Enhancing the Mapreduce Using Cache in Hadoop for Big Data Analytics

J. Janani#1, K. Kalaivani#2

1M.E student, Department of Computer Science, Arasu Engineering College, Kumbakonam,
2 Assistant Professor, Department of Computer Science, Arasu Engineering College, Kumbakonam.
1jananijagadeesan1991@gmail.com
2kalai4best@gmail.com

Abstract----**Hadoop is an open-source framework that supports the processing of massive volume of datasets in a distributed environment. Big data and hadoop are the catch-phrases for describing the storage and processing of huge amount of data, where zeta bytes of unstructured data and updates are constantly arriving, that cannot be mined efficiently by the traditional tools and methods. i^2 MapReduce is the first MapReduce-based solution that efficiently supports incremental iterative computation, which is widely used in data mining applications. i^2 MapReduce utilizes key-value pair incremental processing rather than task level re-computation. In this paper, an extension to MapReduce using in-memory for mining big data has been proposed. Compared with the work of i^{2}MapReduce , MapReduce using in-memory performs key-value pair level processing in map phase based on the mining results of iterative algorithms, cache the mapped data in the buffer that reduces the I/O workload of the reducer phase. Mapreduce using cache is enhanced in hadoop environment and uses the hadoop cache as a buffer to store the intermediate data. The evaluation of MapReduce task for the mobile datasets using the iterative algorithm and cache memory has been retrieved fast and executed in time.**

Index terms: MapReduce, bigdata, incremental processing, in-memory.

I. INTRODUCTION

A. Big Data

Big data is an evolving term that describes any voluminous amount of structured, semi-structured and unstructured data in areas including internet search, social network, finance, business informatics, health care, environment and education. Mining of such big data are become popular in order to gain better performance and quality services. Big Data 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. Big Data is mostly used in Retail, Banking, Government, Real estate, Science and research sectors. This helps in decision making, cost/time reduction, market analysis etc.

B. Hadoop

It is an open source platform for storage and processing of diverse data types that enables data driver enterprises to rapidly derive the complete value from all their data.The original creators of Hadoop are Doug cutting (used to be at Yahoo! now at Cloudera) and Mike Cafarella (now teaching at the University of Michigan in Ann Arbor). Doug and Mike were building a project called "Nutch" with the goal of creating a large Web index. They saw the MapReduce and GFS papers from Google, which were obviously super relevant to the problem Nutch and were trying to solve. They integrated the concepts from MapReduce and GFS into Nutch; then later these two components were pulled out to form the genesis of the Hadoop project. The name "Hadoop" itself comes from Doug's son, he just made the word up for a yellow plush elephant toy that he has. Yahoo! hired Doug and invested significant resources into growing the Hadoop project, initially to store and index the Web for the purpose of Yahoo! Search. That said, the technology quickly mushroomed throughout the whole company as it proved to be a big hammer that can solve many problems. In the recent years, lots of frameworks [1], [2], [3], [4], [5] have been developed for big data analytics. From those frameworks, MapReduce [1], is the most widely used framework for the simplicity, generality and maturity with open source implementation such as hadoop. In this paper, we improve the efficiency of MapReduce using the cache memory.

Big data is data, when grows as the technology advances constantly. As new data and updates are constantly arriving, the mining of those data are complex by the traditional tools and methods. Refresh periodically to store the mining results up-to-date. E.g., Page Rank algorithm [6] estimates the ranking scores of the web pages based on the search of the web contents. For every search, web graph structures are constantly evolved. The results of page rank become stale and it decreases the quality of the web content search. Therefore, it is important to refresh the Page Rank estimation in a regular interval. Incremental processing is an efficient approach to refresh those mining results. If the size of the input is large, the processing of those big data is very expensive.

C. MapReduce Overview

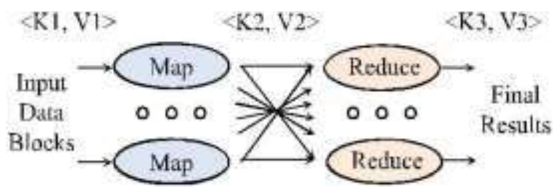

Fig 1: MapReduce workflow

The MapReduce framework has two parts: A function called Map, which takes a kv-pair <k1,v1> as input and computes zero or more intermediate kv-pairs <k2,v2>s. Then all <K2,v2>s are grouped by K2. A function called Reduce, which takes a K2 and list of {v2} as input and computes the reduced final output kv-pairs <k3, v3>s explains in fig 1. A MapReduce is heart of Hadoop, it usually reads the input data of the MapReduce function and computes the final results to the HDFS (Hadoop Distributed File System), which divides a file into equal-sized of 64MB blocks in a cluster environment.

