VOL. 10, NO. 12, JULY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

OPTIMIZING MAPREDUCE FUNCTIONALITY IN BIGDATA USING CACHE MANAGER

Devi. L and S. Gowri
Faculty of Computing, Sathyabama University, India
E-Mail: devikanth.btech@gmail.com

ABSTRACT

The MapReduce framework generates a large amount of intermediate data. These data thrown away after the tasks finish. MapReduce is unable to utilize these data. To improve the efficiency of MapReduce functionality by reducing repeated jobs in data nodes, we develop cache management system inside the MapReduce framework. In which, tasks submit their intermediate results to the cache manager. Before executing the actual computing work, task queries the cache manager. In a Data Aware cache, cache request and cache reply mechanisms are designed. Implementing Cache by extending Hadoop, it improves the completion time of MapReduce jobs. It detects the occurrence of repeated job in the incremental data process. Also, stops the repeated work and minimize the processing time so that to provide the optimized usage of MapReduce nodes.

Keywords: big data, reduce function, data node, name node, map function, intermediate results, incremental data.

1. INTRODUCTION

1.1 BigData

BigData as the name describes a large data sets that is growing beyond the ability to manage and analysis using with the traditional data processing tools. Big data represents large and incremental volume of information that is mostly untapped by existing data warehousing systems and other analytical applications. These data is being gathered from different sources like web search, mobile devices, software logs, cameras, etc. As of 2012 2.5 Exabyte data created by every day and the size of the growth gets doubled by every next year.

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 the data to get the required information.

There are much architecture used in BigData and Google introduced a new process called 'MapReduce', which allocates the tasks parallel to the nodes and collect, which is a very successful framework. Later this framework was adopted by Apache open source project called Hadoop.

Larger organizations interested in capturing the data to add significant values like the business. BigData is mostly used in Retail, Banking, Government, Real estate, Science and research sectors. This helps in decision making, cost/time reduction, market analysis etc.

1.2 Hadoop

It is an open source platform for storage and processing of diverse data types that enables data driver enterprises to rapidly derive the complete value from all their data.

1.3 Overview of Hadoop

The original creators of Hadoop are Doug cutting (used to be at Yahoo! now at Cloudera) and Mike Cafarella (now teaching at the University of Michigan in Ann Arbor). Doug and Mike were building a project called "Nutch" with the goal of creating a large Web index. They saw the MapReduce and GFS papers from Google, which were obviously super relevant to the problem Nutch and were trying to solve. They integrated the concepts from MapReduce and GFS into Nutch; then later these two components were pulled out to form the genesis of the Hadoop project.

The name "Hadoop" itself comes from Doug's son, he just made the word up for a yellow plush elephant toy that he has. Yahoo! hired Doug and invested significant resources into growing the Hadoop project, initially to store and index the Web for the purpose of Yahoo! Search. That said, the technology quickly mushroomed throughout the whole company as it proved to be a big hammer that can solve many problems.

2. RELATED WORK

2.1 Bigtable

Referred paper "A Distributed Storage System for Structured Data" by Fay Chang and Jeffrey Dean"- BigTable have been developed by Google as a distributed storage system, structured as a large Table petabytes in

size. It is being used since 2005 in many Google services to store items such as URLs, many versions of webpages; over 100 TB of satellite image data; hundreds of millions of users; and performing thousands of queries a second.

Semi- Structured storage design is used in BigTable. It is a big map indexed by row key, column key and timestamp used for lookup, insert and delete. It has 'n' number of column families and various attributes of web pages are stored in column families. It is indexed by row key and split into multiple subtables. The subtables are called as tablets.

BigTable is not an RDBMS; instead it's a distributed, persistent and multidimensional database. The data is stored in row key, column key and a 64bit timestamp. The key gets generated by the database or by the application. For example in the Google Webtable (for Google search) the reverse URL is used as the row key (com.google.www; com.facebook.www). Various attributes is stored in column families of the webpage. Each column contains multiple versions of the timestamp helps to retrieve the recent version. The data key points are to some content from the webpage.

