A Novel Collaborative Caching Technique To Improve Performance on Data Storage In Hadoop

Dr.N.Gopika rani
Assistant Professor(Sl.Grade),
CSE department
PSG College of Technology
Coimbatore
9994153301
ngopika79@gmail.com

Dr.N.Hema priya
Assistant Professor(Sl.Grade),
IT department
PSG College of Technology
Coimbatore
nhemapriya@gmail.com

Abstract

Explosion of data and computing resources necessitates the use of platforms like Hadoop to handle large data sets. Hadoop comprises of HDFS and MapReduce framework. HDFS uses the MapReduce paradigm to store huge amounts of data in a scalable manages a commodity hardware. Map Reduce framework is used to perform analysis and carry computations parallel on huge datasets. Analysis performed on existing Hadoop configuration, which shows that processing of map-reduce is very slow due to increased access latency for data. This access latency in Hadoop slows processing speed. This necessitates the use of a caching scheme on the distributed platform to reduce latency. Hadoop currently uses Modified ARC Caching. The proposed technique is Collaborative caching with Least Unified Value (LUV) cache replacement policy. The novel cache replacement technique uses Least Unified Value, increases the performance of Distributed Caching System in Hadoop.

Keywords—Collaborative Caching, MapReduce, Least Unified Value, Hadoop.

1. Introduction

Analysis of massive amounts of data produced during online activities is resource intensive and complex. Online analytics tasks require a specialized framework like Hadoop to access and handle large data sets. Hadoop is an open source scalable distributed framework based on clustered architecture. Its master/slave architecture decouples metadata and application data. Metadata is stored on server NameNode and application data resides on datanode in hadoop. Hadoop is being utilized by reputed industries like the Amazon, Google, Facebook, and Yahoo etc. [20]. Facebook is uses a modified Hadoop architecture in real time for better throughput and latency. Hadoop has gained more importance because of its high reliability, scalability and parallel computations on huge data sets.

1.1 Data Storage on Hadoop

Hadoop applications uses the Hadoop Distributed File System (HDFS).HDFS is distributed system facilitate usage of data across hadoop clusters[15]. HDFS is the platform for managing data for big data applications. HDFS initially takes the data from client application, it separates the information into blocks and distributes them to datanodes of cluster for parallel processing. The system maintains replica of data so that one copy of data should be available on different rack. If any datanode fails in the cluster, it allows some other node to continue the failed process. Fig 1 shows the working of HDFS.

1.2 MapReduce

Hadoop uses MapReduce to perform analysis and parallel computations on the data. According to PACMan [21], map phase in Map Reduce involves reading raw information from the disk and this task is I/O based. Map phase generates <key, value> pairs as output that are combined by Reduce phase to generate a single result. Optimizing MapReduce Job Performance [6], explains that optimization done for improving the performance of MapReduce Tasks. Processing time of MapReduce job can be improved by adding more nodes to the cluster, but that is not a cost effective solution [5].

1.3 Collaborative Caching

We know that cache data access is much faster when compared to disk access. Hence caching concept can be used for improving the performance of map reduce tasks. Collaborative caching is implemented with centralized cache management which is managed by namenode [17]. Every datanode is allocated with local data cache which considered as globally single cache. Not only local data can be cached on datanode, but also data of other nodes can be cached in remote caches. If cache miss occurs in local data cache, then it can be accessed the data from remote data cache.

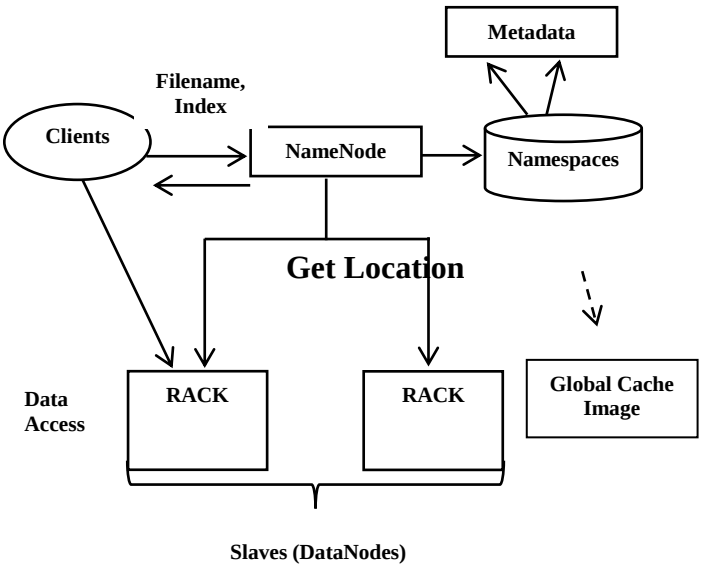

Fig.1 HDFS architecture

Every node is created with its own data cache. All caches are connected with each other to form a global cache. NameNode acts as central co-coordinator of cache, but it allows the cache manager to decide for selecting the remote cache on DataNodes. New approach allows efficient use of resources instead of using more number of nodes. So to improve the performance, more slots are to be added.

