Big Data Hadoop: Aggregation Techniques

Vidya Pol

Department of Computer Engineering, KJ College of Engineering Management & Research, Savitribai Phule University, Pune, India

Abstract: The term ‘Big Data’, refers to data sets whose size (volume), complexity (variability), and rate of growth (velocity) make them difficult to capture, manage, process or analyzed. To analyze this enormous amount of data Hadoop can be used. However, processing is often time-consuming. One way to decrease response time is to executing the job partially, where an approximate, early result becomes available to the user, before completion of job. The implementation of the technique will be on top of Hadoop which will help to sample HDFS blocks uniformly. We will evaluate this technique using real-world datasets and applications and we will try to demonstrate the system’s performance in terms of accuracy and time. The objective of the proposed technique is to significantly improve the performance of Hadoop MapReduce for efficient Big Data processing.

Keywords: privacy preservation, security, e-healthcare systems, data mining, image feature extraction.

1. Introduction

Big data is a term that refers to data sets or combinations of data sets whose size (volume), complexity (variability), and rate of growth (velocity) make them difficult to be captured, managed, processed or analyzed by conventional technologies and tools, such as relational databases. Hadoop MapReduce programming model is being used for processing Big Data, which consists of data processing functions: Map and Reduce. Parallel Map tasks are run on input data which is partitioned into fixed sized blocks and produce intermediate output as a collection of <key, value> pairs. These pairs are shuffled across different reduce tasks based on <key, value> pairs. Each Reduce task accepts only one key at a time and process data for that key and outputs the results as <key, value> pairs. The Hadoop MapReduce architecture consists of one JobTracker (Master) and many TaskTrackers (Workers). The MapReduce Online is a modified version of Hadoop MapReduce which supports Online Aggregation and reduces response time. Traditional Map Reduce implementations materialize the intermediate results of mapper and do not allow pipelining between the map and the reduce phases. This approach has the advantage of simple recovery in the case of failures, however, reducers cannot start executing tasks before all mapper have finished. This limitation lowers resource utilization and leads to inefficient execution for many applications. The main motivation of Map Reduce Online is to overcome these problems, by allowing pipelining between operators, while preserving fault tolerance guarantees. Redis is an open-source, networked, in-memory, key-value data store with optional durability. It is written in ANSI C.

The name Redis means REMote DIctionary Server. In its outer layer, the Redis data model is a dictionary which maps keys to values. One of the main differences between Redis and other structured storage systems is that Redis supports not only strings, but also abstract data types like lists of strings, sets of strings (collections of non-repeating unordered elements), sorted sets of strings (collections of non-repeating elements ordered by a floating-point number called score), hashes where keys and values are strings. The type of a value determines what operations (called commands) are available for the value itself. Redis supports high-level, atomic, server-side operations like intersection, union, and difference between sets and sorting of lists, sets and sorted sets. The main goal of the project work is to implement Online MapReduce and Redis on the top of the Hadoop, which will improve the performance of Hadoop for efficient Big Data processing.

2. 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 MapReduce jobs in the cloud by locating intermediatedata 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 et al. 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 et al. have designed an algorithm to schedule operations based on the key distribution of intermediate key/value pairs to improve the load balance. Larson et al. 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. In addition to data partition, many efforts have been made on local aggregation, in-mapper combining and in-network aggregation to reduce network traffic within MapReduce jobs. Conde et al. have introduced a combiner function that reduces the amount of data to be shuffled and merged to reduce tasks.
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. Costea et al. have proposed a MapReduce-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. Different from existing work, we investigate network traffic reduction within MapReduce jobs by jointly exploiting traffic-aware intermediate data partition and data aggregation among multiple map tasks.

