ISSN: 2277-9655
(I2OR), Publication Impact Factor: 3.785

IJESRT
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A NOVEL APPROACH TO IMPROVE PERFORMANCE OF HADOOP USING EFFICIENT DATA AWARE CACHING FOR BIG DATA APPLICATION

Shakil B. Tamboli [1], Smita Shukla Patel [2]
MEIT Student, IT Department, Smt.Kashibai Navale College of Engineering, Pune,India
Assistant Professor, IT Department, Smt.Kashibai Navale College of Engineering, Pune,India

ABSTRACT

Continuous generation and explosion of large amount of data has focused the attention of researchers and scholars from industrial, academic area to capture the potential business opportunities and academic benefits from Big Data. 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.

There are various applications which need recursive operations on input data. 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.

KEYWORDS: Efficient Data Aware Caching, Value Degree, File Vector, Locality Sensitive Hashing, Create Signature, Cache Coherence

INTRODUCTION

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 [1].

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 [14]. 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 [3].

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 [16].

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 [15].

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 [2].

In this paper new proposed architecture modified Hadoop marginally to add in-memory caching with custom data structure. A determined worker process is accepted to confirm the in-memory cache is also in the same process which did the map operations, i.e. worker processes are not killed after Job complete for easier sharing of the in-memory caches.

The remaining paper is written as below. The related work is described in section II. Section III presents proposed system and components role for improving performance of Hadoop jobs using an efficient data aware caching for big data application. Section IV contains performance evaluation, implementation and experimental settings details. Section V highlights results and discussions based on it. Conclusion and future scope is highlighted in section VI.

RELATED WORK

Active research to improve performance in data intensive applications with MapReduce is widely adopted in scientific and research domain. In paper [3] the design is proposed based on concept cache request and reply protocol and provisioning cache layer to extend the MapReduce framework for efficiently identifying and accessing cache items in a MapReduce task. In this system cache manager customizes the indexing of data objects for applications to describe their operations and contents of their partial results.

C-Aware cache management and storage algorithm proposed in paper [4]. The speed of cache media and network load conditions considered for establishing this strategy. The system analyzes historical information of accessed cached data from network and forecasts the future access to the cache and storage server performance based on historical information.

In paper [5] collaborative caching approach adopted to lower job execution times on DataNode. In collaborative caching mechanism cache distributed over the clients, dedicated servers or storage devices form a single cache to fulfil the requests. Effective data local jobs techniques is achieved in this arrangement. Local as well as remote data was cached on DataNodes and fed as an input to MapReduce jobs. New cache layer formed along with cache of NameNode, DataNode, Remote DataNode and disk. The global cache is formed from all the participating DataNode's machines. NameNode is the central coordinator of this global cache, but permits remote caching decision to be taken by Cache Manager on DataNodes. New protocols for DataNodes are introduced in the system. This new methodology permits efficient use of resources in the form of addition of more slots instead of increasing the number of nodes to improve performance.

Smart cache solution from paper [6] implements two phase structure, but takes care to avoid unnecessary searching overhead in the second phase. SmartCache's success observed from key principle which states that if one itemset is very close to, although not above, the threshold, it might exceed the threshold in other splits. This principle states that it is beneficial to store more itemsets in the second phase to avoid repetitive counting. SmartCache's first phase is presented as a cache layer which saves the counting information generated from phase 1 to HDFS files, which are used by phase 2 to speed up its execution. And the second phase is designed as an online analyzer module to decide the cost effective number of itemsets' counting information to be saved.

HaLoop system is presented in paper [7] to efficiently handle the iterative nature of applications. HaLoop insisted on two simple intuitions for better performance by MapReduce. In first iteration a MapReduce cluster can cache the invariant data and then reuse that data in further iterations. And in second stage a MapReduce cluster can cache reducer outputs making checking for a fix point more efficient, without an extra MapReduce job.

The PACMan system states that when multiple jobs are processed in parallel, job's running time can be decreased only when all the inputs related to running a job are cached [8]. Therefore cache all the input data related to that particular job or do not cache the input data at all. PACMan caching service coordinates access to the distributed caches [8]. For removing the inputs which are least used, LFU-F algorithm, LIFE sticky policy proposed. The system

has a new parameter called wave-width of the job reffered for the total tasks which may be executed in simultaneously at a time.
The Dynamic Caching [9] mechanism allows simultaneous access to data. The system is implemented with the help of Memcached open source cache technologyand used in the form of set servers which stores the mapping of block-id to datanode-ids.

