ISSN: 2321-7782 (Online)
Volume 3, Issue 1, January 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Lifetime Management of Cache Memory using Hadoop

Snehal Deshmukh[1]
Computer, PGMCOE,
Wagholi, Pune,
India

Kasturi Sutar[2]
Computer, PGMCOE,
Wagholi, Pune,
India

Nimeratha Kadam[3]
Computer, PGMCOE,
Wagholi, Pune,
India

Maya Jadhav[4]
Computer, PGMCOE,
Wagholi, Pune,
India

Prof. Rupali Patil[5]
Computer, PGMCOE,
Wagholi, Pune,
India

Abstract: Hadoop is a framework for distributed processing of large data set across the clusters of commodity computers using simple programming model. Google MapReduce and Hadoop MapReduce are two different implementations of MapReduce framework. MapReduce framework creates a intermediate data. After finishing the task such intermediate data can not be easily accessible. Motivated by this observation, in this paper we propose Using the map reduce framework a data aware cache system for a big data application a cache layer for efficiently identifying and accessing cache items in a MapReduce job provides aims at extending the MapReduce framework.

Keywords: Hadoop, MapReduce, TaskTracker, JobTracker, Cache Memory.

I. INTRODUCTION

Hadoop is framework that allows for the distributed processing of large data set across the clusters of commodity computers using simple programming model There are two main component of Hadoop HDFS and Map Reduce. HDFS is a storage part and MapReduce is processing part. There are two main Types of MapReduce first one is Google MapReduce and second Hadoop MapReduce. Google MapReduce and Hadoop MapReduce are two different implementations of MapReduce framework. Hadoop MapReduce is open source ,Google MapReduce is not and actually there are not so many available details about it. Hadoop uses as a standard distributed file system the HDFS(Hadoop Distributed File System)while Google MapReduce uses GFS (Google File System).Hadoop is Implemented in Java. Google MapReduce seems to be in C++. cluster and manages the blocks commodity computer using a simple programming model. Hadoop implements MapReduce framework with two categories of components: a Job Tracker and many Task Trackers. The Job Tracker commands Task Trackers to process data in parallel through two main functions: Map and Reduce. HADOOP uses also Distributed File System (HDFS) is highly fault tolerance and high throughput. HDFS contain two main components Namenode and Datanode. Name node is a Master Node. In name node contain Job Tracker. It maintains the blocks which are present on the data nodes. Datanode is a slave node. In data node contain task tracker .It actually keep the data and process the data .Slave which are deployed on each machine and provide the actual storage. It is also responsible for serving read and writes requests for the Client. Client is an application which will used to interact Job Tracker and task tracker via the Namenode and Datanode. HDFS architecture contains main part called as rack. Rack is work as storage area where multi data node put together, group at the node, collection of data in the cluster physical collection at data node store in single location. There are many companies who uses Hadoop like YAHOO, GOOGLE, FACEBOOK, AMAZON, IBM.

In "Big and Distributed data application" large amount of input data is presented and it is split to the Task Tracker. Every data file is called as "Records". In MapReduce phase all input data is distributed to all Task Tracker. After Mapping, shuffling and sorting is done by using intermediate file created by the Task Tracker. Again it submitted to the Task Tracker for the Reducing phase. Finally by using MapReduce the Reduced output is generated in the disk.

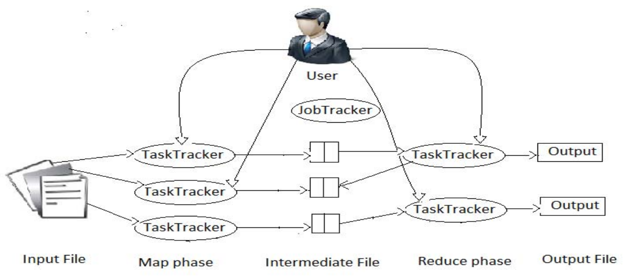

Fig:1 Data processing without cache memory.

