International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Survey on Optimal Data Storage of Cache Manager for Big Data Using Map Reduce Framework

Rupali Pashte[1], Ritesh Thakur[2]

[1]Savitribai Phule Pune University, Institute of Knowledge COE, India

[2]Professor, Savitribai Phule Pune University, Institute of Knowledge COE, India

Abstract: ***MapReduce is an open-source framework used for implement big-data sets. The very famous word big-data that points to the wide-ranging diffused applications that works on unmatched huge data sets.While processing on this big-data in-between abundant knowledge that being neglected after getting process finished. In this paper we analyze various techniques emerged for simplified data processing on large data sets (big-data) and along with new and very effective technique that sensible of data cache model (framework). In this model process collects its in-between data to the cache manager. This new proposed cache model reduces the execution time of the task as it checks in cache, which could keep and store up execution.***

Keywords: Dache, MapReduce, Cache-management, big-data

1. Introduction

Now a days, there are many applications required large amount of data as input data, such applications known as big-data applications. Google provides a job scheduling framework MapReduce for huge diffused computing on this big data application. MapReduce takes large amount of data as input and divides into number of jobs. MapReduce is very simple programming model and effective to implement large scale data applications. As name suggest 'MapReduce' works into two phase Map and Reduce. In map phase, divided jobs are feed to the workers and generates the results with abundant intermediate information. These results are getting shuffled by MapReduce after completion of the Map phase and passed to next phase i.e. Reduce phase. In many existing systems generated abundant intermediate data can be neglected after completion of the task because it can't be handled by MapReduce. In Reduce phase, results are getting computed by multiple workers and final results are writing on the disk. In this proposed system we are using the optimal storage of the cache. Resulted abundant intermediate information is forward to the cache manager and stored into the cache storage for future access of the processed result if the data matched in map phase. Existing systems, many of them are never used the intermediate data further because these data never utilize by them. Hadoop Common is a Distributed File System (DFS) that provides storage for file.

A HDFS will consider only one master node and number of worker nodes. In HDFS architecture, JobTracker and NameNode are responsible for managing running task and managing HDFS respectively. The JobTracker and NameNode are present on the same machine. The divide jobs are handled by JobTracker in MapReduce. The existing system is not supported to the incremental input resulted computation overhead. Inspired by this we propose optimal use of cache memory with intension identifying and selecting the correct information or result in MapReduce task which is processed by Map or Reduce phase. While using this concept, we need to be face two challenges.

Indexing: MapReduce work on large data setsresulted different intermediate information because multiple jobs are processed on multiple workers with different operation. Main challenge in this step is that how to index these different intermediate results in cache memory that identify the correct operation and content of the produced result of the job and this is very easy part. Protocols for request and reply: As we know, while jobs getting processed by Map or Reduce phase generated data can be very vast. How to transfer this abundant data while any worker request becomes complicated. The designed protocol should collect the needed cache result with process as up to mark hence delay for transmission and overhead being minimized.

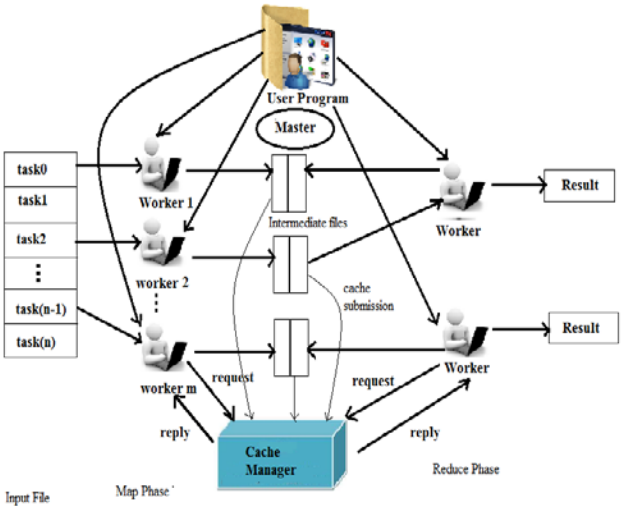

Figure 1: System Architecture

2. Optimized Cache Storage

As we know, intermediate data is very big and we have to store that abundant data into the limited sized cache memory. We can use different protocols and different customized indexing in cache memory. To store the newly created cache

items old cache items need to be deleted as storing capacity of the cache memory not enough.

