Global Journal of Computer Science and Technology: B
Cloud and Distributed
Volume 15 Issue 2 Version 1.0 Year 2015
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Improving the Read Performance of the Distributed File System through Anticipated Parallel Processing

By B. Rangaswamy, Dr. N. Geethanjali & Dr. T. Ragunathan
Sri Krishnadevaraya University, India

Abstract- In the emerging Big Data scenario, distributed File systems (DFSs) are used for storing and accessing information in a scalable manner. Many cloud computing systems use DFS as the main storage component. The Big Data applications de-ployed in cloud computing systems more frequently perform read operations and less frequently the write operations. So, improving the performance of read access has become an im-portant research issue in DFS. In the literature, many client side caching with appropriate pre fetching techniques are proposed for improving the performance read access in the DFS. A speculation-based approach which uses client side caching is also proposed in the literature for improving the performance of read access in the DFS. In this paper, we have proposed a new read algorithm for the DFS based on anticipated parallel processing. We have evaluated the per- formance of the proposed algorithm using mathematical and simulation methods and the results indicate that the pro-posed algorithm performs better than the speculation-based algorithm proposed in the literature.

Keywords: distributed system, speculation, asynch- ronous reading performance.

GJCST-B Classification : C.1.4, C.2.4

Strictly as per the compliance and regulations of:

Improving the Read Performance of the Distributed File System through Anticipated Parallel Processing

B. Rangaswamy [α], Dr. N. Geethanjali [σ] & Dr. T. Ragunathan [ρ]

Abstract- In the emerging Big Data scenario, distributed File systems (DFSs) are used for storing and accessing information in a scalable manner. Many cloud computing systems use DFS as the main storage component. The Big Data applications de-ployed in cloud computing systems more frequently perform read operations and less frequently the write operations. So, improving the performance of read access has become an im-portant research issue in DFS. In the literature, many client side caching with appropriate pre fetching techniques are proposed for improving the performance read access in the DFS. A speculation-based approach which uses client side caching is also proposed in the literature for improving the performance of read access in the DFS. In this paper, we have proposed a new read algorithm for the DFS based on anticipated parallel processing. We have evaluated the per- formance of the proposed algorithm using mathematical and simulation methods and the results indicate that the pro-posed algorithm performs better than the speculation-based algorithm proposed in the literature.

Keywords: distributed system, speculation, asynch-ronous reading performance.

I. Introduction

Lot of data (text, images, video and audio) is getting gen-erated due to the extensive use of social media applications. This phenomena is referred as Big Data in the literature.The availability of smart phones which support many at-tractive applications facilitate the users to upload the mul-timedia data into the web in a exible manner. The main problem here is the availability of scalable storage solutions which provide required storage capacity and efficient read and write facilities. Distributed File systems (DFSs) have been emerged as the scalable storage facility for storing Big Data and for accessing them in an efficient manner. Many cloud computing systems use DFS as the main storage component.

Author α : Research Scholar Department of Computer Science and Technology Sri Krishnadevaraya University Ananthapur, Andhra Pradesh, India. e-mail: burujula1971@gmail.com
Author σ : Associate Professor Department of Computer Science and Technology Sri Krishnadevaraya University Ananthapur,Andhra Pradesh, India. e-mail: geethanjali.sku@gmail.com
Author ρ : Professor and Dean Department of Computer Science and Engineering ACE Engineering College Hyderabad, India.
e-mail : ragu_savi@yahoo.com

The Big Data applications deployed in the cloud computing environment more frequently perform read operations and less frequently carry out write operations. Hence, improving the performance of the read operations in a Big Data environment has become one of the important research is-sues. So, for the DFS which is used for storing and accessing Big Data, it is important that it carries out the read access in a faster manner so that Big Data application execution time can be reduced.

The DFS uses disk as the main storage device and data transfer rate of disk is very less in comparison with that of the dynamic or static random access memories used in the computer systems. To reduce the input/output (I/O) access time, many client side caching techniques have been proposed in the literature. These techniques allow the client node to download the requested Files from the server and store the same in the client side cache so that further read re-quests issued by the applications running in the client nodes will be satis read by reading the content from the local cache (client side cache). To avoid stale data problems in the client side caching techniques one of the following methods will be used: (i) Cache synchronization or cache invalidation pro-tocol (ii) Checking up with the server whether the data in the client side cache is valid or stale. If the data available in the cache is stale then the data has to be fetched from the server's disk.

In the literature, a speculation-based technique has been proposed for improving the performance of read access in the DFS [6]. In this technique, the client application reads the data from the local cache and proceed with its execution (speculative execution). Simultaneously, the server systemis also contacted to check whether the data in the local cache is stale or valid by comparing the time stamp values of the cached copy and the copy available in the sever's disk. If the data in the local cache is found to be valid, then the speculative execution is allowed to continue. If the data in the local cache is found to be stale, then the data is read from the server's disk and the speculative execution will be rolled back.

