Volume 7, Issue 3, March 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Effective Analysis of Traffic Optimality Using Bigdata

Assistant Prof. S. Aiswarya, K. Yamuna, D. Sowndariya, G. Mahalakshmi
Department of CSE, Velammal Institute of Technology,
Chennai, Tamilnadu, India

Abstract—The paper proposes the MapReduce programming model which simplifies large-scale data processing on commodity cluster by exploiting parallel map tasks and reduce tasks and also reduces the network traffic. Most of the researches have been done to improve mapreduce jobs, but they have not concentrated on network traffic which deals with the performance and efficiency. 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 main aim of this paper is to focus on the reduction of network traffic by using the load balancing algorithm.

Keywords: Mapreduce, Network Traffic, Load Balancing, HashFunction, Cluster.

I. INTRODUCTION

Mapreduce is a programming model and an associated implementation for processing and generating big data sets with parallel, distributed algorithm on a cluster. . MapReduce 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, bioinformatics and cyber-security. The MapReduce algorithm contains two important tasks, namely map and reduce. Map task a set data and convert into another set of data, where individual elements are broken down into tuples(key/value pairs).secondly, reduce task which takes the output from a map as the input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job. 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.

Hadoop Distributed File System (HDFS) is developed to store a huge volume of data. Files are divided into blocks and the replicated blocks are then stored on many Data Nodes may lead to load imbalance among the Data Nodes. Moreover, the built-in-load-balancing algorithm may reduce the performance and consume lots of network resources. Therefore in this paper we consider all the situation that may influence the load-balancing state and propose a new load-balancing algorithm in the proposed algorithm a new role named balance node is introduced to help in matching heavy-loaded and light-loaded Data Nodes, so those light-loaded can share part of the load from heavy-loaded ones.

II. LITERATURE SURVEY

Shanjiang Tang, Bu-Sung Lee, and Bingsheng " Dynamic Job Ordering and Slot Configurations for MapReduce Workloads" , The author proposed two classes of algorithms to minimize the makes pan and the total completion time for an offline Map Reduce workload his first class of algorithms focuses on the job ordering optimization for a Map Reduce workload under a given map/reduce slot configuration and second class of algorithms considers to optimization for map/reduce slot configuration for a Map Reduce workload resulting execution cost will be low.

J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade,"Deadline-based mapreduce workload management", The author introduced a scheduling technique and implemented a prototype called Adaptive Scheduler that can adaptively manage the workload performance with the awareness of hardware heterogeneity, distributed storage to meet r's deadline requirement. The scheduler's ability to favor data-locality in the scheduling algorithm; and relative performance characterization for those applications that can benefit from executing on specialized processors, environments and commonly used data sets into flexibly deployable virtual

J. Leverich and C. Kozyrakis. On the energy (in) efficiency of Hadoop clusters. Author proposed a method for cluster energy management for Map Reduce jobs by selectively powering down nodes with low utilization. Their method uses a cover set strategy that exploits the replication to keep at least one copy of a data-block. As a result, in low utilization periods some of the nodes that are not in the cover set can be powered down.

Andréa Matsunaga, Maurício Tsugawa and José Fortes. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. IEEE Fourth International Conference on eScience, 2008. This paper proposes and evaluate an approach to the parallelization, deployment and management applications that integrates several emerging technologies for distributing computing. The proposed approach uses the MapReduce paradigm to parallelize tools and manage their execution, machine virtualization to encapsulate their execution machines.

III. PROPOSED SYSTEM

A parallel program is a set of processes which aims to perform one or more tasks of the program. Particularly, the task is the smallest division of a parallel program. Thus, the development of a parallel program requires at first, decomposing the overall program into a collection of tasks and assigning each task to a process. This step is called the partitioning.Its optimization is based on the balancing of workloads between different processes and reducing inter-processes communication. As the number of processes generated by a program is often different from the number of processor in charge of processing the request, there is a risk of overloading or under loading of a processor by processes. Consequently, the optimization of Cloud resources cannot be done without an efficient load balancing strategy. In summary, the major goal of load balancing is to develop performed algorithms able to achieve both efficiently the partitioning and mapping steps. In other words, the load balancing algorithms should:

- Enhance the performance significantly;
- Ensure a partial or complete backup plan in case of system failed ;
- Keep up the system stability;
- Put up future update of the system.

