PREFETCHER FOR TUNING MAPREDUCE FRAMEWORK IN BIGDATA

S.Tamil Selvan[1] | J.Kokilavani[2]

[1](AP / CSE, M.P.N.M.J. Engineering College, Chennimalai, Erode, India, stamilselvancse@gmail.com)

[2](AP / MCA, Vivekanandha Institute of Information and Management, Tiruchengode, India, vanijagan2k4@yahoo.co.in)

Abstract— *The big-data refers to the large-scale distributed data processing applications. Google's MapReduce and Apache's Hadoop, is an open-source framework that operates extraordinarily on large amounts of data. MapReduce framework is the framework that generates a large amount of intermediate data. Such abundant information is thrown away after the tasks finish, because MapReduce is unable to utilize them. In order to enhance efficiency of MapReduce functionality, we propose a data-aware prefetcher framework for big-data applications. In this framework tasks submit their intermediate results to the prefetcher. A task queries the prefetcher before executing the actual computing work. A novel prefetch description scheme and a prefetch request and reply protocol are designed. Experimental results show that Prefetcher significantly improves the completion time of Hadoop MapReduce job.*

Keywords—Big-data, MapReduce,Hadoop, Prefetcher,Intermediate results

1. Introduction

1.1 Big data

Big data is a budding term that describes any voluminous amount of structured, semi structured and unstructured data in areas including internet search, social network, education, finance, health care and business informatics. The main characteristics of BigData are Volume, Variety, Velocity, Variability, Veracity and Complexity. This describes the data is big in Volume, has multiple categories, speed of gathering data to meet the requirement, consistency/quality of the data and the complexity in collecting, processing data to obtain the required information. Google MapReduce[1] is a programming model and a software framework for large-scale distributed computing on large amounts of data. Figure 1 illustrates the high-level work flow of a MapReduce job ,input data is first split and then feed to workers in the map phase. Individual data items are called records. The MapReduce[6] system parses the input splits to each worker and produces records. After the map phase, intermediate results generated in the map phase are shuffled and sorted by the MapReduce system and are then fed into the workers in the reduce phase. Final results are computed by multiple reducers and written to the disk.

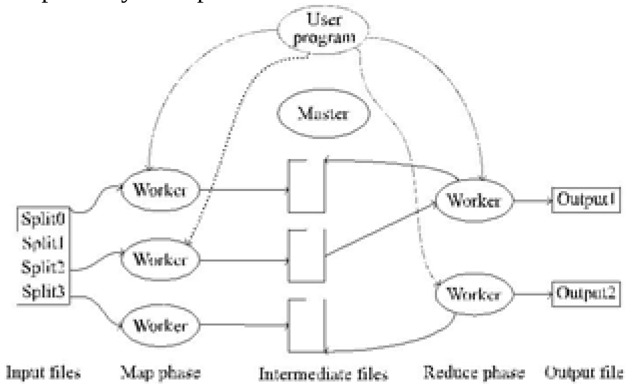

Fig. 1 A illustration of the MapReduce programming model

2. LITERATURE REVIEW

1. Large-scale Incremental Processing Using Distributed Transactions and Notifications [2] Daniel Peng et al. proposed, a system for incrementally processing updates to a large data set, and deployed it to create the Google web search index. By replacing a batch based indexing system with an indexing system based on incremental processing using Percolator, Auther process the same number of documents per day.

2. Design and Evaluation of Network-Leviated Merge for Hadoop Acceleration [5] Weikuan Yu et al. proposed, Hadoop-A, an acceleration framework that optimizes Hadoop with plugin components for fast data movement, overcoming the existing limitations. A novel network-levitated merge algorithm is introduced to merge data without repetition and disk access. In addition, a full pipeline is designed to overlap the shuffle, merge and reduce phases. Our experimental results show that Hadoop-A significantly speeds up data movement in MapReduce and doubles the throughput of Hadoop.

3. Improving Mapreduce Performance through Data Placement in Heterogeneous Hadoop Cluster [3]Jiong Xie et al. proposed that ignoring the data locality issue in heterogeneous environments can noticeably reduce the MapReduce performance. In this paper, author addresses the problem of how to place data across nodes in a way that each node has a balanced data processing load.Given a data intensive application running on a Hadoop MapReduce cluster, our data placement scheme adaptively balances the amount of data stored in each node to achieve improved data-processing performance. Experimental results on two real data-intensive applications show that our data placement strategy can always improve the MapReduce performance by rebalancing data across nodes before performing a data-intensive application in a heterogeneous Hadoop cluster.

4. Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization [4]

Zhenhua Guo et al. proposed, Benefit Aware Speculative Execution which predicts the benefit of launching new speculative tasks and greatly eliminates unnecessary runs of speculative tasks. Finally, MapReduce is mainly optimized for homogeneous environments and its inefficiency in heterogeneous network environments has been observed in their experiments. Authors investigate network heterogeneity aware scheduling of both map and reduce tasks. Overall, the goal is to enhance Hadoop to cope with significant system heterogeneity and improve resource utilization.