The organization of the paper is as follows; Section 2 describes related work, Section 3 explains the system architecture, Section 4 provides Proposed system, Section 5 explains the Experimental Results and Section 6 presents conclusion and future work followed by acknowledgement and references.

2. Related Work

Hadoop collaborative caching aims to improve the performance of Map Reduce Tasks. With local cache in every node to reduce the data access latency. Caching only part of the input will not help in improving the performance [1]. Rather performance can be improved only if the cached block is used fully [4]. The Adaptive Replacement Cache (ARC) algorithm dynamically balances recency and frequency by using two Least-Recently Used (LRU) queues in response to changing access patterns [10]. Collaborative caching is a caching service that coordinates access to the distributed caches [2]. This service aims at minimizing total execution time of job by evicting those items whose inputs are not completely cached. For evicting the inputs which have been minimally used, Modified Adaptive Replacement cache algorithm has been proposed.

Hadoop collaborative caching techniques added with new functionalities consists of five components. The components are Cache manager, Global Cache Image, Data Node, NameNode. The cache manager is responsible for managing caches by performing lookup in

local as well as global cache image. Upon Request from cache manager, replacement policy for cache is applied when cache is fully utilized [9]. Currently hadoop uses first in first out replacement policy. Hadoop collaborative caching, a proposed methodology is used to improve the performance of MapReduce jobs. This caching technique places the data on data nodes by using cache managers. This technique uses LRU caching with prefetching mechanism to enhance the job performanace[2]. In simple collaborative scheme, if there is a local cache miss, then DataNode notifies DFSClient of the next cached DataNode to access the data. This paper also explains the Modified ARC. Basic mechanism is to separate the caches into two different parts namely cached items and history items [11]. Cached items having the actual data and history part contains the references of cached data. Hence the cached part is further divided into Recent Cache with its Recent History and Frequent Cache with its Frequent History. A Recent cache is a cache where the block accessed for the first time is placed. A second reference to the same data will cause the block to be placed in the frequent cache [9]. The design includes prefetching by looking up, neighboring blocks for a request. If neighboring blocks are missing then cache requests NameNode to look for replicas and if available, requests are sent to DataNode to cache blocks [19]. Cache hit ratios can only be tested for 350 -650 MB which is the major limitation. Least Unified value uses the complete reference history of documents in terms of reference frequency [12]. It allows a better implementation in both space and time complexity. This LUV replacement policy follows the heap data structure, so time complexity is O ($\log_2 n$), where n denotes the number of elements in the cache[13].

A cache replacement strategy decides which object to evict from the cache when no space is available to store additional objects. It is based on several factors: recency, frequency, cost for fetching and size. To determine the effectiveness of a replacement strategy, certain metrics are measured such as the cache hit ratio, byte hit ratio, delays [18]. However for the Least Recent Value (LRV) algorithm, the time complexity varies according to the performance measures. While the algorithm is efficient for the byte hit rate measures, for other measures the complexity of algorithm is O(n). For this reason, a recent version LRV restricts their optimization of the algorithm to only byte hit rate measure [14] .For Caching algorithm to be practical, it is important that the time complexity of the algorithm should not be excessive, preferably not higher than O($\log_2 n$) , where n is the number of data in the cache[9].Adaptive replacement policy which has low overhead on system and is easy to implement. This model is named Weighting Replacement Policy (WRP) which is based on ranking of the pages in the cache according to three factors, such as cache hit, byte hit rate, data size [8] LFU works based on how many times page has been referenced and it selects the least number referenced page [3].LRU fails when recently used block is not used frequently. It cannot deal with larger than available cache size [16].

3. System Architecture

The proposed system architecture explains about working of HDFS with LUV replacement policy. Initially the client requests for block locations to the NameNode. The Namenode act as coordinator of datanodes. The namenode reports to the cache manager for accessing the data from cache. If data is not available in local cache then it contacts Global

Cache Image for remote access of data .In case cache is fully utilized, LUV replacement policy is used to evict the items from cache. The working of the proposed system is shown in Fig.2

3.1 Cache Manager

Every datanode is provided with their cache managers who have responsibilities to manage the local caches as well as global cache. Replacement policy and eviction policy is applied when cache is fully used. A buffer is maintained to cache the file references consists of metadata and application data. Data blocks in the caches are replaced or evicted when cache is full. Replacement method applied here is LUV. In cache the data blocks are arranged based on the value calculated from past reference history and data access time.

3.2 DataNode

DataNode provides a cached block report of its local cache to NameNode after periodic interval. As a response to this report NameNode commands DataNode to update global cache image.

3.3 NameNode

NameNode manages datanode cache by piggybacking cache and uncache commands on the DataNode heartbeat. The namenode asks its cache directives from user to determine which paths should be cached. It also stores the set of cache pools which are administrative part used to group the cached items for resource management.

