ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 6, Special Issue 12, July 2017

On Transit-Attentive Separation and Gathering in Mapreduce for Big data Application

Vinay Kumar K.S[1], Prof. Deeksha Hegde B[2]

PG Student, Department of Computer science & Engineering, S V I T, Bengaluru, India[1]

Assistant Professor, Department of Computer science & Engineering, S V I T, Bengaluru, India[2]

ABSTRACT: The MapReduce programming model simplifies large-scale data processing on commodity cluster by exploiting parallel map tasks and reduce tasks. Although many efforts have been made to improve the performance of MapReduce jobs, they ignore the network traffic generated in the shuffle phase, which plays a critical role in performance enhancement.Defaultly big data application will be using hash function which used to partition intermediate data among reduce tasks, but it is not traffic-efficient because network topology and data size is not considered. In this paper, the study is done to reduce network traffic cost for a MapReduce job by designing a intermediate data partition scheme. A decomposition-based distributed algorithm is proposed to deal with the large-scale optimization problem for big data application. Finally the simulation results are verified under offline casesfor network traffic reduction.

KEYWORDS: Distributed Algorithm, Hash Function, Mapreduce, Network Traffic

I. INTRODUCTION

MapReduce 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 have been adopted by leading companies, 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. 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 because it is unaware of the location on the map and reducer tasks. To tackle this problem incurred by the traffic-oblivious partition scheme, author take into account of both task locations and data size associated with each key. By assigning keys with larger data size to reduce tasks closer to map tasks, network traffic can be significantly reduced.

To further reduce network traffic within a MapReduce job, author 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.

In this paper, authors jointly consider data partition and aggregation for a MapReduce job with an objective that is to minimize the total network traffic. In particular, author proposes a distributed algorithm for big data applications by decomposing the original large-scale problem into several sub problems that can be solved in parallel.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 6, Special Issue 12, July 2017

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 cases.

1.1. LITERATURE SURVEY

Most of the work which is done to improve the performance by optimizing its data transferring. Balance[1] 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 [2] have presented Purlieus, a 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. 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. To overcome this shortcoming, Ibrahim [3] 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 [4] have designed an algorithm to schedule operations based on the key distribution of intermediate key/value pairs to improve the load balance. Lars [5] have proposed and evaluated two effective load balancing approaches to data skew handling for MapReduce-based entity resolution. Unfortunately, all above work focuses on load balance at reduce tasks, ignoring the network traffic during the shuffle phase.Different from existing work, author investigates network traffic reduction within MapReduce jobs by jointly exploiting traffic-aware intermediate data partition and data aggregation among multiple map tasks.

1.2 SYSTEM ARCHITECTURE

MapReduce is a programming model based on two primitives: map function and reduce function. A MapReduce 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 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 MapReduce job by jointly considering aggregator placement and intermediate data partition.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 6, Special Issue 12, July 2017