Google File System is used in Bigtable and the internal file format for storing data is called SSTable. The application defines the number of versions to keep based on the timestamp. Alternatively the application can also specify how long entries to be stored.

2.2 MapReduce

In the paper "Simplified Data Processing on Large Clusters" by Jeffrey Dean and Sanjay Ghemawatat - MapReduce is designed for processing large volumes of data by dividing the work into a set of independent tasks. The input data format is specified by the user and it is application-specific. Output is set of <key, value> pairs. Map and Reduce are the two functions used. The Map function applied on the input data and it produces a list of intermediate <key, value> pairs. These intermediate pairs are applied by the reduce functions using the same key. The output pairs may be zero or more and it's produced by some kind of merging operation. Finally, with their key value, the output pairs are sorted. The programmer provides only the Map function in the simple MapReduce programs. Other functionalities like grouping of intermediate pairs that have same key and sorting the final result is provided at the runtime. Unit of work in MapReduce is Job. A Job has two phases, they are map and reduce phase. For example, MapReduce job counts the words across documents. Map phase counts the words in the document and reduce will collate the data into word counts. At map phase, inputs are divided into splits and are made to run parallel across Hadoop clusters. Hadoop Distributed File System (HDFS) is the file system used in Hadoop. The Reduce tasks collate the final results and stores the results in HDFS.

2.3 Improving MapReduce performance

MapReduce performance is improved by prefetching before scheduling based on virtual reality. For big data sets the communication between Data nodes and Name node affect the performance of MapReduce functionality. Hadoop schedules tasks to the nodes near the data locations which will preferentially decrease the data transmission overhead and this works well in homogeneous and in dedicated MapReduce environments. Unfortunately, it's difficult to take advantage of data locality in heterogeneous or shared environments. The performance of MapReduce in heterogeneous or shared environments is improved by data pre-fetching mechanism which is proposed in this paper. In the pre-fetching mechanism the data is fetched to the corresponding compute nodes in advance. It is proved that data transmission overhead is reduced effectively with the theoretical analysis. The main idea of data pre-fetching mechanism proposed in this is to overlap the data transmission process with data processing process. By this way, the overall performance of MapReduce could be improved.

By analyzing the process of map tasks in Hadoop, when the input data is not local the serial execution of data transmission and data processing is discovered to cause overhead. Map tasks can be pre-fetched to nodes where map tasks are executed. Experiments are carried out in both heterogeneous and shared environments. Parameters includes map run time, data transmission time, total input data size and input split size of map tasks in the experimental applications. The experiment results show: up to 94% of data transmission time is reduced and 15% performance improvement in jobs' execution.

2.4 Smart speculative execution strategy

MapReduce performance is improved by smart speculative execution strategy. MapReduce is widely used parallel computing framework for large data processing. MapReduce two major performance metrics are job execution time and cluster throughput. This can be seriously impacted by straggler-machines in which the tasks take an unusually long time to finish. The common approach for dealing this straggler problem is speculative execution, which is by simply backing up those slow running tasks on alternative machines. We do have multiple speculative execution strategies that has been proposed, but they have some drawbacks: (i) average progress rate to identify slow tasks (ii) Difficult to handle the situation when there exists data twist among the tasks, (iii) Will not consider whether backup tasks can finish earlier when choosing backup worker nodes To improve the effectiveness of speculative execution they developed a new strategy, maximum cost performance (MCP). In MCP, the following methods are used to identify stragglers accurately and promptly: (i) Slow tasks are selected by using the progress rate and the process

bandwidth within a phase, (ii) By using exponentially weighted moving average (EWMA) the process speed and the task's remaining time is calculated, (iii) Cost-benefit model is used to determine which task to backup based on the load of a cluster. Cluster of 101 virtual machines running in a variety of applications on 30 physical servers, we evaluated MCP and the experiment results show that compared to Hadoop-0.21, MCP can run jobs up to 39 percent faster and improve the cluster throughput by up to 44 percent.