3.4 DFS client

DFSclient can connect to the Hadoop file system and perform some fundamental tasks. To get the block locations for a particular file, it has to contact with NameNode. For every request, it provides the result for cache locations and non-cache locations. After receive the cache locations, and then it obtains the list for reading the cached block.

3.5 Global Cache Image

It maintains the metadata for cached block in datanode. If data is not found in the local cache , then cache manager lookup the global cache image check if data is available or not.

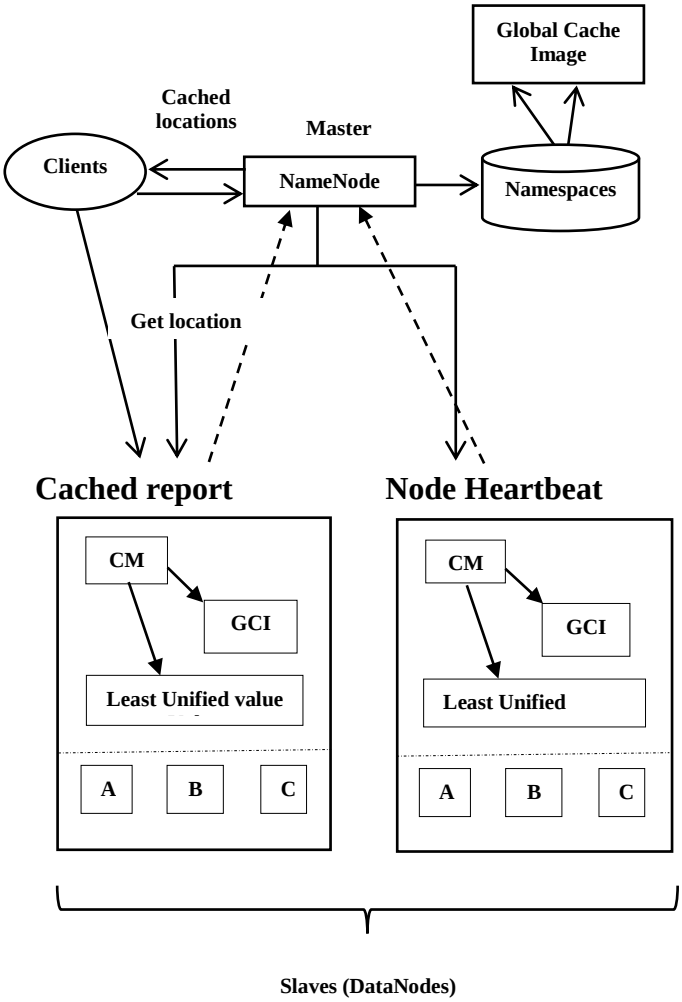

Fig.2 Proposed system Architecture

GCI – Global Cache Image
CM – Cache Manager

4. Proposed System

4.1 Caching Scheme

In order to reduce the execution, it introduces the collaborative caching with LUV replacement policy. It improves the performance of Hadoop when compared to the other replacement policies. The proposed system uses collaborative caching with Least Unified Value replacement algorithm. It works based on the object value which is calculated from past reference history of the data item. Every data item in cache has unique object value and is stored in heap order. The eviction is applied based on the object value. The proposed caching scheme using LUV approach reduces the time of execution of the MapReduce job.

The Name Node in the Hadoop system is responsible for getting file locations from Namespaces to NameNode reports to the cache manager check whether the data is available or not. If data is present in the cache, it returns the blocks.

In case of a Cache miss, the cache manager receives the request from NameNode to access the file from the disk and add block to the local cache The blocks in cache are maintained in form of a heap. While adding the data in the cache ,it immediately calculates the value based on the past reference history and request time of particular data. Based on calculate value, blocks are stored in a heap. Based on the value only the eviction policy is applied for replacement. The lower value must be evicted first from the cache. The parameter mainly used for calculating is relative cost and probability of request of k times.

4.2 Least Unified Value Algorithm

The basic idea behind is to calculate the values from past reference history and request time. The data structure maintained here is heap data structure. So the datas are arranged in heap order in cache [7].

Working of Least Unified Value

- Initially client requests for block, the NameNode issues the correct block location of metadata.
- The cache manager searches the requested data block in local cache, if it is not present then go for remote cache.
- If cache miss occurs in the both the cache, data would be accessed from disk and added to the cache.
- If cache is fully utilized, Least Unified value replacement policy is applied.

The following equation represents the object value,

$$V(i) = W(i).p(i) \qquad (1)$$

W (i) - Relative cost to fetch the object from its original server

P(i) - Probability for interval between current access time and last reference time.

$$W(i) = c(i)/s(i) \qquad (2)$$

c(i) - fetching cost of object *i* from the server

s(i) - size of object

$$p(i) = \sum_{k=1}^{fi}(tc - tk) \qquad (3)$$

t_c - Current access time.

t_k - the oldest request time in a window of *k* request times

4.3 Working of Least Unified value algorithm

It initially checks the data present in the local cache and valid state. If it available, then the LUV value for the data object is calculated. Adding this value to the heap, it follows the heap rules. If it is not in the local cache access data from the remote cache. In the situation of data is not available in the cache, cache manager reports NameNode with cache miss.

