ABSTRACT:

After the Map phase and before the beginning of the Reduce phase is a handoff process, known as shuffle and sort. Here, data from the mapper tasks is prepared and moved to the nodes where the reducer tasks will be run. When the mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. 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.

Keywords: Map phase, Reduce phase, MapReduce, shuffle phase

[1] INTRODUCTION

Recently MapReduce has emerged as one of the most popular computing frameworks for Big Data processing because of the simple programming model and also includes the automatic management of the parallel execution. The MapReduce framework and the open source implementation is widely adopted the major and leading companies like Yahoo!, Facebook and Google. The computation in MapReduce framework has been divided into two main phases, that is map and reduce phases, that in turn are carried out by different map tasks and reduce tasks, respectively. During the map phase, map tasks are launched in parallel in order to convert the input splits to intermediate data to form the key/value pairs. The local machine stores these key/value pairs then they are organized into multiple data partitions, one per reduce task. During the reduce phase,
each of the reduce task fetches its part of data partitions from all map tasks to generate the final result. In between the map phase and the reduce phase, there is a shuffle step. During this step, the data produced by the map phase are ordered, partitioned and transferred to the appropriate machines executing the reduce phase. During this time, it causes a huge volume of traffic from the network traffic pattern carried out from all map tasks to all reduce tasks. The network traffic imposes a serious constraint on the efficiency of data analytic applications. To explain with example, when we have a tens of thousands of machines, the data shuffling could account for 58.6% of the cross-pod traffic and it amounts to over 200 petabytes in total in the analysis of SCOPE jobs. There will be considerable performance overhead in case of the shuffle-heavy MapReduce tasks, which could be up to 30-40%. A hash function shuffles the intermediate data by default in Hadoop, which then leads to a high network traffic as it ignores network topology and data size associated with each key.

To tackle the problem of high network usage, incurred by the traffic-oblivious partition scheme, we take into account of both task locations and data size associated with each key in the project. 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, we consider to aggregate data with the same keys before sending them to remote reduce tasks. By aggregating the data of same keys before sending them reducers we reduce network traffic as shown in the Fig. 1.
STATE OF THE ART

A system similar to MapReduce uses CamCube’s distinct properties for attaining high performance. Camdoop [1] running on cam-cube has higher performance than Camdoop running over a traditional switch (proven using a small prototype). In majority of the cases Camdoop running over a Camcube has a higher performance, even when current clusters attained full bisection bandwidth. This is because packets on the path are aggregated resulting in significantly decreasing the network traffic. In big data centers, one of the important framework for data processing [2] that has emerged is MapReduce. MapReduce is an algorithm of consisting of three phases: Map, Reduce and Shuffle. As MapReduce systems are largely deployed, many projects to improve its performance have been in place and they look at one among three phases to enhance performance. The main storage component of the Hadoop framework is Hadoop Distributed File System (HDFS). HDFS is framed for processing and maintaining the huge datasets efficiently among cluster nodes. The computation infrastructure of Hadoop has to cooperate with MapReduce [3], the data has to be uploaded to HDFS from local file systems. However when data size is large, this upload procedure takes more time, causing delay for key tasks. This project named Zput mainly proposes a faster mechanism to upload data that uses the approach of metadata mapping and can greatly improve uploading.

Recent times have seen advancement of technologies with high throughput generating large amounts of data i.e. biological data requiring interpretation and analysis [5]. Oflate, to decrease the data complexity and also to make it easier to interpret data, nonnegative matrix factorization (NMF) is used. Different fields in biological research use it. This project comes up with CloudNMF, which is an open source & distributed NMF implantation over a framework of MapReduce. Based on experimental studies, CloudNMF could deal with large data, is scalable.
[3] PROBLEM PP

The processing of large-scale data is simplified using the MapReduce programming model which works very well on the commodity cluster which exploits the processing by processing of map tasks and reduce tasks in parallel. There have been many efforts that have been made towards in order to enhance performance of jobs of MapReduce; in shuffle phase network traffic that is generated is ignored many a times. It plays a key role in improving the performance. Historically, partition of intermediate data is done using a hash function in reduce tasks. However, data size and network topology for each key are not considered. Hence this is not efficient from traffic perspective.

Following are the two issues this project aims to resolve:

1. Network traffic: Cost of the network traffic has to be decreased using a novel intermediate data partition method.

2. Aggregator placement problem: An algorithm called the distribution algorithm which is designed to solve the issue of optimize the big-scale decomposition of the big data applications.

Figure: 2. Proposed MapReduce Model with Aggregators
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. The model is as shown in the Figure 3. The figure shows the MapReduce with the aggregators placed for processing for traffic aware scheme.

![Figure 3. Distributed Algorithm](image)

Figure: 3. Distributed Algorithm

In this section, we verify that our distributed algorithm can be applied in practice using real trace in a cluster consisting of 5 virtual machines with 1GB memory and 2GHz CPU. Our network topology is based on three tier architectures: an access tier, an aggregation tier and a core tier (Fig. 4). The access tier is made up of cost effective Ethernet switches connecting rack VMs. The access switches are connected via Ethernet to a set of aggregation switches which in turn are connected to a layer of core switches. An inter-rack link is the most contentious resource as all the VMs hosted on a rack transfer data across the link to the VMs on other racks. Our VMs are distributed in three different racks, and the map-reduce tasks are scheduled as in Fig. 6. For example, rack 1 consists of node 1 and 2; mapper 1 and 2 are scheduled on node 1 and reducer 1 is scheduled on node 2. The intermediate data forwarding between mappers and reducers should be transferred across the network. The hop distances between mappers and reducers are shown in Fig. 4, e.g., mapper 1 and reducer 2 has a hop distance 6. Data forwarding between mappers and reducers should be transferred across the network. The hop distances between mappers and reducers are shown in Fig. 4, e.g., mapper 1 and reducer 2 has a hop distance 6.
The intermediate data from all mappers is transferred according to the traffic-aware partition scheme. We can get the total network 2690:48 in the real Hadoop environment while the simulated network cost is 2673:49. They turn out to be very close to each other, which indicates that our distributed algorithm can be applied in practice.

[6] CONCLUSION

In this system, we look into so that we can reduce network traffic cost for a MapReduce job by designing a novel intermediate data partition scheme. Furthermore, we jointly consider the aggregator placement problem, where each aggregator can reduce merged traffic from multiple map tasks. 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 partition and aggregation in a dynamic manner. The partition and aggregators help to add to distance aware routing for processing the data for the big data applications. Placing the aggregators as close to the nodes and the client would also add to the network traffic reduction and in turn helps to reduce the cost of the data processing.
REFERENCES


Author[s] brief Introduction
I am Rajeshwari Adrakatti, studying Master of Technology, City Engineering College, Bangalore.

Corresponding Address-
Prashant Inamdar,
#2543, KDC Residency Apartments,
13th MAIN, Kumarswamy Layout 2nd stage,
Bangalore-78.
Mob: 9740122880
(Pin code and Mobile is mandatory)