ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse91942020.pdf
https://doi.org/10.30534/ijatcse/2020/91942020

Performance Optimization for Distributed Hybrid Storage Systems using a Predictive Approach

Maen M. Al Assaf[1]
[1]King Abdullah II School for Information Technology, The University of Jordan, Amman, Jordan
m_alassaf@ju.edu.jo

ABSTRACT

Distributed data centers are considered very important for data storage services in the contemporary computing world especially with the increased amount of data that are needed to be stored and retrieved. Data retrieval speed performance is a sensitive issue when considering the huge amount of data that need to be retrieved from several nodes over the network. Data prefetch has proved to be an important technique for reducing data reading time from the distributed nodes. In such distributed environment, data fetching time from a node to an another consists of the disk reading time and the network transmission time. Multi-layer (hybrid) storage provides high performance solutions for big data centers. We introduce a solution PPDHSS that implements predictive-probability graph to predictively prefetch the data that are expected to be accessed by the application in the near future from the lower level hard disk of the storage-side nodes (slower) to the top level solid state disk (faster) in parallel which the application data reading requests that comes from the client-side node by taking advantage of the storage system's parallelism. Our performance evaluation in which we used real traces shows that our system can reduce the data fetching time from storage side-nodes to the client-side nodes without the need of using caches of large size.

Key words: Predictive prefetching · probability graph, distributed storage environment.

1. INTRODUCTION

In contemporary world, Big Data concept became important in different domains especially in business world [18]. Research related to Big Data processing and storage is considered a dominant research were there exist a significant need for solutions that aim to improve the performance [19]. There exists an increased amount of data that need to be stored and retrieved in big data centers that consists of distributed hybrid (multi-levels) storage nodes. Storage-side nodes of parallel hybrid layers that vary in speed, size, and cost are important to allocate the data based on their predicted importance. Potential data that will be actually accessed soon is considered more important. Data prefetch research played an important role in solving performance bottleneck problems in read intensive data centers [1][5]). Several prefetch techniques were proposed to preload data from disks before the application issues the actual read request. Mainly, there are two prefetch methods; predictive and informed. In predictive method, the system predicts what data will the application needs soon based on the patterns of previous requests history [2]. On the other hand, informed method makes use of the hints that are given from the application to fetch the data in advance [1][3][5]. This research implements probability graph predictive method (PG) that continuously record the history of the data requested by the application at the client-side node [2][6]. We study the performance impact of this scheme on distributed hybrid (multi-level) storage environment. In such systems, data are allocated in remote storage-side nodes in a form of blocks and fetched to the application (client) side node via the network. Data fetching time includes both, storage disk reading time and network transmission time. We predictively prefetch the data from lower levels (slow) of the storage-side nodes to the top one (fast) in parallel with the application read requests in order to reduce the data read time; and eventually, reducing the application's running time.

1.1 Research Motivation

Several issues motive us to do this research. The growing use of hybrid layers in contemporary big data centers especially with the increased amount of data that need to be stored in daily life domains. Predictive prefetch process does not need hints from the applications as informed prefetch process does. In addition, data fetching time in distributed environment is increased due to the network transmission time. Reducing disk reading time is valuable to lessen the entire data fetching time. Predictive prefetch process can be invoked to predictively preload that data that is expected to be accessed soon from the bottom to the top level of the hybrid multi-levels environment. This can be done parallelly while the application is issuing its data read request as there is enough bandwidth in the parallel storage setting.

1.2 Research Contributions

Our PPDHSS research provides a mechanism based on probability graph-predictive prefetch process to predict and prefetch the data that are expected to be needed by the application in the near future from the bottom level to the top one in a two levels distributed hybrid (multi-levels) storage environment. This process is performed simultaneously while the application is issuing its data read requests. Our storage-side nodes setting consists of two levels; SS and HD.

2. RELATED WORK

This section presents some of the existing work and concepts related to our research presented in this paper.