D. Distributed Cache

Distributed Cache is a facility provided by the Map-Reduce framework to cache files (text, archives, jars etc.) needed by applications. Applications specify the files, via urls (hdfs:// or http://) to be cached via the jobconf. The Distributed Cache assumes that the files specified via urls are already present on the File System at the path specified by the url and are accessible by every machine in the cluster. The framework will copy the necessary files on to the slave node before any tasks for the job are executed on that node. Its efficiency stems from the fact that the files are only copied once per job and the ability to cache archives which are un-archived on the slaves. Distributed Cache can be used to distribute simple, read-only data/text files and/or more complex types such as archives, jars etc. Archives (zip, tar and tgz/tar.gz files) are un-archived at the slave nodes. Jars may be optionally added to the classpath of the tasks, a rudimentary software distribution mechanism. Files have execution permissions. In older version of Hadoop Map/Reduce users could optionally ask for symlinks to be created in the working directory of the child task. In the current version symlinks are always created. If the URL does not have a fragment the name of the file or directory will be used. If multiple files or directories map to the same link name, the last one added, will be used. All others will not even be downloaded. Distributed Cache tracks modification timestamps of the cache files. Clearly the cache files should not be modified by the application or externally while the job is executing.

II. RELATED WORK

Iterative processing. A number of distributed frameworks have recently emerged for big data processing [2], [4], [5]. We discuss the frameworks that improve MapReduce. HaLoop [9], a modified version of Hadoop, improves the efficiency of iterative computation by making the task scheduler loop aware and by employing caching mechanisms. iMapReduce [10] supports iterative processing by directly passing the Reduce outputs to Map and by distinguishing variant state data from the static data.
iMapReduce allows users to specify the iterative operations with map and reduce functions, while supporting the iterative processing automatically without the need of users' involvement. More importantly, iMapReduce significantly improves the performance of iterative algorithms by reducing the overhead of creating a new task in every iteration, eliminating the shuffling of the static data in the shuffle stage of MapReduce, and allowing

asynchronous execution of each iteration, i.e., an iteration can start before all tasks of a previous iteration have finished. i2MapReduce[11] improves upon these previous proposals by supporting an efficient general purpose iterative model. Pregel [3] follows the Bulk Synchronous Processing (BSP)model. The computation is broken down into a sequence of super steps. In each super step,a Compute function is invoked on each vertex. It communicates with other vertices by sending and receiving messages and performs computation for the current vertex.This model can efficiently support a large number of iterative graph algorithms. Compared to i2MapReduce, the BSP model in Pregel is quite different from the MapReduce programming paradigm. It would be an interesting future work to exploit similar ideas in this paper to support incremental processing in Pregel-like systems. Incremental processing for one-step application. Besides Incoop [7], several recent studies aim at supporting incremental processing for one-step applications. In contrast, i2MapReduce exploits a fine-grain kv-pair level re-computation that is more advantageous. Incremental processing for iterative application. In comparison, we extend the widely used MapReduce model for incremental iterative computation. Existing Map- Reduce programs can be slightly changed to run on mapreduce using cache for incremental processing. Incoop detects changes to the inputs and enables the automatic update of the outputs by employing an efficient, fine-grained result reuse mechanism. To achieve efficiency without sacrificing transparency, they adopt recent advances in the area of programming languages to identify systematically the shortcomings of task level memorization approaches, and address them using several novel techniques such as a storage system to store the input of consecutive runs, a contraction phase that make the incremental computation of the reduce tasks more efficient, and a scheduling algorithm for Hadoop that is aware of the location of previously computed results.

III. PROPOSED SYSTEM

A. Problem Statement

We consider the MapReduce approach in big data processing, where the data sets are stored in the cluster environment. MapReduce approach is used for mining big data. In optimized MapReduce using cache for big data analytics, it extracts the URL from online dynamic websites and it converts it into the data. In the Map phase, it Map the data based on the <key,value> pair and store it in cache. The cached data are partitioned and then combine in this phase. In the reduce phase, the output data from map phase are merged and sorted and then given as input to the reducer to get an optimized output data. MapReduce based on cache reduces the I/O workload in the reduce function.

