Map Reduce Optimizer with an Intermediary Cache Manager

Ms Vishakha Mehendale[1] Prof Mrs. Shital Dhamal[2]
[1,2]Department of Computer Engineering
[1,2]SES'SFOE College of Engineering, Diksal, Raigad, Mumbai University, India

Abstract— Big data is a term used to address the data set whose size is beyond the ability of traditional software technologies to capture, store, manage and process within a tolerable elapsed time. In the world of big data, we observe daily based data which is generated daily e.g. Google, Facebook, and Amazon etc. This large volume of data is un reliable to store, manage and analyze which runs on commodity hardware. As data is in large size it takes more time to execute. 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 proposed an Intermediary 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 an Intermediary cache Management, cache request and cache reply mechanisms are designed. It detects the occurrence of repeated job in the incremental data process.

Key words: Big data, MapReduce, Incremental Processing, Cache

I. INTRODUCTION

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.

II. RELATED WORK

The paper [1] proposed a mechanism which is used to access the cached data with less time and resources. All the local caches can be coordinated by the distributed cache called DoCache. It uses the data oblivious caching algorithm for processing data because algorithm is much easier to analyze than real cache's characteristic such as replacement policy. For application data, it works by divide and conquer method. Paper [2] aims at reducing the recompilation time of MapReduce job. It proposed a HADOOP distributed system for large data processing, which has two types of cache memory one is local memory, and the other is distributed centralized cache memory it uses LRU cache replacement to remove least recently used frames when cache is full.

This paper [3] compares the performance of HDFS with a commercial cluster file system called VERITAS with variety of workload and MapReduce applications. They developed the cluster file system connector module for Apache Hadoop platform. In this both VERITAS cluster file system and SF- CFS can run in parallel to share file system namespace. This [4] paper proposed Azwraith, a hierarchical MapReduce approach aiming to optimize data locality and task parallelism of MapReduce application on Hadoop. In this model each map or reduce task is assigned to a single node is treated as a separate MapReduce job and further decomposed in map and reduce tasks.

In this paper [5] emphasis is given on security for nodes in MapReduce framework. It has introduced additional component Accountable MapReduce. It employs an Auditor Group to conduct an A-test on every worker in the system. If malicious behavior occurs AG is able to detect it and provide verifiable evidence. The paper [6] focuses on MapReduce applications with huge amount of intermediate key value pairs and relatively low amount of computations situation. Here run time is not dominated by application code in map and reduce. The overhead is of library itself on commodity multicore computer. The core challenge is the organization of MapReduce intermediate data. The paper has presented a new MapReduce library called Metis whose intermediate data structure is a hash table with b+ tree in each entry. Tiled-MapReduce (TMR) [7] has extended the general MapReduce programming model with tiling strategy. It is more effective for MapReduce to iteratively process small pieces of data in turn than processing a large pieces of data at one time on shared memory multicore platforms. Tiled- MapReduce presented number of different methods; executed optimizing techniques which are directed on multicore to improve the memory, cache and CPU resources.

III. PROBLEM STATEMENT

MapReduce has been widely recognized for its elastic scalability and fault tolerance, with the efficiency being relatively disregarded. The MapReduce framework generates a large amount of intermediate data. These data thrown away after the tasks are finished. MapReduce is unable to utilize these data. To improve the efficiency of MapReduce functionality by reducing repeated jobs in data nodes, we proposed an Intermediary 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 an Intermediary cache Management, cache request and cache reply mechanisms are designed. Implementing Cache extends functionality of MapReduce, 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.

IV. OUR SOLUTION

Our proposed system is "Map Reduce Optimizer with an Intermediary Cache Manager", is an intermediary cache system for big-data applications using the MapReduce framework. It aims at extending the MapReduce framework and provisioning a cache layer for efficiently identifying and accessing cache items in a MapReduce job. It identifies the source input from which a cache item is obtained. The intermediate results obtained by processing file splits should be cached. In this system, we present a method for reducers to utilize the cached results in the map phase to accelerate the execution of the MapReduce job.

The Map Reduce Optimizer with an Intermediary Cache Manager has different modules as Map phase, Cache Manager, Reduce Phase.

A. *Input to Map Phase*

Map Reduce gains popularity for its simple programming interface. The input data are get selected by the user in the cluster. The input files are splitted. And then that is given as the input to the map phase. The input to the map phase are very important. These inputs are processed by the map phase. In map phase the data is sorted by ascending order. Using sorted data the reducing will be worked.

B. *Analyze in Cache Manager*

The request is transferred to the cache manager. The request is analyzed in the cache manager. Cache manager searches the request from the mapping phase in the database. If the data is present in the cache manager means then that is transferred to the map phase. The intermediate data for the request is transferred to the mapping phase. If the data is not present in the cache manager means then there is no response to the map phase.

C. *Results to Cache Manager*

The input data is processed in the map phase. The intermediate result is generated in the map phase. That intermediate result are transferred and stored in the cache manager.