2.1 Hybrid (multi-level) storage Environment

A hybrid (multi-level) storage environment is commonly used in big data centers as it provides a hierarchy of devices that are different in their size, cost, and speed. As we go up in the hierarchy, cost in dollars and data retrieval speed increase and size decreases. So, it is better to store the most important data that is most frequently demanded in the top level in order to provide high performance in their retrieval process [1][2][3][4][5][11][12]. When a data block is requested by the application at the client-side node, the system search for it in the top level of the storage-side nodes, if it is not found, the system descends to the lower levels until it is founded and retrieved.
In distributed environments, hybrid storage is also used where the multi-level storage devices are distributed in a form of storage-side nodes connected through the network. Data blocks are in such setting are replicated or stripped all over the storage-side nodes [1]. Applications on client-side nodes request the data to be fetched from storage-side nodes over the network. Such systems are highly important to maintain high performance, availability, reliability, fault tolerance, and scalability [10].

2.2 Data Prefetch

There exist two main data prefetch techniques: Predictive and Informed [15]. Predictive approach predicts the data that will be requested in the future by the application based on its historical requests' patterns. An approach used to record the history of the application's previous data requests is called probability graph (PG) [2][6][13]. Markov chain prediction approach was also used by other researchers [8]. Informed approach takes hints from the application on its future data requests in order to preload the data in advance.
Data Prefetch process in hybrid (multi-levels) storage environment is performed by moving the important data blocks from a bottom level in the hierarchy to the top one. This process is technically challenging as it importantly needs to control both prefetch process aggressiveness and accuracy [7][9][11] [12].

3. SYSTEM DESIGN ISSUES

This section presents the hardware and the software design on which our PPDHSS system functions.

3.1 Hardware Design

The system consists of several nodes that are connected over a network. A node is either an application (client)-side node or a storage-side node. The array of storage-side nodes forms the distributed parallel hybrid (multi-levels) storage environment/ system. The application on the client-side node keeps requesting data by issuing data read requests for the data blocks that are located at the storage-side nodes. The storage-side nodes architecture consists of a two-level storage disks that consists of Solid-State Disk (SS) in the top level and Hard Disk (HD) in the bottom level as shown in (Figure. 1). In terms of performance, SSs have higher data retrieval speed performance. In this research, we are not going to implement a cache at client-side node to cache the data that are brough from the storage-side nodes through the network as we consider that will be more helpful in optimizing the performance. In addition, we will implement one client-side node (i.e. one application); however, our approach will function in case several client-side nodes are connecting.
At the top level of each storage-side node (i.e. SS), there exists a reserved portion (we call it: SS cache) that implements (LRU) least-recently-used policy to cache both of; the application' data read requests and the data that our PPDHSS mechanism prefetch to the SS from the HD.

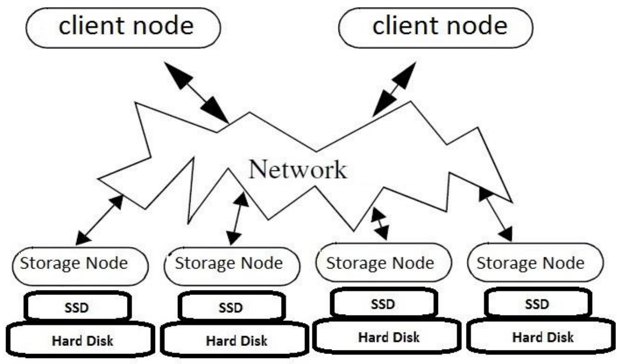

Figure 1: PPDHSS's hardware design. Quoted from [1].

As we will illustrate next, data are stored in the distributed hybrid (multilevel) storage environment in a form of fixed sized data blocks. So, the size of the cache implemented in SS level is measured by number of cache blocks. Each cache block has the same size of the fixed size data block. Our system uses data stripping to parallelize data in the distributed storage-side nodes. This leads us to consider that each cache block is also stripped in the distributed SS level in the form of equal size distributed chunks; where each cache block chunk at each storage-side node's SS can buffer a prefetched chunk of a particular stripped data block available in the same storage-side node' HD.