3. MOTIVATION

MapReduce provides a standardized framework for implementing large-scale distributed computation, called as, the big-data applications. However, there is a limitation of the system, i.e., the inefficiency in incremental processing[7]. Incremental processing refers to the applications that incrementally grow the input data and continuously apply computations on the input in order to generate output. There are potential duplicate computations being performed in this process. However, MapReduce does not have the technique to identify such duplicate computations and accelerate Task execution. Motivated by this observation, in this paper we propose, a data-aware prefetcher for big-data applications using the MapReduce framework, which aims at extending the MapReduce framework and provide a prefetch layer for efficiently identifying and accessing prefetch items in a MapReduce job.

4. NEED

A scheme to describe the prefetcher, a prefetch request and reply protocols are designed. Prefetcher is implementing by extending Hadoop. It improves the completion time of MapReduce jobs by preventing the repeated jobs.

4.1 Prefetcher Management Phase

Prefetcher works as a centralized system. All the unique input and output data performed by clients are feed in to the prefetcher[8]. The data in prefetcher is stored as a log which contains the input and the place where the output is available. Each client checks the prefetcher before it starts the functioning. If the prefetcher contains that task then the client machine can easily retrieve information from it, else the prefetcher accept data from the client. prefetcher prevents the occurrence of repeated tasks. The above process is clearly illustrated in the figure2

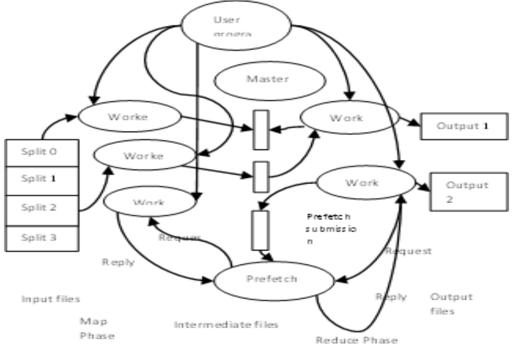

Fig. 2 Architecture of a data-aware prefetcher framework

Below example portray the MapReduce functionality in detail. For an example,we are considering one line as each. However, this is not necessarily true in a real-time scenario. Map() in the below case holds the occurance of each word captured as (the, 1), (blue, 1), (empty, 1) and so on. The output of Map() is Intermediate Results. Reduce phase produce the final sum of words.

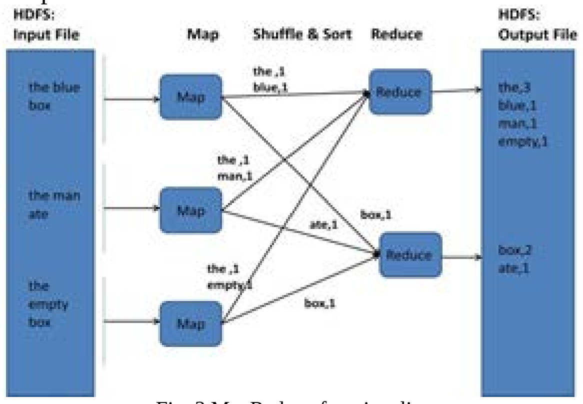

Fig. 3 MapReduce functionality

Input: First the input data are split into fixed number of pieces and then they are feed to different workers (data nodes) in the mapreduce environment. Records are individual data items. Each worker process the input file as per the user program.

Map phase: In this phase, each input split is fed to the mapper who has the function map (). This map function count the occurrence of each word and each occurrence is captured and arranged as (Key, value) pairs. After processing the intermediate results are stored in the data node's hard disk.

Prefetch request and reply protocol: We use prefetch request and reply protocol to get the results that are stored in data nodes. Before processing the splits, the data node sends the request to prefetcher. All the unique input and output data performed by clients are feed to the prefetcher. The data is stored as a log in prefetcher which contains the input and the place where the output is available. Each client checks the prefetcher before it starts the functioning. If the prefetcher contains that task then the client machine can easily retrieve information from it, else the prefetcher accept task from the client. If data is already processed, the prefetcher sends the positive reply to the data node. Otherwise send the negative reply. If negative reply obtained, the data node do the process on the split file. If positive reply obtained, the data node need not process the splits. So, no need to process the repeated data. Prefetcher ensures the repeated input split files need not process more than one time. Finally all the intermediate files are reduced by data node and the final result is stored in Name node.

Reduce phase: In this step, for each unique key, the framework calls the application's Reduce () function. The Reduce can iterate through the values that are associated with that key and produce zero or more outputs. In the word count example, the input value is taken by reduce function, sums them and generates a single output of the word and the final sum. The output of the Reduce is writing to the stable storage, usually a distributed file system.