2.1Adaptive Replacement Cache:

MapReduce works on multiple jobs parallel and produces the different intermediate result. These intermediate data is very big in size and storing at customized index of the cache memory. As the size of the cache memory is limited, so we need to replace the cache index data by the newly created intermediate data for that we are going to be use extended LRU replacement algorithm i.e. adaptive replacement cache algorithm with better performance to optimal use of cache storage. An adaptive replacement cache algorithm [10] that keeps track of both recently used and frequently used pages with history of the deleted pages for both. In LRU, it stores only entry for last recently used pages. New entry of the page is added to the top of the list after last entry has been deleted. If cache hit then it move to the top by pushing all entries in the list down. ARC is used to improve the LRU strategy by dividing cache list into two sub-lists, t1 for recently used and t2 for frequently used entries. These two lists are extended with a ghost list b1, and b2 which is attached to the bottom of the list respectively. These two extended list are used to keeping track of the recently deleted or evicted cache entries.

This ghost list contains metadata and the actual data itself.
All lists are used in the ARC are,
t1=stores the recent entries.
t2= stores the referenced frequent entries at least twice.
b1=entries recently deleted from cache list t1.
b2=entries recently deleted from cache list t2.

New list l1 is combination of t1 and b1 that stores the combined history of recent references. Similarly, l2 is stores the combined history of t2 and b2.

```
. . . [   b1 <-[     t1   <-!->    t2    ]-> b2   ] . .
      [ . . . . [ . . . . . . ! . .^. . . ] . . . . ]
                [   fixed cache size (c)    ]
```

[] brackets show the actual size of the cache, which is fixed. New entries are added into the list t1 to the left of '!' mark that shows the actual size of the l1 list, and'^'indicates the target size for t1 list that may be equal to, smaller than, or larger than the actual size of the t1.

New entries are added to the list t1 and page hits are added to the list t2. If the hit is occur in the b1 then size of the list t1will increase by pushing'^' to the right and last entry of the list t2 is pushed into the list b2. If hit is occur in the list b2 then t1 will shrink by placing the '^' to the left and last entry of the list t1 will pushed into the list b1. In this algorithm, cache miss is not affect '^', but '!' boundary is placed closer to '^'.

2.2 Same Map Task Cache

In optimal use of cache manager, there are two types of cache items and they have different complexities as they produced at different phase. In Map phase it is easy to use cache memory as they processed in well-formed. Cache memory in Map phase is not use directly used if file splitting is not context free.

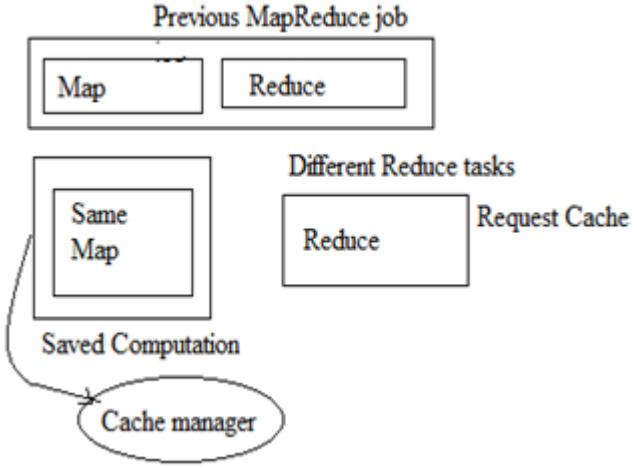

Figure 2: Two jobs have same Map task.

When Reduce phase uses the cache memory it taking two situation in consideration, first is when current MapReduce job of the reducer is totally different from the previous one and second is when reducers can take the advantage of the previous MapReduce job as they append the results from cache items to produce the final result.

3. Dache: A Data Aware Caching

An observation regarding Hadoop and NoSQL database applications is that they generate and store a large amount of intermediate data [1], and this abundant information is thrown away after the processing finishes. Inspired by this observation, Yaxiong Zhao et al proposed a data-aware cache framework for big-data applications, which they called Dache. In Dache, tasks submit their intermediate results tothe cache manager. Monetary values of computational resources are well captured in existing cloud computing services, for example, in AmazonAWS [7] and Google Compute Engine[8]. Active SLA [9] is an admission control frame work that takes into account the monetary gain of admitting jobs to run on a cloud-based database service provider.

4. MapReduce: Simplified Data Processing on Large Clusters

They designed a MapReduce program that can be automatically paralyzed and executed on large cluster of the machines [2].As this model is easy to use even for programmers don't have experience with parallel and distributed systems and divided jobs are well balanced using load balancing technique. In MapReduce model of the Jeffrey Dean and Sanjay Ghemawat cache memory is not taking in consideration hence computation overhead increases.