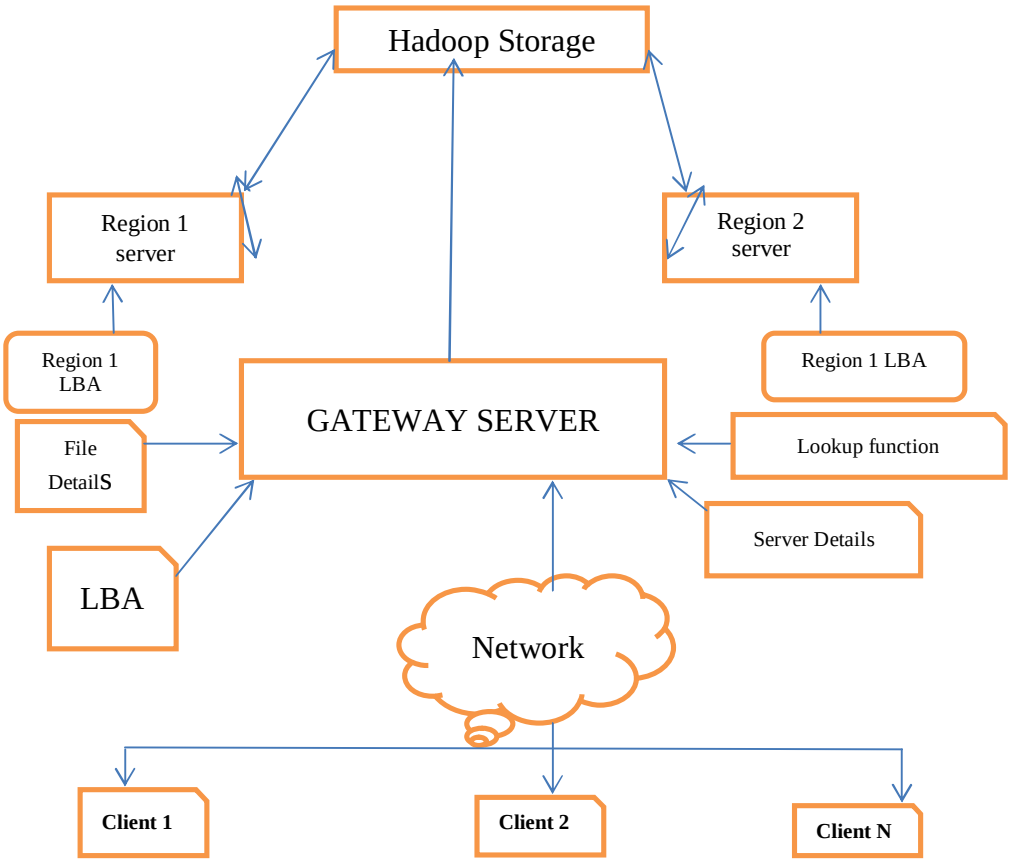

Fig: 1 System Architecture

As shown in above Fig: 1 Gateway server will get the entire client request message through network. The gateway server is having all the process of the mapreduce task. According to the details in the Logic block address and server details the partition data is routed to the servers in the different regions.

II. ALGORITHMS

Initially author will be using MD5 for generating hash details of the data and author propose a distributed algorithm by decomposing the original large-scale problem into several sub problems that can be solved in parallel.

2.1 Pseudo Code for MD5 Algorithm

Step 1: Get the Message
Step 2: Convert message into the bits.
Step 3: Append Padding Bits (Make the message bit length ought to be the correct numerous of 512 bitsand in addition 16 word Blocks).
Step 4: Divide add up to bits into 128 bits obstructs each
Step 5: Initialize MD Buffer. A four word cradle (A, B, C, D) is utilized to register the message process ,add up to 128 bits.
Step 6: Do AND, XOR, OR, NOT operations on A,B,C, D by giving three data sources furthermore, get one yield.
Step 7: Do the Step 6 until get the 128 bits hash (16 bytes).
Step 8: Stop

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 6, Special Issue 12, July 2017

2.2 Pseudo Code for Distributed Algorithm

Step 1: Initialize the time slot equals to 1 and Lagrangian multiplier of node belongs to set of aggregationlayer to arbitrary nonnegative value
Step 2: For timeslot t lesser than T the do the operation
Step 3: Administrational solve the sub problem of data partition and sub problem of aggregator placementon multiple machines in a parallel manner;
Step 4: Update the values of Lagrangian multiplier with gradient method and send the result to all Sub problem
Step 5: Set timeslot equals to timeslot plus 1
Step 6: End for loop

Author develops 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.

III. RESULT ANALYSIS

In this section, author conduct extensive simulations to evaluate the performance of our proposed distributed algorithm Distributed algorithm by comparing it to the following two schemes.

- **HNA:** Hash-based partition with No Aggregation. As the default method in Hadoop, it makes the traditional hash partitioning for the intermediate data, which are transferred to reducers without going through aggregators.
- **HRA:** Hash-based partition with Random Aggregation. It adopts a random aggregator placement algorithm over the traditional Hadoop. Through randomly placing aggregators in the shuffle phase, it aims to reducing the network traffic cost compared to the traditional method in Hadoop. To our best knowledge, this author is the first to propose the scheme that exploits both aggregator placement and traffic-aware partitioning. All simulation results are averaged over 30 random instances.