3.2 Software Design

PPDHSS's implements three modules: disk manager, SS cache manager, and predictor. The three modules are implemented in the storage-side nodes. While the application issues requests for data reads, PPDHSS receives the request in the three software modules. The disk manager seeks for the requested data block in the storage system from top to bottom.

The SS cache manager controls the cache that is implemented in the SS level to buffer the application's reads as well as the prefetched data from the HD. It also performs LRU policy when the SS cache reaches to its full capacity. The predictor receives each application data read request and executes the probability graph-predictive algorithm to determine the data blocks that have good cache to be requested by the application soon in the near future [2]. Those requests are sent to the disk manager to read them from the HD to the SS unless if the data block is already existing in the SS level. In such parallel storage setting, there is a maximum bandwidth of the number of data blocks that can be prefetched from the SS to the HD in parallel with the application data read requests without being congested. Later we will discuss this issue and its measurements.

3.3 Assumptions

We assume that the data blocks are already stored in the bottom level (HD) as it has a significant large storage space in comparison to the SS. This represent a worst-case scenario since we can have better performance if some of the data are already stored in the SS. [1][2][5].
As we have the small cache implemented in the SS level (SS Cache), our data prefetch mechanism will cache copies of prefetched data beside the application's data read request the are read from the HD (SS misses). No moves for the entire data blocks from the HD to the SS takes place as that will waste the SS level's capacity and we will consume the parallel storage' bandwidth; especially if some of these data blocks need to be returned to the HD level later [2]. Using our SSs and HDs installed in our research lab's distributed storage setting, we found that SS provides higher performance than HD when using big sized data blocks. Other researchers have noticed the same observation too. For example, HDFS uses data blocks of sizes starting from 64 MB [14]. Hadoop uses Hadoop archives (HAR) tool to combine the data blocks in case they are already small [17]. Based on our research lab setting, we will use data blocks of size 200MB. [1][5].

4. THE PPDHSS ALGORITHM

For such storage systems, we proposed our PPDHSS that implements probability graph- predictive approach (PG) in order to do data prefetch for important data blocks to the top level of the storage-side nodes. Important data blocks are those that are very likely to be requested by the application soon. This will reduce data fetching time from remote storage-side nodes to the client-side node and therefor, reduces the application's running time. This process goes simultaneously while the application on the client-side node is requesting the data.
In a distributed storage environment, application reads its needed data from the remote storage-side nodes. Reading time for each data block includes network transmission time beside the disk read time. Our mechanism reduces disk read time as a part of the overall data loading time.
At storage-side nodes, our experiments indicate that parallel storage environment may encounter bandwidth congestion in case a huge number of data are requested in one time. So, there is an ultimate limit of data read requests that can be processed in the same time in the parallel storage environment. As the application issues only one data read request at the moment, the unused bandwidth can be used for doing the prefetch process from the low to the top level in order to increase the chance in which the application finds its future read requests in the top (fast) level of the storage hierarchy.

4.1 Definitions

Our distributed environment consists of two types of nodes: user-side nodes and storage-side nodes where each storage-side node contains two storage disks levels (SS and HD) (Figure. 1).
We define (BwMax) to be the max number of data read request that can be processed in the storage system without causing a congestion. As the application always issue one read request at each time for a particular data block to be read, the remaining value of the max bandwidth (BwMax - 1) can be used to process the data prefetch requests in the storage-side nodes. Since each storage-side node can perform one data read request for one block inside its hybrid system at a time without being congested, (BwMax**)** value is equal to the number of storage-side nodes. The SS cache size is represented by the term (CacheSize) which is counted by the number of cache blocks. Each cache block has the same size of the fixed data block size that is stored in the storage-side node.
An application running time comprises both, the processing time and the data loading time from the storage-side nodes over the network. The time interval between each two consequent data read requests is the sum of the loading and the processing times for the current under manipulation data block. We will not consider processing time in our study as our system is mainly concerned for read intensive applications.
As our distributed hybrid storage-side nodes consist of HD and SS, $T_{HD-network}$ is the time of loading one data block from a remote storage-side node HD level to the client (application) side node, and $T_{ss-network}$ be the same when reading that from the remote node' SS. $T_{ss-network}$ value is less when compared to $T_{HD-network}$ due to the high speed of SSs. T_{HD-ss} is the time needed to load one data block from the HD and to have it copied to the SS one. In addition, the process of data prefetch from HD to the SS level is performed simultaneously with the application data read requests [1].

