MCache : Hadoop with Map Cache

Sayali Ashok Shivarkar
Computer Network
Sinhgad College of Engineering,Pune, Maharashtra

***Abstract*— Apache's hadoop system is pretty good and most popular for storing and processing large amount of data. Hadoop has two main systems: HDFS and Mapreduce. HDFS is uses for storing data and MapReduce is used for processing this large amount of data. An observation regarding mapReduce application is that they generate large amount of intermediate data and this big data is thrown away after processing is done. Motivated by this observation, we are introduced cache for Map task which called as MCache. In MCache, task submits their result to cache manager. Whenever new task arrives it request cache manager to find similar task, which may be save completely its execution and time also. Introducing Mcache significantly improves the completion time of Map task and also saves a significant chunk of CPU execution time.**

Keywords—Hadoop,MapReduce,Cache Manegement.

I. INTRODUCTION

Today is age of data. Data is everywhere and data is going to increase day by day. Photo and videos are uploaded in facebook and other social networking sites per day are near about 1 lakhs and more than that. This terabytes and petabytes of data need to be analyzed to know which web site is popular, which book is in demand, what kind of ads and movies appeals to people. Previous tools are become inadequate too processing this huge amount of data, so hadoop is introduced to store, analysis and processes this big data.

For many originations like Yahoo, facebook, LinkedIn, and Twitter hadoop become a core part of the computing infrastructure.

Hadoop is an open source framework for writing and running distributed applications. Hadoop is based on two key technologies HDFS and MapReduce. To process large amount of data in serial manner is not possible so MapReduce process that data in parallel to accomplish work in less time which is main objective of hadoop. MapReduce requires special file system that is Hadoop Distributed File System.

MapReduce is basically divided into two part: Map and Reduce. In map phase master node takes the input and divided into small sub-problems and it takes input as key and value pair. In Reduce phase master node collects the answer to all sub-problems and combines them in such way to from a output which is answer to the problem it was originally trying to solve.

In this process there are potential duplicate computations begin performed and MapReduce does not have any mechanism to find such duplicate computation. One way to avoid processing of such duplicate task is to introduce cache which keeps track of all execution, which not only help to find duplicate computation but also saves execution time.

II. BACKGROUND

a. MapReduce :

MapReduce is a software framework for processing a big data set in distributed fashion. The core idea behind MapReduce is mapping your data sets into <key, Value> pairs and then reducing overall pair to the single which is the result we are trying to obtain. The overall data need to be mapped in key and value pair. Key and value may be any type of data: string, integer etc. [2]
Fig 1. Map/Reduce framework shows working of Map and Reduce. First it split the input into segment, passing each segment to a different machine. Each machine execute map task on data i.e. key and gives value. For e.g if we wanted word count in a text file then our <key, value> will be <word, count>

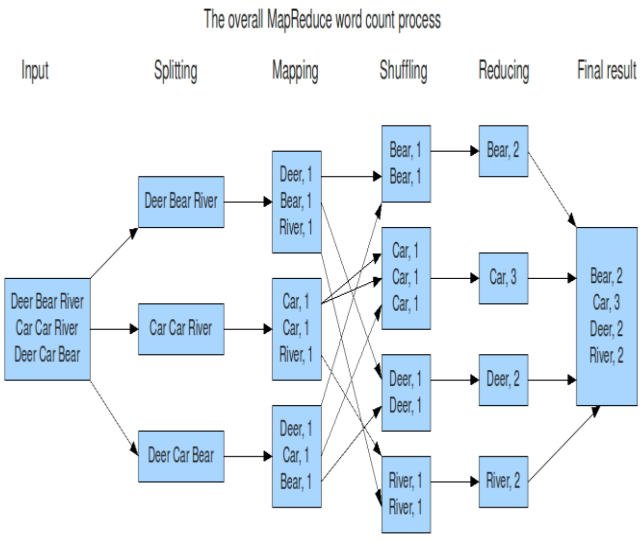

Fig 1. Working of Map/Reduce [1]

Our map task generate a <word, count> pair for each word in input stream but it does not aggregate word count, reducer does this job. Emitted <key, value> pairs are then "shuffled",

which basically means that pairs with the same key are grouped and passed to a single machine, which will then run the reduce script. But output of map phase does not store anywhere, after processing of map task it thrown away.
The reduce script takes a collection of <key, value> pairs and “reduces” them according to the user-specified reduce script.
In our word count example, we want to count the number of word occurrences so that we can get frequencies. So in first phase we have text that is divided into three parts and given to three mappers which generates word and count as output then that is shuffled and given to reducer and then reducer gives final result.