According to their working strategy, load balancing algorithms are classified on two categories, static or dynamic load balancing. There are two main performance indicators of load balancing: stability and efficiency. The stability measures essentially, the capability of an algorithm to force any initial workload distribution into equilibrium state. The efficiency indicates the time necessary to either reduce the variance or arrive at the equilibrium state. The response time, which indicated the time necessary from registering the input to providing a response to it, and throughput, which represents the number of tasks achieved in a given time, represents the most significant key performance indicators in a distributed system.

A. System Architecture

A classic Hadoop cluster includes a single master node and multiple slave nodes. The master node runs the Job Tracker routine which is responsible for scheduling jobs and coordinating the execution of tasks of each job. Each slave node runs the Task Tracker daemon for hosting the execution of Map Reduce jobs. The concept of "slot" is used to indicate the capacity of accommodating tasks on each node. In a Hadoop system, a slot is assigned as a map slot or a reduce slot serving map tasks or reduce tasks, respectively. At any given time, only one task can be running per slot. Scheduler will dispatch the jobs to task tracker for map reducing the number of available slots per node indeed provides the maximum degree of parallelization in Hadoop

- Input Phase − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- Map − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- Intermediate Keys − The key-value pairs generated by the mapper are known as intermediate keys.
- Combiner − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- Reducer − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
- Output Phase − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

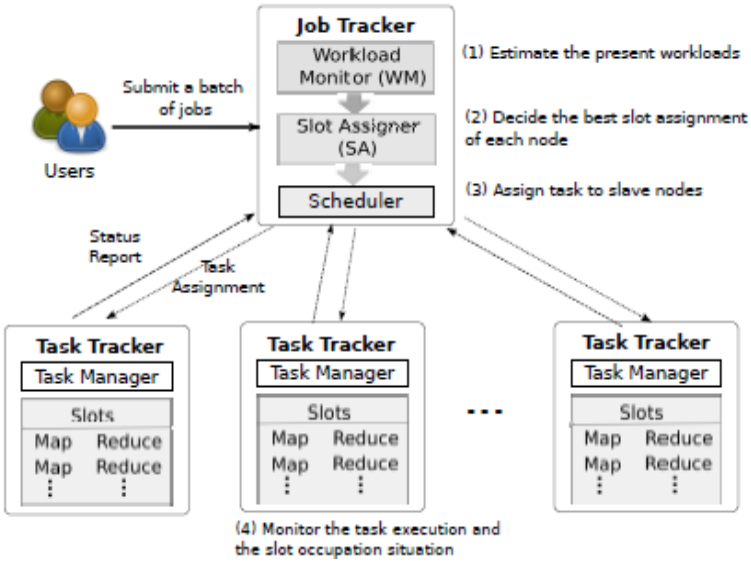

Fig. 1 Overview of System Design

B. Load Balancing Algorithm

To conduct our study we apply the Taguchi experience plans. This methodology inspired from the industrial sector offers the advantage organizing and making easier the study of a complex system such Cloud computing. The Taguchi's concept utilizes optimal number planned experiments in order to examine the settings of influential process control parameters. For each set factors, the appropriate experiments are achieved and the performance measurements are recorded. Afterward, this approach use figures and tables to analyze the obtained measurements and to predict the best combination to optimize the effect 01' factors. Lastly, a few additional experiments are achieved in order to validate the combination predicted.

An efficient load balancing policy should allocate users' request to closest data center and allows optimization of request size. As depicted the proposed load balancing architecture is organized on three layers. The first layer includes the main controller and the second controller. The main controller is the first element to receive the users' tasks. The main controller classifies users' tasks by priority, region and technical characteristics and then dispatches users' tasks on regional load balancers as mentioned hereafter (see Algorithm land . In addition, the main controller communicates regularly with the second controller in order to update information about the state of the system. The second controller has to communicate frequently with the regional load balancers in order to refresh the information about system state. Through different agents, the second balancer updates the load level of the different nodes according to a frequency T. Then it informs the main controller about the modified state of a node. This configuration aims to lighten the main controller load.

