ISSN 2278-3091
Volume 9, No.5, September - October 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse69952020.pdf
https://doi.org/10.30534/ijatcse/2020/69952020

A Novel Speculation-Based Read Algorithm with Rank based Replacement for Improving Performance of the Distributed File System

Rathnamma Gopisetty[1], Thirumalaisamy Ragunathan[2], C Shoba Bindu[3]
[1]Research Scholar at Jawaharlal Nehru Technological University,
Anantapur, India, rathnagurram@gmail.com.
[2]Department of Computer Science and engineering,
SRM University, Andhra Pradesh, India.
[3]Department of Computer Science and engineering, Jawaharlal Nehru Technological University,
Anantapur, India.

ABSTRACT

Modern web-based information systems generate large data and they require cloud computing kind of environment to store and access this huge data in order to provide their millions of online users in a scalable manner. Cloud computing systems make use of distributed file systems (DFSs) in their infrastructure to store and access large data. Hence, DFS has become an important component in the modern web-based information systems deployed in the cloud computing environment. Most of the users of the information system frequently perform read operations and infrequently perform write operations on the DFS. Hence, improving the read operations performance in the DFS has become an important research problem in the Big Data scenario. To improve the read operations performance in the DFS, prefetching and client side caching techniques are used. In the recent works, to reduce read operation access time of distributed file system, many collaborative client side caching techniques are discussed. In this paper, we have proposed a novel speculative read algorithm. Speculative read algorithm along with rank-based replacement algorithm can improve the read operations performance in the distributed file system.

Key words: Client side caching, Distributed file system, Hierarchical Collaborative Caching, Prefetching, Rank based Replacement Algorithm, Speculative Read.

1. INTRODUCTION

In the promising big data scenario, web-based information systems (WISs) are being deployed in cloud computing systems to cater billions of online users efficiently. Cloud computing systems offer scalable storage and computing power so that deploying WIS will be beneficial to the owners of the organizations and also the users of the WIS. Cloud computing system uses distributed file system (DFS) at the back end to store the large data in a distributed manner so that storage will be scalable. The distributed file system (DFS) environment consists of data nodes (DNs) and name nodes (NNs). The DNs are used to store the data permanently in the hard disks and NNs are used to store metadata. Note that, the read or update requests submitted by the users are carried out in the DNs.

The WIS receives very frequently the read requests and infrequently the write/update requests. For this reason, improving read operations performance, carried out on the DFS has become an important issue in the current context. Two significant techniques, Client-side caching and prefetching, are presented in the literature for improving read operations performance in the DFS environment. It is possible to minimize the average read operations access time in the DFS environment, by maintaining client-side caches (local caches) in the DNs main memory and by following collaborative caching by combining the local caches kept in all the DNs. In [10], authors discuss regarding the use of global cache which is maintained in one of the dedicated computer systems in the DFS environment to minimize the average access time of read operation. In [51], authors discuss the advantages of combining local, collaborative and global caches to store the frequently accessed data which will result in reducing average read access time further.

In this paper, we have presented a novel speculation-based algorithm for reducing the read access time and also the write/update overhead of the client-side caching system. We have investigated the algorithms performance through simulation. The results indicate that the advocated speculative read algorithm along with rank-based replacement algorithm can improve the read operations performance in the distributed file system.

The rest of the paper is organized as follows: section 2 explains the related works. The speculative read algorithm and modified write procedure are detailed in section 3 and section 4 respectively. Our evaluation methodology and results are presented in section 5 and conclude the paper in section 6.

2. RELATED WORKS

There are many researches on how to improve the read access time of distributed file systems using client-side caching prefetching and speculative techniques.
Big-data systems use hard disk drive (HDD) to store huge amount of data to be utilized by big-data applications. For example, Google File System (GFS), Hadoop Distributed File System (HDFS), stores massive amounts of data on large number of data nodes. HDDs alone become inadequate to meet the exigent requirements of the big-data applications. PACMan [43] uses main memory-based caching to enhance application efficiency and performance of cluster based file systems. Global memory caching systems (GMS) project [44], NOW project [45] and Dynamic Caching using Memcached servers [8] have employed coordinated caching using the memory of remote machines. The caching methods presented in the past focus on coordinated caching by taking into consideration only the local caches [44], [45]. And the Memcached system proposed in [8] employs global servers that are independent of one another. These techniques do not exploit collaboration among local and global caches in a combined manner. To tackle with the performance limitations of HDDs and global memory caching systems, we are presenting a hierarchical global collaborative caching (HCGC) technique [51] that makes use of collaborative local caching and the collaborative global caching in the DFS.

The growing gap between processing speeds and disk access times [1] is causing data intensive applications to spend most of their execution times on disk I/O to fetch data from disk. I/O prefetching is a renowned technique to solve the disk stalling problem even though the data is need to be fetched from multiple disks distributed across various geographical locations.

Existing prefetching methods are broadly classified into predictive prefetching and informed prefetching. Predictive prefetching predicts the upcoming Read/Write access patterns based on the past data accesses of applications [2]. Informed prefetching makes prefetching choices based on application's upcoming access hints [3], [4].

The I/O access performance capabilities of a storage system also depend on the placing of the data. Generally a high-level storage store hot data which is to be accessed in the near future. Previous studies [5]–[10], [51] show that there is more benefit of having a multi-level caches or hierarchical caches. In hierarchical cache systems if the requested data is not found in the higher level storage, the subsequent storage level is checked until the requested data is fetched.

By instigating prefetching and speculative executions on the data might decrease the I/O stall time of target standard executions. Modern processors use Speculative execution technique to alleviate pipeline stall problem by executing the next likely instructions in advance [11]–[17]. Examples of speculative execution are Branch prediction [11], [12] [17]–[20] and load value speculation [15], [16].

A large body of research has been devoted to enhance the performance of storage systems by reducing the disk I/O operations. A speculation-based method discussed in [21], starts the speculative execution using the local cache contents first. The timestamps obtained from the server disk and local caches are compared to commit the execution. An aggressive hint-driven prefetching system [22] is proposed to pre-execute speculatively the application's code to determine and supply hints for its upcoming read accesses, but the applications need to be explicitly modified to issue the hints. Big Data systems such as Google Map Reduce, Apache Hadoop, and Apache Spark depend on speculative execution [23]–[25] to mask slow tasks in turn to cut down job execution time.