PROPOSED SYSTEM

Proposed framework of system

The proposed framework presents an efficient data aware caching technique with the help of custom data structure and value degree cache replacement algorithm to manage storage space allocated to cache. A high-level architectural diagram is shown in figue 1. The system accepts the source input from cache in which cache item is stored by file name and operations applied on the input. As well as this cache item produced by the workers in the map phase is indexed properly. Reduce phase considers the partition operations applied on the output in the map phase as well as reducers utilize the cached results in the map phase to accelerate the execution of the MapReduce job. Efficient data aware caching is implemented in the Hadoop by utilizing the related components. The implementation performs a non-intrusive approach, so it only needs least changes to the application code.

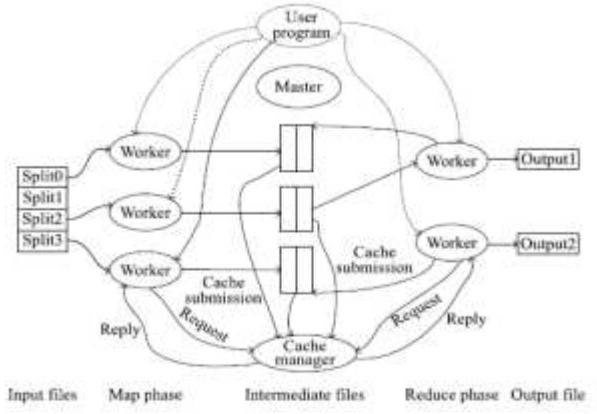

Figure 1: Proposed system architecture for caching

Proposed Methodology of system

Efficient data aware caching is one such mechanism in which the cache distributed over the clients or dedicated servers or storage devices form a single cache to serve the requests. This technique facilitates more data local jobs. The local as well as remote data is cached on DataNodes in such a way that introduced a new layer of hierarchy called NameNode cache, DataNode's cache, Remote DataNode's cache and the disk.

A single cache or global cache is formed from all the participating DataNode's machines. NameNode is the central coordinator of global cache, allows taking decisions of remote caching by Cache Manager on DataNodes with new DataNode protocols. The new system thus makes efficient use of resources just by adding new slots from TaskTracker instead of increasing the number of nodes to improve performance.

Caching of data is made faster and easier by the reference caching technique means not evicting the old cached data. A HDFS block is a collection of two files namely meta file and block file. Meta file means checksum value of the data and block file means actual data. If these file references are cached, it helps in locating the meta files faster for checksum checks, faster caching of data from disk to memory since not much time is spent in searching for files when data stored is about petabytes. Thus the mechanism reduces the overall time.

Value degree cache replacement policy is used in order to maximize cache hit ratio and to improve efficiency.
The intermediate data created by worker nodes execution of a MapReduce task is stored in cache and this part of cached data is then stored in a Distributed File System (DFS). The content of a cache item is well defined by the original data and the operations applied. Normally, a cache item is described by a 2-tuple i.e. Origin and Operation.

Execution flow

The data flow for Hadoop job is shown in figure 2. The jobtracker is responsible for scheduling the job to run by assigning map and reduce tasks available on task trackers. If the data is in HDFS reading requires network communication with the namenode. The map tasks de-serialize the input data to generate a stream of key/value pairs that is passed into the mapper. Once map output has been flushed out to disk, reducer tasks start fetching their input data. This requires disk and network I/O.

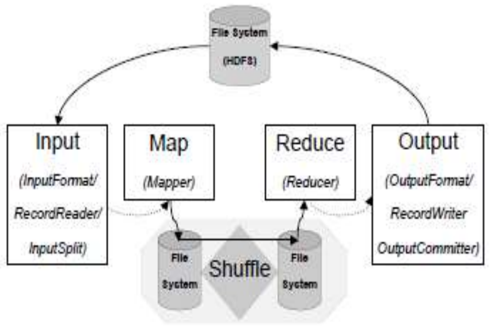
File System (HDFS)
Input (InputFormat/ RecordReader/ InputSplit)
Map (Mapper)
Reduce (Reducer)
Output (OutputFormat/ RecordWriter OutputCommitter)
File System
Shuffle
File System

Figure 2: Data flow in a Hadoop system

The data flow in proposed system is shown in figure 3. The cache in proposed system is mostly transparent to the user, as it is intended to work with unmodified Hadoop jobs.

