International Journal of Computer Applications (0975 8887)
Volume 116 - No. 8, April 2015

Improving the Performance of Read Operations in Distributed File System using Global Cache Mechanism and Anticipated Parallel Processing

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

Distributed file system (DFS) is the main storage component of a distributed system (DS). DFS provides scalable storage to the DS and it is also the main component of any cloud storage system. Improving the performance of read operations in a DFS is very important as many web-based applications deployed in the cloud perform read operations quite frequently. Many pre fetching and caching techniques are used to improve the performance of the read operations in a DFS. Speculation-based techniques have also been proposed in the literature for reducing the read access time. In this paper, we propose a novel anticipation-based parallel processing technique by considering global caching technique for a DFS. The results of performance analysis through mathematical and simulation modeling indicate that the proposed technique improves the performance of read operations better than the speculation-based algorithm proposed in the literature.

General Terms:

Algorithms.

Keywords:

Distributed System, Speculation, Asynchronous reading Performance

1. INTRODUCTION

Many IT organizations around the globe deploy their web applications in cloud environment in order to improve the availability, scalability and performance of their applications. A distributed file system (DFS) is the main storage component of a cloud computing system. Google file system (GFS) [4] and Hadoop distributed file system (HDFS) [10] are the important distributed file systems which are currently used in the enterprise level cloud storage systems.

It is found that very frequently the client programs perform read operations and infrequently they perform update operations in a cloud storage system. So, improving the performance of the read operations in a DFS is an important requirement in the emerging cloud computing scenario. Many techniques have been proposed in the literature for improving the performance of the read operations in the DFS. Pre fetching and client-side caching [2] [6] are the two important techniques used for improving the performance of the read operations in the DFS [9] [7] [8][5], [3] and [1]. These techniques allow a client program to read the requested data blocks from the main memory of the system where it is getting executed by avoiding disk I/O.

A speculation-based technique has been proposed for improving the performance of read access in the DFS [7]. This technique permits the read requests of the client application program (CAP) to read the requested data blocks from the cache (LC) maintained at the data node where that CAP is getting executed provided the requested blocks are available in LC. If the requested data blocks are not available in LC or if the data blocks are obsolete, then the CAP has to read the required data blocks from the disk(s) of the server system(s) where the file system is kept. This technique follows synchronous approach and hence the CAP has to wait for the response given by the server system. In our proposed technique we make use of this waiting time of the client to carry out additional anticipated execution by reading the data blocks available in the global cache maintained in a separate system in the DFS environment. Then, based on the time stamp value returned by the server system, it will be decided to terminate which execution. If both the executions are not matching with the time stamp value returned by the server then the CAP has to read the data blocks from from the disk(s) of the server system(s) where the file system is kept.

We have carried out mathematical analysis of the algorithms and also we have conducted simulation experiments. The results of both mathematical analysis and simulation modeling indicate that the proposed algorithm improves the performance of read operations better than the speculation-based algorithm proposed in the literature.

This paper is organized as follows. In the next section, we discuss our proposed approach in detail. In section 3, we have done the detailed performance evaluation of the algorithms through mathematical and simulation modeling. Section 4 concludes the paper.

2. PROPOSED ALGORITHM BASED ON ANTICIPATED PARALLEL PROCESSING USING GLOBAL CACHE MECHANISM

In this section, first, we discuss the architecture the DFS which we have considered for proposing our algorithm. Next, we discuss the advantages of the anticipated parallel execution and then we describe the proposed algorithm.

2.1 Architecture of Distributed File System

We have considered that the DFS consists of number of data nodes organized in multiple racks forming a cluster. We have also considered that data is stored in data nodes and meta data (global directory) is stored in the name node and client application programs are executed in the data nodes. In order to support reliability feature, this DFS supports secondary name node. In case, the name node fails, data can be recovered by using secondary name node. The data nodes, name node and secondary name nodes are placed in the racks of the cluster. The client programs executed in the data nodes communicate with the name node by using the DFS client program which is also getting executed in the data nodes. We have assumed that the DFS maintains a separate node where a global cache is maintained and the data nodes maintain their own local caches.

