International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

AN EFFICIENT TRAFFIC-AWARE SEPARATION AND AGGRIGATION USING MAPREDUCE FOR BIG DATA APPLICATIONS

M.Gayathri[1], M.Nishitha[2], E.Srinivas Reddy[3] ,V.Mounika[4]
[1]B.Tech C.S.E TKREC Hyderabad Email: gayathri.manusani@gmail.com
[2]B.Tech C.S.E TKREC Hyderabad Email: maddatinishitha0207@gmail.com
3B.Tech C.S.E TKREC Hyderabad Email: srinivas3827@gmail.com
[4]Assistent Professor ,CSE Dept, TKREC, Hyderabad, TS-India, Email: mounikatkrcse@gmail.com

1.ABSTRACT

The MapReduce programming model simplifies the processing large datasets. Mapreduce is typically used to do distributed computing on cluster of computers, 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. Traditionally, a hash function is used to separate intermediate data among reduce tasks however, it is not traffic-efficient. In this paper, we study to reduce network traffic cost for a MapReduce job by designing a novel intermediate data separation scheme. A decomposition-based distributed algorithm is proposed to deal with the large-scale optimization problem for big data application and an online algorithm is also designed to adjust data separation and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost.

Keywords: MapReduce, Separation, Aggregation.

2.INTRODUCTION

Bigdata is a term used to describe a collection of data sets. The size and complexity of big data makes it difficult to use traditional database management and data processing tools. Data is being created in much shorter cycles from hours to milliseconds. There is also a trend underway to create larger databases by combining smaller data sets so that data correlations can be discovered.

Big data has become the new frontier of information management given the amount of data today's systems are generating and consuming. It has driven the need for technological infrastructure and tools that can capture, store, analyse and visualize vast amounts of disparate structured and unstructured data. These data are being generated at increasing volumes from data intensive technologies including, but not limited to, the use of the Internet for activities such as accesses to information, social networking, mobile computing and commerce. Corporations and governments have begun to recognize that there are unexploited opportunities to improve their enterprises that can be discovered from these data.

2.1 Objective

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, such as machine learning [6], [7], [8], bioinformatics [9], [10], [11], and cyber-security [12], [13]. 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. For example, with tens of thousands of machines, data shuffling accounts for 58.6 percent of the cross-pod traffic and amounts to over 200 petabytes in total in the analysis of SCOPE jobs [14]. For shuffle-heavy MapReduce tasks, the high traffic could incur considerable performance overhead up to 30-40 percent as shown in [15].By default, intermediate data are shuffled according to a hash function [16] 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, 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. Although a similar function, called combiner [17], 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 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, the network traffic will be reduced.

3.Proposed work

The system that existing consist of intermediate data are shuffled according to a

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

hash function in Hadoop, which would lead to large network traffic because it ignores network topology and data size associated with each key.

Due to disadvantages in present existing system, a system is proposed to reduce the network traffic.

The system that is proposed helps for an efficient network traffic, 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 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.System architecture

Hadoop is a java based programming model which living big data. Hadoop having map-reduce and hadoop distributed file system (HDFS).

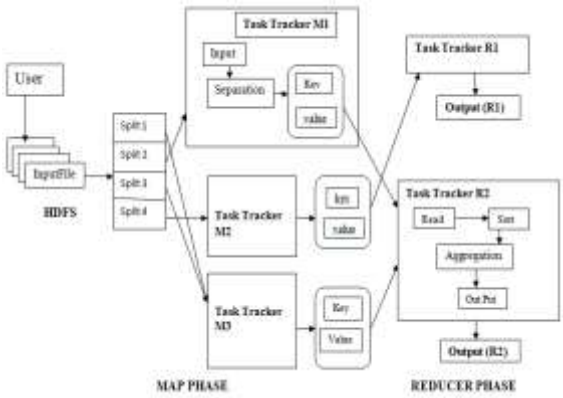

Fig System Architecture

The user stores the input files in HDFS. The data produced by the map phase are ordered, separated and transferred to the proper machines executing the reduce phase is represented in the Fig. The resulting network traffic pattern from all map tasks to all reduce tasks can cause a great bulk of network data flow enforce a difficult sate on the efficiency of data analytic applications. This work proposed an efficient Traffic Aware separation and Aggregation to minimize network sequence of operation cost for Big Data applications using Map-Reduce.

5. Algorithm

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 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 $v_j^p (j \in A, p \in P)$ 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 v_j^p with the gradient method (15), and send the results to all subproblems;
5: set $t = t + 1$;
6: end for

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

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

Online Algorithm:

Until now, we take the data size and data aggregation ratio 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**

In this section, we divide the execution of a MapReduce job into several time slots with a length of several minutes or an hour. The parameters collected at time slot t with no assumption about their distributions. As the job is running, an existing data partition and aggregation scheme may not be optimal. To reduce traffic cost, we may need to migrate an aggregator from machine j to j0 with a migration cost. Meanwhile, the key assignment among reducers is adjusted. When we let reducer process the data with key p instead of reducer k that is currently in charge of this key, we use to denote the cost migrating all intermediate data received by reducers so far.