Replacement algorithms play a significant role to overcome the performance problems caused by the speed difference among processor and memory. Many replacement algorithms were proposed to determine what to eject from memory when the cache is full and to replace the new file block with the existing file block.

The replacement policies are generally categorized [26] as recency-based policies, frequency-based policies, size-based policies, function-based policies and randomized polices. In recency-based policies, recency is used as the key decision making aspect. LRU and LRU variant replacement policies fall under this category. Frequency-based policies consider the popularity or frequency count as the most important factor and LFU is one of the examples of this class. Size based policies use the object size as the main factor, and this replacement policy removes larger objects first, from the cache. Function-based policies assign a utility value for each object. This value is computed using different factors such as time, frequency, size, cost, latency. GD-Size is the best example policy in this class. Randomized polices evicts an object selected arbitrarily from the cache.

Low inter-reference recency Set (LIRS) proposed by [27], uses the access recency information known as inter-reference recency to anticipate the pages which are more likely to be accessed in the near future. The clock-based algorithms, for example CLOCK [28], CLOCK-PRO [29], CAR [30], and ARC [31], use a reference bit or a reference counter to find the victim page. Clock-based algorithms record only history access information of pages, but will not consider the order in which the pages got accessed. The scarcity of affluent history information can damage the hit ratios.

Detection-based adaptive block replacement scheme (DEAR) [32], and AFC [33], analyze the memory accesses to find specific patterns and adopts diverse replacement conditions for each pattern like MRU for sequential accesses and LRU or LFU for other kind of patterns.

Fuzzy inference techniques by [34], [35] and Artificial Intelligence techniques by [36], [37] are also proposed to make the replacement decisions. An intelligent replacement strategy combined with GDSF and SVM is proposed for web caching in [38] and [39]. The cache replacement decisions will be made using the object re-accessed probability. These

intelligent algorithms are computationally difficult to take cache replacement decisions.

In the Big data and cloud computing era, efficient multi level cache management is one of the biggest challenges. The replacement algorithm has most important impact on the efficiency of a caching. A number of replacement algorithms such as first in first out (FIFO), most recently used (MRU), least recently used (LRU), and least frequently used (LFU) emerged as more effective. The most generally used replacement policy is LRU (Least Recently Used) and it records the pattern of recent referenced blocks. The important limitation of LRU policy is it keeps track of the present accesses of the blocks without considering the file block access frequency, (file level access patterns) and file block support. In face of this problem, we have proposed support based replacement algorithm and rank based cache replacement algorithm [51]. In these algorithms the replacement choices are made using the support value of a file block.

All of these methods bring significant intangible benefits to the conventional replacement algorithms. In many cases they need more complex implementations, additional data structures and also entail data update for each memory access.

3. PROPOSED SPECULATIVE READ ALGORITHM

Speculative execution is a significant issue in computer architecture. It enables parallel execution of serial programs. Executing instructions ahead of their typical schedule is called as speculative execution. The origin of speculative execution is speculative instruction execution with branch prediction [52]. The style of speculative execution is more influenced and led to the development of the thread level parallelism. Control, data dependence and data value speculation are the three important ways of implementing thread level speculations [53].

Speculative execution is also applied on relational databases [54], [55], [56]. Speculative query execution allows the queries to be performed in advance in the database system to reduce the response time.

Today's world is lead by data. People along with machines are producing massive amounts of data every second by communicating messages, uploading multimedia content like photos and videos, collecting sensor data from various types of sensors and so on. In order to pace with the changes in the data patterns and to accommodate the requirements of big data, the platform for storage and processing also requires great advancements. To dynamically support and speed-up execution of user applications we propose speculative read algorithm.

This section illustrates how the proposed speculative read algorithm works. Speculative execution permits serial tasks to be executed in parallel. To implement speculative execution, the system anticipates the result of a certain operation and carries out its execution using the predicted value. After the operation execution, the obtained result is compared with the predicted result. The system commits the speculative state, if the prediction was correct. Otherwise, rollbacks the system state to a prior consistent state.

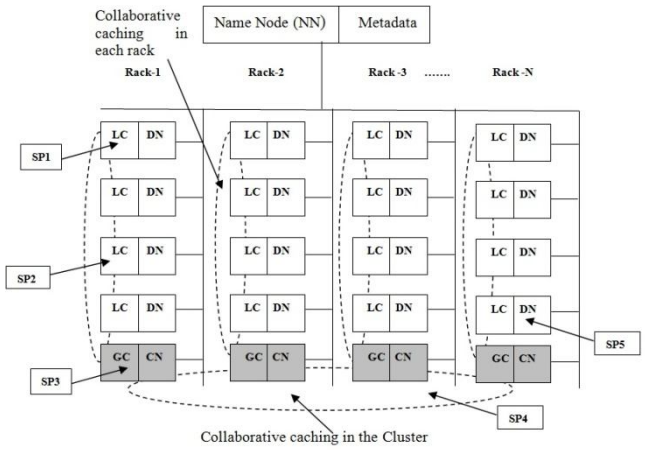

Figure 1: Speculative Threads created in the Cache Hierarchy during the implementation of Speculative Read Algorithm.

Speculative execution is used to improve performance in various systems including distributed file systems [21]. The hierarchical global collaborative caching (HCGC) algorithm, searches for the requested file block to complete the read operation, in a sequential fashion starting from lower level cache to higher level cache of the multi level caches available in the system. It would be beneficial if we execute the read operation concurrently as the hierarchical caches present in the system are pre-filled with the popular file blocks and the possibility to find these file blocks in any one (or more) of the caches of the system.

The motivation behind speculative read algorithm is to reduce the read latency. The proposed speculative read algorithm allows the application to concurrently execute the read operation on the contents of local cache (LC), the collaborative local cache (LCC), the global cache (GC), the collaborative global cache (GCC) and disk to reduce the read operation latency.