2.2 Anticipated Parallel Execution

In anticipated parallel execution method, we execute a task before it is known whether that task will be required at all. Later, the executed task will be allowed to continue its work or it will be terminated and the work is undone based on the results produced. Anticipated parallel execution method reduces the delay in execution and hence the performance of the system can be improved. Modern pipelined processors use anticipated parallel executions for improving the throughput performance of the CPU.

2.3 Proposed Algorithm

In this subsection, we describe regarding the client-side caching and global caching techniques followed in the DFS. Next, we discuss the three parts of the proposed algorithm in detail.
Assumptions:

The proposed DFS consists of a name node, multiple data nodes and a cache node. These nodes are arranged in racks and communication is established through a switched local area network. The name node stores the meta data (global directory) of the DFS.
The data nodes store the data blocks and they are also capable of executing client application programs. The data nodes communicate with the name node and cache node using the DFS client program getting executed in the data nodes. These data nodes also maintain cache in the local main memory where the frequently and recently accessed data blocks of the files by those data nodes are captured. The cache node maintains a global cache in which frequently and recently accessed data blocks of the files by the data nodes present in the DFS are captured. We also consider that three copies of a file is maintained in three different data nodes and file level caching technique is implemented in both the local cache and the global cache. Each data node maintains a cache directory (CD) in which the meta data of the cached files are stored and the cache node also maintains global cache directory (GCD) in which the meta data of the files accessed by various data nodes are stored. Note that, we have considered that cache coherence protocol is not implemented in this system in order to avoid communication and other overheads.
Two parts of the algorithm:
Our algorithm consists of two parts. The first part describes the steps to be followed for the main thread of execution of the *read* procedure of the DFS. The second part describes the steps to be followed by the anticipated executions (AE1 and AE2).

/* A client program CL running in DN1 has issued *read* procedure to read the contents of the file F1 */
I) Algorithm for main thread of execution:

Step 1. If AE1 and AE2 are not created or AE1 and AE2 are terminated then the following steps are executed or else go to Step 5.

Step 2. DFS client running in DN1 contacts the name node to get the addresses of the data nodes (DNs) where F1 is stored.

Step 3. 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.

Step 4. The content available in F1 is transferred to CL and also cached in CL.

Step 5. Stop

(II) Algorithm for anticipated executions (AE1 and AE2):

Step 1. If F1 is available in the local cache, CL can read it (Anticipated parallel execution (AE1)).
Let us consider that time stamp value of this cached copy of F1 is T1. Otherwise go to Step 4.
Step 2. AE1 will continue until the time stamp value of F1 is received from the name node (T2) or its completion.
Step 3. If T1 ¿= T2 then AE1 will be allowed to complete its execution and goto Step 7.; or else AE1 will be terminated.

Step 4. CL verifies the GCD to get the time stamp value of F1 if F1 is cached in the global cache.

Step 5. If F1 is available in the global cache, then CL can read the time stamp value (T3) of F1 from the GCD.

Step 6. If T3 ¿= T2 then Anticipated parallel execution2 (AE2) will be started and it will be allowed to complete its execution;

Step 7. Stop.

(Note that request message is sent to name node to send the time stamp value of F1 and then part II will be started. If F1 is not available in local cache and also in the global cache then part I will be executed.)

3. PERFORMANCE EVALUATION

In this section, first, we discuss regarding assumptions that we have made for evaluating the performance of the algorithms. Next, we discuss the results of the mathematical analysis. Finally, we discuss the simulation results.