II. Literature Review

1. Large-scale Incremental Processing Using Distributed Transactions and Notifications [3] 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, Author process the same number of documents per day.

2. Design and Evaluation of Network-Leviated Merge for Hadoop Acceleration [7] 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 [5] 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 [6] 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.

III. Motivations

Hadoop is a framework for distributed processing of large data set across the clusters of commodity computers using simple programming model. . MapReduce framework creates a intermediate data. After finishing the task such intermediate data cannot be easily accessible and there is so many duplicate data presented in this process. But in MapReduce phase does not have ability to identify the duplicate data and perform job execution.

Motivated by this observation, in this paper we propose Using the map reduce framework a data aware cache system for a big data application a cache layer for efficiently identifying and accessing cache items in a MapReduce job provides aims at extending the MapReduce framework.

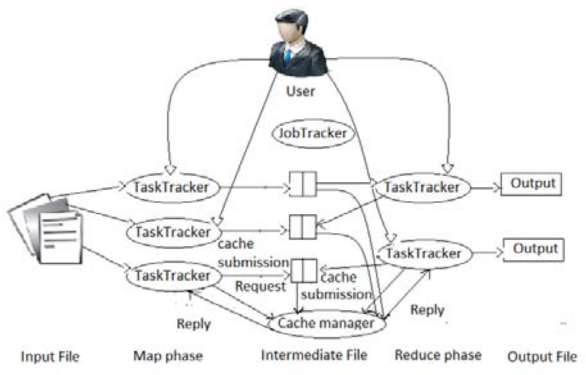

Fig:2 Data processing with cache memory.

We defined Dache, a data –aware cache system for big data applications Such as storing and processing using the MapReduce framework. Dache aims at improving the MapReduce framework and supply a cache layer for efficiently identifying and accessing cache items in a MapReduce . The architecture of cache description scheme should assign a changeable indexing that enables the applications to describe their functions and the product of their generated near by results.

Cache request and reply scheme using protocol

The size of the intermediate information can be very big. When such information is requested by other task trackers, to finding how to transport this information becomes complex. So we use protocols. Protocols able to communicate by sending request to the cache manager from the task tracker processes potentially that needs information and the reply is to be send from cache manager to the task trackers who needs the information for further processing. by using this protocols the transformation time and overhead are decreased.

In this paper, we shows a new architecture for cache description scheme. A high-level description scheme is presented in Fig. 2. This scheme determines the source input from which a cache item is obtained, and the functions performs on the input, so that a cache information item produced by the task tracker in the map phase is indexed properly. In the reduce phase, we devise a mechanism to take into consideration the partition operations applied on the output in the map phase. We implement cache memory in the Hadoop project by extending the relevant components.

IV. Cache Description

Cache description scheme in map phase

Cache refers to the intermediate information that is produced by task tracker during the execution of a MapReduce task. A piece of cached information is to be saved in a Distributed File System (DFS). The content of a cache item is described by the original information and the functions performed. a cache item is described by a 2-tuple: (initial point, function). Initial point is the name of a file in the Distributed File System function is a one by one list of available functions performed on the starting point of file. The exact format of the cache description of different applications differs according to their specific contents. The application developer designed and implemented this cache description who are responsible for implementing their MapReduce tasks.

Cache description scheme in reduce phase

The input given to the reduce phase is also a list of key pairs. The schemes applied on the reduce phase is much same as that of map phase cache description scheme. The original input and the performed functions are required. The intermediate results of the map phase saved in the DFS is the original input which is required for reduced phase. The performed functions are determined by unique IDs that are specified by the user. The results those are generated in map phase which after cached, cannot be directly used as the final output. in incremental processing. In the Map phase the intermediate results are to be generated likely to be intermixed in the shuffling phase, which does not matches between the original input of the cache items and the newly generated input. A method is to apply a finer description of the original input of the cache items in the reduce phase. The details should include the original data files generated in the Map phase.