The second layer embraces regional load balancers. Those components receive various tasks sent by the main controller and schedule tasks received through the different nodes of the Cloud following an ant colony optimization algorithm. The third layer is composed by two levels of nodes. The first level receives the tasks sent by regional load balancer. Then, they divided the tasks received on elementary subtasks in order to minimize the size of users' requests. This approach is driven by the conclusion of the Taguchi analysis which point out that the "Request size" factor has a high impact on the performance response time.

C. Load Balancing Algorithm Architecture

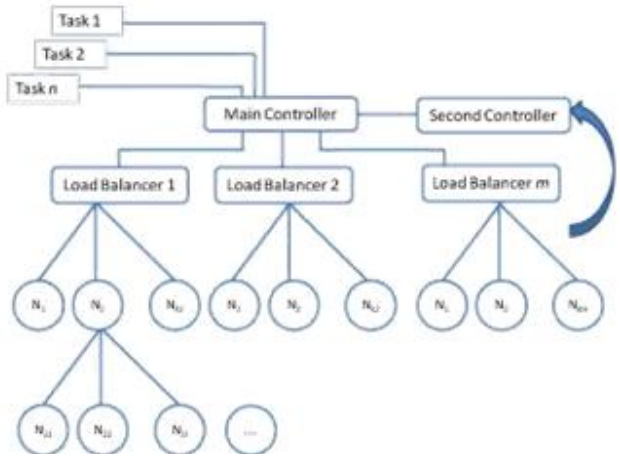

Fig. 2 Load Balancing Architecture

D. Algorithm I: Task scheduling of the Main Controller

```
Begin
While Task do
CalculateTechnicalratio (Task);
While (value of Timer) < T do
SearchcloserLoadBalancer (Task);
if RegionalBalancerState == Idle then
Send Task to RegionalBalancer;
else
Search tür next RegionalBalancer;
Ca1culateWaitingTime(Task); Ca1culateExpectingCompletionTime (Task);
if Waiting Time> Expecting Completion Time then
Send Task to NextRegionalBalancer;
else
Put Task on the RegionalBalancer queue
end if
end if
end while
end while
end
```


IV. RESULT AND ANALYSIS

The final processing is the analysis of job by the mapreduce framework using the proposed load balancing algorithm which will Reduce processing time and increase throughput by decreasing the data traffic between nodes and balancing the loads of the cluster. It is used to improve the performance of Map Reduce applications in big data. It serves as an heavy traffic optimal solution. Hence decomposing the original large-scale problem into several sub problems that can be solved in parallel. Thus high throughput, fast response, load balance, low cost and low price with help of load balancing algorithm .