5. RESULT AND PERFORMANCE ANALYSIS

There are several steps for installing and configuring Hadoop. First install the following software, and then configure hadoop.

- VMware Workstation 10
- Create new virtual machine and install ubuntu OS 14.04
- Install Java SE 7
- Install SSH
- Install Apache Hadoop 2.7.1

Case-1: CPU Utilization of Hadoop & Prefetcher

In this case, CPU utilization ratio is measured by averaging the CPU utilization ratio of the process of the MapReduce jobs over time. From the figure, it is clear that Prefetcher saves significant amount of CPU cycles. With a larger incremental size, the CPU utilization ratio of prefetcher grows significantly too.

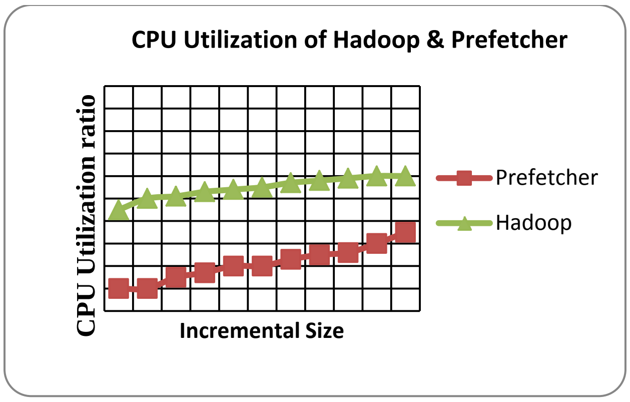

Fig. 4 CPU utilization of Hadoop&Prefetcher

Case-2: Completion time of Hadoop & Prefetcher

This case proves that prefetcher indeed removes redundant tasks in incremental MapReduce jobs and reduces job completion time. The redundancy of input file is checked by prefetcher which in turn tremendously increase the performance.

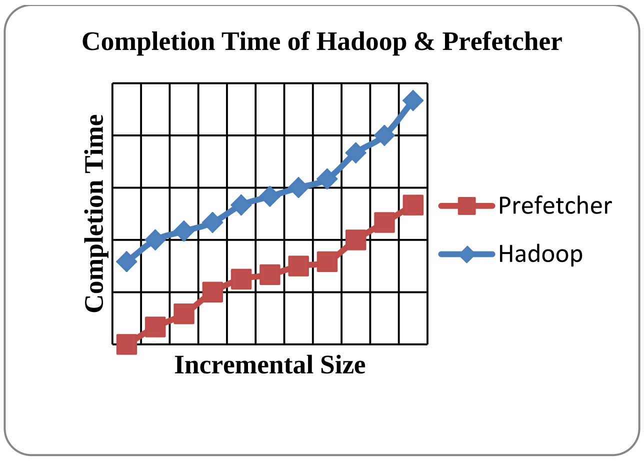

Fig. 5 Completion time of Hadoop &Prefetcher

6. CONCLUSIONS

This paper focuses on the problem of inefficiency in incremental processing. Incremental processing refers those applications that incrementally grow the input data and continuously apply computations on the input in order to generate output. There are lots of duplicate computations being performed in this process. MapReduce does not have a mechanism to find out these computations. The data aware prefetcher in MapReduce framework helps to overcome this problem and provide high efficiency in incremental processing. It prevents the repeated tasks to process and increment the performance.

REFERENCES

[1] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. of ACM, vol. 51, no. 1, pp. 107-113, 2008.

[2] D. Peng and f. Dabek,"Large Scale incremental Processing using distributed Transaction and notification", in Proc. of OSDI'2010, Berkeley, CA, USA, 2010

[3] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", Department of Computer Science and Software Engineering Auburn University, Auburn, AL 36849-5347

[4] Zhenhua Guo, Geoffrey Fox "Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization" School of Informatics and Computing Indiana University Bloomington Bloomington, IN USA

[5] Weikuan Yu, Member, IEEE, Yandong Wang, and Xinyu Que, "Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration", IEEE Transactions on Parallel and Distributed Systems

[6] Y. Zhang, S. Chen, Q. Wang, and G. Yu, "i2mapreduce: Incremental mapreduce for mining evolving big data," CoRR, vol. abs/ 1501.04854, 2015.

[7] Y. Bu, B. Howe, M. Balazinska and M.D Ernst,"Hadoop: Efficient iterative data processing on large clusters," in proc, VLDB Endowment, 2010, vol. 3,no.1-2, pp.285-296.

[8] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. N. Rao, V. Sankarasubramanian, S. Seth, C. Tian, T. ZiCornell and X. Wang. 2011. Nova: Continuous pig/Hadoop workflows, in Proc. of SIGMOD'2011, New York, NY, USA.