V. Case Study with Hadoop Map Reduce

Map cache

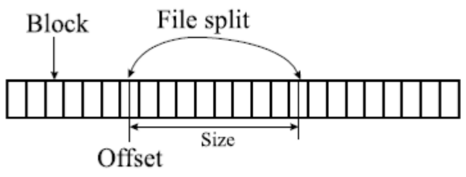

Fig 3: A file in a DFS which are fixed size data block and this file is stored as multiple blocks.The original file name offset and size is identified by file split.

Apache Hadoop (HDMR) is an open-source implementation of the MapReduce distributed parallel processing algorithm originally designed by Google. In Map phase users input is split into multiple file splits which are then processed by an equal number of Map task tracker processes which is known as data-parallel processing procedure. As depicted in Fig. 3, based on user-provided rules a file split partitions one or more users input files . processing file splits should be cached which is obtained from intermediate results. Each file split is determined by the original file name, offset, and size. complications implements in describing cache items by this determination scheme . In an ideal situation, each cache item would only correspond to a single input file, which makes identifying it with the above scheme straightforward. This scheme is slightly modified to work for the general situation. The original field of a cache item is changed to a 3-tuple of (file name, offset, size)..

Reduce cache

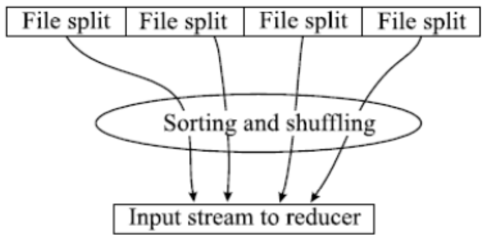

Fig 4:By sorting and then shuffling multiple files of mapper is obtained by the input stream to a Reducer. The input to the reducer is identified by using this mapping.

Whole input of the MapReduce job is to be given as a input to the reducers . Therefore, to describe the original file to the reducers we could simplify the description by using the file name with a version number; to distinguish incremental changes version number of the input file is used right a way approach is to encode the size of the input file with the file name. Since we assume that appending new data at the end of the file, i.e., only incremental changes are allowed, during different MapReduce jobs. To identify the changes made the size of the file is enough .As shown in Fig. 4 to generate the input for the reducers; file splits are sorted and shuffled. Although this process is internally operated by the MapReduce framework, the users are able to describe a shuffling remedy by supplying a devider which is Fig. 4 sorting and shuffling multiple output files of task trackers, The input stream to a reducer is obtained . This mapping is used to determine the input to the reducer. In Hadoop implemented as a Java object. The devider examines the key of a record and identifies in the reduce phase which reducer should process this record. Therefore, the cache description should be attached with the devider, which can be implemented as a linerized object in Hadoop. Different deviders partitioned the same input file splits that are produce different reduce inputs, therefore cannot be treated as the same. Finally, the deviders is assigned index of the reducer. The whole description is a 3-tuple: (file splits, deviders, reducer index).

VI. Protocol

The map and reduce phase can be utilized in different scenarios that generate the partial result. There are two types of cache item: map cache and reduce cache. When it comes to sharing under different scenarios it has different complexity. The operations applied are generally well-formed because Cache items in the map phase are easy to share. The cache manager reports the previous file splitting scheme used in its cache item when processing each file split. Needs to split the files according to the same splitting scheme in order to utilize the cache items. However, if the new the new MapReduce job uses a different file splitting scheme to MapReduce job, the map results cannot be used directly, unless the operations applied in the map phase are context free. By context free, we mean that the operation only generates results based on the input records, which does not consider the file split scheme. This is generally true.

We identify two general situations When considering cache sharing in the reduce phase. The first is when the reducer's complete different jobs from the cached reduce cache items of the previous MapReduce jobs. In this case, after the mappers obtained submit the results from the cache items. The partitioner provided by the new MapReduce job to feed input to the reducers it MapReduce framework uses. In the Map phase the saved computation is obtained by removing the processing. Requires additional mappers to process usually, new content is appended at the end of the input files. However, this does not require additional processes other than those introduced above. The second situation is when the reducers can actually take advantage of the previously-cached reducing cache items; the reducers determine how the output of the map phase is shuffled by using the description scheme. Automatically identifies the best-matched cache item by cache manager to feed each reducer, which is the one with the maximum overlap in the original input file in the Map phase.

