in M_j} m_i^p$ is the upper bound of O_j^p. The MILP formulation after linearization is:

$$\min \sum_{p \in P} \left(\sum_{i \in M} \sum_{j \in \mathcal{A}_i} f_{ij}^p d_{ij} + \sum_{j \in \mathcal{A}} \sum_{k \in R} g_{jk}^p d_{jk} \right)$$
$$\text{subject to:} \quad (1) - (7), (9), \text{and } (10).$$

Theorem 1. *Traffic-aware Partition and Aggregation problem is NP-hard.*

Proof: To prove NP-hardness of our network traffic optimization problem, we prove the NP-completeness

of its decision version by reducing the set cover problem to it in polynomial time.

The set cover problem: given a set U = fx1; x2; : : : ; xng, a collection of m subsets S = fS1; S2; : : : ; Smg, Sj _ U,1 _ j _ m and an integer K. The set cover problem seeks for a collection C such that jCj _ K and S i2C Si = U.

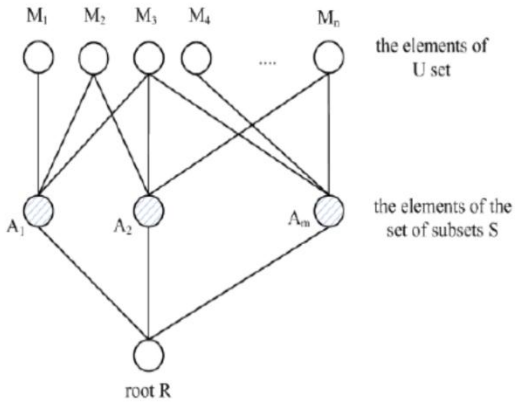

Fig. 4. A graph instance.

For each xi 2 U, we create a mapper Mi that generates only one key/value pair. All key/value pairs will be sent to a single reducer whose distance with each mapper is more than 2.

For each subset Sj , we create a potential aggregaor Aj with distance 1 to the reducer. If xi 2 Sj , we set the distance between Mi to Aj to 1.Otherwise, their distance is greater than 1.

The aggregation ratio is defined to be 1. The constructed instance of our problem can be illustrated using Fig. 4. Given K aggregators,

V. DISTRIBUTED ALGORITHM DESIGN

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., xpij and ypk) and constraints (e.g., (1) and (7)) in our formulation, leading to a large-scale optimization problem that is hardly handled by existing algorithms and solvers in practice.

In this section, 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 distributively solvable subproblems that are coordinated by a high-level master problem.

To achieve this objective, we first introduce an auxiliary variable zp j such that our problem can be equivalently formulated as:

$$\min \sum_{p\in P}\left(\sum_{i\in M}\sum_{j\in A_i} f_{ij}^p d_{ij} + \sum_{j\in A}\sum_{k\in R} g_{jk}^p d_{jk}\right)$$

$$\text{subject to:} \quad x_{ij}^p \le z_j^p, \forall j \in N, i \in M_j, p \in P, \quad (11)$$

$$z_j^p = z_j, \forall j \in N, p \in P, \quad (12)$$

$$(1)-(5),(7),(9),\text{and } (10).$$

The corresponding Lagrangian is as follows:

$$\begin{aligned} L(\nu) &= \sum_{p\in P} C^p + \sum_{j\in N}\sum_{p\in P} \nu_j^p (z_j - z_j^p) \\ &= \sum_{p\in P} C^p + \sum_{j\in N}\sum_{p\in P} \nu_j^p z_j - \sum_{j\in N}\sum_{p\in P} \nu_j^p z_j^p \\ &= \sum_{p\in P}\left(C^p - \sum_{j\in N} \nu_j^p z_j^p\right) + \sum_{j\in N}\sum_{p\in P} \nu_j^p z_j \quad (13) \end{aligned}$$

where ν_j^p are Lagrangian multipliers and C^p is given as

$$C^p = \sum_{i\in M}\sum_{j\in A_i} f_{ij}^p d_{ij} + \sum_{j\in A}\sum_{k\in R} g_{jk}^p d_{jk}.$$

5.1 Network Traffic Traces

In this section, we verify that our distributed algorithm can be applied in practice using real trace in a cluster consisting of 5 virtual machines with 1GB memory and 2GHz CPU.

Our network topology is based on three tier architectures: an access tier, an aggregation tier and a core tier(Fig. 6). The access tier is made up of cost effective Ethernet switches connecting rack VMs.

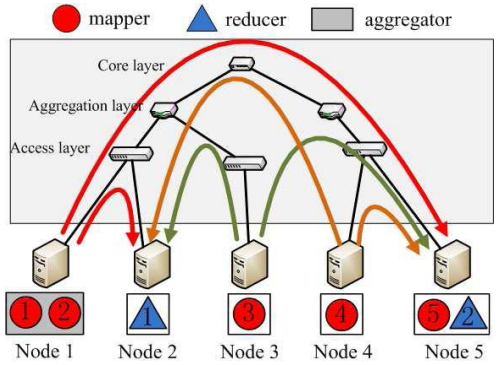

Fig. 6. A small example.

Our VMs are distributed in three different racks, and the map-reduce tasks are scheduled as in Fig. 6. For example, rack 1 consists of node 1 and 2; mapper 1 and 2 are scheduled on node 1 and reducer 1 is scheduled on node 2. The intermediate data forwarding between mappers and reducers should be transferred across the network.

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.

VI. 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.

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 roceedings 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] T. White, Hadoop: the definitive guide: the definitive guide. " O'Reilly Media, Inc.", 2009.