4.2 The Predictive Probability Graph (PG)

Our PPDHSS makes use of predictive data prefetch algorithm that is based on PG approach to prefetch predicted data blocks to the top level in the storage-side nodes; hence, reducing the applications' running time. As mentioned previously, the predictor module that performs this task is implemented in the storage-side nodes. PG contentiously records the application' past data requests patterns in order to predict near future requests. PG uses a directed-weighted-graph that consists of nodes and edges that are used to estimate the near future request probabilities of the data blocks. Each previously requested data block by the application is allocated a node in the PG. Directed-weighted-edges are used to make connections across the nodes to be used in the probability calculations of which a particular block will be requested. While the application is issuing data read requests, the predictor module keeps updating the graph and executing the prefetch process. There are two main parameters attached to this process, 1- Look-Ahead-Period (LKP) 2- Minimum-Chance Percentage (MNC). LKP determines the correlation among the consequent requested data blocks. MNC determines the volume of the upcoming data prefetch request when a data block is requested by the application. Increased LKP and decreased MNC leads to make the prefetch process more aggressive. Over aggressive prefetch process threaten the prefetch process accuracy as un-needed data blocks will be brought [2][6].

4.3 Putting All Together

While the application at the client-side node requests data blocks that are stored in the storage-side nodes, the predictor, SS cache manager, and the disk manager modules get each request. The predictor continues its operation in updating the PG setting and in issuing the prefetch requests for those data blocks that are deemed to be requested soon by the application. The disk manager checks if the requested block is already in the SS level to be provided to the application. Otherwise, it will be directly provided from the HD level. The volume and the aggressiveness of each prefetch request are determined by both LKP and MNC. Our system does not allow more than (BwMax – 1) volume of a prefetch request in the same time. Our system takes into account the case when a prefetch request is issued for a data block that is already in the SS. Our system also takes into account the case where a data block was modified by the application for the sake of consistency. The SS cache buffers the application's data requests and the prefetched data. SS cache manager applies LRU least-recently-used policy whenever needed in order to open a space for other potential blocks. The time gap between each two consequent application' data requests comprises of both, processing time and data loading time over the network. In performance evaluation, we set the processing time to zero as we assume a more severe scenario of an aggressive read intensive application. (Figure. 2) describes PPDHSS algorithm work.

```
Input parameters: BWMax, CacheSize, MNC, LKP.

while the application is issuing data reading request to the storage nodes do
    if their exists bandwidth shortage in the storage nodes then
        drop the last data block of the current HD to SS prefetch request (by the disk manager)
    end if
    if prefetched data block is altered by application writing
    then
        discard the prefetched data block from the SS cache (by the disk manager)
    end if
    provide the application over the network with its requested data block by searching the SS level first, if it
    is not there, read it from the HD level directly. (by the disk manager)
    if the application data reading request is found in the HD level at the storage node then
        cache it in the SS cache and update the cache by applying LRU policy (by cache managers)
    end if
    update the probability graph (by the predictor)
    issue a prefetch request from the HD level to the SD cache for the predicted data blocks that are only found
    in the HD level using the given MNC and LKP values (by the disk manager) and update SS cache by
    applying LRU policy (by the cache manager)
    if the number of data blocks requested by the prefetching request exceeds BwMax -1 then
        drop the extra requests from the end side (by the disk manager)
end while
```

Figure 2: The PPDHSS algorithm. Quoted from [2] with some modifications.

5. PERFORMANCE EVALUATION

This section illustrates our simulation process and results for PPDHSS system.