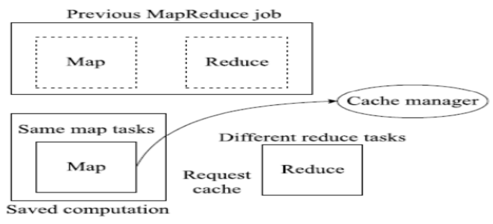

Fig 5:The situation where two MapReduce jobs have the same map tasks, which could save a fraction of computation by requesting caches from the cache manager.

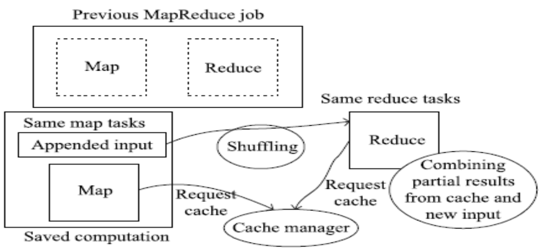

Fig 6:The situation where two MapReduce jobs have the same map task as well as reduce tasks. The reducers combine results from the appended input and cache items to produce the final results.

VII. Management of Cache Item

How much time particular cache item should be place in the DFS is determined by Cache manager. The storage space will waste by holding a cache item for infinite amount of time when there is none another MapReduce task using the cache items intermediate result. There are two types of policies to understand the lifetime management of cache item as follow. A cache item can be promoted to permanent file and store it in DFS is done by Cache manager, it happens when MapReduce task is using cache item as its final result. In this case, the cache manager does not manage the lifetime of the cache item. The mapping between the Cache description and the actual storage location is also managed by the cache manager.

Fixed Storage items

In this policy the fixed storage space is given to cache item. When there is no enough storage space for storing the new cache item then old cache item is removed from it. The policy of removing old cache item can be modeled as a Classic cache replacement problem. In our first implement the Least Recently Used (LRU) is used. Determination of cost of allocation of fixed storage quota can be done by a pricing model that captures the monetary expense of using that amount of storage space for. In a public Cloud service such pricing models are available. In next model 3.3.2 we will discuss it in more detail.

Optimal utility items

Hitting a plateau due to the diminishing return effects is likely to increasing the storage space. A utility-based measurement is used to determine the optimal space required for cache item which increases the benefits of dache and respect the constraint cost. This scheme estimates the saved computation time, by caching a cache item for a given amount of time. The monetary gain and total cost can be derives from these two variables. The difference of subtracting total cost from gain should be positive. To accomplish this, the computational resources are used for accurate pricing model. Although cloud computing use such model, the conventional computer do not. The computational values are well kept in exciting cloud computing services for e.g., Amazon and Google Computer Engine. This model would be perfect for many organizations that rely on a cloud service provider for their infrastructure. According to the report of Amazon, the more numbers of organization using their services that helps them to achieve billion of revenue. Thus this pricing model is very useful in real world application. On the other side, if any organization relay on their private IT infrastructure, this model will be incorrect and can be used as reference.

$$\text{Expense}_{t_s} = P_{\text{storage}} \times S_{\text{cache}} \times t_s$$
$$\text{Save}_{t_s} = P_{\text{computation}} \times R_{\text{duplicate}} \times t_s$$

Above equation shows how to evaluate the expense of storing cache and the corresponding saved expense in computation. The details of computing above variable are as follow. The profit of storing a cache item for t_s amount of time is evaluated by accumulating the charged expenses of all saved computation task int_s. The number of the similar task that is submitted be the user in t_s amount of time. The optimal life of a cache item is the maximumt_a, such that the profit is positive. The total benefit of this scheme are that user will not be changed more and at the same time the computation time is reduced, which in turn reduce the response time and increase the user satisfaction.