3. PROPOSED SYSTEM

Data aware cache overcomes the limitations of the existing system. Cache aims at extending the MapReduce framework. It provides a cache layer for efficiently identifying and accessing cache items in a MapReduce job. In which, tasks submit their intermediate results to the cache manager. Before actual computing work Task queries the cache manager before executing the.

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

3.1 Advantages

- **Redundant tasks:** The proposed system helps MapReduce framework to avoid redundant tasks to retrieve the existing data from the cache manager.
- **Efficiency:** By implementing cache in Hadoop, the efficiency for computing incremental data is improved to greater extent.

This significantly saves the completion time as well as the work load assigned to the nodes and reduces the task of MapReduce.

3.2 System architecture

Input data is first split and then feed to workers in the map phase. Records are individual data items. Cache manager works as a centralized system. All the unique input and output data performed by clients are feed in to the cache manager. The data in cache is stored as a log which contains the input and the place where the output is available. Each client checks the cache before it starts the functioning. If the cache contains that task then the client machine can easily retrieve information from it, else the cache accept task from the client.

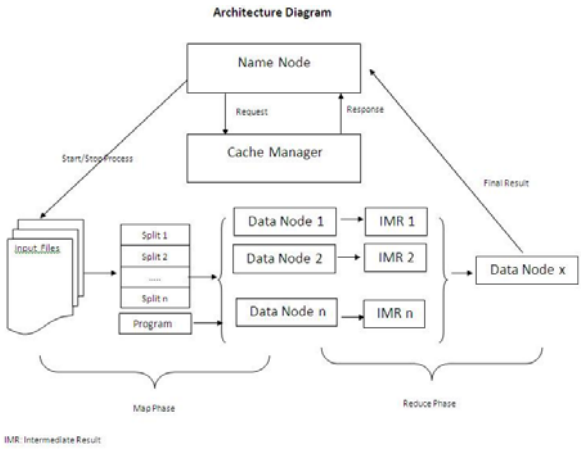

Figure-1. System architecture.

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 (SQL, 1), (DW, 1), (SQL, 1) and so on. The output of Map() is IMR-Intermediate Results. Reduce phase produce the final sum of words.

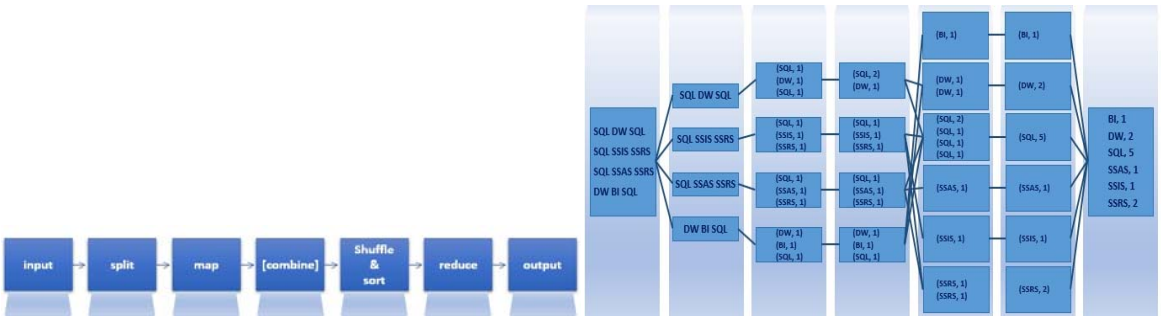

Figure-2. MapReduce functionality.

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 () has the logic on how to process the input data. For example, map () is containing the logic to 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.

Cache management phase: Cache manager works as a centralized system. All the unique input and output data performed by clients are feed in to the cache manager. The data in cache is stored as a log which contains the input and the place where the output is available. Each client checks the cache before it starts the functioning. If the cache contains that task then the client machine can easily retrieve information from it, else the cache accept data from the client. Cache prevents the occurrence of repeated tasks. Thus it decreases the Processing time of system.