B. Proposed Methodology

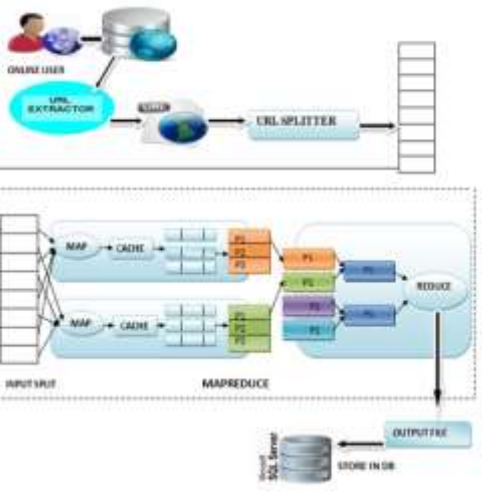

Fig 2: Architecture diagram

Mapreduce is the framework for the processing of big data; we propose an extension to i2mapreduce based on the in-memory concept. First of all, mine the URL of the mobile datasets from the dynamic websites using the URL extractor. After the extraction, the URL are sent to the URL splitter, it splits the URL and converts it into content based on the annotations. The splitted data are mapped with the key-value pair and stores it in the defined cache memory. The cached items, (i.e) the intermediate data from multiple maps are managed by the cache manager. Then the cached data are partitioned and combined based on the key-value pair, merges the data based on the partition involved in the multiple map phase. Finally the cached intermediate data are sent to the reduce phase, the process reduces the i/o workload of the reducer phase. The reducer decreases the size of the data based on the intermediate key. The information needed by the user is extracted from the dynamic mobile datasets by the mapreduce process. The above process is clearly illustrated in the fig 2.

III CACHE DESCRIPTION

Input: First the input data are split into fixed number of pieces and then they are feed to different workers (data nodes) in the mapreduce environment. Records are individual data items. Each worker process the input file as per the user program.

Map phase: In this phase, each input split is fed to the mapper who has the function map (). This map () has the logic on how to process the input data. For example, map () is containing the logic to count the occurrence of each word and each occurrence is captured and arranged as (Key, value) pairs. After processing the intermediate results are stored in the data node's hard disk.

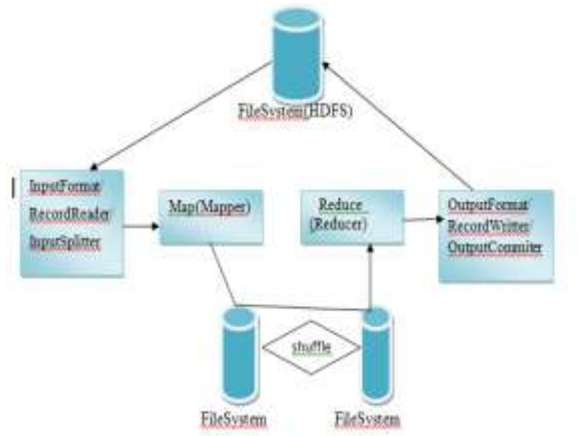

Fig 3: Data flow in hadoop system

Cache management phase: Cache manager works as a centralized system. All the unique input and output data performed by clients are feed in to the cache manager. The data in cache is stored as a log which contains the input and the place where the output is available. Each client checks the cache before it starts the functioning. If the cache contains that task then the client machine can easily retrieve information from it, else the cache accept data from the client. Cache prevents the occurrence of repeated tasks. Thus it decreases the Processing time of system.

Cache request and reply protocol: We use cache request and reply protocol to get the results that are stored in data nodes. Before processing the splits, the data node sends the request to Cache Manager. All the unique input and output data performed by clients are feed to the cache manager. The data is stored as a log in cache which contains the input and the place where the output is available. Each client checks the cache before it starts the functioning. If the cache contains that task then the client machine can easily retrieve information from it, else the cache accept task from the client. If data is already processed, the Cache Manager sends the positive reply to the data node. Otherwise send the negative reply. If negative reply obtained, the data node do the process on the split file. If positive reply obtained, the data node need not process the splits. So, no need to process the repeated data. Cache Manager ensures the repeated input split files need not process more than one time. Finally all the intermediate files are reduced by data node and the final result is stored in Name node.

