Scope International Journal of Science, Humanities, Management and Technology. ISSN : 2455-068X
Vol.4 Issue 3 (2018) 141 - 144. Submitted 15/08/2018. Published 07/09/2018

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

P.Ashok Kumar.,MCA.,MPhil,Assistant Professor Department Of Computer Applications Avs college of arts and science, Salem

S. Ragunathan.,MCA.,MPhil.,Head of the department, Department Of Computer Science, Avs college of arts and science, Salem

ABSTRACT

MapReduce 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 MapReduce 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.

1. 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, such as Yahoo!, Google and Facebook, for various big data applications, such as machine learning bioinformatics and cyber security. 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.

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 MapReduce 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.

2. RELATED WORK

Most existing work focuses on MapReduce performance improvement by optimizing its data transmission. Blancaet 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 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.

3. LITERATURE SURVEY

J. Dean And S. Ghemawat, "Mapreduce: Simplified Data Processing On Large Clusters,": MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality,": Scheduling map tasks to improve data locality is crucial to the performance of MapReduce. Many works have been devoted to increasing data locality for better efficiency. However, to the best of our knowledge, fundamental limits of MapReduce computing clusters with data locality, including the capacity region and theoretical bounds on the delay performance, have not been studied. In this paper, we address these problems from a stochastic network perspective. Our focus is to strike the right balance between data-locality and load-balancing to simultaneously maximize throughput and minimize delay.

F. Chen, M. Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems,": MapReduce has emerged as an important paradigm for processing data in large data centers. MapReduce is a three phase algorithm comprising of Map, Shuffle and Reduce phases. Due to its widespread deployment, there have been several recent papers outlining practical schemes to improve the performance of MapReduce systems. All these efforts focus on one of the three phases to

obtain performance improvement. In this paper, we consider the problem of jointly scheduling all three phases of the MapReduce process with a view of understanding the theoretical complexity of the joint scheduling and working towards practical heuristics for scheduling the tasks. We give guaranteed approximation algorithms and outline several heuristics to solve the joint scheduling problem.

4. PROPOSED SYSTEM

In this paper, we jointly consider data partition and aggregation for a MapReduce 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.

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 in this locality-awareness reduces network traffic in the shuffle phase generated in the cloud data center.

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 MapReduce-like system to decrease the traffic by pushing aggregation from the edge into the network.

5. RESULT ANALYSIS

In this paper, we studied the map task scheduling problem in MapReduce with data locality. We derived an outer bound on the capacity region and a lower bound on the expected number of backlogged tasks in steady state. Then, we developed a scheduling algorithm constituted by the Join the Shortest Queue policy and the Max Weight policy. This algorithm achieves the full capacity region and minimizes the expected number of backlogged tasks in the considered heavy-traffic regime. Therefore, the proposed algorithm is both throughput-optimal and heavy-traffic optimal. The proof technique was novel since non preemptive task execution and random service times were involved. Simulation results were given to evaluate the throughput performance and the delay performance for a large range of total arrival rates.

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. 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.

REFERENCE

[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 Proceedings 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] 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 mapreduce 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.