Cache request and reply protocol: We use cache 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 Cache Manager. All the unique input and output data performed by clients are feed to the cache manager. The data is stored as a log in cache which contains the input and the place where the output is available. Each client checks the cache before it starts the functioning. If the cache contains that task then the client machine can easily retrieve information from it, else the cache accept task from the client. If data is already processed, the Cache Manager 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. Cache Manager 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.

4. RESULT AND PERFORMANCE ANALYSIS

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

- VMware Player 5.0.
- Create new virtual machine and install Cent OS 6.3
- Install Java SE 7
- Install Eclipse Juno Release 1.0
- Install Apache Hadoop 1.0.3

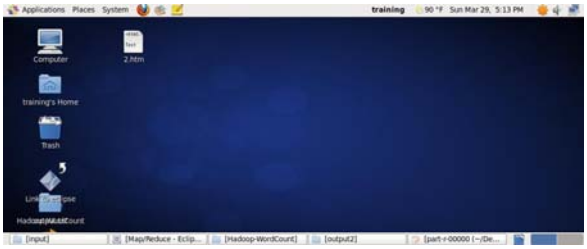

Figure-3. Hadoop configuration.

Below is the Hadoop console output, this actually splits the tasks into several records and allocate it to the available data nodes. This console output will update the status of Map () and Reduce () and the task completion status.

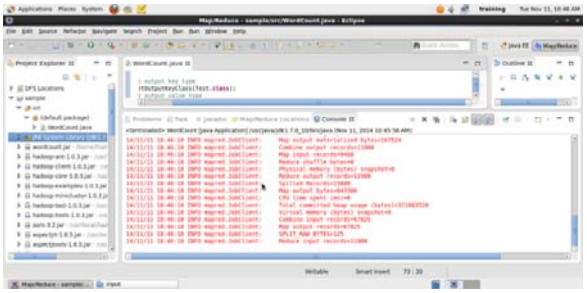

Figure-4. Hadoop console output.

Bigdata usually process large size and incremental data's in nature. Incremental data means, the data to be processed is increasing continuously. The systems which process the bigdata get the undetermined number of input files. Consider the Hadoop system can accept 10 input files for understanding the concept of cache system inside the Hadoop framework.

4.1 Case-1: More than 4 files are duplicates (the files having the same content and may have different file names or extensions)

In this case, Map phase unwantedly process the same file content again, it split the entire file into different pieces, and send it to the datanodes to process. Datanodes are unaware of redundant data. So the wastage of time is

more and performance of the MapReduce is decreased without any mechanism to identify the redundancy. If the cache mechanism is used inside the Hadoop, the redundancy of input file is checked which in turn tremendously increase the performance.

If 5 files are duplicates,

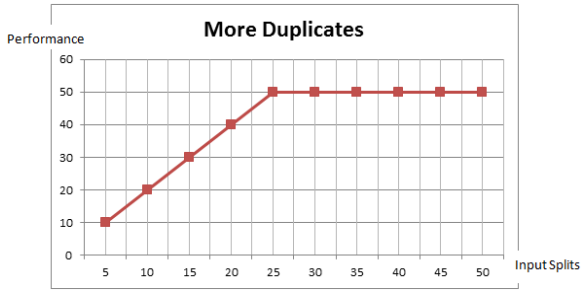

Figure-5. More redundant input.

If each file has 5 splits, the Hadoop process only 25 splits rather than unwanted 50 splits. So it reaches maximum performance as early as possible. So by introducing cache in more number of duplicates the performance is more powerful.

4.2 Case-2: One or two files are duplicates

In this case Map phase unwantedly process less amount of repeated data. Once again datanodes are unaware of redundant data, so the wastage of time is less compare to previous case and performance of the MapReduce is decreased to some level. If the cache mechanism is used inside the Hadoop, then it will check the redundancy of input file and increases the performance by somewhat.