In this paper, we propose anticipated parallel processing-based algorithm which carries out executions by considering the local cache of the node (LN) where the client application program is getting

executed and also the local cache of the node (NN) which is placed near to LN (where the same data is available). Based on the time stamp value available in the server for the data, the cache content of LN or NN will be considered. If the data available in LN and NN are stalethen the data will be read from the server's disk. We have evaluated the performance the proposed algorithm through mathematical analysis and simulation experiments. The re-sults indicate that our proposed algorithm performs better than the earlier speculation-based algorithm proposed in the literature.

This paper is organized as follows. In the next section, we describe the techniques discussed in the literature for im-proving the performance of the DFS. In section 3, we discuss our proposed approach in detail. In section 4, we have done the detailed performance evaluation of the algorithms using mathematical analysis and simulation modeling. Section 5 concludes the paper.

II. Relatedwork

In this section, we discuss First we describe the techniques discussed in the literature for improving the performance of the DFS.

Many client-side caching techniques have been used to im-prove the performance of distributed File systems. A cooper-ative caching technique is discussed in the paper [2]. In this type of technique, the server maintains a directory which stores the details of File blocks stored in each and every local caches available in client nodes. Whenever a client applica-tion program issues read request for a block, First the local cache is verified and then the cache directory maintained in the server is verified to see whether the requested File block is available or not. If the File block is not available in the local cache and in any of the caches maintained in the client nodes then it will be read from server's disk where the DFS is deployed. This technique is suffering from the problem known as single point of failure.

In order to eliminate the single point of failure, researchers have come out with a technique known as "Decentralized Caching Technique" which was proposed in [8]. The authors proposed a hint based approach in which the cache directory of the local cache maintains hints regarding in which local cache of the client nodes the File block probably be found. This technique proposed the meta data in the form of hints to be distributed to client nodes and hence the single point of failure can be eliminated.

A new type of caching technique called collective caching was discussed in [4]. If the subtasks of a client application runs in multiple client nodes then the caches available in these client nodes may be logically combined to act as a single cache so that all the subtasks can read the File blocks from this uni_ed cache provided these blocks are available there.

In [7] an aggressive proactive technique was proposed for the effective pre fetching of _le blocks based on hints. In [3], locality aware cooperative caching was proposed.

The Hadoop DFS (HDFS) [9] is a open-source project and it is a cluster-based File system. The HDFS is an attractive File system and provides scalable storage solutions for Big Data applications.

In [6], a speculation-based method was proposed to improve the performance of the DFS. This technique uses only the local cache for the speculative execution purpose.

III. Proposed Algorithm based on Anticipated Parallel Processing

In this section we discuss regarding anticipated parallel ex- ecution, disadvantages of speculation-based algorithm and then the proposed algorithm.

a) *Anticipated Parallel Execution*

The main idea behind anticipated parallel execution is to do some task before it is known whether that task will be required at all. Later we come to know that whether the task is required or not by checking various conditions. If the task is required then the effect of the task execution is used and the results produced by the task are considered. If the task is not required then the effect of the task execution is undone and the results produced by the task are not utilized. This type of task execution will reduce the waiting time for mmany cases and hence the performance can be improved.

Anticipated parallel execution technique is followed in mod-ern pipelined processors particularly for the efficient han-dling of conditional branch instructions. In this type of pro-cessors, the conditional branch instructions are allowed to go through the various stages of the pipeline. Here the assumption is that the condition may not satisfied and hence the branch will not take place. Whether the condition is satisfied or not for an instruction is known at the execution stage of the pipeline. If the condition is not satisfied, the instruction executions continue in the pipeline. If the con-dition is satisfied then the pipeline is drained and then the instruction will be fetched from the target address (branch address) [1]. Anticipated parallel executions are used in op-timization phase of the compilation process [5].

b) *Disadvantages of Speculation-based algorithm*

In the literature a speculation-based method has been pro-posed for improving the performance of read operations [6]. In this paper, the authors assumed that caches are main-tained in the client systems and the server will be contacted to check whether the content in the local (client side cache) is stale or valid by checking the time stamp of the cached copy and the copy available in the server's disk. Whenevera client

application program requests for a File and if the File is available in the local cache then one speculative execution will be started which reads the content from the local cache and proceed its execution. Meanwhile the server system is contacted to know whether the cached copy of theFile is stale or valid. If the cached copy of the File is valid then the speculative execution will be allowed to continue. If the cached copy of the File is stale then the speculative execution will be rolled back and then the file content is read from the server's disk and then the execution will continue. In this algorithm, the client program checks only the local cache and if the content is not available there the it accesses the content from the server's disk. Note that, the same content may be available in other client nodes connected in the DFS environment. This speculation-based algorithm does consider the availability of data in other client nodes. So, there is a scope of proposing an improved read algorithm by considering the local caches present in other systems.

c) *Proposed Algorithm*

In this subsection, we discuss _rst regarding the assumptions of the caching system maintained in the DFS. Next, we describe the three parts of the proposed algorithm and then the proposed algorithm.