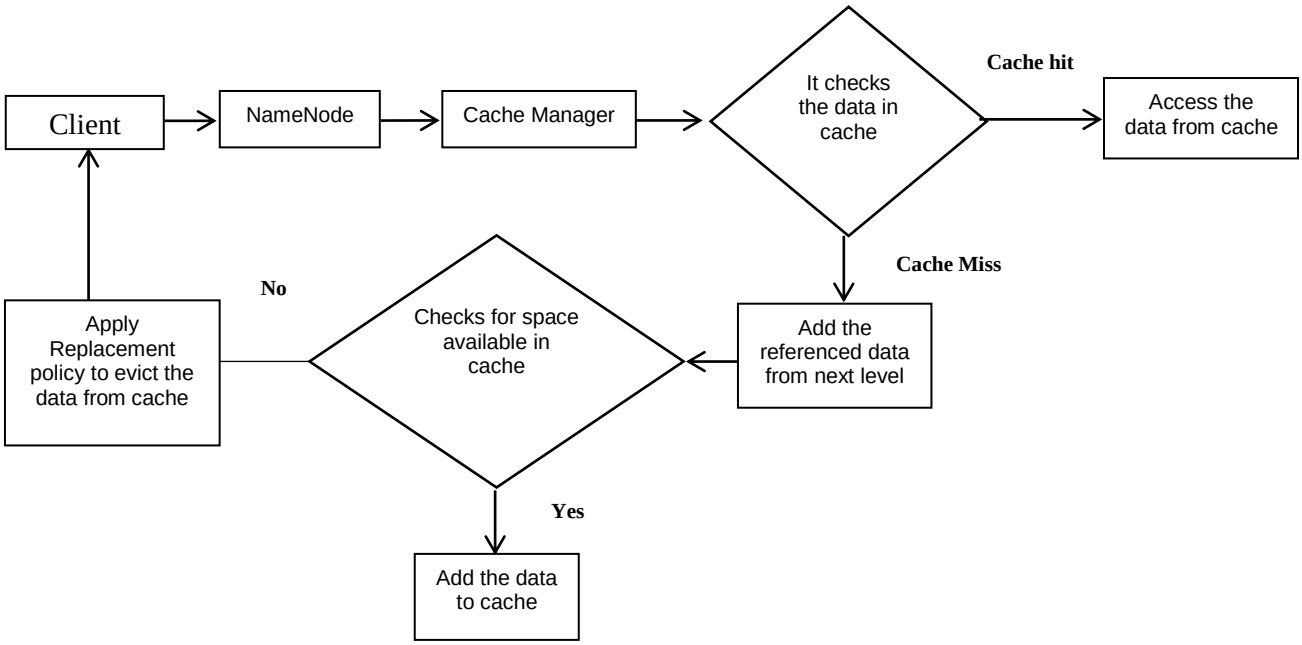

Fig.3 Flowchart for LUV replacement policy

In such a case the data should be accessed from disk and a new heap value is calculated. The block is inserted and the heap is adjusted according to the heap rules. In case the cache is full then minimum value is replaced by new data item and heap is adjusted. The proposed replacement algorithm for cache is shown in Fig 4.

D_x = Data object
M_x = Message for data object
D_y = Data object to be replaced
LUV = Value calculated for the data object'
L = minimum value
if D_x is in local cache
Calculate the LUV value
Return to the client
Else if D_x is in remote cache
Calculate the LUV value
Return to the client
Else D_x is not in the cache
Then cache miss occurs
Access the data from disk
Calculate the LUV value
Checks for the space available in Cache
If there is no enough space
Find min value = L
Evict the object which contains
LUV value=L
Bring d_x into cache
Return D_x to the client
Else
Calculate the LUV value
Add the data iem to the cache
Return d_x to the client

Fig.4 New algorithm for LUV in Hadoop

5 Experimental Results

The experimental results have been carried out in a hadoop cluster of 7 nodes with 2 cpu's with Word Count Application .This application is used to find the frequency of words present in the given input. By using "Inbuilt time" command in Linux, we analyze the results for various inputs. Experiment was repeated for varying file system from 100 MB to 1000 MB. It was conducted for block size of 128 MB. Table 1 and Table 2 shows the result of MapReduce job for the given sample files.

The master node act as both master as well as slave. Master node also contains datanode to store the application data. Caching part is present in the both master and slave node together to form as single cache. This improves MapReduce job performance .Local data is accessed when jobs are executed in same node as input. Remote access occurs when the data is present in the cache of nodes than the input. Local data access is better than remote access. In case of

collaborative caching all nodes are provided with cache. If file size increases more blocks are present and hence the cache can get full. Replacement algorithm is used to evict the items from the cache. Least Unified Value as is used as
replacement algorithm. It reduces the execution time for MapReduce jobs.

