International Journal of Computer Applications (0975 - 8887)
Volume 121 - No.3, July 2015

Multi Level Caching and Anticipated Parallel Processing-based Algorithm for Improving the Performance of the Distributed File System

B. Rangaswamy
Sri Krishnadevaraya University
Ananthapur, Andhra Pradesh, India

N. Geethanjali
Sri Krishnadevaraya University
Ananthapur, Andhra Pradesh, India

T. Ragunathan
ACE Engineering College
Hyderabad, India

ABSTRACT

Large amount of data is getting generated due to the extensive use of web applications by billions of users around the globe. The organizations which has deployed web applications are pondering over solutions for scalable storage and faster access of large data. Distributed file systems (DFSs) have been emerged as efficient storage solutions so that the data can be stored and accessed efficiently. Modern cloud-based data centers have been using the DFS as main storage component. Improving the performance of read operations have become important as most of users of the web applications carry out read operations in the web. Caching and speculation-based approaches are proposed in the literature for improving the performance of read operations in the DFS. In this paper, we have proposed an anticipated parallel processing-based algorithm which considers the presence of multi level caches namely local cache, local cache of neighbouring node (nearby cache) and global cache. We have carried out the performance evaluation of the algorithms using mathematical analysis and simulation modeling. The results of the analysis indicate that the proposed algorithm performs better than the speculation-based algorithm proposed in the literature.

General Terms:

Algorithms.

Keywords

Distributed file system,Speculation, Multi level caching, Performance evaluation

1. INTRODUCTION

More and more users have turned to use web applications to carry out their daily activities. Smart mobile phones have been emerged as one of the important and inseparable gadget in the life of the people. Lot of audio, video, image and text data are generated and communicated using these mobile phones have made significant changes in the life style of the people. Organizations which deploy web applications for the use of people are pondering over the issue of storage and subsequent access of the data. Large storage capacity and faster access of data are the two important factors considered by these organizations for choosing a storage solution to get adapted.

Distributed file systems (DFSs) are currently being used in most of the search engines for storage and faster access of both structured and unstructured data. Cluster-based file systems like Hadoop DFS (HDFS)[9] has become more popular to cater the needs of the data intensive applications. With the support of parallel and distributed programming framework namely MapReduce framework, the HDFS has become an ideal storage solution for data intensive applications. Note that, most of the web applications in this type of environment performs read operations on the data and less frequently the data is updated. So, improving the the performance of the read access has become one of the major research issues in the emerging Big Data scenario.

Many pre fetching and client side caching techniques are proposed in the literature for improving the performance of the read operations in the DFS. [2] [5] [8] [6] [7][4], [3] and [1]. Pre fetching technique is used for pre fetching the requested data and store the same in the local caches maintained in the data nodes. Caching technique permits the client application program to read the requested data from the local cache (client side cache) if it is available there; Otherwise the requested data will be fetched from the file server system's disk. The advantage with caching technique is that it will reduce the number of disk accesses and network communication overhead.

Since the data is distributed and stored in the DFS, researchers have made efforts to carry out parallel or speculative processing for improving the performance of the read operations in the DFS. In [6], the authors proposed a speculation-based read technique for improving the performance of the read operations in the DFS. This technique considers the presence of the local caches in the data nodes for proposing speculation-based read algorithm. In the literature cooperative and collaborative caching algorithms are proposed for improving the performance of the DFS.

In this paper, we have proposed an anticipated parallel processing-based algorithm which considers the presence of multi level caches namely local cache, local cache of neighbouring node (nearby cache) and global cache and the possibilities of parallel executions. We have carried out the performance evaluation of the algorithms using mathematical analysis and simulation modeling. The results of the analysis indicate that the proposed algorithm performs better than the speculation-based algorithm proposed in the literature.

This paper is organized as follows. In the next section, we discuss the architecture of the DFS that we have considered for proposing our algorithm. In section 3, we discuss the details of the proposed algorithm. In section 4, we describe the performance evaluation of the algorithms through mathematical and simulation modeling. Section 5 concludes the paper.