5.1 System Parameters

The following are our system' platform and the underlying hardware resources: (1) Linux OS (2) Western Digital HD (3) Intel SS (4) Dell Network Switch.
As mentioned, the size for each data blocks stored in the SS or in the HD levels will be 200MB. Using a large size data blocks is a realistic observation in the contemporary life especially with the huge increase in data that need to be stored in big data center where data are stored in form of huge size data blocks. In the subsequent subsections, we consider that the data block size equals to 200 MB. As we mentioned previously, the size of the SS cache reserved for our technique is determined in terms of number of cache blocks where each one has a size same as the size of a data block. Hence, a SS cache block size equals to 200 MB. As we will discuss later, predictive prefetch process provides its best performance improvement when using small caches. Hence, cache reserved in the storage-side nodes SS level will not consume its capacity.
As mentioned, application processing time on loaded data is set to null as we are concerned in this research about the applications that do aggressive data reads. In case we add some value for the processing time, this will help the prefetch processes; as the application data requests frequency will decrease.
In our laboratories, we found that, T_{HD-ss} equals to 4.5 seconds, ($T_{HD-network}$) equals to 4.43, and ($T_{ss-network}$) equals to 4.158 when considering a 200MB fixed size for each data block. As we can see, remote SS shows higher efficiency [1].

5.2 The PPDHSS Simulation

In this simulation, we are going to evaluate the system performance in terms of application total running time; first: without implementing PPDHSS, second: when using PPDHSS under different SS cache sizes, and third: using PPDHSS under different levels of aggressiveness. A decreased application running time indicates better performance. Our simulator is trace driven implemented in C. The system consists of one client-side node and BwMax number of storage-side nodes connected via a network. As we will see, our solution provides significant performance improvement when SS cache is small. So, we will set BwMax to 5 and SS cache size to 5 cache blocks (i.e. 1 GB aggregated SS cache size where each cache block on each storage-side node has a size of 200 MB). Recall that each SS cache block is stripped on the SS level through all over the storage-side nodes and can buffer a stripped chunk of a stripped data block. Increased BwMax will by default increase the performance in which; as we will see; will not be much needed.

We use in this simulation LASR1 from the LASR trace repository [16]. The trace consists of approximately 11600 reading requests. Each distinct request represents a data block stored in the system. As we are neglecting the processing time, the time gap between each two consequent requests is the loading time of the currently requested data block to the client-side node either is founded in the SS or the HD level.

5.2.1 performance without PPDHSS

Based on our assumption where all data blocks are initially placed in the HD level. Without implementing PPDHSS, the running time of the application will be **51724.7 seconds**. Next, we will see that PPDHSS can improve performance in about 2% when using small sized cache and 4% when using moderate sized one.

5.2.2 PPDHSS with increased SS cache sizes

Whenever the SS cache size increases, application running time will decrease as more SS cache hits take place. We executed our simulation when using different values of SS cache sizes from 1 - 100 and when setting the prefetch aggressiveness to a moderate level (i.e. LKP = 1 and the MNC = 0.5). (Figure. 3) shows a decreased running time trend as the SS cache becomes larger. The most interesting observation is that the system shows its best significant improvement when using small SS cache sizes (i.e. 1 - 13). Performance continues to improve till the SS cache size reaches to 40, then, more large size will not add much added value. So, we can conclude that no large SS caches are needed.

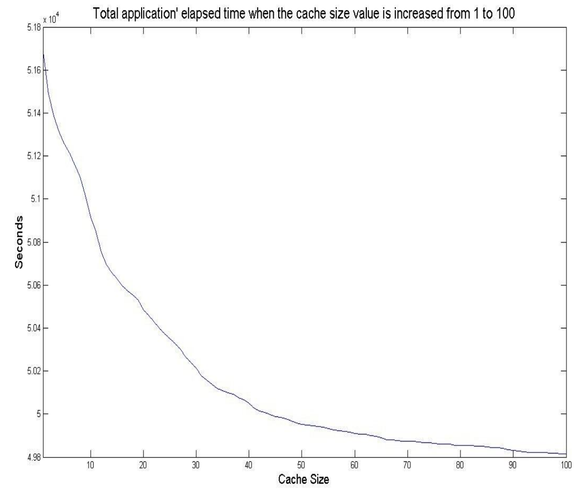

Figure 3: App running time in seconds when the SS cache size from 1 - 100. *BwMax* = 5. LKP = 1. MNC = 0.5.