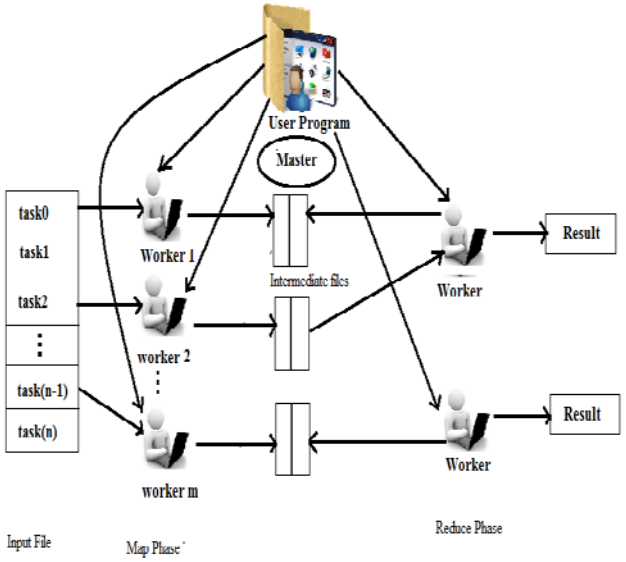

Figure 3: Last Processing on large cluster

5. A Hierarchical Approach to Maximizing MapReduce Efficiency

As in multi-core platforms in Hadoop has limitations in exploiting data locality and parallization of the tasks [3].As we know, Hadoop is open-source for implement the MapReduce, uses the Java Virtual Machines to run the actual MapReduce task. Hadoop implements the writable interface to take the advantage of Serialization and deserialization, resulting overhead in creation and destroy an object. In some applications that need to processing the same slice of the data n-times or iteratively to obtain final output.

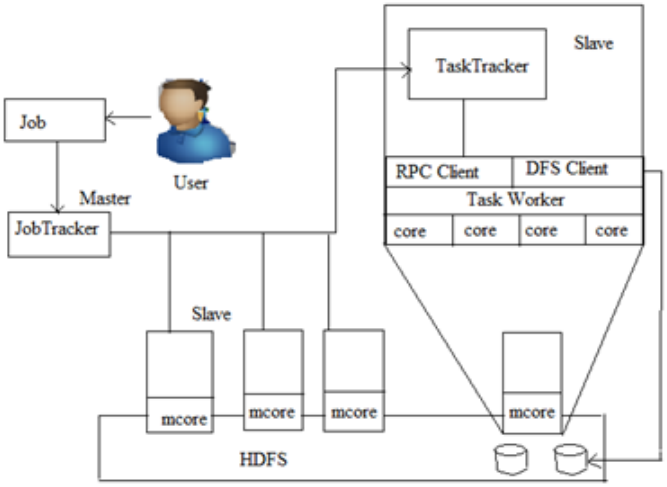

Figure 4: Hierarchical Approach

In many MapReduce systems it does not consider data locality across multiple processing iterations, therefore same data can loaded many times from HDFS to workers that process the data. In this system they extended Hadoop using this concept. Maximizing parallelism and data locality is the main aim of this system. In hierarchical approach, input file is distributed by JobTracker (Master) and assigned to worker (slave). In this system next these single workers are act as separate MapReduce system and split the allocated job by master further into number of sub-jobs that increase the runtime of the MapReduce by acting separate MapReduce job and is divide into a Map and Reduce task on a single worker.Figure-3 describes the hierarchical approach by Azwraith that integrates an efficient MapReduce runtime for Hadoop.

6. Secure MapReduce in Cloud Computing

In this system Zhifeng Xiao et al proposed secure MapReduce that can be find out the malicious node in the MapReduce. While working, the accountability test is held by Auditor Group to find out the malicious node in real time. This system is clicked on very sensitive place of the MapReduce which is very important to increase system performance [4].They proposed new component known as Auditor Group(AG) which can take test to find malicious node in the MapReduce. Customer cloud resources split it into multiple slices which is a bunch of machines allocated to a customer. An AG is allocated to each divided piece of cloud which runs MapReduce. Because of that, independence and privacy of the customers is maintained. The Auditor Group manager is responsible for creation of the new Auditor Group, management, and close. The Auditor Manager takes different parameters into account while creation of the new Auditor Group for newly created piece of the cloud computing. These parameters are user's data size, time and other requirements.