5.1 MapReduce Without caching

Table I shows that the results obtained from MapReduce job execution time. The results are taken from existing Hadoop framework. Here the jobs are executed in Hadoop system default

S.no	File Size	Single node cluster(msec)	Multi node Cluster(msec)
1	100 KB	285	165
2	500 KB	362	278
3	100 MB	450	321
4	500 MB	734	592
5	1 GB	1254	1189
6	2 GB	1980	1760
7	3 GB	2876	2512

Table 1 MapReduce Job Time without caching

5.2 MapReduce With Caching

Table II explains about MapReduce job execution time for given sample files. From the above results we conclude that collaborative caching with LRU replacement produces the better results compared to the existing system.

Table 2 MapReduce Job Time with caching (LRU)

S.no	File size	Single node cluster (msec)	Multi node Cluster(msec)
1	100 KB	164	69
2	500 KB	245	119
3	100 MB	336	272
4	500 MB	498	421
5	1 GB	1090	1034
6	2 GB	1790	1557
7	3 GB	2643	2341

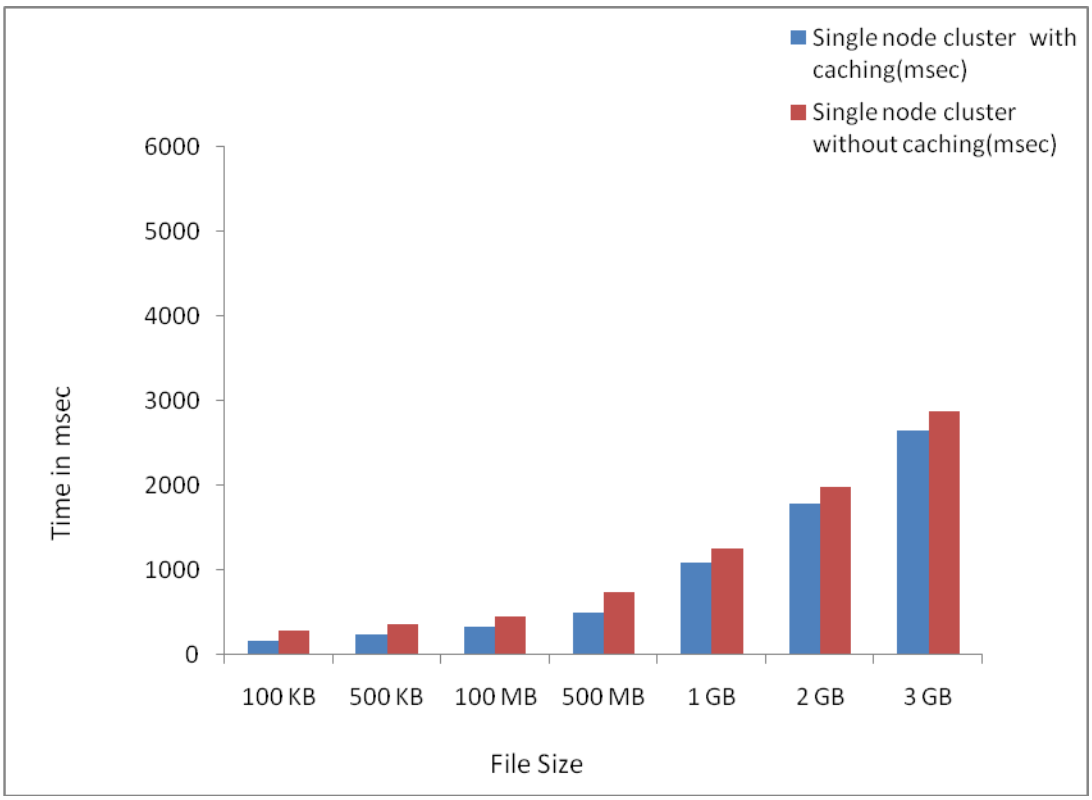

Fig 5. Job execution time with / without caching on Hadoop (Single node Cluster)