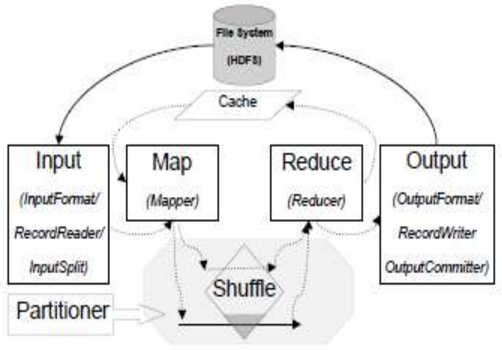
File System (HDFS)
Cache
Input (InputFormat/ RecordReader/ InputSplit)
Map (Mapper)
Reduce (Reducer)
Output (OutputFormat/ RecordWriter OutputCommitter)
Partitioner
Shuffle

Figure 3: Data flow in proposed system

The proposed system provides an in-memory key/value cache between multiple jobs in a job sequence for effective communication. It caches the key/value pairs in memory (related with the input file name) before passing to mapper. Thus in a subsequent job, when the same input is requested, this data will be directly obtained from the cache.

The engine makes effort to avoid the time and space overhead of (de)serialization by locally shuffling data. The system allows permits programmer to control keys partitioning among reducers. The default implementation uses a hash function to map keys to partitions. The system provides partition guarantee by mapping partitions to places in deterministic strategy.

Models created in proposed system

Following models bring system in total control.

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 [11] 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 [12].
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.

The data structures applied in proposed system are Data structure for locality sensitive hashing function, Data structure for SubClustering, Data structure for storing mapping information, Data Structure for CacheTable, Data Structure for storing intermediate result. All above structure will be either array of structure or linked list or object of classes. The internal data structures will be used to store result obtained by map task it need to store locally because it improves data locality.

Improvements in Hadoop
The system proposed, implemented and integrated with Hadoop works for improving performance. Thus it proved as successful in lowering job execution times in the overall system. In traditional system data is issued from disk reducing the overall performance. Therefore an attempt is made to cache data and issue from cache like remote caches. The contributions added to improve the system are basically remote memory caching, more data local jobs and reference caching. In remote memory caching, caching of input data at the DataNode level lowers job execution time. A distributed cache structure is adopted so that DataNodes caches are maintained by Cache Managers. 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.

Mathematical Model of proposed system
In mathematical model criteria considered is as; the k tuples are selected and cached in the cluster, where the value of k is belongs to the size of cache. If there are n cache nodes and the average size of cache on each node is a, then the total cache size S in the cluster will be

$$S = n \times a \qquad (1)$$

Let the size of tuple be l and k is computed as:

$$k = S / l \qquad (2)$$

Traditional cache updating algorithms consider only few factors hence are inefficient while replacing cached data. Hence the approach is designed to estimate *the value degree Value i* of a cache tuple as per formula below.

$$Value\ i = Fi \times Ti / (Tci - Tli) \qquad (3)$$

The variables used in formula are Fi is frequency of tuple i accessed, Ti is delay time of fetching tuple i from the disk, Tci is current time and Tli is the last access time. From Formula 3, it is perceived that *Value i* increases as the access frequency Fi and delay time Ti increase and decreases as the interval of fetching data (Tci-Tli) increases. The following updating algorithm is based on the concept of *value degree* formulated in Formula 3.