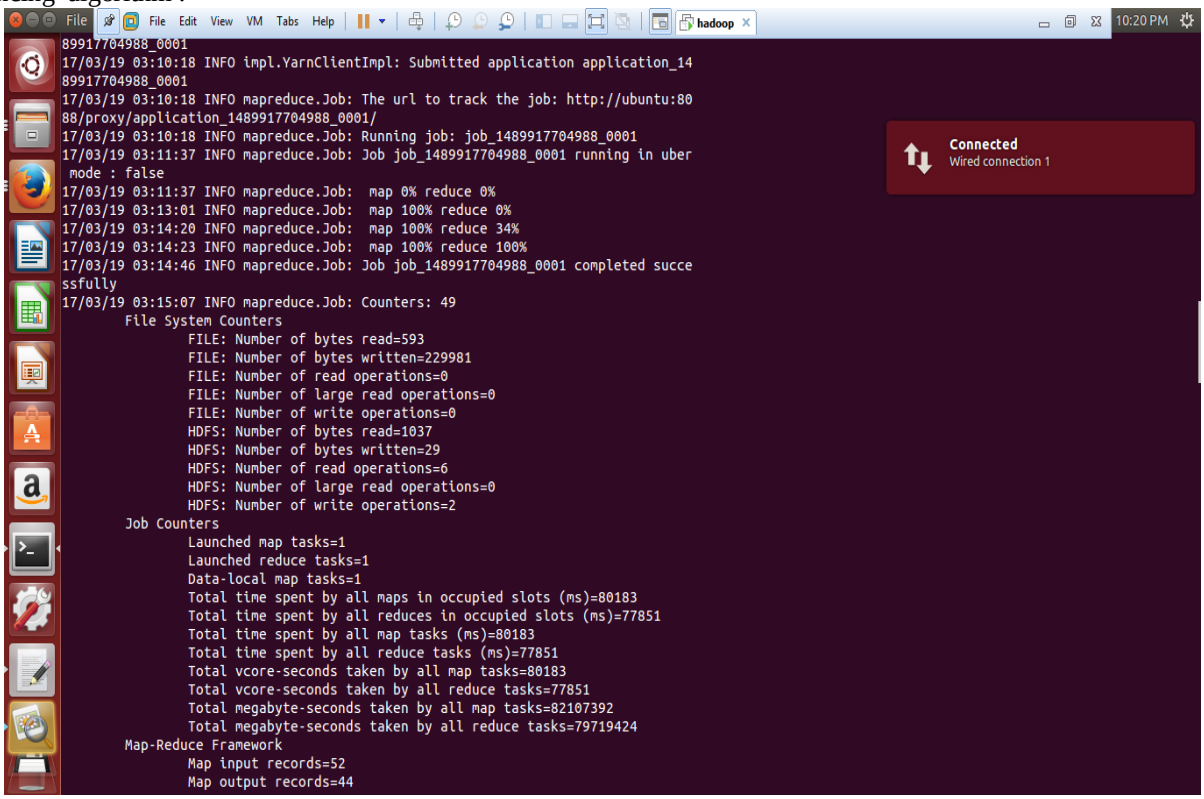

Fig. 3 Output of Mapreduce Framework

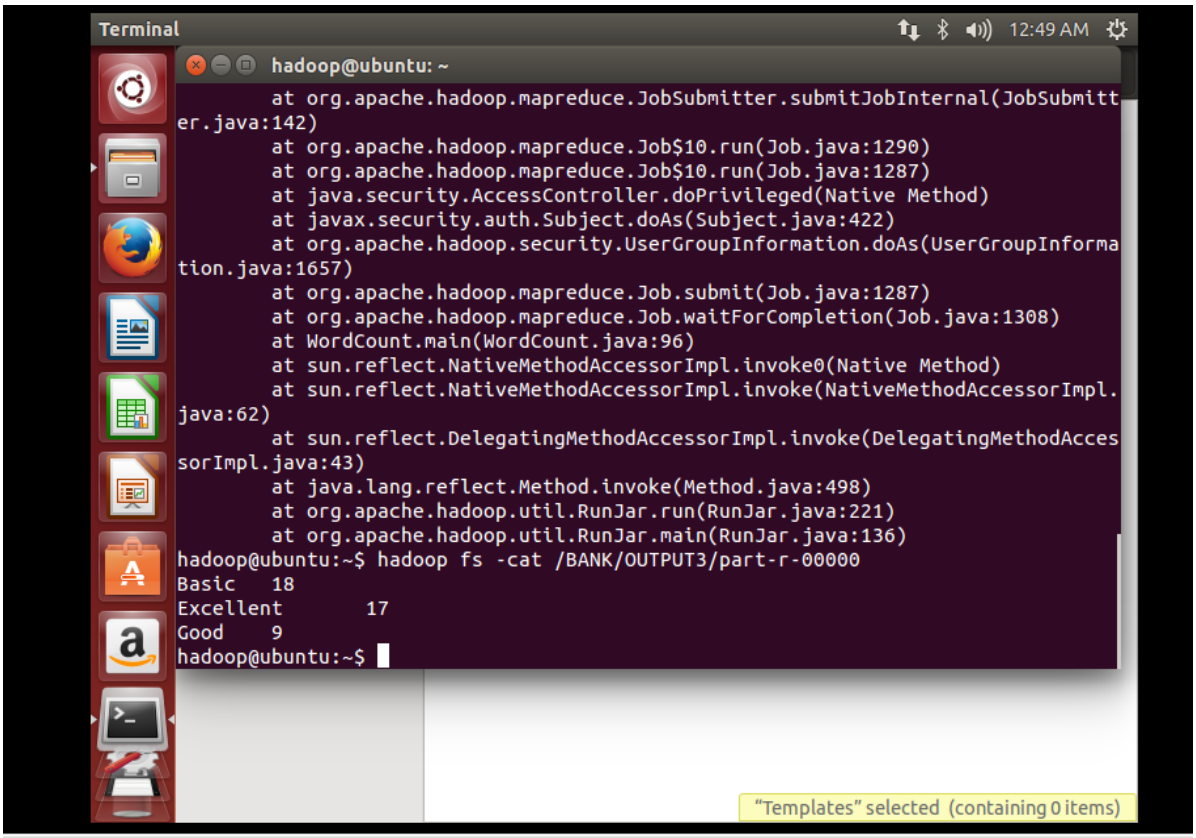

Fig. 4 Final View of the Output

V. CONCLUSION

Given the increasing number of users in the Bigdata and the geographic scope of data centers, it becomes more and more difficult to maintain a satisfactory level of availability, security and responsiveness through conventional models of load balancing. Through this paper we propose an improved architecture for load balancing by offering to share the functions of the main controller representing the entry point into two parts. Firstly, the main controller will keep the function of partitioning users tasks through different regional load balancers. Secondly, the auxiliary controller will insure the updating state of the system through various agents. Otherwise, we choose to combine different algorithms in order to optirnize the partitioning and mapping steps. Briefly, the proposed architecture of the Cloud is organized on three levels and each level has a specific role and specific algorithm. The main controller applies an algorithm based on the geographic location of both user and data center in order to choose the cIosest load balancer. The regional load balancer applies a second policy based on ant colony optirnization algorithm to allocate the appropriate nodes to users' tasks. Through this approach, we aim to optirnize the whole response times of services in the Bigdata

REFERENCES

[1] On Traffic-Aware Partition and Aggregation in MapReduce for Big Data ApplicationsHuan Ke, Student Member, IEEE, Peng Li, Member, IEEE, Song Guo, Senior Member, IEEE, and Minyi Guo, Senior Member, IEEE.

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

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