The proposed speculative read algorithm along with the support based prefetching [10], hierarchical collaborative global caching and the rank based replacement algorithms [51] works as follows. When a client application program executing on a DN need a file block fb, then the local cache manager concurrently check with the local cache (LC), the collaborative local cache (LCC), the global cache (GC) and the collaborative global cache (GCC) for the file block, by conversing with the pertinent cache managers and also creates speculative executions (SPs).

Speculative threads are created on behalf of each speculative execution and named as SP1, SP2, SP3, SP4, SP5, SP6, and SP7. Speculative thread SP1 represents the speculative execution initiated at local cache (LCDN). SP2 is a speculative thread, which represents the speculative execution

initiated at one of the local cache on the same rack (local collaborative cache-LCC). The speculative execution started at global cache (GC) present on same rack is carried out by the speculative thread SP3. SP4 is the speculative thread created to represent the speculative execution started at global collaborative cache (global cache present on any rack-GCC). Speculative execution started at a nearest data node (DN) is represented by the speculative thread SP5.

Speculative thread SP1 starts its execution if the requested file block Fb is available in the local cache (LCDN) otherwise it will get terminated. SP2 execution will be started if the requested Fb is available in any one of the local cache present on the same rack i.e the collaborative local cache (LCC). SP3 execution will be started if the requested Fb is available in the global cache (GC) connected to the same rack. SP4 execution will be started if the requested Fb is available in any one of the global cache present on any rack i.e the global collaborative cache (GCC).

Simultaneously, the DFS user application program executing on the DN starts the speculative execution by invoking SP5 thread for reading the file block from a nearest DN disk by contacting the NN and also collects the timestamp of the file block. After that, the time stamp collected from the NN is compared with the time stamp of the file blocks in the local cache, collaborative local cache, global cache, and collaborative global cache. If any file blocks time stamp value is matching with the time stamp collected from the NN, then the corresponding speculative execution initiated in the cache is allowed to complete. All the remaining speculative executions initiated in the other caches will be terminated. If time stamp is not matching with any of the cached file blocks then the speculative execution initiated at the disk is committed. Algorithm 1 describes the speculative read algorithm. Figure 1 describes the speculative threads created in the Cache Hierarchy during the implementation of Speculative Read Algorithm.

Algorithm 1 Speculative read algorithm

/* A user application program AP executing on a data node DN has issued read operation for File block *fb* of file *F* */
1: Timestamp t1=getNameNodeTimeStamp(*fb*)
2: Timestamp t2=get-LC-TimeStamp(*fb*)
3: Timestamp t3=get-LCC-TimeStamp(*fb*)
4: Timestamp t4=get-GC-TimeStamp(*fb*)
5: Timestamp t5=get-GCC-TimeStamp(*fb*)
6: Timestamp t6=get-closest DNs-TimeStamp(*fb*)
/* Create speculative execution threads SP1, SP2, SP3, SP4, and SP5 and all these threads will be executed in parallel */
7: if (*fb* is present in the LCDN) then SP1 starts execution
8: else SP1 is terminated
9: end if
10: if (*fb* is present in the LCC) then SP2 starts execution
11: else SP2 is terminated
12: end if
13: if (*fb* is present in the GC (present on same rack)) then SP3 starts execution
14: else SP3 is terminated
15: end if
16: if (*fb* is present in the GCC (global cache present on any rack)) then SP4 starts execution
17: else SP4 is terminated
18: end if
19: if (*fb* is present in the closest DNs disk)) then SP5 starts execution
20: else SP5 is terminated
21: end if
22: if (t1! = null) and (t2! = null) and (t2 == t1) then
23: Terminate SP2, SP3, SP4, SP5
24: Wait for SP1 to complete
25: else if (t3! =null and t3 == t1) then
26: Terminate SP1, SP3, SP4, SP5
27: Wait for SP2 to complete
28: else if (t4! =null and t4 == t1) then
29: Terminate SP1, SP2, SP4, SP5
30: Wait for SP3 to complete
31: else if (t5! =null and t5 == t1) then
32: Terminate SP1, SP2, SP3, SP5
33: Wait for SP4 to complete
34: else if (t6! =null and t6 == t1) then
35: Terminate SP1, SP2, SP3, SP4
36: Wait for SP5 to complete
37: else File not found
38: end if

4. MODIFIED WRITE PROCEDURE

The procedure followed for implementing modified write operation is discussed in this section. When a user application program executing on a data node has issued a write/update operation on file block then the file block is first copied into the local cache, global cache and to fulfil the requirement of replication factor the file block is written to local disk, remote disk on same rack and remote disk on different rack. The write procedure is described in algorithm 2.

Algorithm 2 Write Procedure

/* A user application program AP executing on a data node DN has to write/update a File block *fb* of file *F* */
1: if (write (new file block *fb*) then
2: Write in local cache
3: Write in global cache
4: write to local disk
5: Write in remote disk on same rack (replica on same rack)
6: Write in remote disk on different rack (replica on different rack)
7: end if

The traditional write procedure writes the data only to the disks available in the DFS. We have modified the write operation such that the newly written or updated copy is also placed in the local cache and also in the global cache along with one or more disks according to the replication factor followed in the DFS. So that the subsequent read operations invoked on the new blocks will get benefited from the data copies present in the local cache , global cache also avoids the disk accesses. This will also increases the system performance by reducing the read operation latency.

5. PERFORMANCE EVALUATION

We cover the assumptions for implementing the speculative read algorithm in this section. The implementation aspects of the proposed algorithms are covered next.

5.1 Assumptions

1) To avoid the communication overhead, no protocols are used to synchronize the caches.
2) The DFS user program converse with the NN to collect the DNs addresses where the requested file blocks are available.
3) The size of Local and global caches are predetermined.
4) During the write of a file block, it will first be written to the local cache, global cache and then written to the DNs disk according to the write-through strategy.
5) 80% of read and 20% of write operations.
6) The minimum threshold value considered for prefetching is 0.6.