Practically modification in stored data is a real situation, to timely cache the aggregated data it needs to regularly update the cache. During the update, the last *n* tuples ranking with the *value degree* computed with Formula 3 are replaced. This operation is executed during the process of periodic update. Figure 4 presented the algorithm for updating cache data.

```
Algorithm: ALG—update cached tuples
Input:aggregation query Q;
Output:query result R;
    UpdateCache(Q)
    while hasNext[cache] is not empty
      if key[Q]==key[cache(i)]
        R ← cache(i)
        Getback(R)
        F[cache(i)] ← F[cache(i)]+1
        Tl[cache(i)] ← currentTime
        Return
    while hasNext[file] is not empty
    if key[Q]==key[file(i)]
        R ← file(i)
        Getback(R)
        DeleteFromCache(random(min(Vi)))
        cache(i) ← InsertIntoCache(file(i))
        F[cache(i)] ←1
        Tl[cache(i)] ←currentTime
        Return
    End
```

Figure 4: Algorithm for updating cached data

Cache coherency is another characteristic of proposed system [10]. When node is in failure state then one possible solution is to cache the same data on a backup node and put the backup node into the cluster as a standby. But to remove redundancy of nodes in proposed system, a flag is added in the aggregation result files to label whether a tuple has already been cached. So if a node is failure, it is necessary to find the backup of data files on the failed node in the cluster and re cache it according to the flag. In this way, the cached data can be recovered in a short time.

IMPLEMENTATION

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.

EXPERIMENTAL SETTINGS

The experimental setup configured for the proposed system is a machine with 3 cores CPU, each core running at 2.10 GHz, 3GB memory, and a SATA disk along with installation of Ubuntu operating system, Hadoop 2.0 framework, Java 6 and Net beans editor. The number of mappers, reducers and replication factor is auto set by Hadoop framework in the experiment. The application to benchmark the speedup of efficient data aware caching over Hadoop is word count. It counts the number of unique words in large input text files. It is an IO intensive application requires loading and storing a sizeable amount of data during the processing. The inputs of application are the files varing in sizes from 1MB to 10MB.

RESULTS AND DISCUSSION

The results obtained from proposed efficient data aware caching system have shown remarkable improvements. The execution time differs between default Hadoop and proposed system as file size increases. Issuing data from cache is faster than disk as requests are served from references. If the blocks are found in cache (local or remote) instead of accessing from disk, the overall execution time declines and observed in a larger difference between Hadoop and proposed efficient data aware caching system. Another important parameter decreases the overall job execution time due to job being executed as data local tasks.

Data is appended to the input file. The size of the appended data varies from 1MB to 10MB. The response time of proposed system is shown in figure 5 and observed that with increase in file size, the map reduce timings have decreased as the system has caching provision.

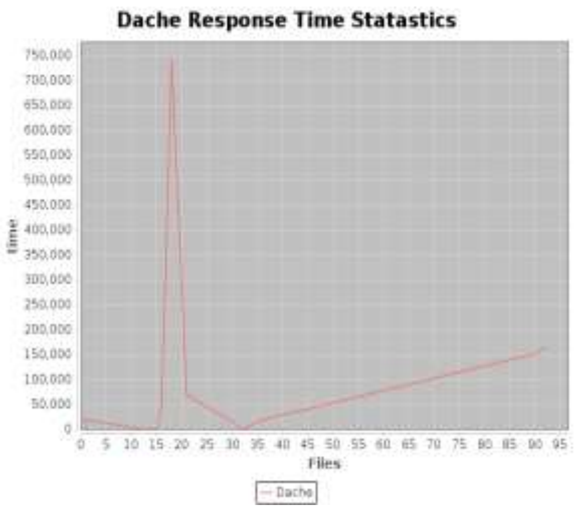

Figure 5: Response time of proposed system for word count application

Similarly it is observed in figure 6 that response time of Hadoop increases as file size increases due to no caching presense in it.

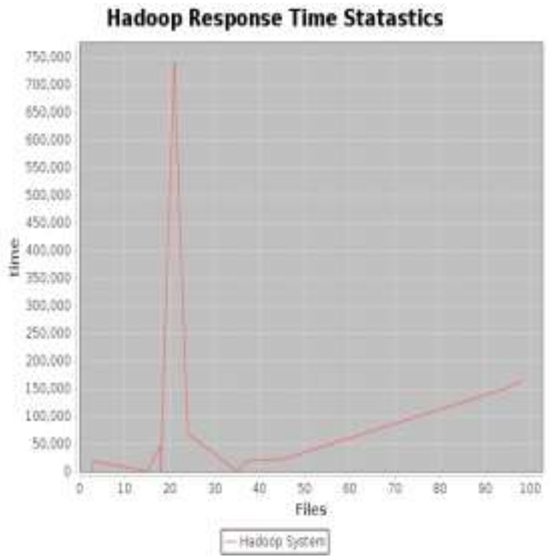

Figure 6: Response time of Hadoop for word count application

The completion time of jobs executed by Hadoop and proposed system having caching mechanism are shown in figure 7. And from graph it is observed that the job takes substantial less time to execute with proposed system than Hadoop.