2. ARCHITECTURE OF DISTRIBUTED FILE SYSTEM

In this paper, we have considered cluster-based DFS. This DFS consists of one name node (NN) where the meta data (global directory) of the files stored in the DFS are available. A secondary NN may be used optionally to support fault tolerance purpose. In this type of DFS, one or more data nodes (DNs) are present. These DNs are used for two important purposes. The first purpose of a DN is to store the data and the second purpose is to execute client application programs. Each DN maintains a local cache (client side cache). Note that, the operations of the local cache are managed by the cache manager program running in that DN. This cache manager maintains a cache directory where the meta data of the files stored in the local cache are available. In this cache directory, address of the nearby DN is also stored We have considered that only file level caching is followed in the DFS. The client application program getting executed in a DN can access the file blocks with the help of the DFS client program running in that DN. The requests given by the DFS client program will be served by the DFS server program running in the NN. The data nodes and NN will communicate with each other quite frequently to know the functioning status of the data nodes and to know the file blocks stored in the data nodes. This communication is essential for balancing the work load of the data nodes. Note that, a file is replicated three times and these replicas are kept in three different DNs so that the reliability feature can be implemented in the DFS. Note that, replicating files is also useful for improving the performance of the DFS by providing the opportunities for parallel processing. We have also considered that the DFS maintains a global cache in a separate server system.

3. PROPOSED ALGORITHM

In this section, we cove the assumptions that we have made for proposing the algorithm. The details of the proposed algorithm are covered next.

3.1 Assumptions

(i) Cache synchronization or invalidation protocol is not used in order to avoid the communication overhead.
(ii) The DFS client program communicates with the NN to collect the addresses of the DNs where the requested file copies are available.
(iii) Caching is done only during read operations and write operations do not initiate any caching activity.
(iv) Least recently used (LRU) policy is followed to replace a file in the cache. The local, global and nearby caches present in the DFS follow this LRU policy.

3.2 Proposed Algorithm

Whenever a client application program running in a DN requests for a file the local cache manager simultaneously verifies with the local cache, nearby cache and global cache by communicating with the appropriate cache managers by creating anticipated parallel executions (APEs) . Simultaneously, the DFS client program running in that DN communicates with the NN to read the meta data (addresses of the data nodes where the file is available and time stamp of the file). Next, the time stamp value returned by the NN is checked up with the time stamp values of the copies in the local, nearby and global caches. If the time stamp value of any one of the cached copies is matching then the APE meant that cache will be allowed to continue and the APEs meant for the remaining caches will be terminated. If time stamp value is not matching with that of all the cached copies then the file content will be fetched from one of the DNs which is very near to the host DN where that file content is available. Next, we describe the algorithm in detail.
The proposed algorithm:
This algorithm creates four threads namely main thread to read the file from the disk and anticipated parallel executions (APEs) for reading from local, global and nearby caches.

```
/* A client application program (CAP) running in a data node (DN1) has issued read request to read the contents of a file F1 */
/* The threads MT, APE1, APE2 and APE3 are created. After the creation these threads are executed simultaneously. */
Step 1. (a) APE1 reads F1 from local cache if F1 is available in that cache and time stamp value of F1 is T1; else APE1 is terminated.
(b) APE2 reads F1 from the nearby cache if F1 is available in that cache and time stamp value of F1 is T2; else APE2 is terminated.
(c) APE3 reads F1 from the global cache if F1 is available in that cache and time stamp value of F1 is T3; else APE3 is terminated.
(d) The DFS client program running in DN1 contacts the name node (NN) to get the addresses of the data nodes (DNs) where F1 is stored. Time stamp of F1 given by NN is T. (This is the main thread of execution and we name this as MT)
/* All the four operations specified in this step are executed in parallel. */
Step 2. (a) If T1 is not equal to T then APE1 is terminated; else APE2, APE3 and MT are terminated. After reading is completed APE1 will return the F1 content to the CAP.

(b) If T2 is not equal to T then APE2 is terminated; else APE1, APE3 and MT are terminated. After reading is completed APE2 will return the F1 content to the CAP.

(c) If T3 is not equal to T then APE3 is terminated; else APE1, APE2 and MT are terminated. After reading is completed APE3 will return the F1 content to the CAP.

(d) If APE1, APE2 and APE3 (all anticipated parallel executions) are terminated then (i) The DFS client program contacts one of the nearest data node among DNs to read the contents of F1 from the disk storage system of that data node.
(ii) The content of F1 is delivered to CAP and copy of F1 is stored in the local cache of DN1.

Step 3. Stop.
```

4. PERFORMANCE EVALUATION

4.1 Assumptions

We have made the following assumptions for mathematical analysis and simulation modeling.
(i) File block size is 4 KB.
(ii) Data nodes and name nodes are connected in a local area network.
(iii) Average communication time required to transfer a 4 KB block from a remote node to the host node is 4 ms.