5.2 Experimental Setup

The computing and storage infrastructure of the DFS considered to conduct the experiments consists of 10 racks R1, R2... R10 and each rack consist of ten DNs. Each rack also consists of a cache node with the global cache to facilitate the implementation of HCGC algorithm. We have also assumed that NN is connected to rack R1.

5.3 Specifications

We have considered the following specifications for computing read access time for a file block. We presumed that the file block size should be of 4 KB for the DFS and the communication delay (CD) requisite to transfer a file block from a distant data node to the local data node as 4 ms. Based on the specifications of the switched local area network [46], the time requisite to transfer time stamp and metadata is considered as 0.125 ms. By considering the latest Seagate disk storage devices [47], the disk access time is 12 ms. We have presumed that the main memory access time is 0.005 ms, according to the latest DDR4 dynamic random access memory technologies [48]. The access time to read 4KB file block from the distant memory as 4.01 ms. And access time for transferring metadata as 0.125 milliseconds. The communication delay (CD) requisite to transfer a file block from a remote data node (present in a different rack) to a local data node is 6 milliseconds. We have also considered that the cache invalidation time is 0.125 milliseconds.

5.4 Synthetic Log Generation

In order to generate a more realistic log, we used MediSyn [49], [50]. We have assumed that, there are 100 files present in the DFS and each file consists of 25 to 1000 blocks. The file and block popularity ranks are generated by using the random permutation and rejection method. We have used a generalized Zipf distribution function to generate file and block frequencies. We have fixed popularity parameter (p) value as 0.8 and maximum file frequency (M_f) value as 500, maximum file block frequency (M_b) value as 1000, and scale parameter k as 30. The request arrival for a day is modelled as a non-homogeneous Poisson distribution function and each interval consists of 1000 requests. After sessions are generated, we have merged all the sessions according to the arrival time to build the complete log.

For conducting simulation experiments, we have contemplated that a log with 100000 sessions and each session has got a number of read or write requests. Note that, each read or write requisition consists of 5 to 15 file blocks. We have to calculate local support and global support values for all the files by considering a DN. We have contemplated that the popular files will have support value greater than 60%. After that popular file blocks are computed for all the popular files. Local caches of the DNs is filled with the popular file blocks and the remaining popular file blocks and the globally popular file blocks are stored in the corresponding global caches present in the cache node of the rack in which that DN is present.

In each experimentation, 100000 log entries are created using non-homogeneous poisson distribution function, and each interval consists of 1000 read/write requests and computed the average read operation access time.

5.5 Simulation Results

In this section we present the results of simulation experiments of speculative read algorithm using support based prefetching, hierarchical collaborative global caching, LRU replacement, support based replacement, and the rank based replacement algorithms.

Figure 2 portray the average read access time (ARAT) of speculative read algorithm with LRU, support and rank based replacement techniques. To conduct these experiments, we set the size of local cache as 100 global cache as 500, total number of files available in the system as 100 and varied the number of blocks of a file from 25 to 100.

Through the experiments we can see that the proposed speculative read algorithm with support and rank-based replacement techniques performs better than speculative read algorithm with LRU replacement.

The reason behind this is, the cache content usually becomes stable, after filled with the file blocks of maximum support value (in support based replacement) and top n ranked file blocks (in rank based replacement).

Figure 3 depicts the cache hit ratio (CHR) of the speculative read algorithm with LRU, support and rank-based replacement techniques. To conduct these experiments, we set the size of local cache as 100 global cache as 500, total number of files available in the system as 100 and varied the number of blocks of a file from 25 to 100.

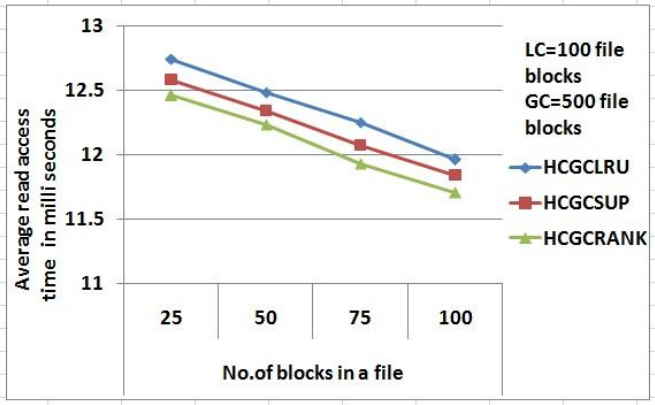

Figure 2: Number of blocks of a file Vs ARAT (LC size is 100 and GC size is 500 blocks).

Experiments can prove that the local cache hit ratio of the proposed speculative read algorithm with rank-based replacement algorithm is better than speculative read algorithm with support and LRU replacement algorithms. We can also observe that collaborative local cache hit ratio of the proposed speculative read algorithm with rank-based replacement technique is better than speculative read algorithm with support and LRU replacement algorithms. It is also observed that local cache hit ratio, collaborative local cache hit ratio and global cache hit ratio of speculative read algorithm with rank based replacement algorithm is better than speculative read algorithm with support and LRU replacement algorithms. We also observe that speculative read algorithm with rank based replacement algorithm has got a good hit ratio value for collaborative global caches.

Figure 4 illustrates the average read access time of speculative read algorithm with LRU, support and rank based replacement techniques. To conduct these experiments, we set the size of local cache as 100 global cache as 500, total number of files available in the system as 100 and varied the number of blocks of a file from 200 to 500.

Speculative read algorithm with rank based replacement technique performs better than speculative read algorithm with support based replacement.

Speculative read algorithm with support based replacement technique performs better than speculative read algorithm with LRU replacement. We can also observe that, the proposed speculative read with rank-based replacement performs better than the remaining three algorithms.

Speculative read algorithm with support based replacement technique performs better than speculative read algorithm with LRU replacement. We can also observe that, the proposed speculative read with rank-based replacement performs better than the remaining three algorithms.