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in Proc. ACM Symp. Operating Systems Principles (SOSP), Bolton Landing, NY, 2003, pp. 29–43.

[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in IEEE Symp. Mass Storage Systems and Technologies (MSST), Incline Villiage, NV, May 2010, pp. 1–10

[6] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in Proc. European Conf. Computer Systems (EuroSys), Paris, France, 2010, pp. 265–278.

[7] C. Abad, Y. Lu, and R. Campbell, "DARE: Adaptive data replication for efficient cluster scheduling," in IEEE Int. Conf. Cluster Computing (CLUSTER), Austin, TX, 2011, pp. 159–168.

[8] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in Proc. IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing (CCGRID), Melbourne, Australia, 2010, pp. 94–103.

[9] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In EuroSys, 2012.

[10] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou. Comet: batched stream processing for data intensive distributed computing. In SoCC'10, 2010.

[11] H. Herodotou, F. Dong, and S. Babu. MapReduce programming and cost-based optimization? Crossing this chasm with Starfish. PVLDB, 4(12), 2011.

[12] Q. Ke, V. Prabhakaran, Y. Xie, Y. Yu, J. Wu, and J. Yang. Optimizing data partitioning for data-parallel computing. In HotOS XIII, 2011.

[13] D. G. Murray, M. Isard, and Y. Yu. Steno: automatic optimization of declarative queries. In PLDI, 2011.

[14] J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou. Optimizing data shuffling in data-parallel computation by understanding user-defined functions. MSR-TR-2012-28, 2012.

[15] J. Zhou, P.-˚A. Larson, and R. Chaiken. Incorporating partitioning and parallel plans into the SCOPE optimizer. In ICDE, 2010.