(iv) Meta data transfer time from name node to data node is 0.125 ms.
(v) Time required for Reading 4 KB of data from local disk is 12 ms.
(vi) Main memory access time is 0.0006 ms.
(vii) Local cache hit ratio is called as *lc*.
(viii) Global cache hit ratio is called as *gc*.
(ix) Cache hit ratio of nearby cache is called as *nc*.
(x) The client application program is getting executed in the data node CDN.
Next, we describe the results of our mathematical analysis of the proposed and speculation-based algorithms. In the last subsection, we discuss the results of the simulation experiments.

4.2 Mathematical Analysis

Average read access time for a 4 KB data block of DFS (without caching and anticipated parallel processing) = time required to access name node to collect meta data + reading 4 KB data block from a specific data node from the disk + reading 4 KB data block from the main memory of source data node + transferring the 4 KB data block to the destination data node + time required for transferring the data block to client's address space).
Average read access time for a 4 KB data block of DFS (with speculation-based approach) = lc * (time required to access the local memory + time required to access name node to collect time stamp) + (1-*lc*) * (time required to access the local memory + time required to access name node to collect time stamp+ time required to access name node to collect meta data + reading 4 KB data block from a specific data node from the disk + reading 4 KB data block from the main memory of source data node + transferring the 4 KB data block to the destination data node + time required for transferring the data block to client program's address space).
For the proposed approach we have to calculate the time required to access 4 KB of data block from global cache or nearby cache which is equivalent to time required to access main memory + time required to transfer the data from the remote node to the local node + time required for transferring the data block to client's address space.
Average read access time for a 4 KB data block of DFS (proposed approach = *lc** (time required to access name node to collect time stamp) + *nc* * (time required to transfer the 4 KB data from the remote memory to CDN's memory + time required for transferring the data block to client program's address space and local cache) + *gc* * (time required for transferring data block to CDN from the global cache + time required for transferring the data block to client program's address space) + (1-*lc*- *nc* - *gc*) * (time required to access name node to collect meta data + reading 4 KB data block from a specific data node's the disk + reading 4 KB data block from the main memory of source data node + transferring the 4 KB data block to the CDN + time required for transferring the data block to client's address space and local cache)
Based on the formulas discussed above we have evaluated the performance of the proposed anticipated parallel processing-based and the speculation-based algorithms proposed in the literature.

In Figure 1, we have fixed the *lc* and *nc* values as 0.2 and observed the performance of the algorithms by varying the *gc* values from 0.1 to 0.5. For the *gc* values 0.1 and above the proposed algorithm (APMC) performs better than the speculation-based algorithm (SP). As we use multi level caches, the hit ratio has been improved and hence the proposed algorithm performs better than the speculation-based algorithm proposed in the literature.

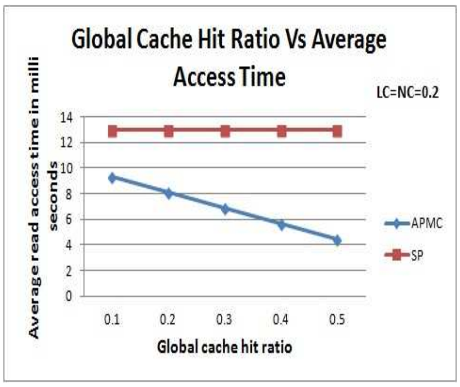

Fig. 1. Global cache hit ratio versus Average access time.