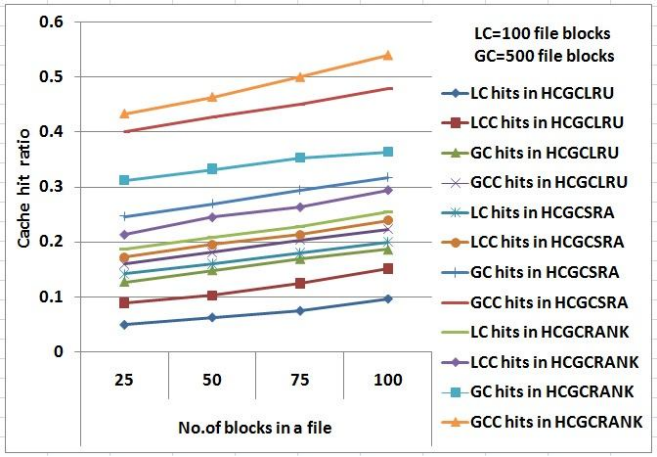

Figure 3: Number of blocks of a file Vs CHR (LC size is 100 and GC size is 500 blocks).

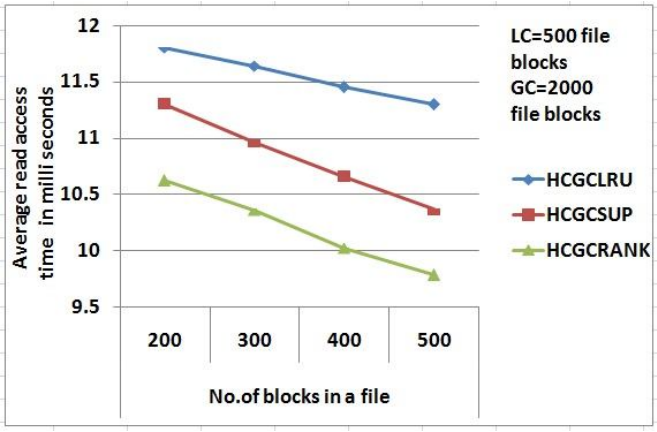

Figure 4: Number of blocks of a file Vs ARAT (LC size is 500 and GC size is 2000 blocks).

The local cache size is set as 500 global cache as 2000, total number of files available in the system as 100 and varied the number of blocks of a file from 200 to 500 and potted the graphs for cache hit ratio.

In Figure 5 we have plotted graph with the cache hit ratios of local and collaborative caches, of speculative read algorithm with rank based replacement algorithm is better than speculative read algorithm with support-based replacement and speculative read with LRU.

We also find that the global cache hit ratios of speculative read algorithm with rank-based replacement technique is better than speculative read algorithm with support-based replacement and speculative read with LRU.

We also find that the collaborative global cache hit ratio of speculative read algorithm with rank-based replacement technique is better than remaining algorithms. So, we can say that speculative read algorithm with rank-based replacement technique surpasses the remaining two algorithms.

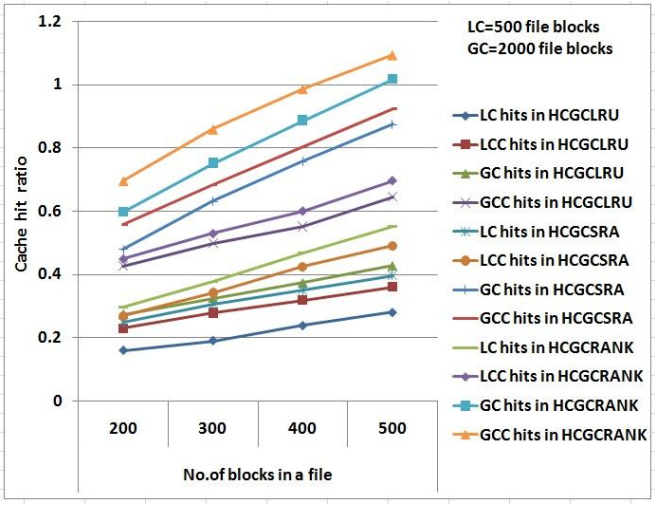

Figure 5: Number of blocks of a file Vs CHR (LC size is 500 and GC size is 2000 blocks).

For this and next experiments, we have fixed the local cache size as 1000 blocks, global cache size as 5000 , varied the number file blocks in a file from 250 to 1000 and plotted the graph which are shown in Figure 6. In Figure 6, we can examine the average read access time performance of the speculative read algorithm with LRU, support and rank-based replacement techniques. Speculative read algorithm with rank based replacement technique performs better than speculative read algorithm with support based replacement. The performance of speculative read algorithm with LRU replacement support and rank-based replacement increase as the size of the global cache also increases. We can also observe that, the proposed speculative read algorithm with rank-based replacement performs better than the remaining three algorithms.

In Figure 7, we have plotted the graph with cache hit ratios of local and collaborative caches, of speculative read algorithm with rank based replacement algorithm is better than speculative read algorithm with LRU and HCGC with support based replacement. We also find that the local, collaborative local, global cache hit ratios of speculative read algorithm with rank based replacement technique is better than speculative read algorithm with LRU and HCGC with support based replacement. We also find that the collaborative global cache hit ratio of speculative read algorithm with rank based replacement technique is better than remaining algorithms. So, we can say that speculative read algorithm with rank-based replacement technique performs better than remaining three algorithms.

From the experiments results, we can see that the cache hit ratio of all algorithms is in the increasing order. We believe that this is due to the support based prefetching technique and it leads to number of hits in the cache hierarchy.

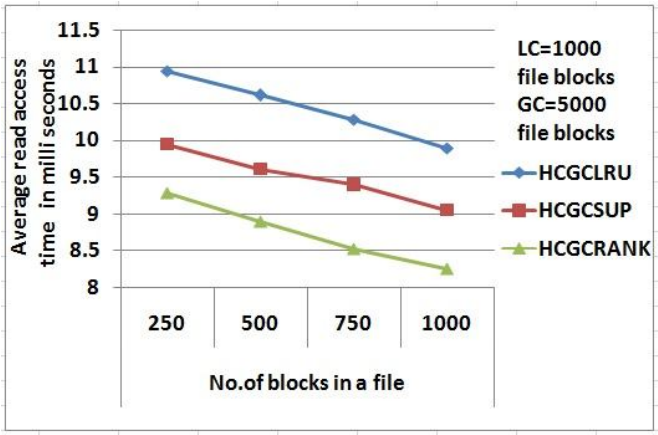

Figure 6: Number of blocks of a file Vs ARAT (LC size is 1000 and GC size is 5000 blocks).