i. *Assumptions*

We have considered a cluster-based DFS to propose our algorithm. In the DFS, we have assumed that one name node (name sever system) and two or more data nodes are present. The purpose of the name node is to store the meta data (global directory - File attributes and other details). The data nodes are used for storing the files and executing user (client) application programs. The name node and data nodes are connected through local area network. All the data nodes are maintaining their own local caches and cache operations are managed by a cache manager module deployed in the data nodes. The cache managers maintain cache directory (CD) in which the information regarding which Files are stored in the local cache is available. In the CD of a data node, the address of the nearest data node is also stored. Here, we have considered only the File level caching (entire File will be downloaded from the server's disk and stored in the cache). We have assumed that caching is done only during read access and write operations will not initiate any cache operation. We have also assumed that the no cache synchronization or invalidation protocol is followed in order to avoid communication delay. Each client program whenever it reads the content form the cache, it has to verify with the name node whether the content read from the cache is valid or stale. We have also assumed that three copies of the same _le is kept in three different data nodes in order to support the reliability feature.

ii. *Three parts of the algorithm*

Our algorithm consists of three 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 execution (AE1) and the third part describes the steps to be followed by the anticipated execution (AE2).

/* A client program (C) running in a data node (D1) has issued *read* procedure to read the contents of the _le F2 */

I) Algorithm for main thread of execution

if AE1 and AE2 are not created or AE1 and AE2 are terminated *then*

D1 contacts name node to get addresses of data nodes where F2 is stored.
C contacts the nearest data node to read F2.
F2 is transferred to C and cached at local cache.
end if

(II) Algorithm for anticipated execution (AE1)

if F2 is available in the local cache *then*
Anticipated parallel execution AE1 is created
C reads F2
Read Time stamp value of F2 into T1.
Wait for name node to send time stamp value of F2 recorded
in its directory (T2)
if T1 >= T2 *then*
AE1 will continue its execution
else
AE1 will be terminated
end if
end if

(III) Algorithm for anticipated execution (AE2)

C veri_es the CD of D1 to get the address of the nearest data node (D2) where F2 is available.
if F2 is available in D2 *then*
Anticipated parallel Execution (AE2) starts.
Read time stamp value of F2 from D2 into T3.
Wait for name node to send time stamp value of F2 recorded
in its directory (T2)
if T3 >= T2 *then*
AE2 will continue its execution
else
AE2 will be terminated
end if
end if

(Note that request message is sent to name node to send the time stamp value of F2 and then both the algorithm steps II and III are executed in parallel.)

IV. Evaluation of Performance

We have analyzed the performance of the algorithms through mathematical and simulation modeling. In this section, we discuss first regarding the assumptions. Next, we discuss regarding performance evaluation through mathematical model. Finally, we discuss regarding the results of the simulation experiments.

a) Assumptions

We have made the following assumptions by considering various factors related to main memory, disk and local area network. (i) Block size is 4 KB. (ii) All data and name nodes are connected in a network. (iii) Average communication delay is 4 ms. (iv) Transferring meta data from name node to requested data node is 0.125 ms (v) Average block access time for disk is 12 ms (vi) Average block access time formain memory is 0.005 ms (vii) Local cache hit ratio is *lc* and remote cache hit ratio is *nc*.

b) Mathematical Model

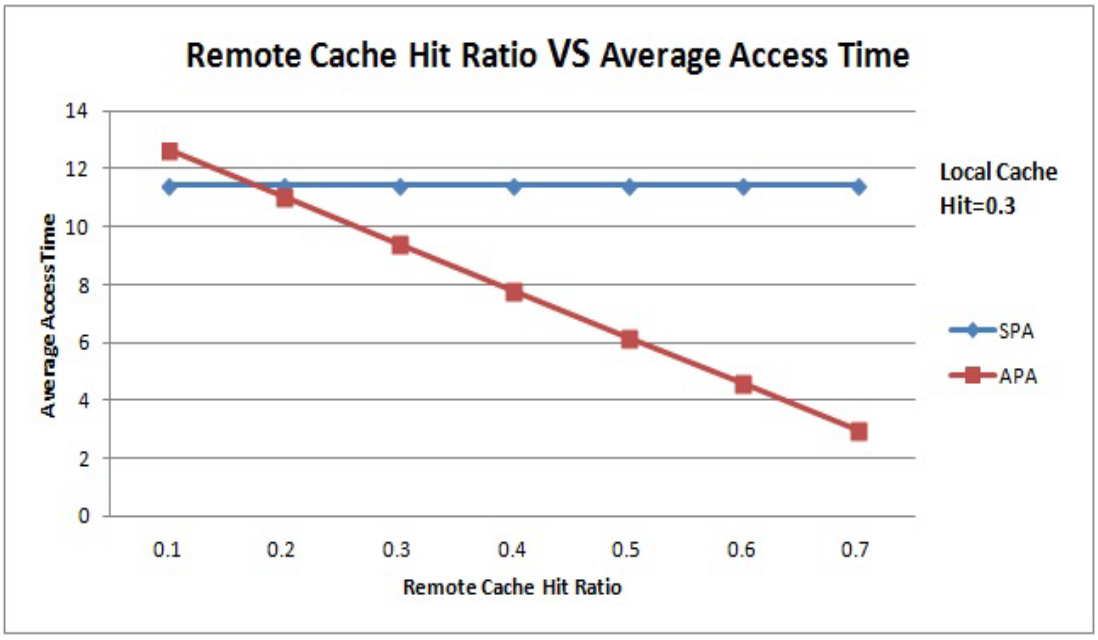
Based on the assumptions discussed in the above subsection we calculate the average access time for the speculation and anticipated parallel processing-based algorithms. We call average block read access time as ABRAT. We have calculated the time required to access a file block from the remote data node as 4.01 ms.