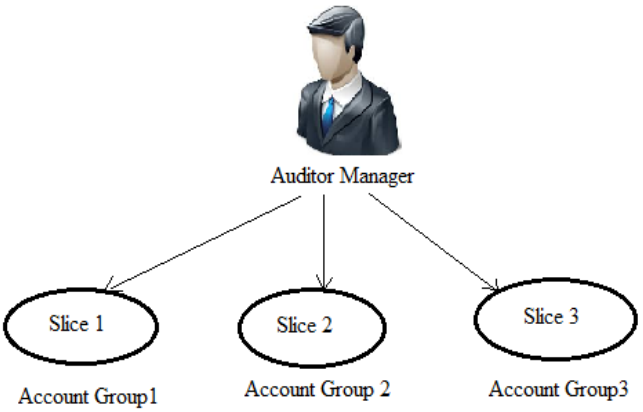

Figure 5: Audit Group in cloud computing

The master is responsible for provide information needed for conduct accountability test. The Group Head is responsible for allocate the accountability test task to the workers also known as Group Members and they provide the status report to master. The performance of the MapReduce will improve automatically as the unauthorized tasks are not executed in cloud. Main drawback of this system is that it can give false output can disfigure everything. This system also introduced some new components and need to add on top of original MapReduce may generates several new problems.

7. Shared Disk Big Data Analytics

In this system, Anirban Mukherjee et al in their study compared a widely used clustered file system: VERITAS Cluster File System (SFCFS) with Hadoop Distributed File System (HDFS) usingpopular MapReduce benchmarks like Terasort, DFS-IO andGridmix on top of ApacheHadoop[5]. This system does not need any changes in original MapReduce applications. Big Data analytics can be made up and processing very fast by setting few parameters in the

configuration of Apache Hadoop. Apache Hadoop platform has a very simple architecture as the clustered file system connector module is developed by them. VERITAS clustered file system is used instead of HDFS functionality in the Hadoop stack. MapReduce framework and File system communicate with other by using APIs and that proposed SF-CFS to the Hadoop class. This could be achieved because the Map-Reduce framework always talks in terms of a well-defined File System [10] API for each data access. Both HDFS and their clustered file system connector module implement this File System class shown in figure-5.

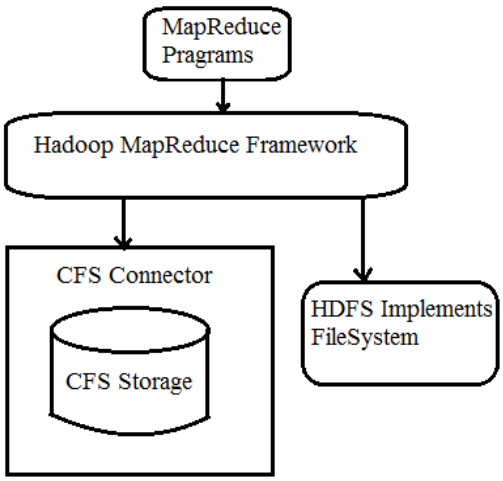

Figure 6: SC-CFS Hadoop Connector

8. Conclusion

In this paper we have analyzed several methods discovered by several researches in solving the problem of performance in MapReduce. In these researches have some problems that can be solved by using cache memory is beginning to become a very exciting application in MapReduce.

References

[1] Yaxiong Zhao, Jie Wu, and Cong Liu "Dache: A Data Aware Caching for Big-Data Applications UsingtheMapReduce Framework," ISSN 1007-0214l, 05/10pp39-50, Volume 19, Number 1, February 2014.
[2] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," OSDI 2004.
[3] ZhiweiXiao."A Hierarchical Approach to Maximizing MapReduce Efficiency,"2011 International Conference on Parallel Architectures and Compilation Techniques.
[4] ZhifengXiao, "AccontableMapReduce in Cloud Computing". 2011 IEEE.
[5] AnirbanMukherjee, "Shared Disk Big Data Analyticswith Apache Hadoop" 2012 IEEE.
[6] Hadoop, http://hadoop.apache.org/, 2013.
[7] Amawon web services, http://aws.amazon.com/, 2013.
[8] Google compute engine, http://cloud.google.com/ products/computeengine.html, 2013.
[9] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacig¨um¨uS, Activesla: A profit-oriented admission control framework for database-as-a-service providers, in Proc. of SOCC'2011, New York, NY, USA, 2011.
[10] Nimrod Megiddo, Dharmendra S. Modha,IBMAlmaden Research Center, "Outperforming LRU with an Adaptive Replacement Cache Algorithm".

Author Profile

Rupali Patil received the B.Tech. from Dr. Babashaheb Ambedkar Technological University, Lonere, Raighad (MH) and pursuing M.E. degrees in Computer Engineering from Institute of Knowledge College of Engineering, Pune in 2005 and 2014, respectively.

Prof. Ritesh Thakur Working as assistant professor in Institute of Knowledge College of Engineering, Pimple-jagtap, Pune