5.2.3 PPDHSS with aggressive prefetching

As mentioned, the values of MNC and LKP determine aggressiveness of the prefetch process. Increased LKP and decreased MNC increase the aggressive prefetch.

(Figure. 4) shows the case when using a small SS cache (i.e. = 1 block) and performs aggressive prefetching (i.e. when MNC = 0.1 and LKP = 1 or 2 or 3). Since the cache is quite small, aggressive prefetch will replace useful blocks with non-useful ones. Decreasing the aggressiveness (i.e. MNC = 0.2 and LKP = 2 or 3), will provide better performance as more valuable data will stay longer in the SS cache. Increased LKP to values (2 or 3) and increased MNC to a level higher than 0.2 will negatively affect the performance because of the reduced aggressiveness and accuracy. So, PPDHSS is not bringing enough valuable data to the SS cache.

It is clear that the case when LKP = 2 shows better performance than the case when LKP = 3 due to the increased accuracy of the prefetch decisions. LKP = 1 case shows increased improvement trend in performance especially with the increase in MNC value. Hence, the aggressiveness goes down and the accuracy goes up and the system retains valuable data for longer durations.

(Figure 5), shows an increased SS level cache size to 2. The observations explained in Figure. 4 case (SS cache size =1) happen. But as the SS cache size goes up, it is natural that it will be able to buffer more data, and hence positively affect the performance. It will also be more able to benefit from more aggressive prefetch. We observed that when MNC = 0.2 and LKP = 3, our system provides its best performance optimization as a result of moderate aggressive prefetch process.

We simulated the system when using larger sizes of SS cache that span to 100 blocks. PPDHSS shows similar trend in performance improvement. Due to space limitation, we will mention our general notes on the performance results of those cases and illustrate some figures. In each of those cases, SS cache size becomes significantly large and able to keep more data. Hence, we surely can achieve extra performance improvement even if the prefetch process decisions are not that much accurate. As the SS cache becomes larger, aggressive prefetch process becomes able to leverage the increased cache size and to achieve more cache hits. In many cases, we observed that PPDHSS provides its best positive effect on the performance when the MNC = 0.1 and LKP = 3. When the aggressiveness of the data prefetch process decreases, it will not bring enough data; which negatively affects the performance. It is also good to mention that when the MNC goes up to 0.6 or above and LKP = 3, PPDHSS shows its lowest performance improvement due to the low accuracy and aggressiveness of the prefetch process. So, prefetch process is bringing few and unneeded data blocks. Figures 6-12 show different scenarios of increased SS cache sizes. It is clear that performance trend improves with the increased size SS cache.

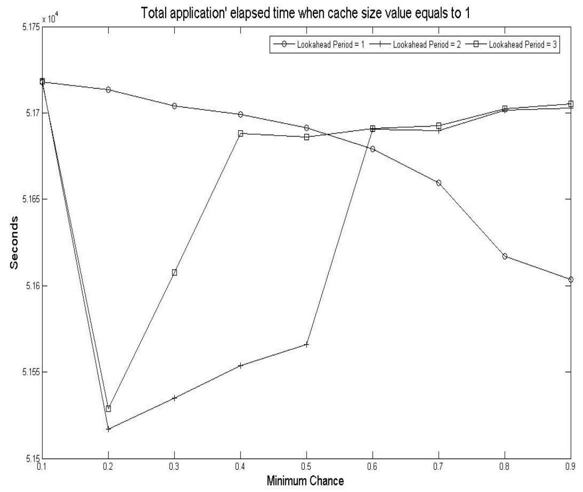

Figure 4: App running running time in seconds when SS cache size = 1 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