3.1 Assumptions

Let us consider that a file consists of only one block and the block size is 4 KB for the purpose of performance evaluation. Here, we have assumed that all the data nodes and the name node are connected in a local area network. We have considered that the average communication delay (ACD) required for transferring

4 KB of data as 4 ms and for transferring time stamp and meta data information as 0.125 ms based on the recent analysis by considering the switched local area network. Also, the average time required to access a 4 KB data block from the disk storage system is 12 milliseconds by considering the latest disk storage devices. Also, we consider that the average time required to access 4 KB of data block from the main memory as 0.0006 ms by considering latest dynamic random access memory technologies. Let us consider that local cache hit ratio as *lc* and global cache hit ratio as *gc*.

3.2 Mathematical Analysis

Average read access time for a 4 KB data block of DFS (without anticipated parallel processing and global caching) = 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). By following above formula the average access time is computed as = 16.13 ms (0.125 ms + 12 ms + .0006 ms + 4 ms + .0006 = 16.1262 ms)

Average read access time for a 4 KB data block of DFS (with speculation - earlier 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).
By following above formula, the average access time is computed as = (16.26 - 16.13 *lc*)ms (Formula 1). Note that, we have ignored the time required to start speculative execution in these calculations.
For the proposed approach we have to calculate the time required to access 4 KB of data block from global cache (remote memory) which is equivalent to time required to access 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. This time is computed as 4.0012 ms (0.0006 ms + 4 ms + 0.0006 ms).
Average read access time for a 4 KB data block of DFS (our anticipated parallel processing and global caching -based approach = *lc** (time required to access the local memory + time required to access name node to collect time stamp) + *gc* * (time required to send the request message to the name node to get the time stamp of F1 + time required for transferring data block to CL from the global cache + time required for transferring the data block to client program's address space) + (1-lc-gc) * (time required to send the request message to the name node to get the time stamp of F1 + 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)
By following above formula, the average access time is computed as = (16.26 - 16.13*lc* - 12.13*gc*)ms (Formula 2)

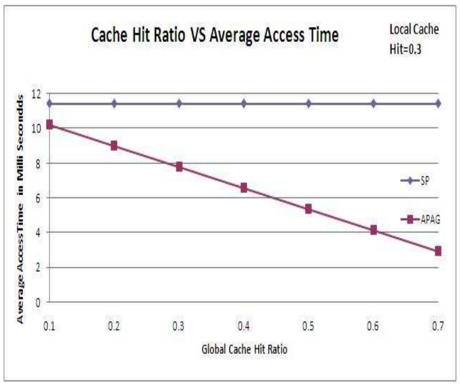

Fig. 1. Global cache hit ratio versus Average access time (local cache hit ratio is 0.3.

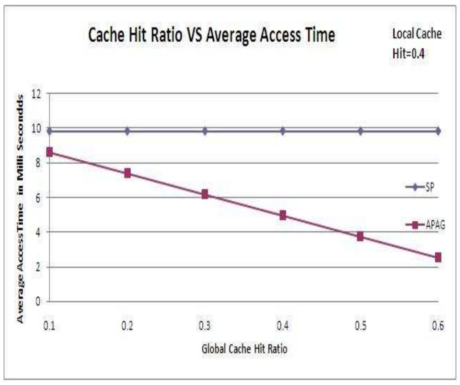

Fig. 2. Global cache hit ratio versus Average access time (local cache hit ratio is 0.4.)

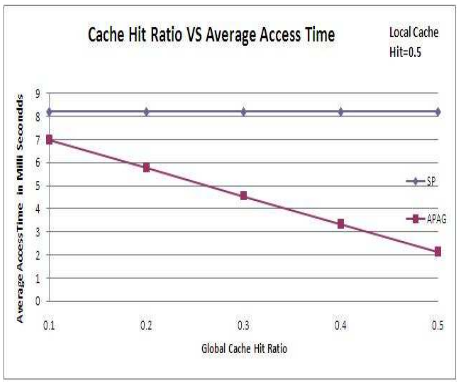

Fig. 3. Global cache hit ratio versus Average access time (local cache hit ratio is 0.5.