Reduce phase: In this step, for each unique key, the framework calls the application's Reduce () function. The Reduce can iterate through the values that are associated with that key and produce zero or more outputs. In the word count example, the input value is taken by reduce function, sums them and generates a single output of the word and the final sum. The output of the Reduce is writing to the stable storage, usually a distributed file system.

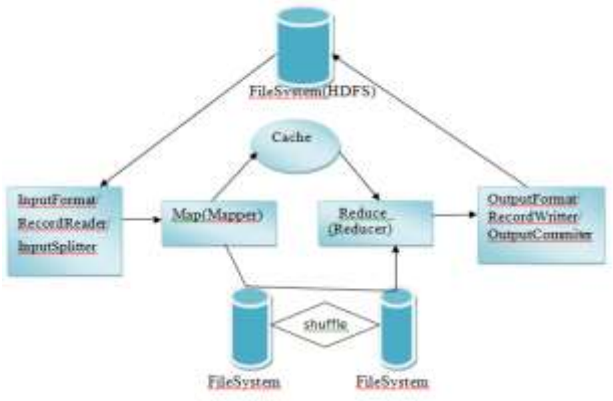

Fig 4: Data flow in proposed System

Steps involved in proposed system

1. Preprocessing File- In file preprocessing stop words are removed and stemming is performed so that proper 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. So file vector manages above details.
3. Create Signature- To find similar file it should be compared with existing files available among the millions of files to make comparison process faster. To create signature bit vector is used and initialized to zero first then hashed with file vector so that decision will be taken regarding whether existing file to be incremented or decremented.

4. Use Locality sensitive hashing to find nearest neighbor- In large clustering environment to compare file signature; locality sensitive hashing technique is used to ensure that only nearest neighbor need to be checked to place file [8].
5. Store file with related files- Name Node maintains subclustertable which store subclusterid and file placed on that cluster and if subclusterid is not found then new subcluster will be created.

The various data structures implemented are locality sensitive hashing function, subclustering and storing mapping information, cachetable, storing intermediate result in the form of either array of structure or linked list or object of classes.

V. EXPERIMENTAL RESULTS

There are several steps for installing and configuring Hadoop. First install the following software, and then configure hadoop.

☐ VMware Player 12
☐ Create new virtual machine and install ubuntu OS
☐ Install Java SE 7
☐ Install Eclipse Juno Release 1.0
☐ Install Apache Hadoop 2.0.

Fig-5. Hadoop Window.

Below is the Hadoop console output, this actually splits the tasks into several records and allocate it to the available data nodes. This console output will update the status of Map () and Reduce () and the task completion status.

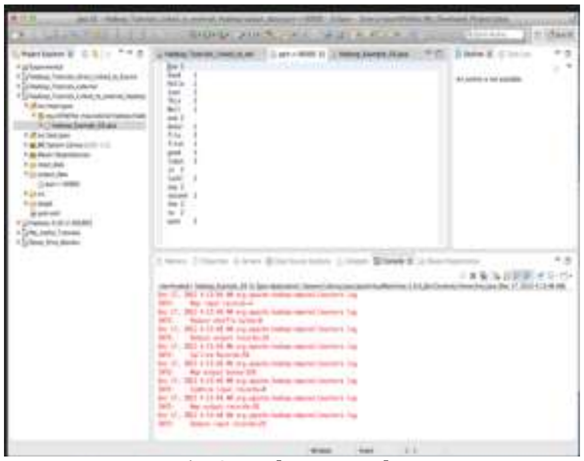

Fig-6. Hadoop console

In the experimental results, the existing method such as imapreduce and i2mapreduce with the proposed method mapreduce using cache are compared with the computation time in the graph shown in fig4.

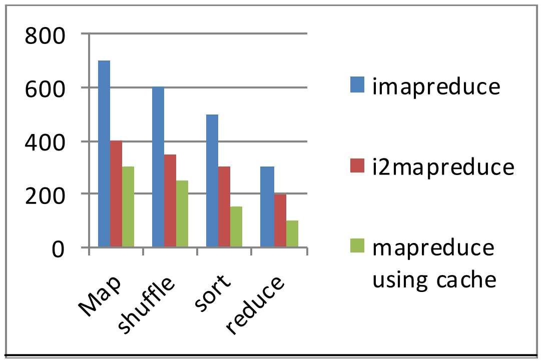

Fig 7: runtime of individual stages

This graph shows comparative based on the retrieval of data in their corresponding computation time. Based on the comparison mapreduce using the cache increases the efficiency and retrieve the abundant information thrown away after the map phase in the mapreduce framework. It takes less amount of time for the big data processing.