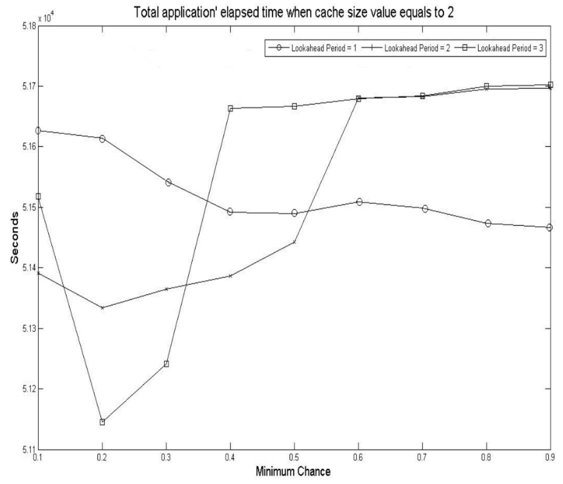

Figure 5: App running time in seconds when SS cache size = 2 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

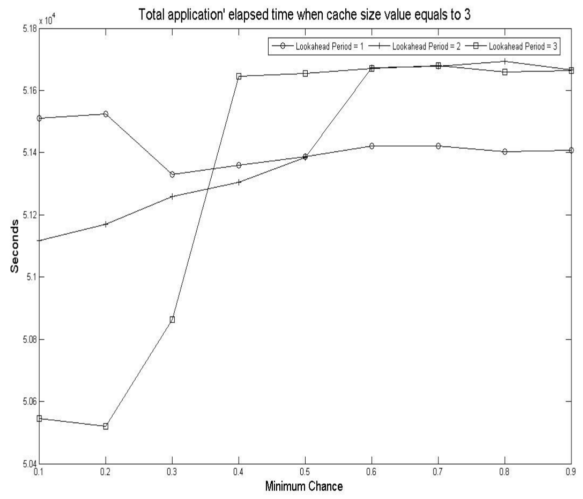

Figure 6: App running time in seconds when SS cache size = 3 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

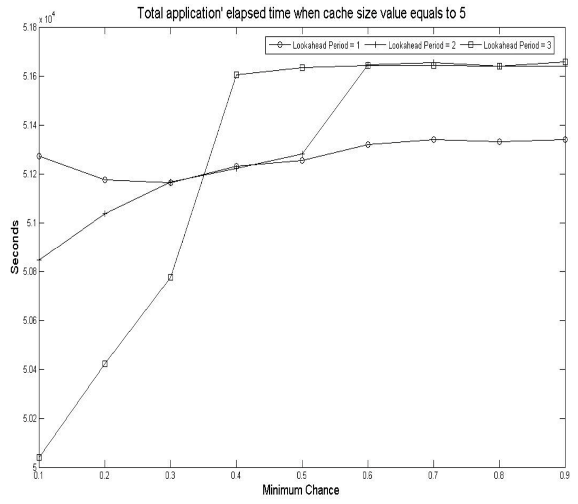

Figure 7: App running time in seconds when SS cache size = 5 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

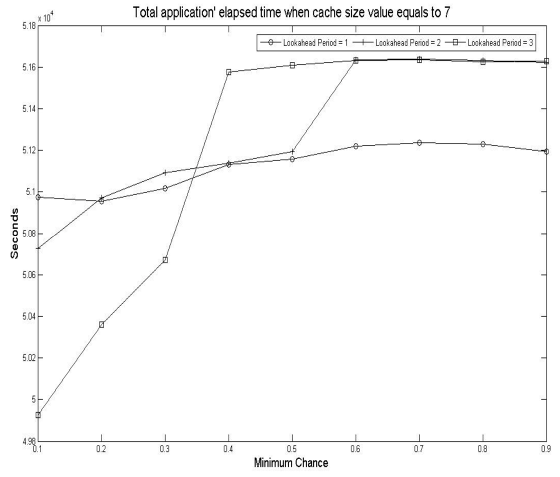

Figure 8: App running time in seconds when SS cache size = 7 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