Based on the formulas (Formula 1 and Formula 2) discussed above we have evaluated the performance of speculation-based algorithm and our proposed algorithm based on anticipated parallel processing and global caching mechanism by fixing the local cache hit ratio (cl) and by varying the global cache hit ratio (gc).

In Figure 1, we have fixed the lc value as 0.3 and observed the performance of the algorithms by varying the gc values from 0.1

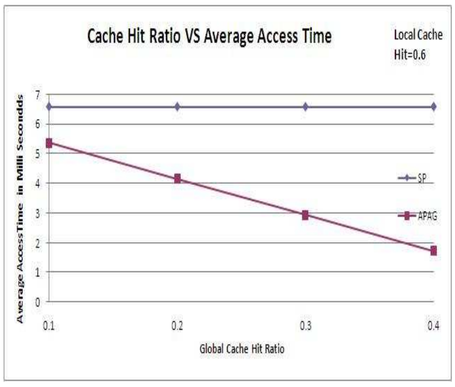

Fig. 4. Global cache hit ratio versus Average access time (local cache hit ratio is 0.6.

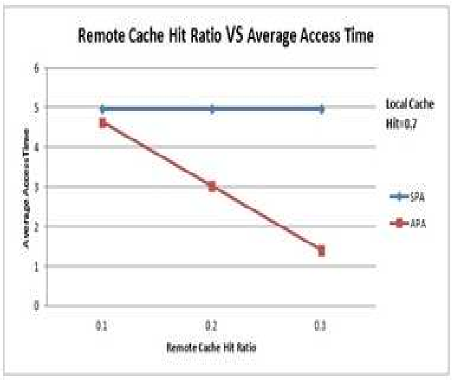

Fig. 5. Global cache hit ratio versus Average access time (local cache hit ratio is 0.7.

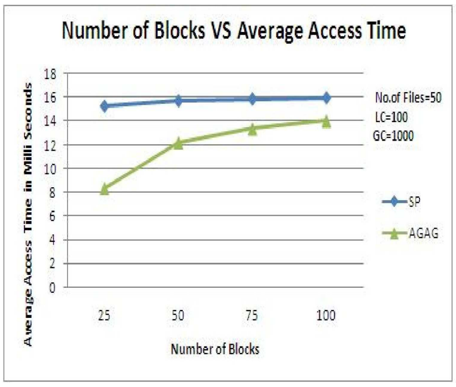

Fig. 6. Number of blocks versus Average access time.

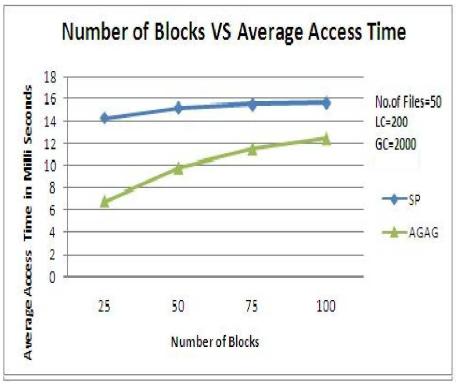

Fig. 7. Number of blocks versus Average access time.)

to 0.7. For the gc values 0.1 and above the proposed anticipated parallel processing and global caching mechanism-based algorithm (APAG) performs better than the speculation-based algorithm (SP). Note that, the performance of speculation-based algorithm is dependent only on lc whereas the the performance of the proposed algorithm is dependent on both lc and gc.
In Figure 2, we have fixed the lc value as 0.4 and observed the performance of the algorithms by varying the gc values from 0.1 to 0.6. We can note that for the cache hit ratio values 0.1 and above the proposed APAG performs better than the SP. Note that, the performance of SP is dependent only on lc whereas the performance of the APAG is dependent on both lc and gc.
In Figure 3, we have fixed lc value as 0.5 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 anticipated parallel processing and global caching mechanism-based algorithm (APAG) performs better than the speculation-based algorithm (SP). We can observe similar trends in Figures 4 and 5.