III. DESIGN OF THE SYTEM

When MapReduce process computation which is divided into map and reduce task. In execution of map task a very large intermediate data is generated and which is thrown after a processing is done. Map cache refers to the intermediate data. A piece of cached data is stored in the distributed file system. The content of the cache item is described by three tuples:{file name, Operation, Resultfile}. The file name is the name of the file on which operation is performed, operation specifies which operation is done on file and result file contain result of the operation. For example, in the word count application mapper / reducer emit {word ,count} which gives word and count of word in that file. MCache store this file in cache for next use as { word .txt , item count, resultcount.txt}.

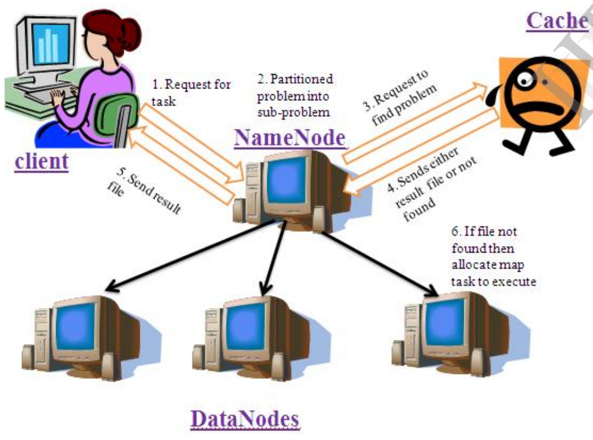
Cache
client
1. Request for task
2. Partitioned problem into sub-problem
3. Request to find problem
NameNode
4. Sends either result file or not found
5. Send result file
6. If file not found then allocate map task to execute
DataNodes

Fig2. Design of MCache System

As shown in fig 2 suppose client request NameNode to execute problem statement NameNode divides that problem statement into number of sub-problems and then it request to cache find sub-problem file name in cache table with operation. If cache finds file name and operation performed on that file in cache table it directly returns the result file name to the NameNode . NameNode find resulted file and directly gives to Reducer for next execution. If file name and operation performed on that file is not found then all sub-problem is given to DataNodes for execution then there results are given to reducers. Mapper and Reducer nodes record there result into local storage. When operations completed, the items are forwarded to cache manager. Cache manager stores file name, operation performed on that file and resultant file name in cache table. The cache item should be put on the same machine as the worker process that generates it because this improves data locality mechanism.

If cache manager is not able to find exact match of file name and operation performed on that file then after all processing is finished it copies file name and operation performed on that file with result file in cache table.
The Cache manager need to decide how much time item can be kept in cache table. If item kept in cache for infinite amount of time then it may happen that older result will be used for some reducer task.
Here we allocate fixed storage size for storing cache items. If there is no space in cache to store new result then old cache item are deleted for storing new one.

IV. PERFORMANCE EVALUATION

A. Experiment Setting :

Hadoop is running in pseudo-distributed mode on a server that has an 8-core CPU, each core running at 3GHz, 16GB memory, and a SATA disk. The number of mappers is 12 in all experiments, the reducers’ count varies. We use word count program to know the speedup of hadoop. Word-count counts the number of unique word in large input file.

B. Result:

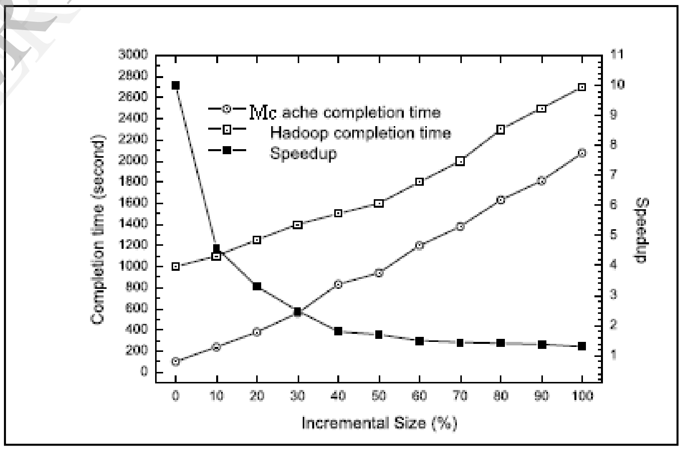

Fig 3. Speed up of Mcache over the hadoop and their completion time of word-count program.

As shown in fig 3. Completion time and speed up of hadoop using Mcache is increased as compare to hadoop completion time and speed up.

V. CONCLUSION

We present the design of Mcache which eliminate all duplicate task and increase speed up and completion time of hadoop . In future we are trying to implement reduce cache as well.

REFERENCES

1. Hadoop. http://hadoop.apache.org/.
2. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. of ACM*, 51(1):107–113, January 2008.
3. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.