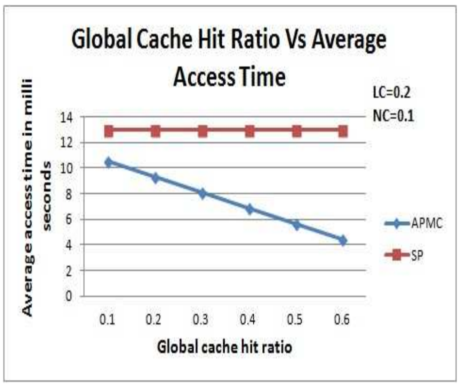

Fig. 2. Global cache hit ratio versus Average access time

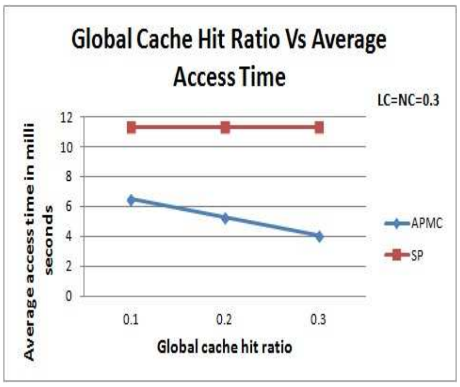

Fig. 3. Global cache hit ratio versus Average access time

In Figure 2, we have fixed the *lc* and *nc* values 0.2 and 0.1 respectively and observed the performance of the algorithms by varying the *gc* values from 0.1 to 0.6. We can observe that for *gc* values 0.1 and above the proposed APMC performs better than the SP.
In Figure 3, we have fixed *lc* and *nc* values as 0.3 and observed the performance of the algorithms by varying the *gc* values from 0.1 to 0.3. For the *gc* values 0.1 and above the proposed algorithm (APMC) performs better than the speculation-based algorithm (SP).
We can observe similar trends in Figures 4 and 5.

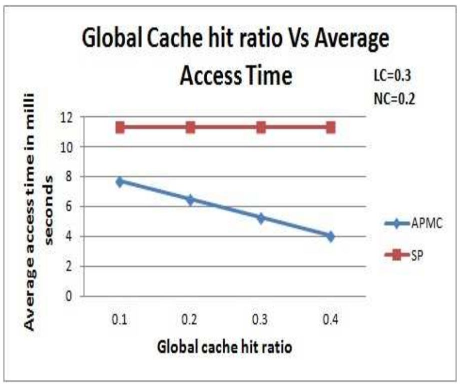

Fig. 4. Global cache hit ratio versus Average access time

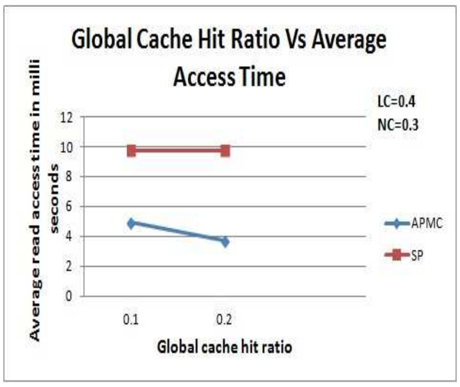

Fig. 5. Global cache hit ratio versus Average access time

4.3 Simulation Results

We have conducted simulation experiments for the proposed and and anticipated parallel processing-based algorithm. We have fixed the number of files available in the DFS, the number of cache blocks maintained in the local, nearby and global caches and varied number of blocks available in the file to measure the performance.
We can see the performance of the proposed algorithm (APMC) and the speculation-based algorithm proposed in the literature (SP) in Figure 6. Here, we have fixed the number of files present in the DFS as 50, capacity of local and nearby caches as 100 blocks and capacity of global cache as 1000 blocks. The number of blocks present in the files are varied from 25 to 100. We can observe that the proposed APMC algorithm requires less average read access time than SP for all the cases.
The performance of the proposed APMC and SP algorithms is depicted in Figure 7 for a different scenario. Here, we have fixed the number of files present in the DFS as 50, capacity of local and nearby caches as 200 blocks and capacity of global cache as 2000 blocks. We have varied number of blocks present in the files from 25 to 100 and observed the performance. We can observe that, APMC requires less average read access time than SP for all the cases.
Figure 8 shows the performance of the proposed algorithm (APMC) and SP algorithms. Here, we have fixed the number of files present in the DFS as 50, capacity of local and nearby caches as 300 blocks and capacity of global cache as 3000 blocks. We have varied number of blocks present in the files from 25 to 100 and ob-

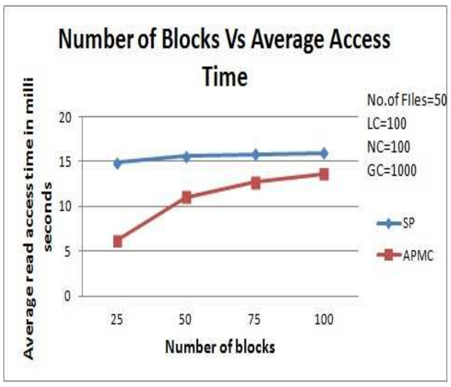

Fig. 6. Number of blocks versus Average access time.