3.1 Simulation results of offline cases

First evaluate the performance gap between our proposed distributed algorithm and the optimal solution obtained by solving the MILP formulation. Due to the high computational complexity of the MILP formulation, author considers 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 is set to 0.5. The distance between any two machines is randomly chosen within [1-60].

As shown in Fig. 2, 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 × 10^4 while optimal solution is only 2.7×10^4, with 46% traffic reduction.

As shown in Fig. 2, 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 × 10^4 while optimal solution is only 2.7×10^4, with 46% traffic reduction.

Author then consider large-scale problem instances, and compare the performance of our distributed algorithm with the other two schemes. Author first describe a default simulation setting with a number of parameters, and then study the performance by changing one parameter while fixing others. Author considers a MapReduce job with 100 keys and other parameters are the same above.

As shown in Fig. 3, the network traffic cost shows as an increasing function of number of keys from 1 to 100 under all algorithms. In particular, when the number of keys is set to 100, the network traffic of the HNA algorithm is about 3.4×10^5, while the traffic cost of our algorithm is only 1.7 × 10^5, with a reduction of 50%. In contrast to HRA and HNA, the curve of DA increases slowly because most map outputs are aggregated and traffic-aware partition

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 6, Special Issue 12, July 2017

chooses closer reduce tasks for each key/value pair, which are beneficial to network traffic reduction in the shuffle phase.

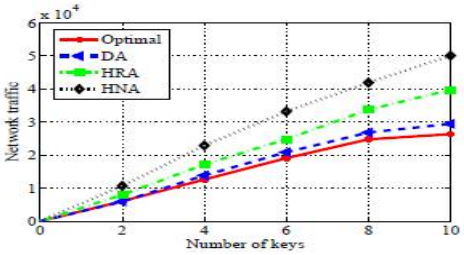

Fig: 2

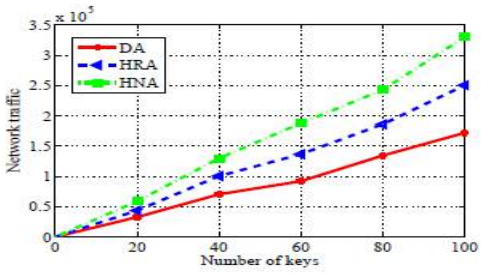

Fig: 3

IV. CONLUSION

In this paper, author study the joint optimization of intermediate data partition and aggregation in MapReduce to minimize network traffic cost for big data applications. Author proposes a three-layer model for this problem and formulates 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, author design a distributed algorithm to solve the problem on multiple machines. Finally, author conducts extensive simulations to evaluate our proposed algorithm under both offline cases.

REFERENCES

[1] A. Blanca and S. W. Shin, "Optimizing network usage in mapreduce scheduling."
[2] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality aware resource allocation for mapreduce in a cloud," in *Proceedings of 2011 International Conference for High Performance Computing,Networking, Storage and Analysis*. ACM, 2011, p. 58.
[3] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning formapreduce in the cloud," in *Cloud Computing Technology and Science (CloudCom),2010 IEEE Second International Conference on*. IEEE, 2010, pp. 17-24.
[4] L. Fan, B. Gao, X. Sun, F. Zhang, and Z. Liu, "Improving the load balance of mapreduce operations based on the key distribution of pairs," *arXiv preprint arXiv:1401.0355*, 2014.
[5] S.-C. Hsueh, M.-Y.Lin, and Y.-C. Chiu, "A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys," *Parallel and Distributed Computing 2014*, p. 3, 2014.
[6] T. White, *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.
[7] S. Chen and S. W. Schlosser, "Map-reduce meets wider varieties of applications," *Intel Research Pittsburgh, Tech. Rep. IRP-TR- 08-05*, 2008.large scale machine learning," *arXivpreprint*