If 2 files are duplicates,

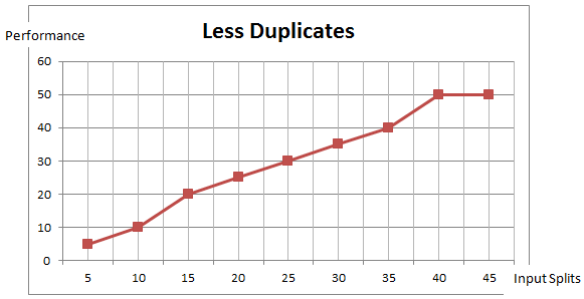

Figure-6. Less redundant input.

If each file has 5 splits, the Hadoop process only 40 splits rather than unwanted 10 splits. So it reaches maximum performance after processing all splits. So by introducing cache when there are minimum numbers of duplicates, the performance is somewhat powerful.

4.3 Case-3: None of the files are duplicates

In this case Map phase process all the data, if the cache mechanism inside the Hadoop will check the redundancy of input file and there are no duplicates then the performance is normal. In this case, cache is not useful and consumes some amount of time.

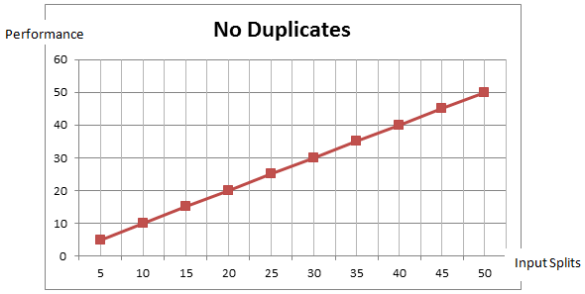

Figure-7. No redundancy.

4.4 Findings

In practical, more number of input files is being processed by MapReduce framework, and there are more possibilities of redundant data. In first case, as per the experimental results, if we use cache mechanism inside the framework then there is better performance.

5. CONCLUSIONS

This project 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 cache 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] 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.

[2] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins. 2008. Pig latin: A not-so-foreign language for data processing, in Proc. of SIGMOD'2008, New York, NY, USA, 2008.

[3] U. A. Acar. 2009. Self-adjusting computation: An Overview, in Proc. of PEPM'09, New York, NY, USA.

[4] T. Karagiannis, C. Gkantsidis, D. Narayanan and A. Rowstron. 2010. Hermes: Clustering users in large-scale e-mail services, in Proc. of SoCC '10, New York, NY, USA.

[5] Memcached-A distributed memory object caching system, http://memcached.org/, 2013.

[6] P. Scheuermann, G. Weikum, and P. Zabback. 1998. Data partitioning and load balancing in parallel disk systems, The VLDB Journal. 7(1): 48-66.

[7] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. 2008. Improving MapReduce performance in heterogeneous environments, in Proc. of OSDI'2008, Berkeley, CA, USA.

[8] H. Herodotou, F. Dong and S. Babu. 2011. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics, in Proc. of SOCC'2011, New York, NY, USA.

[9] S. Wu, F. Li, S. Mehrotra and B. C. Ooi. 2011. Query optimization for massively parallel data processing, in Proc. of SOCC'2011, New York, NY, USA.

[10] D. Logothetis, C. Olston, B. Reed, K. C. Webb and K. Yocum. 2010. Stateful bulk processing for incremental analytics, in Proc. of SOCC'2011, New York, NY, USA.

[11] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin and L. Zhou. 2010. Comet: Batched stream processing for data intensive distributed computing, in Proc. of SOCC'2011, New York, NY, USA.

[12] D. Battr'e, S. Ewen, F. Hueske, O. Kao, V. Markl and D. Warneke. 2010. Nephele/pacts: A programming model and execution framework for web-scale analytical processing, in Proc. of SOCC'2010, New York, NY, USA.

[13] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu and H. Hacig¨um¨uS. 2011. Activesla: A profit-oriented admission control framework for database-as-a-service providers, in Proc. of SOCC'2011, New York, NY, USA.

[14] H. Gonzalez, A. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley and W. Shen. 2010. Google fusion tables: Data management, integration and collaboration in the cloud, in Proc. of SOCC'2010, New York, NY, USA.