3. Proposed Work

The MapReduce Online is a modified version of Hadoop Map Reduce, a popular open-source implementation of the Map Reduce programming model. It supports online aggregation and stream processing, while also improving utilization and reducing response time. Traditional MapReduce implementations materialize the intermediate results of mappers and do not allow pipelining between the map and reduce phases. This approach has the advantage of simple recovery in the case of failures, however, reducers cannot start executing tasks before all mappers have finished. This limitation lowers resource utilization and leads to inefficient execution for many applications. The main motivation of MapReduce Online is to overcome these problems by allowing pipelining between operators, while preserving fault-tolerance guarantees. Although MapReduce was originally designed as a batch-oriented system, it is often used for interactive data analysis: a user submits a job to extract information from a data set, and then waits to view the results before proceeding with the next step in the data analysis process. This trend has accelerated with the development of high-level query languages that are executed as MapReduce jobs, such as Hive, Pig. Traditional MapReduce implementations provide a poor interface for interactive data analysis, because they do not emit any output until the job has been executed. In many cases, an interactive user would prefer a quick and dirty approximation over a correct answer that takes much longer to compute. In the database literature, online aggregation has been proposed to address this problem, but the batch-oriented nature of traditional MapReduce implementations makes these techniques difficult to apply.

4. Simulation Results

We 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, we consider 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. 1, 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 $5.0 \times 10^5$ while optimal solution is only $2.7 \times 10^5$, with 46% traffic reduction. We then consider large-scale problem instances, and compare the performance of our distributed algorithm with the other two schemes. We first describe a default simulation setting with a number of parameters, and then study the performance by changing one parameter while fixing others.

![Figure 1: Network traffic cost versus number of keys from 1 to 10](image1)

![Figure 2: Network traffic cost versus different number of keys](image2)

We consider a MapReduce job with 100 keys and other parameters are the same above. As shown in Fig. 3, the network traffic cost shows an increasing function of the 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 $3.4 \times 10^5$, while the traffic cost of our algorithm is only $2.7 \times 10^5$, 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 chooses closer reduce tasks for each key/value pair, which are beneficial to network traffic reduction in the shuffle phase. We then study the performance of three algorithms under different values of $\alpha$ in Fig. 4 by changing its value from 0.2 to 1.0. A small value of $\alpha$ indicates lower aggregation efficiency for the intermediate data. We observe that network traffic increases as the growth of both DA and HRA. In particular, when $\alpha$ is 0.2, DA achieves the lowest traffic cost of $1.1 \times 10^5$.
other hand, network traffic of HNA keeps stable because it does not conduct data aggregation. The affect of available aggregator number on network traffic is investigated in Fig. 5. We change aggregator number from 0 to 6, and observe that DA always outperforms other two algorithms, and network traffic decreases under both HRA and DA. Especially, when the number of aggregator is 6, network traffic of the HRA algorithm is $2.2 \times 10^5$, while of DA’s cost is only $1.5 \times 10^5$, with 26.7% improvement. That is because aggregators are beneficial to intermediate data reduction in the shuffle process. Similar with Fig. 4, the performance of HNA shows as a horizontal line because it is not affected by available aggregator number. We study the influence of different number of map tasks by increasing the mapper number from 0 to 60. As shown in Fig. 5, we observe that DA always achieves the lowest traffic cost as we expected because it jointly optimizes data partition and aggregation. Moreover, as the mapper number increases, network traffic of all algorithms increases. We shows the network traffic cost under different number of reduce tasks in Fig. 6. The number of reducers is changed from 1 to 6. We observe that the highest network traffic is achieved when there is only one reduce task under all algorithms. That is because all key/value pairs may be delivered to the only reducer that locates far away, leading to a large amount of network traffic due to the many-to-one communication pattern. As the number of reduce tasks increases, the network traffic decreases because more reduce tasks share the load of intermediate data. Especially, DA assigns key/value pairs to the closest reduce task, leading to least network traffic.

5. Conclusion

The proposed system is based on implementation of Online Aggregation of MapReduce in Hadoop for ancient big data processing. Traditional Map Reduce implementations materialize the intermediate results of mappers and do not allow pipelining between the map and the reduce phases. This approach has the advantage of simple recovery in the case of failures, however, reducers cannot start executing tasks before all mappers have finished. As the Map Reduce Online is a modeled version of Hadoop Map Reduce, it supports Online Aggregation and stream processing, while also improving utilization and reducing response time. The limitation of traditional mapreduce’s lowers resource utilization and leads to incident execution of many applications. The main motivation of Map Reduce Online is to overcome these problems, by allowing pipelining between operators.
References


Author Profile

Vidya Vasant Polispursuing her Masters of Engineering in the Computer Networks, Computer Department, KJ College of Engineering Management & Research, and Savitribai Phule University. She received Bachelor of Engineering degree in Information Technology from University Of Pune, Pune, India.