ISSN 2395-1621

Improve Performance of Hadoop using Efficient Data Aware Caching for Big Data

[#1]Amit A. Khodaskar., [#2]Prof.Thombare B.H.

[1]amit_khodaskar@rediffmail.com
[2]babanthombre@gmail.com

[#12]Department of Computer Engineering

Shree Ramchandra College of Engineering, Lonikand,
Pune, India.

ABSTRACT

The Hadoop MapReduce framework is developed and widely accepted by open-source communities for solving massive parallel processing operations with the help of distributed environment. MapReduce systems in various situations are applied to achieve definite performance goals, upgrade existing systems to meet increasing business demands using novel network topology, new scheduling algorithms and resource arrangement schemes. As most of data is unchanged in an iterative programs reloading and reprocessing it results in wasting Input/Output, network bandwidth, and CPU resources and also put extra load for scheduling tasks, reading data from disk, and moving across the network. To avoid the load of these extra operations, new efficient data aware caching framework is introduced with the help of cache programming model and value degree cache replacement algorithm. Data structure is created for caching to store data for temporary purpose needed by software or hardware. Results obtained demonstrate that efficient data aware caching decreases significant completion time of MapReduce jobs. This paper proposes a new method to improve the performance of MapReduce by using distributed memory cache as a high speed access between map tasks and reduce tasks. Map outputs sent to the distributed memory cache can be gotten by reduce tasks as soon as possible. Experiment results show that our prototype's performance is much better than that of the original on small scale clusters. To our knowledge, this is the first effort to accelerate MapReduce with the help of distributed memory cache.

ARTICLE INFO

Article History

Received: 25th December 2016

Received in revised form :

25th December 2016

Accepted: 28th December 2016

Published online :

28th December 2016

Keywords: Map Reduce, Hadoop, Cache, Distributed Cache

I. INTRODUCTION

Recently, MapReduce [1] proposed by Google has emerged as an attractive option.Through a simple interface with two functions, map and reduce, MapReduce facilitates implementing many real world applications. Even more impressive is that the map and reduce operations can be parallelized easily. The map function can be parallelized naturally, and reduce can be parallelized by partitioning the reduction of keys across many machines [2]. Moreover, MapReduce allows programmers to benefit from frameworks for load balancing, communication, task scheduling, and fault tolerance. The response time of MapReduce is very important for short jobs where users want answers as soon as possible, such as queries on logs for debugging, monitoring and business intelligence [3]. Short jobs are very common for MapReduce [1]. In addition, there are many data-analysis systems providing SQL-like queries on top of MapReduce, such as Google's Sawzall [4], Yahoo!'s Pig [5], Facebook's Hive [6]. All those systems are intensive to MapReduce's execution time. Optimizing MapReduce's execution time can also prevent the jobs from occupying system resources too long. At present, the gap between computing capability and I/O capability of machines is getting bigger than ever. But the main concern of MapReduce's architects was put on its fault tolerance ability in large scale clusters. So disks are used heavily in MapReduce to protect the system from failure. And disks play too many other roles in MapReduce. Disks' latency and throughput are seriously affected for those reasons. Because all map outputs put to local disks will be read by reduce tasks remotely, shuffle time of each reduce task has become one of the main

performance bottlenecks in the MapReduce frameworks. At the same time, since MapReduce's scheduling strategies focus on computing capability of each node, memories in heterogeneous clusters are not utilized effectively. We also note that small scale MapReduce clusters, which have no more than dozens of machines, are very common in most companiesand laboratories [7]. Node failures are infrequent in clusters of such size. So it's possible to construct a more efficient MapReduce framework for small scale heterogeneous clusters.

The main objective of the proposed system is to The Performance is improved by adding more slots instead of nodes which helps in more map and reduce tasks to be scheduled parallel. Better execution time is obtained with more data local tasks. TaskTracker contacts JobTracker to check availability of slot. If slot is available task is scheduled. If input for task is on a different node, then it is rack-local and data is streamed from other node. But with caching, tasks scheduled completed earlier. And in reference caching, initial request attended by the references cached contributed to the improvement because these references assist the system to find the data into cache faster.