6.Result analysis

The role of user is 30% i.e., to login into the system, sending an input data or file to the HDFS then performs the mapreduce program.

The role of mapreduce is 50% i.e., map process the input data or file into small chunks and generate key/values, reducer processes the input data which is obtained from mapper, then produces a new set of output which is stored in HDFS.

The role of HDFS is 20% i.e., it mainly focus on storage. It stores the input data and also processed data which is obtained after mapreduce task.

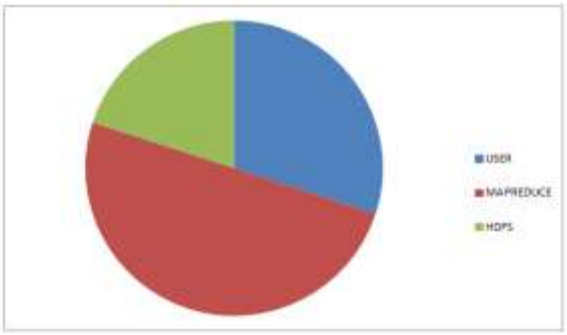

Fig: Result Analysis

7.Conclusion

In this paper, we study the joint optimization of intermediate data separation and aggregation in MapReduce to minimize network traffic cost for big data applications. 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.

8. Future Enhancement

Till now we focused on processing the bigdata using mapreduce task considering partition and aggregation and by sorting the result in HDFS. Further, on traffic aware may be done by using different hadoop tools like hive, pig and also to work on complex data partitioning where intelligent methods have to be employed. This include analyzing computation task, skew record etc. So that optimization of data can be done in mapreduce.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. 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 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] Y. Wang, W. Wang, C. Ma, and D. Meng, "Zput: A speedy data uploading approach for the hadoop distributed file system," in Proc. IEEE Int. Conf. Cluster Comput., 2013, pp. 1–5.

[5] T. White, Hadoop: The Definitive Guide: The Definitive Guide. Sebastopol, CA, USA: O'Reilly Media, Inc, 2009.

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

[7] H. Lv and H. Tang, "Machine learning methods and their application research," IEEE Int. Symp. Intel. Info. Process. Trusted Comput. (IPTC), pp. 108–110, Oct. 2011.

[8] S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber, "Presto: Distributed machine learning and graph processing with sparse matrices," in Proc.

8th ACM Eur. Conf. Comput. Syst., 2013, pp. 197–210.

[9] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in Proc. IEEE 4th Int. Conf. eScience, 2008, pp. 222–229.

[10] 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 Proc. 4th Int. Workshop Data Intensive Comput. Clouds, 2013, pp. 1–5.

[11] R. Liao, Y. Zhang, J. Guan, and S. Zhou, "Cloudnmf: A mapreduce implementation of nonnegative matrix factorization for largescale biological datasets," Genomics, Proteomics Bioinformat., vol. 12, no. 1, pp. 48–51, 2014.

[12] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing map-reduce to high end computing," in Proc. 3rd Petascale Data Storage Workshop, 2008, pp. 1–6.

[13] W. Yu, G. Xu, Z. Chen, and P. Moulema, "A cloud computing based architecture for cyber security situation awareness," in Proc. IEEE Conf. Commun. Netw. Security, 2013, pp. 488–492.

[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," in Proc. 9th USENIX Conf. Netw. Syst. Des. Implemen. (NSDI '12), Berkeley, CA, USA: USENIX Association, 2012, pp. 295–308.

[15] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Mapreduce with communication overlap," J. Parallel Distrib. Comput., vol. 73, pp. 608–620, 2013.

[16] H.-C. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Mapreduce-merge: Simplified relational data processing on large clusters," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2007, pp. 1029–1040.

[17] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in mapreduce," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, pp. 1115–1118.

[18] S. Narayan, S. Bailey, and A. Daga, "Hadoop acceleration in an openflow-based cluster," IEEE SC Companion: High Performance Comput., Netw., Storage Analysis (SCC), pp. 535–538, Nov. 2012.

[19] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Localityaware resource allocation for mapreduce in a cloud," in Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal., 2011, p. 58.

[20] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci., 2010, pp. 17–24.

[21] W. Yan, Y. Xue, and B. Malin, "Scalable and robust key group size estimation for reducer load balancing in MapReduce," IEEE Int. Conf. Big Data, pp. 156–162, Oct. 2013.

[22] S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu, "A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys," in Proc. 12th Australasian Symp. Parallel Distrib. Comput., 2014, pp. 3–9.

[23] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in Proc. 7th USENIX Conf. Netw. Syst. Design Implementation, 2010, vol. 10, no. 4, p. 20.

[24] J. Lin and C. Dyer, "Data-intensive text processing with mapreduce," Synthesis Lectures Human Language Technol., vol. 3, no. 1, pp. 1–177, 2010.

[25] P. Costa, A. Donnelly, A. I. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in Proc. 7th USENIX Conf. Netw. Syst. Design Implementation, 2012, vol. 12, p. 3.