Average Block Read Access Time (with speculation) = *lc* * (Main memory access time + Time stamp collection time) + (1- *lc*) * (Main memory access time + Time stamp collection time + Block access time for Disk + Block transfer communication time + Main memory access time). If we apply the above equation, ABRAT for speculation-based approach is calculated as (16.26 - 16.13 *lc*)ms. (Formula 1)

Note that, we have not considered overhead involved in starting the the speculative execution.

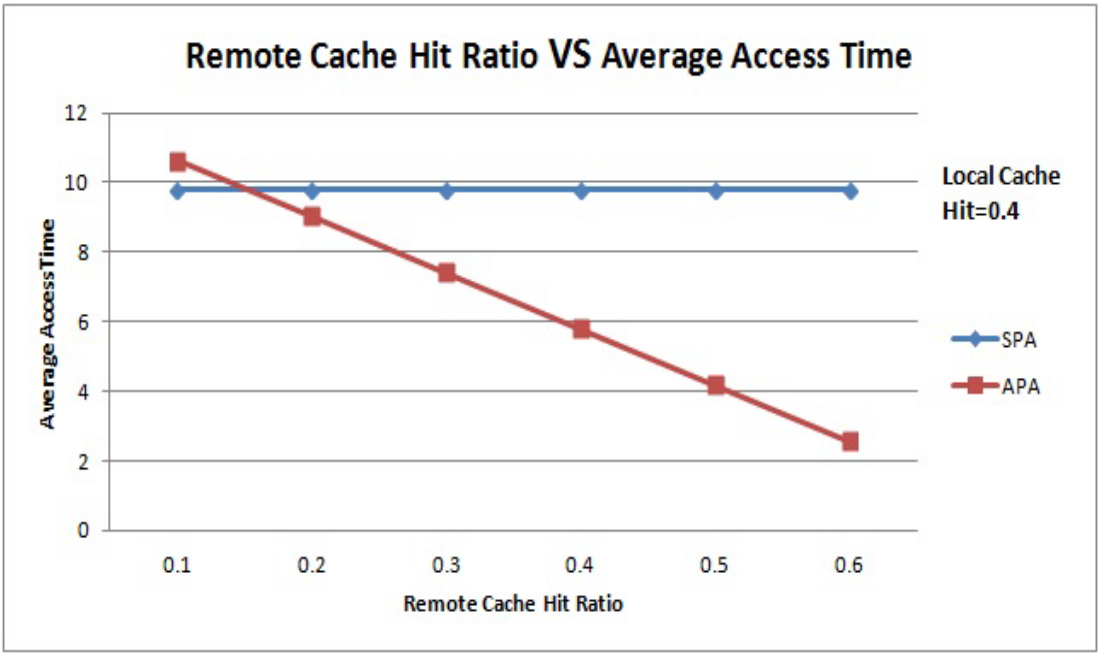
Average Block Read Access Time (with anticipated parallel processing) = *lc** (Main memory access time + Time stamp collection time) + *nc* * (Main memory access time + Time stamp collection time + Block transfer communication time + Remote Main memory access time) + (1- *lc* - *nc*) * (Time stamp collection time + Main memory access time + Remote Main memory access time + Meta Data Collection Time + Block access time for Disk + Block transfer communication time + Main memory access time). The ABRAT for anticipated parallel processing-based approach is computed as (20.26 - 20.01*c* - 16.13*c1*)ms (Formula 2).

Figure 1: Remote Cache Hit Ratio Vs Average Read Access Time
(Local cache hit ratio value is 0.3)



We have varied the local cache hit ratio (*lc*) and remote cache hit ratio (*nc*) and calculated ABRAT values both for speculation- and anticipated parallel processing-based approaches by applying the formulas 1 and 2.

Figure 2 : Remote Cache Hit Ratio Vs Average Read Access Time
(Local cache hit ratio value is 0.4)



We have Fixed the *lc* value as 0.3 and measured the values which is depicted in Fig. 1. For the *nc* values 0.2 and above, the proposed anticipated-parallel processing based algorithm (APA) performs better than the speculation-based algorithm (SPA). In Fig. 2. we have _xed *lc* value as 0.4 and varied the *nc* values from 01. to 0.6. We can observe the similar trend in both the Figures (Fig. 1 and Fig. 2). We have Fixed *lc* value as 0.5 and varied *nc* values from 0.1 to 0.5 and observed the performance of the algorithms. For the nc values 0.11 and above the proposed anticipated parallel processing-based algorithm performs better than the speculation-based algorithm which is depicted in Fig. 3. We can observe similar trends in Fig. 4 and Fig. 5.