We can observe that with the tiny caches, cache can accommodate only limited file blocks, in such contexts, if there are large number of client requests, it leads to more number of misses, so the cache hit ratios of the algorithms are not high. But when the cache size is larger, cache can accommodate more number of file blocks from different files, hat will result in high cache hit rate.

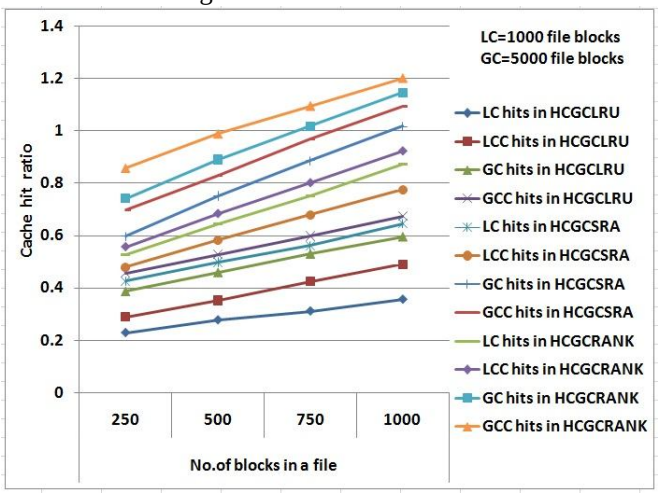

Figure 7: Number of blocks of a file Vs CHR (LC size is 1000 and GC size is 5000 blocks).

Next we discuss about the overhead that occurs with the cache invalidation. When multiple caches have the same file blocks, we need to maintain cache consistency. Change to a cached file block occurs when it got updated by the application program. These changes cause the file block to be out of sync with the other cached copies and or with storage server copies. To solve this hitch, the change should be propagated to all other caches whichever have the file block. The strategy applied to notify all the caches regarding the update is called as cache invalidation policy. This problem is more definite in collaborated caching schemes as there are multiple copies of the same file block existing at different caches.

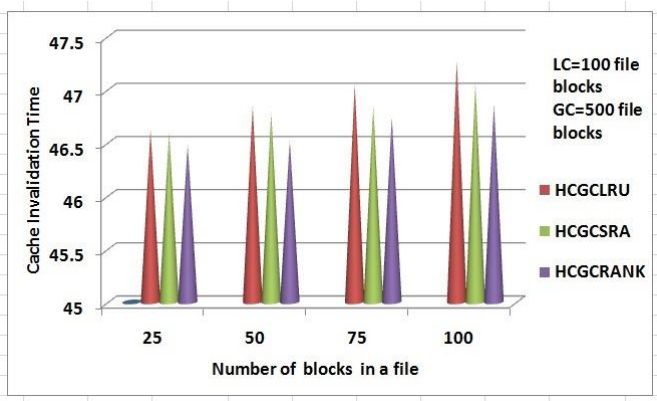

Figure 8: Number of blocks of a file Vs Cache invalidation time overhead (LC size is 100 and GC size is 500 blocks).

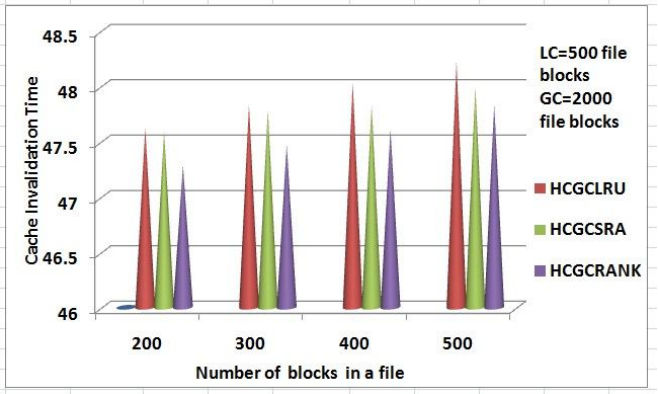

Figure 9: Number of blocks of a file Vs Cache invalidation time overhead (LC size is 500 and GC size is 2000 blocks).

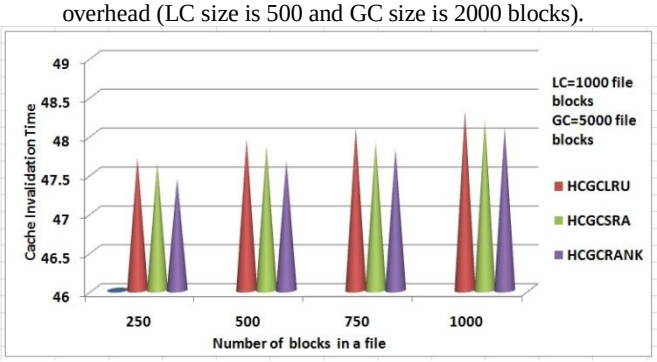

Figure 10: Number of blocks of a file Vs Cache invalidation time overhead (LC size is 1000 and GC size is 5000 blocks).

To compute the cache invalidation overhead, we ran the simulation experiments of hierarchical collaborative global caching algorithm using the support based prefetching, and the rank based replacement algorithms but without speculative execution. Through the simulation experiments we have calculated the cache invalidation time overhead and the obtained results are depicted in figures 8, 9, and 10. From the results, we can observe that the Cache invalidation time overhead increases along with the cache size. We can also observe that the Cache invalidation time overhead is more in HCGC algorithm with LRU replacement when compared to HCGC algorithm with support based replacement algorithm. And the Cache invalidation time overhead is more in HCGC algorithm with support replacement when compared to HCGC algorithm with rank based replacement algorithm. However, the speculative read algorithm does not employ any cache synchronization or invalidation protocol, so cache invalidation time overhead problem will get resolved.