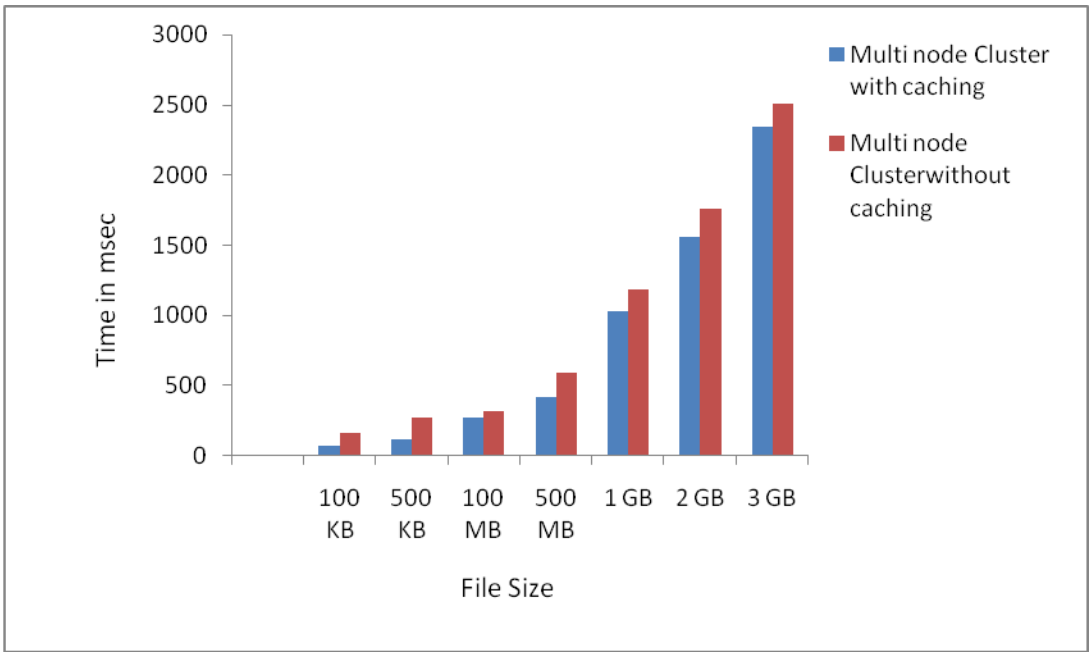

Fig 6. Job execution time with / without caching on Hadoop (Multi node Cluster)

The Table 3 shows that results obtained from proposed caching system for given various sample input files. It concludes that the proposed replacement policy produces better results compared to the existing system.

Table 3 Time Comparison between Hadoop-LUV and Hadoop-LRU

S.No	File Size	Hadoop – No cache (msec)	(Existing) Hadoop-ARC(LRU) (msec)	(Proposed) Hadoop-LUV (msec)
1	100 KB	165	132	69
2	500 KB	278	185	119
3	100 MB	321	311	272
4	500 MB	592	518	421
5	1GB	1189	1121	1034
6	2GB	1760	1640	1557
7	3 GB	2512	2412	2341

Following graph shows that the MapReduce job execution time. It shows a comparison of MapReduce jobs carried out on Hadoop-ARC and Hadoop-LUV. The MapReduce Job carried out was to read files which varied in size. Graph clearly shows that Hadoop-LUV performs better as compared to both Hadoop-ARC.

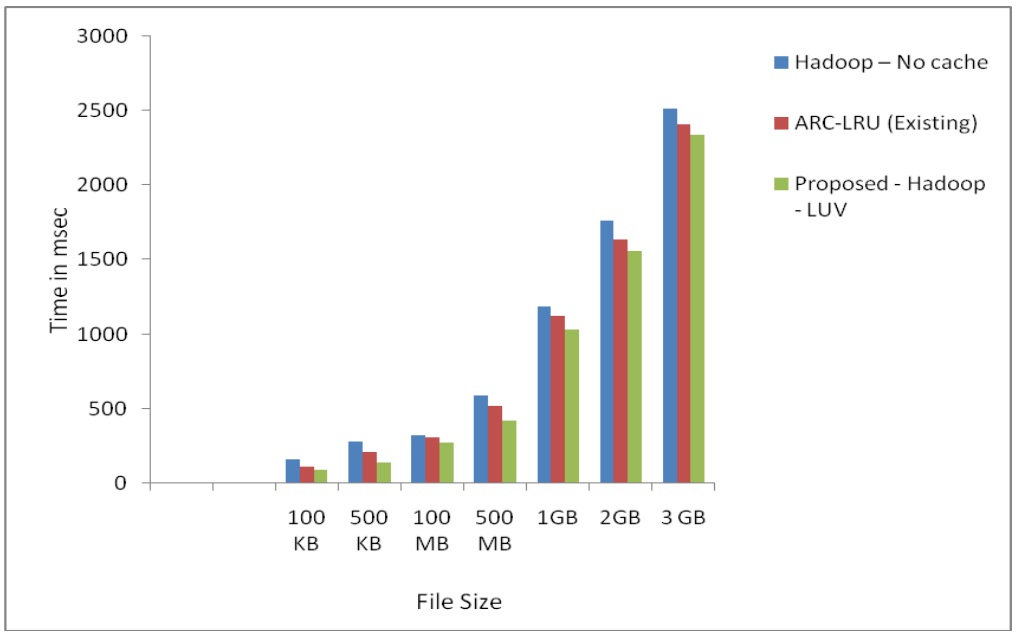

Fig.7 MapReduce job execution time for multi node cluster

5.3 LUV Replacement Strategy

The Table 4 shows that presence of data in cache based on the calculated LUV value. The cache size used here is 1GB. The cache is divided into several blocks of size 128 MB. The data are stored in the blocks according to that size.