D. *Input to Reduce Phase*

The intermediate result are given as the input to the reduce phase. These data are processed in the reduce phase. The reduce phase is the final processing in the map reduce framework. The reduce phase gives the final output. After the mapping phase the reduce phase is executed. It is very important for the final output. It reduce the time of execution in the map phase.

E. *Reduce Phase Output*

The input for the reduce phase is very important. The intermediate result from the map phase is given as the input to the reduce phase. In reducing phase the word count is calculated for the input data. The reduce phase is the last process in the map-reduce. Output from the reduce phase is used as the final output. Finally the processing time and the memory of the mapping and reducing are optimized.

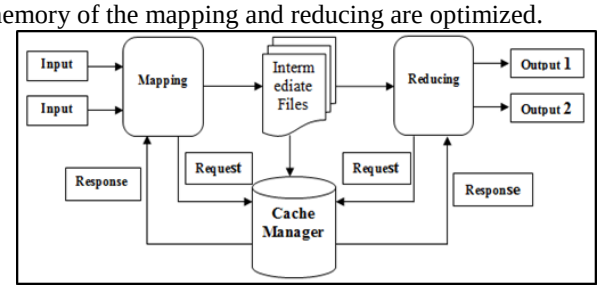

Fig. 4.1: Block diagram of MapReduce Optimizer with an Intermediary Cache Manager

F. *Advantages*

1) In this system, it efficiently identifying the cache identifying and accessing cache items in a Map Reduce job.
2) Cache items in the map phase are easy to share because the operations applied are generally well-formed.
3) It can eliminate all the duplicate tasks in incremental MapReduce jobs.

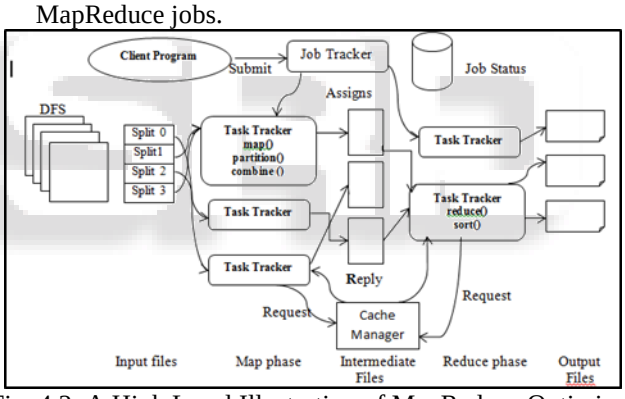

Fig. 4.2: A High Level Illustration of MapReduce Optimizer with an Intermediate Cache Manager

V. EXPERIMENTAL RESULT

In MapReduce framework large files are taken as input. It supports incremental processing hence it will take large files to process. Very first step is to split the file in no of splits. Our system, splits the input into four splits and those four splits are processed simultaneously. To understand the concept we will assume 10 input files. We have following scenarios

A. *Case 1: More than 4 Files are Redundant*

Before implementing intermediary cache manger the map reduce framework will split the file into number of splits. As system is not aware of redundant data operations. It will perform operations repeatedly. But Intermediary Cache Manger if 5 input files are given and no of splits are 4, then instead of processing 40 splits our system will process only 20 splits. Hence by identifying redundancy it saves time and performance of system is increased

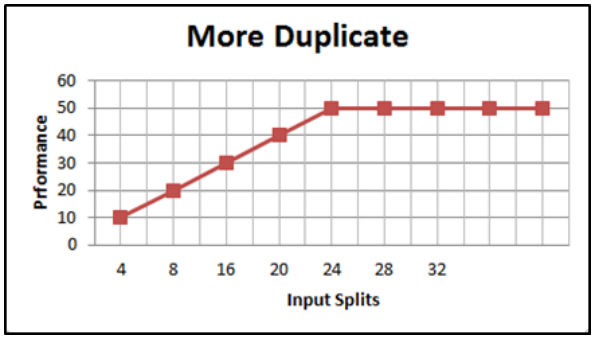

Fig. 5.1: More Duplicate Files

B. Case 2: One or Two Files are Redundant

In this case also mapreduce will process unwanted data because given input has duplicate files. But as compared to previous case no of duplicate files are less, hence less time is wasted. In our system with an Intermediary Cache Manger, if 2 input files are redundant it will process only 4 splits instead of 16. Hence performance is high for redundant data and normal for regular data.

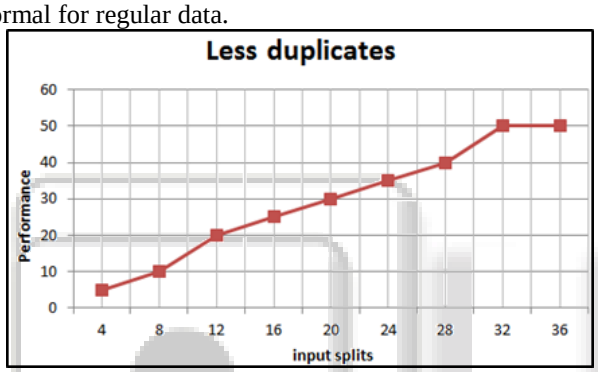

Fig. 5.2: Less Duplicate Files

C. Case 3: when no Redundant Data is given as Input

In this case, no redundant data is applied as input. Hence system has to process all the data. Cache manger will check for redundancy of data but no duplicate files are applied. System performance is normal. And in this case cache will store new entries of data and invests time in searching redundant data.