6. CONCLUSION

Cloud computing systems use distributed file system at the back end for storing large data in a scalable manner. Improving the read operations performance of the distributed file system is one of the key research issues. We proposed a novel speculative read algorithm for hierarchical local and global caching environment to improve the read operation performance in the distributed file system. Experimentation is done by considering the presence of local, global, collaborative local and collaborative global caches. Experimental results shows that the proposed speculative read algorithm can reduce the average read access time of the distributed file system in comparison with the algorithm that does not carry out speculative executions.

REFERENCES

1. William A. Wulf and Sally A. McKee, **Hitting the memory wall: Implications of the obvious**, ACM SIGARCH Computer Architecture News 23, pp.20–24, 1 March 1995.
2. Griffioen, J. and Appleton, R., **Reducing File System Latency Using a Predictive Approach**, Proceedings of the 1994 USENIX Annual Technical Conference, Boston, 6-10 June 1994.
3. [R. Hugo Patterson, **Informed prefetching and caching**, PhD thesis, Carnegie Mellon University, December 1997.
4. M.M.A. Assaf, X. Jiang, X. Qin, M.R. Abid, M. Qiu, J. Zhang, **Informed prefetching for distributed multi-level storage systems**, Journal of Signal Processing Systems, vol. 90, no. 4, pp. 619-640, 2018.
5. D. Muntz and P. Honeyman , **Multi-Level Caching in Distributed File Systems**, Proc. USENIX Winter Conference, 1992.
6. Rafael Alonso and Matthew Blaze, **Dynamic hierarchical caching for large-scale distributed file systems**, Proceedings of the Twelfth International Conference on Distributed Computing Systems, June 1992.
7. Michael D. Dahlin , Randolph Y. Wang , Thomas E. Anderson , David A. Patterson, **Cooperative caching: using remote client memory to improve file system performance**, Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation, Monterey, California, pp.267-280, November , 1994.
8. Gurmeet Singh, Puneet Chandra and Rashid Tahir, **A Dynamic Caching Mechanism for Hadoop using Memcached**, in proceedings of Tahir 2012 ADC,2012.
9. Meenakshi Shrivastava, Hadoop-**Collaborative Caching in Real Time HDFS**, Rochester Institute of Technology, Rochester, NY, USA. Dec 2012.
10. Rathnamma Gopisetty T Ragunathan C.Shobha Bindu, **Support-Based Prefetching Technique for**

Hierarchical Collaborative Caching Algorithm to Improve the Performance of a Distributed File System, Seventh International Symposium on Parallel Architectures, Algorithms and Programming, IEEE,2168-3042/15 , 2015.

11. D. I. August, D. A. Connors, S. A. Mahlke, J.W. Sias, K. M. Crozier, B.-C. Cheng, et al., **Integrated predicated and speculative execution in the IMPACT EPIC architecture**, In Proceedings of the 25th Annual International Symposium on Computer Architecture, pp. 227-237, 1998.
12. I.-Cheng K. Chen, C.-C. Lee, and T. N. Mudge, **Instruction prefetching using branch prediction information**, In Proceedings of the International Conference on Computer Design VLSI in Computers and Processors (ICCD'97), pp.593–601, 1997.
13. A. N. Eden et al., The **YAGS branch prediction scheme**, In Proceedings of the 31st Annual ACM/IEEE International Symposium on Micro architecture (MICRO'98), pp.69-77, 1998.
14. Jose Gonzalez and Antonio Gonzalez, **Speculative execution via address prediction and data prefetching**, In Proceedings of the 11th International Conference on Supercomputing (ICS'97), pp.196-203, 1997.
15. Mikko H. Lipasti, Christopher B.Wilkerson, and John Paul Shen, **Value locality and load value prediction**, ACM SIGPLAN Notices 31, 9 (1996), pp.138-147, 1996.
16. Avi Mendelson and Freddy Gabbay, **Speculative Execution Based on Value Prediction**, Technical Report. EE Department TR 1080, Technion-Israel Institute of Technology, 1996.
17. G. S. Tyson, **The effects of predicated execution on branch prediction**, In Proceedings of the 26th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'94), pp.196-206, 1994.
18. Tullsen D. M., Eggers S.J. and Levy H.M, Simultaneous **Multithreading: Maximizing On-Chip Parallelism**, In Proc.of the 22nd Int. Symp. On Computer Architecture, pp. 392- 403, 1995.
19. Marcuello P., González A. and Tubella J., **Speculative Multithreaded Processors**, 12th Int. Conf. on Supercomputing,Melbourne, Australia, pages 77-84, July 1998.
20. Roth and Sohi G.S., **Speculative Data-Driven Multithreading**, In Proc. of the 7th. Int. Symp. On High Performance Computer Architecture, pp. 37-48, 2001.
21. E. B. Nightingale, P. M. Chen, and J. Flinn, **Speculative execution in a distributed file system**, In Proceedings of the Twentieth ACM Symposium on Operating Systems Principles, SOSP'05, pp. 191-205, New York, NY, USA, 2005.
22. Fay Chang and Garth A. Gibson, **Automatic I/O Hint Generation through Speculative Execution**, In Operating Systems Design and Implementation, pp. 1-14, 1999.
23. Chen Q, Liu C, Xiao Z, **Improving Map Reduce performance using smart speculative execution strategy**, IEEE Transactions on Computers , pp.954-967,2014.
24. T. Phan, G. Pallez, S. Ibrahim, P. Raghavan, **A New Framework for Evaluating Straggler Detection Mechanisms in MapReduce**, ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 2019.
25. X Huang, C Li, Y Luo., **Optimized Speculative Execution Strategy for Different Workload Levels in Heterogeneous Spark Cluster**, Proceedings of the 4th International Conference on Big Data and Computing, pp. 6-10, May 2019.
26. S. Podlipnig and L. B. Osz, **A Survey of Web Cache Replacement Strategies**, ACM Computing Surveys, vol. 35, no. 4, pp. 374-398, 2003.
27. S. Jiang and X. Zhang, **LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance**, In Proc. ACM SIGMETRICS Conf., 2002.
28. Corbat, F. J, **A paging experiment with the Multics system**, In Honor of P. M. Morse, pp. 217-228, MIT Press, Also as MIT Project MAC Report MAC-M-384, 1969.
29. Song Jiang, Feng Chen, Xiaodong Zhang, **CLOCK-Pro: an effective improvement of the CLOCK replacement**, Proceedings of the annual conference on USENIX Annual Technical Conference, Anaheim, CA, pp.35-35, April 10-15, 2005.
30. Bansal, S. and Modha, D. S, **CAR: Clock with Adaptive Replacement**, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST-04), San Francisco, pp. 187-200, 2004.
31. Megiddo, N. and Modha, D. S., **ARC: A Self Tuning, Low Overhead Replacement Cache**, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST-03), San Francisco, pp. 115-130,2003.
32. Jongmoo Choi , Sam H. Noh , Sang Lyul Min , Yookun Cho, **An implementation study of a detection-based adaptive block replacement scheme**, Proceedings of the annual conference on USENIX Annual Technical Conference, Monterey, California, pp. 239-252, June 06-11, 1999.
33. Jongmoo Choi , Sam H. Noh , Sang Lyul Min , Yookun Cho, **Towards application/file-level characterization of block references: a case for fine-grained buffer management**, Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Santa Clara, California, USA, p.286-295, June 18-21, 2000.
34. Sabeghil, M. and Yaghmaee, M. H., **Using fuzzy logic to improve cache replacement decisions**, International Journal of Computer Science and Network, 2006.
35. Davood Akbari Bengar, Ali Ebrahimnejad, Homayun Motameni, Mehdi Golsorkhtabaramiri, **A page replacement algorithm based on a fuzzy approach to improve cache memory performance**, Journal of Soft Computing, 24:955-963, 2020.
36. Aimtongkham, Phet, Chakchai So-In, and Surasak Sanguanpong, **A novel web caching scheme using hybrid least frequently used and support vector machine**, Computer Science and Software Engineering