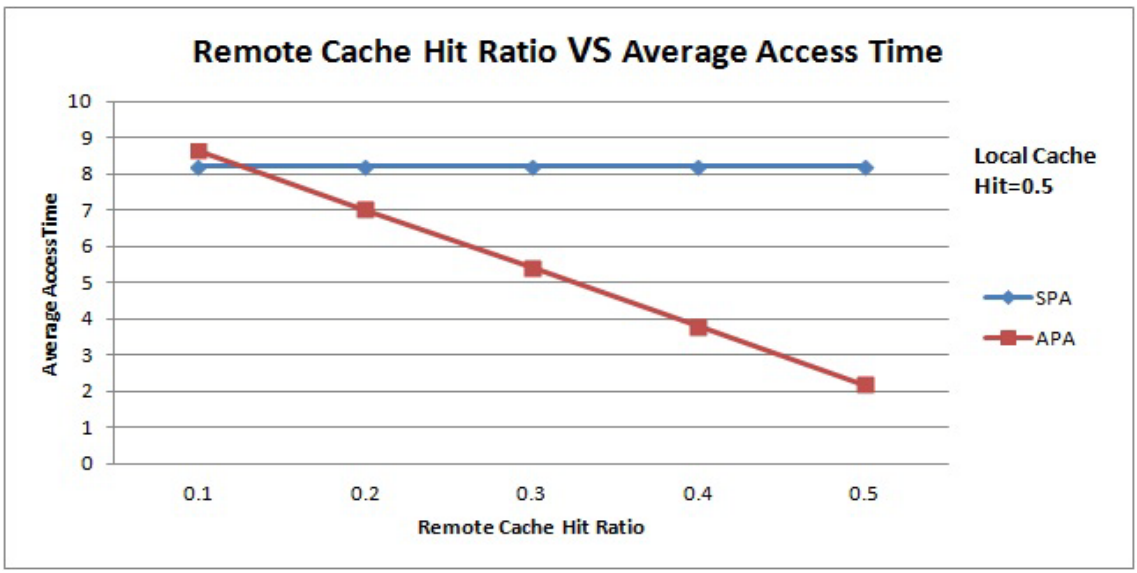
c) Simulation Experiments

We simulated both speculation- and anticipated parallel processing based algorithms. We conducted the simulation experiments by Fixing the number of Files present in the data node and by varying the number of cache blocks of local and remote caches and number of blocks in the File.

The performance of the proposed algorithm (APA) and the speculation-based algorithm proposed in the literature (SP) are shown in Figures 6 to 10. We have Fixed the number of Files present in the DFS as 50 and capacity of LC and NC as 100 blocks and have varied number of blocks present in the Files from 25 to 100 and conducted simulation experiments. The performance is shown in Fig. 6. We observe that APA requires less access time than SP for all cases. Next, we have Fixed the number of Files as 50 and capacity of LC and NC as 200 blocks and varied number of blocks present in the Files from 25 to 100. The observed performance is shown in Fig. 6. We observe that APA performs better than SP. Similar trends can be observed in Fig. 8, Fig. 9 and Fig. 10.

Both the results of evaluation through mathematical and simulation techniques indicate that the proposed anticipated

Figure 3 : Remote Cache Hit Ratio Vs Average Read Access Time
(Local cache hit ratio value is 0.5)



parallel processing based approach performs better than the speculation-based technique proposed in the literature.