VIII. CACHE COMMUNICATION

Map Cache communication

The actual designs of the Hadoop MapReduce framework have various complications. The first is, when cache request are issued by the mapper? As explained above, map cache item are determined by the data chunk and operations performed. The cache requests are sent out before the file splitting phase in order to preserve the original splitting scheme. The jobtracker which is the main controller that handle a MapReduce job sends the cache request to the cache manager. A detail list of cache description is replied by the cache manager. The input file is divided on remaining file section that have no corresponding result in cache item is done by jobtracker. That is, the job tracker uses the same file divide scheme as the same used in the cache item in order to actually use them. In this case, the new added input file should be divided into the same number of mapper task therefore it will not reduce done the entire MapReduce job down. Its result are then joined together to form an aggregated Map cache item. This is done with the help of nested MapReduce job.

Reduce cache communication

The cache request process is more difficult. The first step is to compare the cache items in the cache manager with the requested cache item. As explained in the previous section [2.2] the cache results in the reduce phase may not be directly used due to the increasing changes. As a result, identification of the overlapped of the original input files of the requested cache and the stored cache is done by the cache manager. In our primary implementation, this is done by performing a one after another scan of the stored cache item to find the one with the maximum overlap with request. By comparing request and cache item, it became easy for cache manager to identify the partitioner. The partitioner is available in request and the cache item have to identical i.e., they should use the same partitioning algorithm as well as same number of reducer. It is depicted in the figure 7. The overlapped part means the processing in the reducer could be minimized by obtaining the cached result for that part of the input. The reducer itself processes the incremental part. The final results are generated by combining both parts. The user determines the result by actual method of combination.

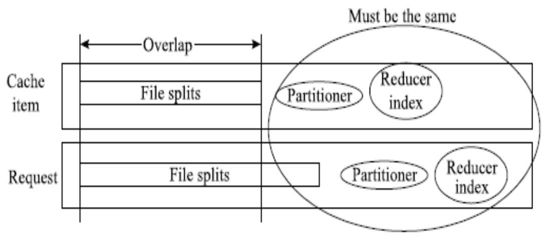

Fig 7:To compare a cache description and a cache request, the cache manager needs to check the partitioner and the reducer indexes and checking that they are the same.

IX. Conclusion

We show the outline and calculation of the data aware cache framework that requires fewer changes to the original MapReduce programming model for improving incremental processing for large data application using MapReduce model. We purposed Dache, data aware cache description scheme, protocol and architecture. This includes little changes in the input format processing and task management of the MapReduce framework. As a final output, application code requires only little modification in order to utilize the Dache. We made dache in Hadoop by extending relevant component. It eliminates the duplicate task in the experiment in the incremental MapReduce job and doesn't require any more changes in the code. In the future we plan to accept to more general scenarios and develop the scheme in the Hadoop project.

References

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 ISSN 1007-0214l l05/10l,Volume 19, Number 1, pp 39- 50, February 2014
2. Hadoop, http://hadoop.apache.org/,2013
3. D. Peng and f. Dabek,"Large Scale incremental Processing using distributed Transaction and notification", in Proc. of OSDI'2010, Berkeley, CA, USA, 2010
4. M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving Mapreduce performance in heterogeneous environments",in Proc. of OSDI' 2008, Berkeley, CA, USA, 2008.
5. 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
6. 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
7. 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] Amawon web services, http://aws.amazon.com/, 2013. [7] Google compute engine, http://cloud.google.com/ Products/computeengine.html, 2013.
8. G. Ramalingam and T. Reps. A categorized bibliography on incremental computation, in Proc. of POPL '93, New York, NY, USA, 1993.
9. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data, in Proc. of OSDI'2006, Berkeley, CA, USA, 2006.
10. S. Ghemawat, H. Gobioff, and S.-T. Leung, The google file system, SIGOPS Oper. Syst. Rev., vol. 37, no. 5, spp. 29-43, 2003.