3.3 Simulation Results

We have developed a simulator for simulating speculation-based, anticipatory parallel processing-based algorithms and by fixing the number of files available in the DFS, the number of cache blocks maintained in the local cache and global cache by varying the number of blocks available in the file.

Figure 6 shows the performance of the proposed algorithm (AGAG) and the speculation-based algorithm proposed in the literature (SP). Here, we have fixed the number of files present in the DFS as 50, capacity of LC as 100 blocks and capacity of GC as 1000 blocks. We have varied number of blocks present in the files from 25 to 100 and observed the performance. We can note that the proposed AGAG requires less average read access time than SP for all the cases.
Figure 7 shows the performance of the proposed algorithm (AGAG) and SP algorithms. Here, we have fixed the number of files present in the DFS as 50, capacity of LC as 200 blocks and capacity of GC 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, AGAG requires less average read access time than SP for all the cases.
Figure 8 shows the performance of the proposed algorithm (AGAG)and SP algorithms. Here, we have fixed the number of files present in the DFS as 50, capacity of LC as 300 blocks and capacity of GC as 3000 blocks. We have varied number of blocks present in the files from 25 to 100 and observed the performance. We can observe that, APAG 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 can perform better than the speculation-based algorithm proposed in the literature by considering the metric ”Average Read Acces Time”.

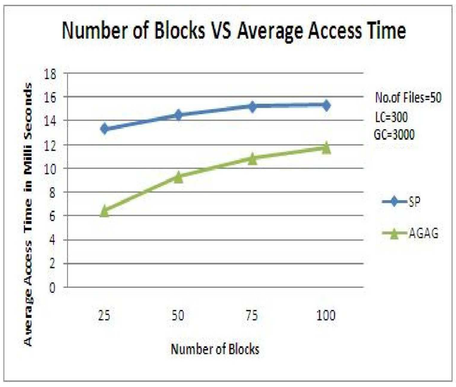

Fig. 8. Number of blocks versus Average access time.

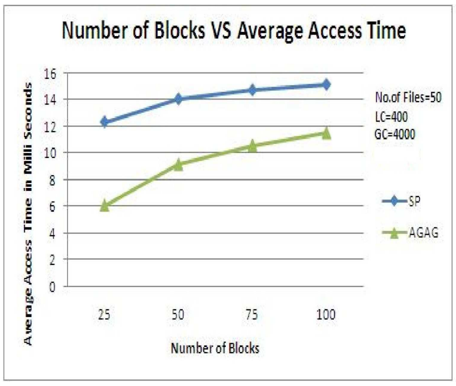

Fig. 9. Number of blocks versus Average access time.

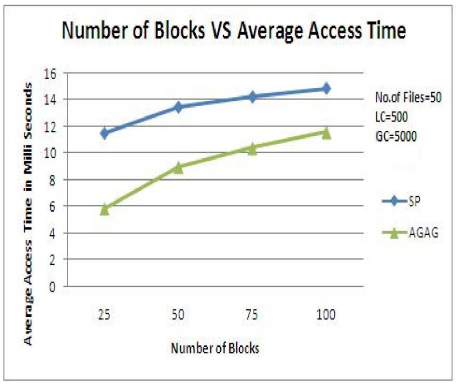

Fig. 10. Number of blocks versus Average access time.

4. CONCLUSION

Distributed file systems provide scalable and reliable storage solutions. Modern cloud computing systems use distributed file system to store and access large amount of data and also to share the data to authorized users. Most frequently read operations are carried out in the cloud computing systems and less frequently update operations are carried out in the system. Hence, improving the performance of read operations is an important research issue.
In this paper, we have proposed a novel anticipation-based parallel processing technique by considering global caching technique for a DFS. The results of performance analysis done through mathematical and simulation modeling indicate that the proposed algorithm improves the performance of read operations better than the speculation-based algorithm proposed in the literature.

5. 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. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.