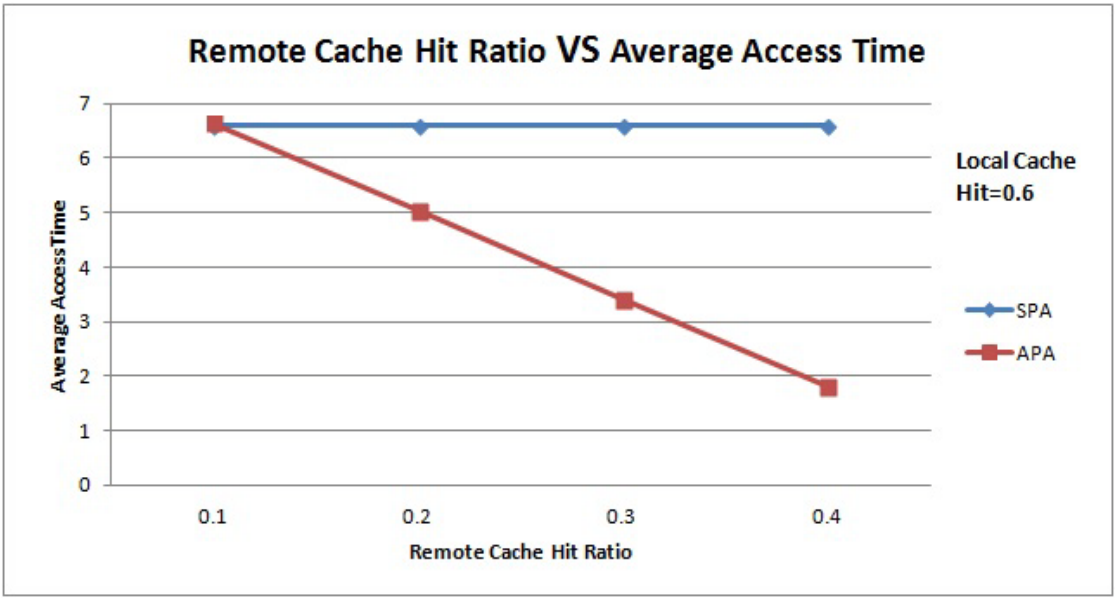
V. Conclusion

In this paper, we have proposed an anticipated parallel processing based read algorithm for improving the performance of the DFS. We have also carried out performance analysis for the speculation-based read and proposed algorithms using mathematical analysis and by conducting simulation experiments. The results of our analysis indicate that our proposed algorithm requires less read access time than the speculation based read algorithm proposed in the literature.

References Références Referencias

1. D. Bernstein, M. Rodeh, and M. Sagiv. Proving safety of speculative load instructions at compile-time. In B. Krieg-Br ~ Aijckner, editor, *ESOP '92*, volume 582 of *Lecture Notes in Computer Science*, pages 56{72. Springer Berlin Heidelberg, 1992.
2. M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A.Patterson. Cooperative caching: Using remote client memory to improve _le system performance. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.
3. S. Jiang, F. Petrini, X. Ding, and X. Zhang. A locality-aware cooperative cache management protocol to improve network _le system performance. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 42{42, 2006.
4. W.-k. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and S. Tideman. Collective caching: application-aware client-side _le caching. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 81{90. IEEE, 2005.

Figure 4: Remote Cache Hit Ratio Vs Average Read Access Time
(Local cache hit ratio value is 0.6)



5. D. Lilja and P. Bird. *The Interaction of Compilation Technology and Computer Architecture*. Springer US, 1994. [6]
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 m191{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 _le system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1{10. IEEE, 2010.

Figure 5 : Remote Cache Hit Ratio Vs Average Read Access Time
(Local cache hit ratio value is 0.7)

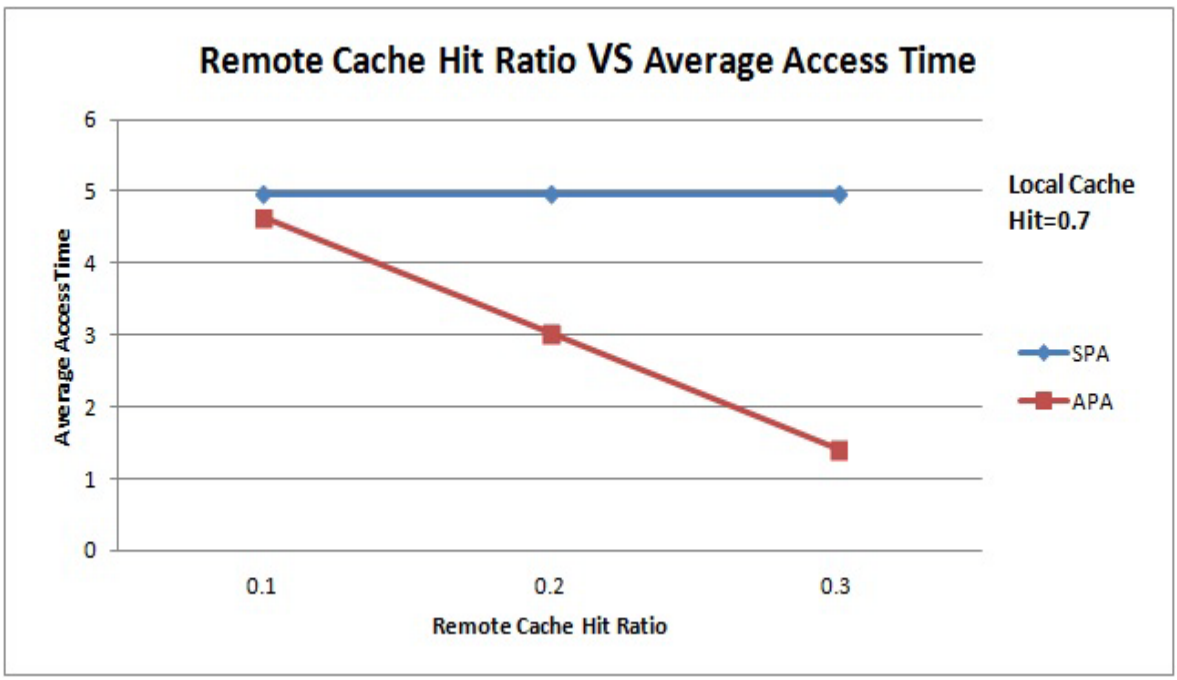


Figure 6 : Number of blocks Vs Average Read Access Time (LC & NC _ 100 blocks)