II. LITERATURE SURVEY

Big data is evolving drastically around us. Researchers, scholars defined big data depending on different perspectives. The very common definition of big data is that the datasets that could not be imagined, understood easily, accomplished and processed by conventional information technology software/hardware tools in an expected time. The volume of information increasing every day as people create such large data with the help of communications like voice calls, emails, texts, uploaded pictures, video, and music [8].

MapReduce is used by researchers at Google from 2004. Due to limitless features of big data, researchers understood that a single machine is unable to serve all data computation/analytic solutions, and new environment like distributed system is required to process and store data in parallel [9].

An open source implementation Apache Hadoop similar to MapReduce became available with free of cost for large scale data analytics, big-data applications and other major parallel computations in which large input data is required. Hadoop is adopted by several distinguished and renowned companies like Yahoo!, Facebook and became mainstay [10].

The Hadoop computational model has several distinguished attributed properties. It is simple that its API stipulates a few entry points for the application programmer specified mappers, reducers/combiners, partitioners, for formatting input and output [11].

In addition to basic large scale computational models, lot of software tools is built around as Hadoop ecosystem. These tools are such as Apache Hive, Apache Giraph, Apache Hama, Apache Mahout all of which harness Hadoop [12].

Since the last ten fifteen years it is observed that Hadoop clusters size is increased, as well as increase in size of RAM memory supported by each machine. The smaller <K, V> size, high amount of data reusability and Hadoop Job interactive ness make it possible to build a robust caching mechanism preferably in -memory for substantial improvement in performance [13].

III. PROPOSED SYSTEM

System architecture:

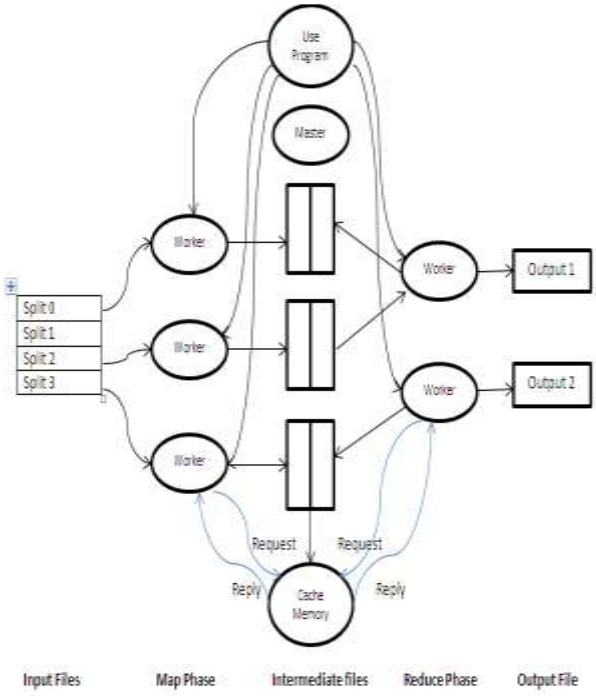

Fig 1. System architecture

Efficient data aware caching system is implemented by using Hadoop incorporated components. Cache manager communicates with task trackers and provides cache items on receiving requests which is implemented in the system. The cache manager uses HDFS, the DFS component of Hadoop, to manage the storage of cache items. In order to access cache items, the mapper and reducer tasks first send requests to the cache manager. Mapper and Reducer classes only accept key value pairs as the inputs which are fixed by Hadoop interface. An open accessed component InputFormat class allows application developers to split the input files of the MapReduce job to multiple file splits and parse data to key value pairs. The component TaskTracker class is responsible for managing tasks, understand file split and bypass the execution of mapper classes entirely. TaskTracker also manages reducer tasks and bypass reducer tasks by utilizing the cached results.

IV. METHODOLOGY

1. Preprocessing File- words are stored in file. In file preprocessing stop words are removed, like 'a ', 'of ', 'the' etc. and stemming is also done. After preprocessing file, collection of words on which operations are performed will be retained.