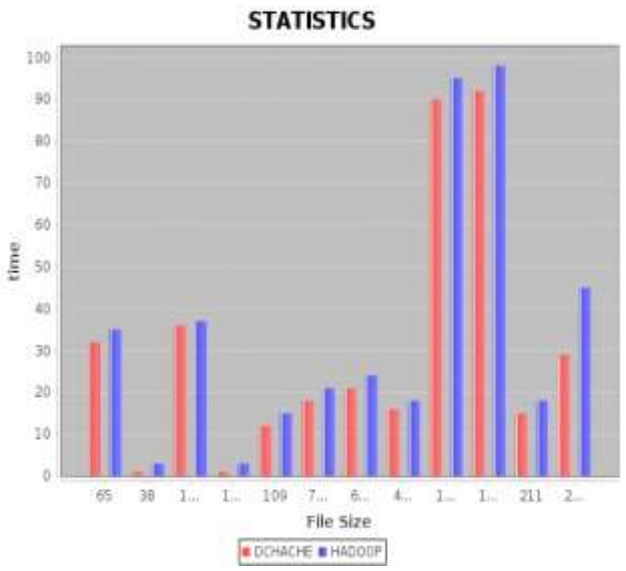

Figure 7: Completion time of Proposed system and Hadoop for word count application

The figure 8 shows the CPU load against file size. It is observed that the proposed system which has caching mechanism incorporated gives less load to CPU as compared with Hadoop system.

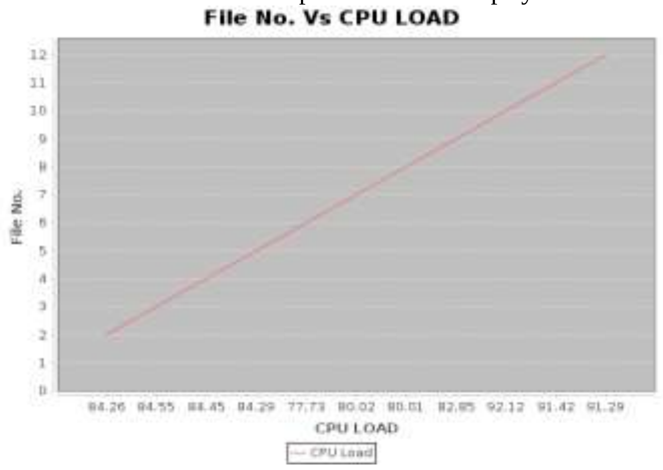

Figure 8: File size vs CPU load of proposed system for word count application

CONCLUSION

An efficient data aware caching system is designed, implemented and evaluated such that only few modifications are required in the original MapReduce programming model for provisioning incremental processing for big data applications. 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.

In future, it is decided to plan proposed system for more general application scenarios.

REFERENCES

[1] Executive Office of the President " Big Data: Seizing Opportunities, Preserving Values" May 2014.
[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] Dunlu Peng, Kai Duan and Lei Xie “Improving the Performance of Aggregate Queries with Cached Tuples in MapReduce”, International Journal of Database Theory and Application Vol. 6, No. 1, February, 2013.

[11] Ritu A.Mundada, Aakash.A.Waghmare “Cache Mechanism To Avoid Duplication Of Same Thing In Hadoop System To Speed Up The Extension”, IJRET: International Journal of Research in Engineering and Technology, Volume: 03 Issue: 11, Nov-2014.

[12] Sayali Ashok Shivarkar “Speed-up Extension to Hadoop System”, International Journal of Engineering Trends and Technology (IJETT), Volume 12 Number 2, Jun 2014.

[13] James Manyika, Michael Chou, Brad Brown, Jacques Bughin, Rihards Dobbs, Charlas Roxburgs, Angela Hung Bayer “Big data: The next frontier for innovation, competition, and productivity” McKinsey Global Institute, May 2011.

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

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

[16] 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.