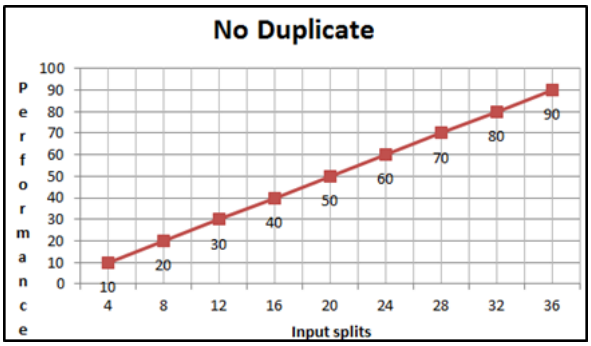

Fig. 5.3: No Duplicate Files

VI. Conclusion

Big data term addresses very large amount of data which is distributed. Applications dealing with such a data faces problems like storage and retrieval of data and analyzing such data is not a realistic. To process big data we have Map Reduce Programming model with elastic scalability and fault tolerance. While processing big data it creates lots of intermediate data. This projects aim a framework for big data applications with an intermediary cache. It ensures efficient incremental processing.

The Map Reduce Optimizer with intermediary Cache manager for big data applications reduces execution time, CPU utilization, and alters the Map Reduce programming model, so that there are very less modification to it. The main improvement in Map Reduce Optimizer with Intermediary Cache manager is that it eliminates duplicate tasks in incremental processing. But the system also has its own drawbacks, where it requires maximum amount of cache and it also needs a better life-time management of cache. Also system performance will tremendously increase in case of highly redundant data. And in normal inputs cache manger invest time in searching of files and system performance is normal.

References

[1] Hemalatha and Sindhuja, "Do-Cache: Mechanism for Hadoop Using Map Reduce in Big Data", published in International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE) ISSN: 0976-1353 Volume 13 Issue 4 –MARCH 2015.

[2] R.Saranya, N.Sundaram, "Distributed Data Cache For Bigdata With LRU Cache Using Mapreduce Technique", published in International Journal of Computer Science and Information Technology Research, ISSN 2348-1196, Vol. 3, Issue 2, pp: (796-801), Month: April - June 2015

[3] Anirban Mukherjee, Joydip Datta, Raghavendra Jorapur, Ravi Singhvi, Saurav Haloi, Wasim Akram,"Shared Disk Big Data Analytics with Apache Hadoop", in 978-1-4673-2371-0/12/$31.00 ©2012 IEEE

[4] Zhiwei Xiao, Haibo Chen, Binyu Zang Parallel Processing Institute, Fudan University "A Hierarchical Approach to Maximizing MapReduce Efficiency".

[5] Zhifeng Xiao and Yang Xiao, The University of Alabama, Tuscaloosa, AL 35487-0290 USA, Accountable MapReduce in Cloude Computing", published in The First International Workshop on Security in Computers, Networking and Communications, 978-1-4244-9920-5/11/$26.00 ©2011 IEEE 1099

[6] Yandong Mao Robert Morris M. Frans Kaashoek "Optimizing MapReduce for Multicore Architectures" Massachusetts Institute of Technology, Cambridge, MA.

[7] Rong Chen, Haibo Chen, and Binyu Zang "Tiled-MapReduce: Optimizing Resource Usages of Dataparallel Applications on Multicore with Tiling" Parallel Processing Institute Fudan University PACT'10, September 11–15, 2010.

[8] Gurmeet Singh, Puneet Chandra and Rashid Tahir "A Dynamic Caching Mechnism for Hadoop using Memcached" Department of Computer Science, University of Illinois at Urbana Champaign.

[9] Zhao Cao, Shimin Chen, Dongzhe Ma, Jianhua Feng, Min Wang "Efficient and Flexible Index Access in MapReduce" (c) 2014, Copyright is with the authors. Published in Proc. 17th International Conference on

Extending Database Technology (EDBT), March 24-28 2014, Athens, Greece.

[10] Ganesh Ananthanarayanan, Ali Ghodsi, AndrewWang, Dhruba Borthakur, Srikanth Kandula, Scott Shenker, Ion Stoica "PACMan: Coordinated Memory Caching for Parallel Jobs" University of California, Berkeley, Facebook, Microsoft Research, KTH/Sweden.

[11] James Manyika, Michael Chou, Brad Brown, Jacques Bughin, Rihards Dobbs, Charlas Roxburgs, Angela Hung Bayer "Big data: The next frontier for innovation, competition, and productivity" McKinsey Global Institute, May 2011.