2. File Vector- When collection of words activity ends in preprocessing, it is very important to evaluate how important a word is to a document in a collection or corpus. The significance increases equivalently to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Tf-idf (Term Frequencies Inverse Document Frequencies) algorithm is a statistical measurement weight of about the importance of word in a document often used in search engine, web data mining, text similarity computation and other applications. These applications are often faced with the massive data processing.

3. Create Signature- To find similar file it should be compared with content of each and every files available among the millions of files which makes process time consuming. So to make process faster compact bit representation of each file vector is created, Signature. To create Signature f bit vector is used and this vector initialized to zero first then it hashed with file vector and then compared the weight of word and decision is taken whether file will be incremented or decremented. Signature file makes process faster which is an advantage.

4. Use Locality Sensitive Hashing to find nearest neighbor- In large clustering environment to compare file Signature to each and every cluster is time consuming. Therefore to avoid comparing of file with each and every cluster locality sensitive hashing technique is used which ensures that only nearest neighbor need to be checked to place file. For this hashing function is used which query file Signature to find nearest neighbor and m number of neighbor is returned to client.

5. Store file with related files- If m neighbor is return to client then only that m neighbor will be compared and after finding cluster where file will be placed, this subclusterid will be given to Name Node. Name Node maintains subclustertable which store subclusterid and file placed on that cluster. If Name Node finds entry then that file will be placed on subclusterid but if subclusterid is not found then new subcluster will be created, file will be stored on newly created cluster and file and cluster Signature will be calculated and this information will be updated to subcluster table. Now suppose client want to execute map task and system should not execute repeated map task for this, cache will be implemented. Cache table will be created which stores file name, operation performed on that file and result file name.

V. CONCLUSION

The proposed efficient data aware caching framework is powerful for cache management. The new cache replacement algorithm is implemented and called it as value degree to calculate the value of tuple being replaced. Results show that there is substantial improvement in performance of Hadoop jobs by reducing completion time and storage overhead using efficient data aware caching for big data application.

REFERENCE

[1] Yaxiong Zhao, Jie Wu, and Cong Liu "Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework" Tsinghua Science and Technology ISSNl l1007-0214l l05/10l lpp39-50 Volume 19, Number 1, February 2014.

[2] Zhu Xudong, Yin Yang, Liu Zhenjun, and Shao Fang "C-Aware: A Cache Management Algorithm Considering Cache Media Access Characteristic in Cloud Computing", Research Article, Hindawi Publishing Corporation, Mathematical Problems in Engineering, Article ID 867167, 13 pages, Volume 2013.

[3] Meenakshi Shrivastava, Dr. Hans-Peter Bischof "Hadoop-Collaborative Caching in Real Time HDFS" Computer Science, Rochester Institute of Technology, Rochester, NY, USA.

[4] Dachuan Huang, Yang Song, Ramani Routray, Feng Qin "SmartCache: An Optimized MapReduce Implementation of Frequent Itemset Mining" The Ohio State University, IBM Research – Almaden.

[5] Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Ernst "HaLoop: Efficient Iterative Data Processing on Large Clusters" Department of Computer Science and Engineering University of Washington, Seattle, WA, U.S.A. 36th International Conference on Very Large Data Bases, September 1317, 2010, Singapore.

[6] 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.

[7] 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.

[8] Executive Office of the President " Big Data: Seizing Opportunities, Preserving Values" May 2014.

[9] [14] Min Chen, Shiwen Mao, Yunhao Liu "Big Data: A Survey" Published online: 22 January 2014, Springer Science+Business Media New York 2014.

[10] Avraham Shinnar, David Cunningham, Benjamin Herta, Vijay Saraswat "M3R: Increased Performance for InMemory Hadoop Jobs", roceedings of the VLDB Endowment, Vol. 5, No. 12,38th International Conference on Very Large Data Bases, Istanbul, Turkey, August 27th 31st 2012.

[11] Tom White "Hadoop: The Definitive Guide",Third edition,Oreilly, ISBN: 978-1-449-31152-0.

[12] Venkatesh Nandakumar "Transparent in-memory cache for Hadoop-MapReduce" A thesis submitted in conformity with the requirements for the degree of Master of Applied Science Graduate Department of Electrical and Computer Engineering University of Toronto, 2014.