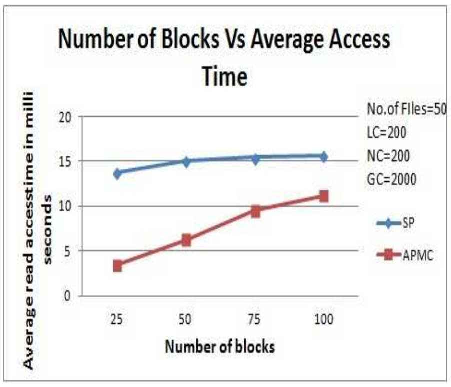

Fig. 7. Number of blocks versus Average access time.

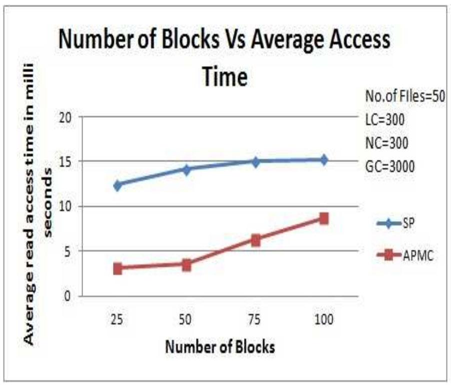

Fig. 8. Number of blocks versus Average access time.

served the performance. We have observed that APMC requires less average read access time than SP for all the cases.
We can observe similar trends in Figures 9 and 10.

Overall, we conclude that the proposed algorithm performs better than the speculation-based algorithm proposed in the literature. This performance enhancement is due to the usage of multi level caches and anticipated parallel processing.

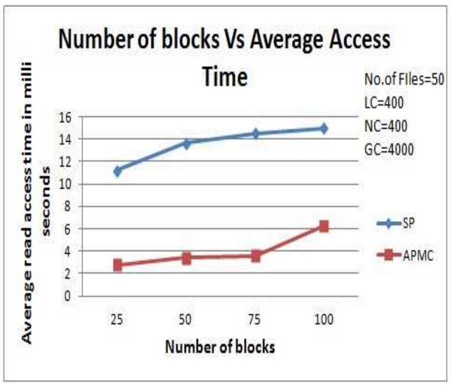

Fig. 9. Number of blocks versus Average access time.

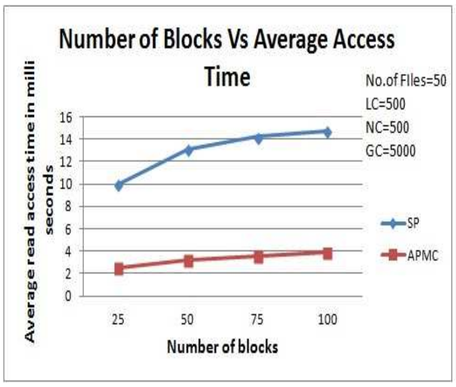

Fig. 10. Number of blocks versus Average access time.

5. CONCLUSION

Distributed file systems are used to provide scalable storage solutions in cloud computing systems. Improving the performance of the read operations in the distributed file systems is one of the important research issues. In this paper, we have proposed an anticipated parallel processing-based read algorithm for improving the performance of the distributed file system by considering the presence of multi level caches. We have done simulation experiments and carried out mathematical analysis. The results indicate that the proposed algorithm performs better than the speculation-based algorithm proposed in the literature. Modern computer systems which are used in distributed environment support multi-core processors which provide abundant processing power and hence implementation of our algorithm in distributed file systems is feasible. As a part of future work, we wish to implement the proposed algorithm in Hadoop distributed file system to prove its efficiency.

6. REFERENCES

[1] B. S. S. X. Chen, Y. Data access history cache and associated data prefetching mechanisms. In *Proceedings fo the AMC/IEEE Conference on Supercomputing, Reno, NV*, pages 1–12, November 2007.

[2] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[3] V. O. G. S. F. Isaila, G. Malpohl and W. Tichy. Integrating collective i/o and cooperative caching into the clusterfile parallel file system. In *In the 18th annual international conference on Supercomputing*, page 5867, June 2004.

[4] S. Jiang, F. Petrini, X. Ding, and X. Zhang. A locality-aware cooperative cache management protocol to improve network file system performance. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 42–42, 2006.

[5] W.-k. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and S. Tideman. Collective caching: application-aware client-side file caching. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 81–90. IEEE, 2005.

[6] 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, pages 191–205, New York, NY, USA, 2005. ACM.

[7] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, pages 79–95, New York, NY, USA, 1995. ACM.

[8] P. Sarkar and J. Hartman. Efficient cooperative caching using hints. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation*, OSDI '96, pages 35–46, New York, NY, USA, 1996. ACM.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.