Id	Pool name	Expiry time	File name
1	Pool1	14.30 pm	Data1
2	Pool1	14.38 pm	Data2
3	Pool1	14.42 pm	Data3
4	Pool1	Never	Data4
5	Pool1	Never	Data5

Table 4 Before LUV Replacement

Id	Pool name	Expiry	File name
1	Pool1	Never	Data1
2	Pool1	Never	Data2
3	Pool1	Never	Data3
4	Pool1	Never	Data4

Table 5 After LUV Replacement

The replacement of data happens ,when cache is fully utilized. Every data present in the cache based on the Calculated LUV value. When cache is fully utilized the lower value of data is replaced by newer one. For example: After run the data5 file in Hadoop , it replaces the expired files in the cache when cache is fully utilized. The Table 5 shows that data5 replaces the files present in the cache which was expired.

5.4 Block Replacement

5.4.1 Least Recently Used (Existing)

The Fig 7 shows that how the LRU replacement is done. Least recently used data will be replaced when cache is fully used. Cache size is 1GB. The cache is divided into blocks of 128 MB size. Fig 6 shows that initially D4 is accessed from the memory and it is cached. Fig 7 explains the replacement of data D7. It is least recent value among other data present in the cache.

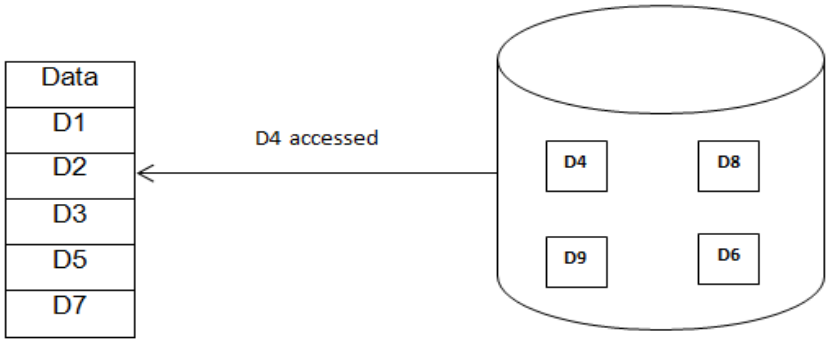

Fig.8. Accessing Data from Disk

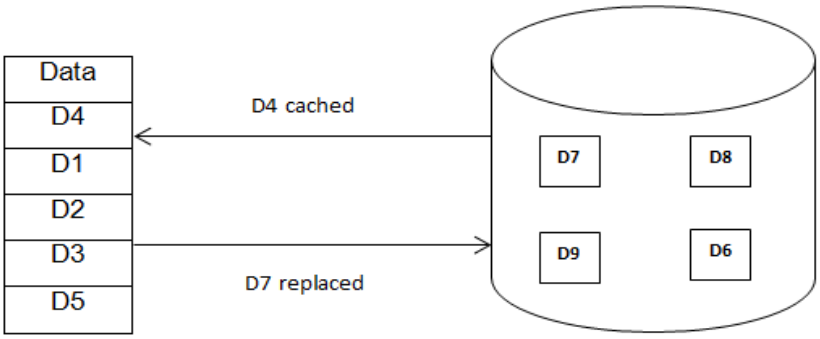

Fig.9. Data Replacement using LRU

5.4.2 Least Unified Value (Proposed)

The fig 8 shows that working of LUV replacement policy. Every data should provide with calculated object value based on the past reference history. The object values are arranged in the heap order. Lower values should be evicted from cache. In Fig.8 shows accessing of data from the disk and cached it. Fig.9 explains about replacement of data D1 using LUV technique.