VI. PERFORMANCE EVALUATION

Overall performance evaluation of the mapreduce using cache is shown in the fig 5. Mapreduce using cache memory increases the performance in each phase such as map, shuffle, and sort and reduce. Performance is based on the time to complete the map and reduce task. It takes less amount of time to complete the job compared with imapreduce and i2mapreduce.

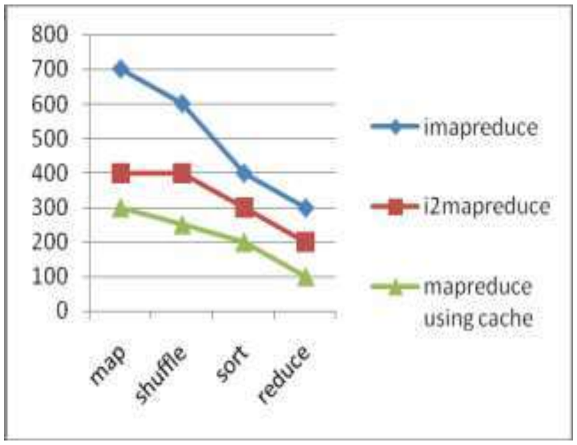

Fig 8: performance evaluation graph.

VIII. CONCLUSION

We have described mapreduce using cache in hadoop, a mapreduce framework for big data processing. Mapreduce using cache reduces workload and increases the efficiency based on the individual phases and it reduces the runtime in each phases of mapreduce framework. Hadoop framework has distributed cache to do mapreduce jobs in order to increase the efficiency and get the reduced output in optimized time.

REFERENCES

[1] J. Dean and S. Ghemawat, Mapreduce: simplified data processing on large clusters in proc. 6th conf. Symp. Opear. Syst.Des. Implementation, 2004,p. 10.

[2] R. Power and J. Li, Piccolo: Building fast, distributed programs with partitioned tables.in proc. 9th USENIX Conf. Oper. Syst. Des.Implementation, 2010.pp, 1-14.

[3] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, Pregel: A system for large-scale graph processing, in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, pp. 135–146.

[4] S. R. Mihaylov, Z. G. Ives, and S. Guha, Rex: Recursive, deltabased data-centric computation, in Proc. VLDB Endowment, 2012, vol. 5, no. 11, pp. 1280–1291.

[5] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J.M. Hellerstein, Distributed graphlab: A framework for machine learning and data mining in the cloud, in Proc. VLDB Endowment, 2012, vol. 5, no. 8, pp. 716–727.

[6] S. Brin, and L. Page, The anatomy of a large-scale hypertextual web search engine, Comput. Netw. ISDN Syst., vol. 30, no. 1–7, pp. 107–117, Apr. 1998.

[7] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, Incoop: Mapreduce for incremental computations, in Proc. 2nd ACM Symp. Cloud Comput., 2011, pp. 7:1–7:14.

[8] Y. Zhang, S. Chen, Q. Wang, and G. Yu, i2mapreduce:Incremental mapreduce for mining evolving big data, CoRR, vol. abs/ 1501.04854, 2015.

[9] Y. Bu, B. Howe, M. Balazinska and M.D Ernst, Haloop: Efficient iterative data processing on large clusters, in proc, VLDB Endowment, 2010, vol. 3,no.1-2, pp.285-296.

[10] Y. Zhang, Q.Gao, and C. Wang, imapreduce: A distributed computing framework for iterative computation, J. Grid comput, vol. 10, no. 1,pp.47-68,2012.

[11] Y. Zhang, S. Chen, Q. Wang, and G. Yu, i2mapreduce:Incrementalmapreduce for mining evolving big data, CoRR, vol. abs/1501.04854, 2015.

ABOUT THE AUTHOR

J. Janani received her **B.E** degree in Computer Science and Engineering from Loyola institute of technology, Chennai. Currently pursuing **M.E** degree in computer science and engineering, at Arasu Engineering College, kumbakonam. Her area of interests includes Medical Image Processing, Data mining and Big data analytics.

K. Kalaivani received her **M.E** degree in Computer Science and Engineering from Oxford Engineering College, Trichy. Currently working as assistant professor with 10 years of experience in Arasu Engineering College. Her area of interests includes Networking,Wireless Sensor Network, Big data analytics.