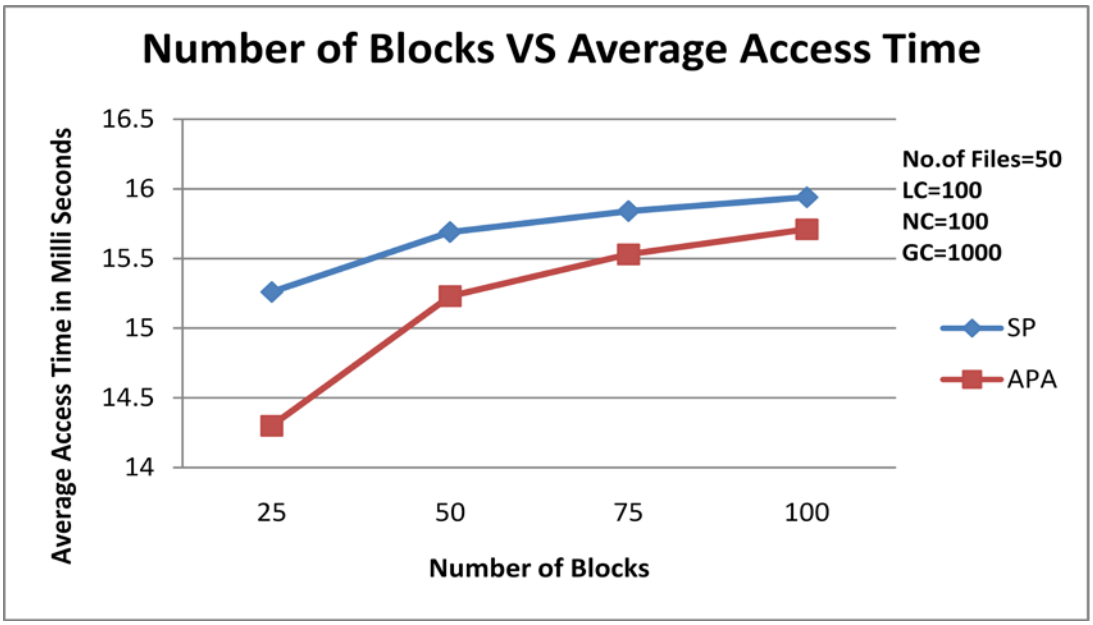


Figure 7: Number of blocks Vs Average Read Access Time (LC & NC _ 200 blocks)

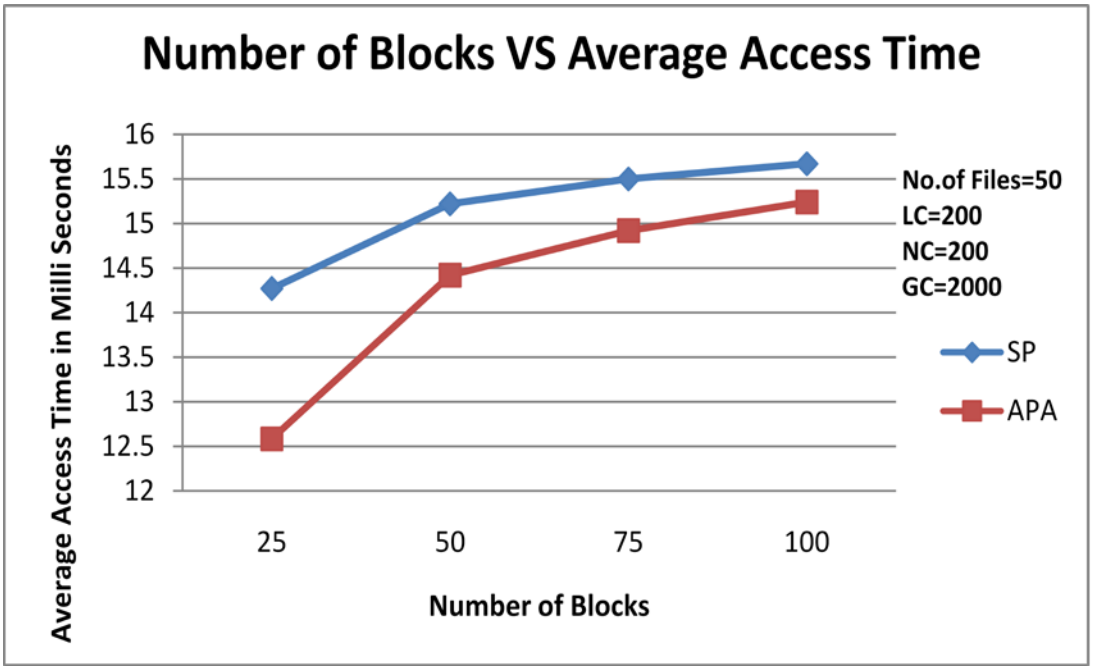


Figure 8 : Number of blocks Vs Average Read Access Time (LC & NC – 300 blocks)