(JCSSE), and also in 13th International Joint Conference on. IEEE, 2016.

37. Wang Y, Yang Y, Han C, Ye L, Ke Y, Wang Q, **LR-LRU: a PACS-Oriented intelligent cache replacement policy**, in IEEE Access, vol. 7, pp. 58073-58084, 2019.
38. [38] a T, Qu J, Shen W et al., **Weighted greedy dual size frequency based caching replacement algorithm**, in IEEE Access, vol. 6, pp. 7214-7223, 2018.
39. Chao W., **Web cache intelligent replacement strategy combined with GDSF and SVM network re-accessed probability prediction**, J Ambient Intell Human Comput 11, pp.581-587, 2020.
40. R. Agrawal and R. Srikant, **Fast Algorithms for Mining Association Rules**, Proc. 20th Int. Conf. on very Large Databases (VLDB 1994), Santiago de Chile), pp.487-499. Morgan Kaufmann, San Mateo, CA, USA, 1994.
41. Yoon, S.-D., Jung, I.-Y., Kim, K.-H., Jeong, C.-S., **Improving hdfs performance using local caching system**, In Second International Conference On Future Generation Communication Technology (FGCT), pp. 153-156. IEEE 2013.
42. Zhang, J., Li, Q. & Zhou,W., **HDCache: A Distributed Cache System for Real-Time Cloud Services**, J Grid Computing, vol. 14, pp.407-428 ,2016.
43. G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica, **PACMan: Coordinated Memory Caching for Parallel Jobs**, In USENIX NSDI, 2012.
44. The Global Memory System (GMS) Project. http://www.cs.washington.edu/homes/levy/gms/.
45. The NOW Project. http://now.cs.berkeley.edu.
46. Mace, C., Limited, S.C., **Latency on a switched ethernet network**. 2014. URL https://w3.siemens.com/mcms/industrial-communication/en/ruggedcommunication/Documents/AN8.pdf.
47. Konstantin V.Shvachko, **HDFS Scalability: The Limits to Growth**, The USENIX Magzine April, Volume 35, Number 2, 2010.
48. Yash Pal, Member, IAENG, **An Analog Method to Study the Average Memory Access Time in a Computer System**, Proceedings of the World Congress on Engineering, London, U.K Vol II, WCE 2013, July 3 - 5, 2013.
49. Wenting Tang, Yun Fu, Ludmila Cherkasova, and Amin Vahdat, **Medisyn: A synthetic streaming media service workload generator**, In NOSSDAV. ACM, 2003.
50. A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, **Deepcache: A deep learning based framework for content caching**, In Netw.Meets AI & ML. ACM, Budapest, Hungary, August 20-24, pp.48-53, 2018.
51. Rathnamma Gopisetty T Ragunathan C.Shobha Bindu, **Improving Performance of a Distributed File System using Hierarchical Collaborative Global Caching Algorithm with Rank Based Replacement Technique**, Article accepted for publication by International Journal of Communication Networks and Distributed Systems, IJCNDS 35604, 03 Mar 2020
52. J.E. Smith, **A study of branch prediction strategies**, ISCA Conference Proceedings, 1981, New York, pp.135-148.
53. J. Šilc, T. Ungerer, B. Robic, **Dynamic branch prediction and control speculation**, Int. Journal of High Performance Systems Architecture, Vol.1(1), pp.2-13, 2007.
54. Reddy, P.K., Kitsuregawa, M. **Speculative locking protocols to improve performance for distributed database systems**, in IEEE Trans. Knowl. Data Eng. 16(2), pp. 154-169, 2004.
55. Ragunathan, T., Krishna, R.P. **Improving the performance of read only transactions through asynchronous speculation**, In: SpringSim Conference Proceedings, Ottawa, pp. 467-474, 2008.
56. Sasak-Okoń A. Wyrzykowski R., Deelman E., Dongarra J., Karczewski K. **Modifying Queries Strategy for Graph-Based Speculative Query Execution for RDBMS**. In: Lecture Notes in Computer Science, vol 12043. Springer, Cham, 2020.