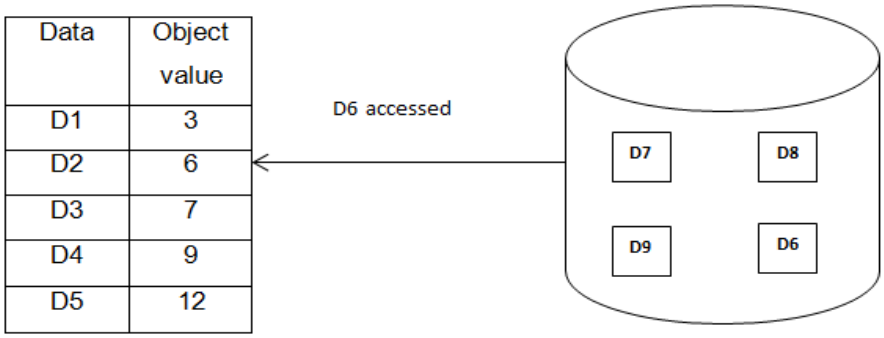

Fig.10. Accessing of data from disk

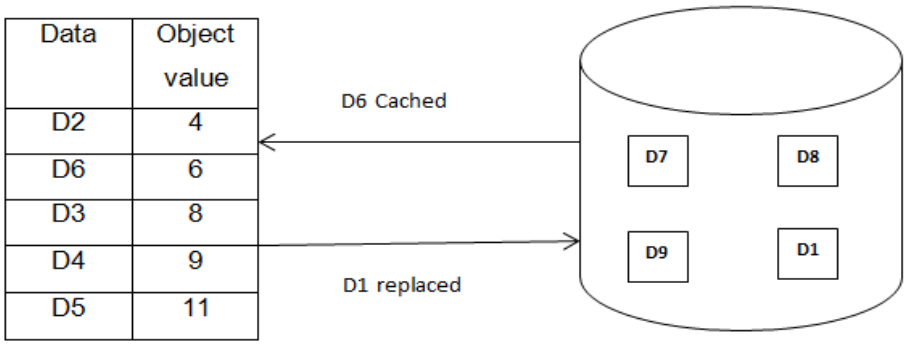

Fig.11. Data Replacement Using LUV Technique

6 Conclusion and Future Work

This work summarizes about the Hadoop collaborative caching with Least Unified value replacement algorithm. The proposed architecture is named as Hadoop collaborative caching .The main objective of the architecture is to reduce the execution time of MapReduce job execution time and to increase the efficiency of the system .It was done by collaborative caching where data is access from local cache as well as global cache. Every datanode contains their dedicated cache managers which are responsible for caching data and it follows the Least Unified Value Cache replacement policy. Every data in the cache has an object value which is calculated by using past reference history of the particular data item and request time of object. The data are arranged in heap order based on the calculated object value. This mechanism is helps better caching for replacement. Hence it improves the overall performance of the MapReduce job execution time. Next is to enhance this system to large clusters with LUV replacement policy.

7 References

[1] Aline Zeistunlian and Ramzi A.Haraty , An Efficient Cache Consistency Approach, World Academy of Science ,Engineering and Technology ,Vol:4 march (2014)

[2] Ananthanarayanan .A.W.D.B.S.K.S. S. G, A. Ghodsiand I. Stoica. ”Pacman: Coordinated memory caching for parallel jobs”, In NSDI, (2012)

[3] Belady. L. A., “A Study of Replacement Algorithms for a Virtual Storage Computer,” IBM systems J.vol. 5, no.2, pp.78-101, (1966)

[4] Christer A. Hansen. “Optimizing Hadoop for the cluster”University of Troms, Norway

[5] David Leong, Collaborative Object Caching for Heterogenous OSD Clusters, Proceedings of the Computer Science and IT Education Conference,pp.425-436, (2007)

[6] Dawei Jiang, Beng Chin Ooi, Lei Shi Sai Wu, "The Performance of the VLDB Endowment,Vol.3,No1.36th International Conference on Very Large Data Bases, September 13-17, (2010), Singapore.

[7] Hyokung bahn,sam H Noh,Sang Lyul Min "Using Full Reference history for efficient Document Replacement in Webcaches",Dec (2012)

[8] Kaveh samie ,"A Replacement Algorithm Based on Weighting and Ranking Cache Objects", International journal of hybrid Information Technology, vol.2 No.2,April,(2009)

[9] Lee.D Choi.J,Kim.J, S.H.Noh ,S.L Min,Y .cho , and C.Kim,"On the Existence of a Spectrum of Policies That Subsumes the LRU and LFU Policies," In Proceedings of the 1999 ACM SIGMETRICS Conference,pp.134-143,(1999)

[10] Megiddo.N and Modha.D.S, "Arc: A Self-Tuning,Low Overhead Replacement Cache," Proc. Second USENIX Conf. File and Storage Technologies, (2003)

[11] Megiddo, N. et.al Outperforming LRU with an adaptive Replacement Cache, Computer IEEE April (2004) pp.58-65

[12] Meenakshi Shrivastava and Hans-Peter Bisch of Hadoop-Collaborative Caching in Real Time HDFS (2012).

[13] Padmapriya.V and Thenmozhi.K," Web caching and response time optimization based on eviction method", International journal of innovative research in science, Engineering and Technology,vol.2,issue 4,april (2013)

[14] Rizzo.L and Vicisano.L ,"Replacement policies for a proxyCache",TechnicalReport UCL-CSRN/98/13,(1998)

[15] SenthilKumar.K.,SatheesKumar.K and Chandrasekaran ,"Performance Enhancement of Data Processing using Multiple Intelligent Cache in Hadoop," International Journal of Innovatoins in engineering ant Technology,ISSN: 2319 – 1058,Vol.4 Issue 1,June, (2014)

[16] Song Jiang and Xiaodong Zhang ,"LIRS: An Efficient Replacement Policy to improve Buffer Cache Performance," National Science Foundation,Dec (2005)

[17] Tom White ,Hadoop:The Definitive Guide, OReilly Media,Inc.,June (2009).

[18] Xu, J. and Hu, Q. (2001). An Optimal Cache Replacement Policy for Wireless Data Dissemination Under Cache Consistency. IEEE. pp.267-274

[19] Yaxiong Zhao*, Jie Wu and Cong Liu ,"Dache: A Data Aware Caching for Bigdata applications Using MapReduce Framework," Tsinghua science and Technology,ISSN 1007-0214 05/10 p39-50 ,vol 19,No.1 Feb(2014)

[20] [Online] http://en.wikipedia.org/wiki/Apache Hadoop

[21] [Online] http://Hadoop.apache.org/docs/r0.20.2/hdfs design.html.