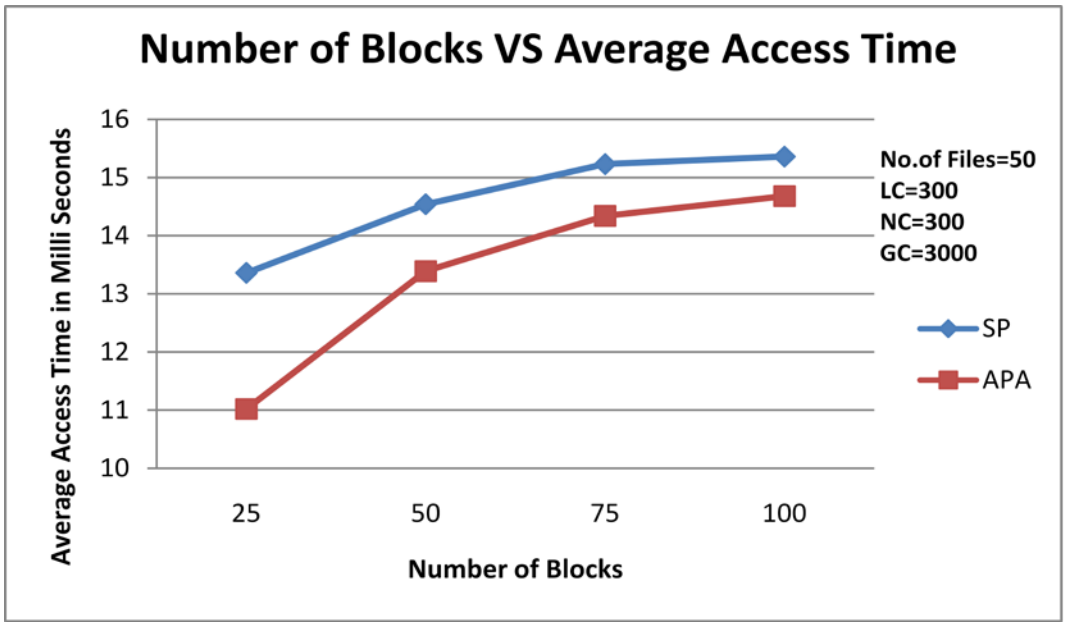


Figure 9: Number of blocks Vs Average Read Access Time (LC & NC – 400 blocks)

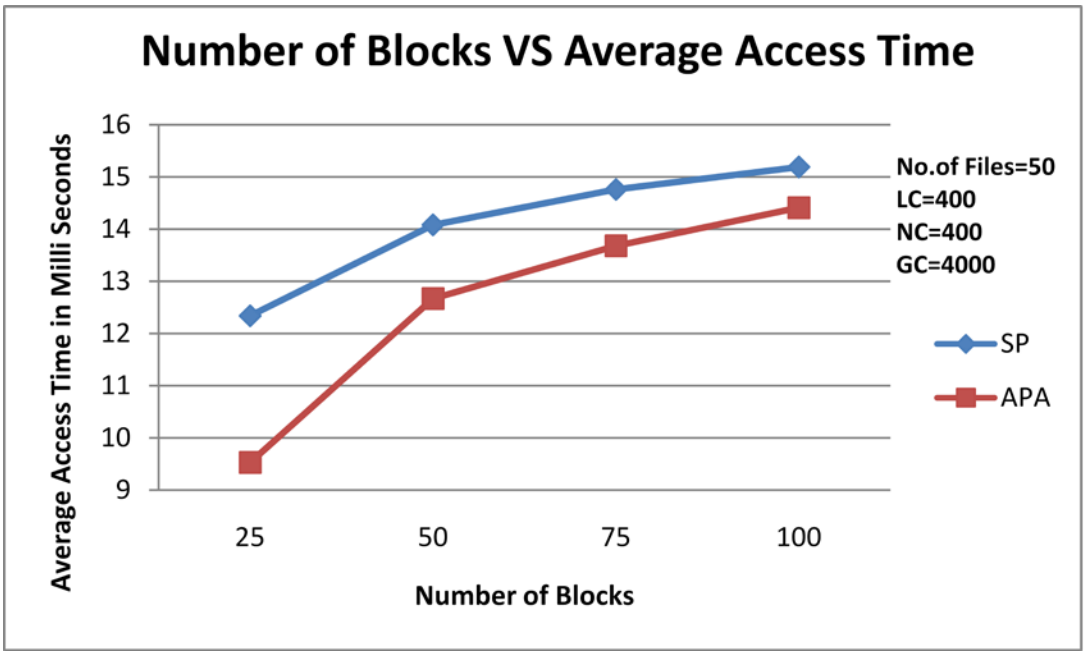


Figure 10: Number of blocks Vs Average Read Access Time (LC & NC – 500 blocks)