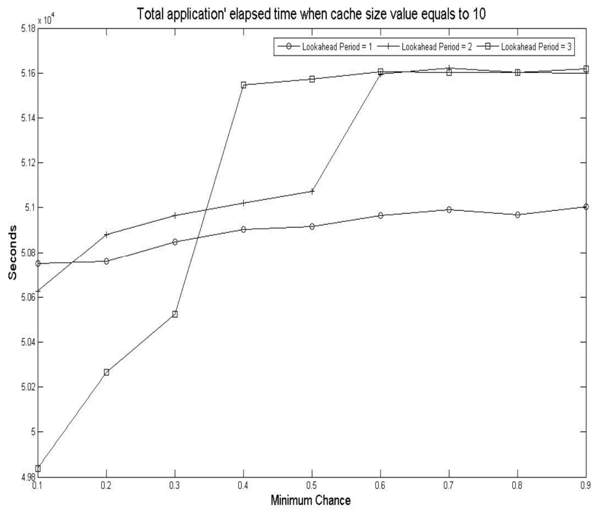

Figure 9: App running time in seconds when SS cache size = 10 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

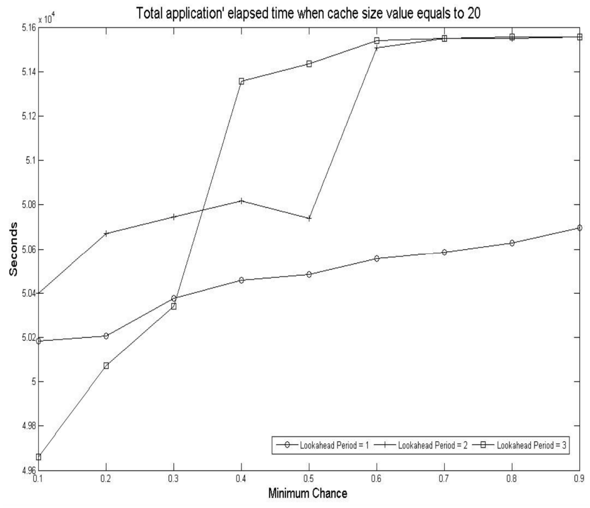

Figure 10: App running time in seconds when SS cache size = 20 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

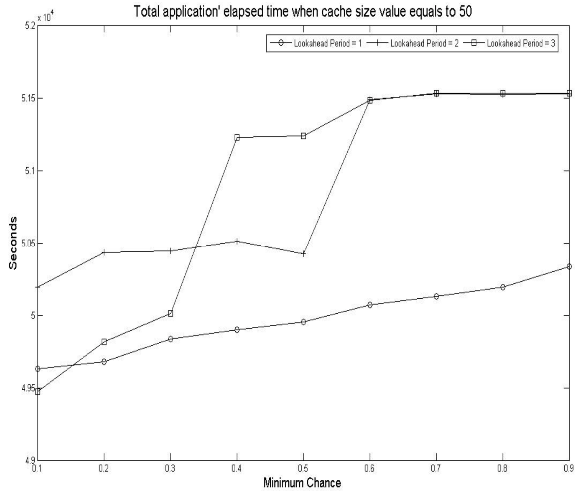

Figure 11: App running time in seconds when SS cache size = 50 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

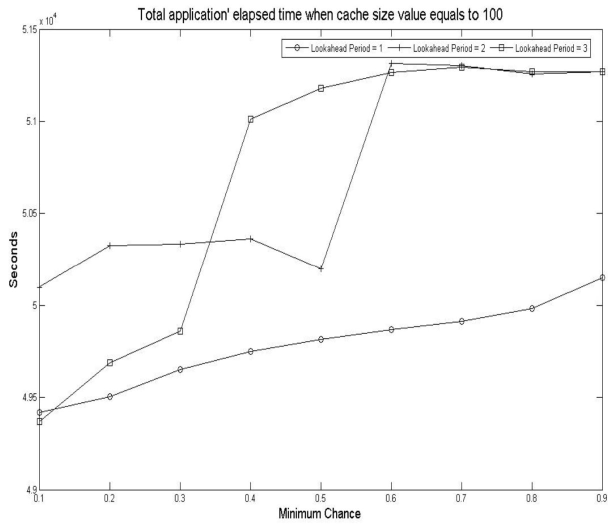

Figure 12: App running time in seconds when SS cache size = 100 and multiple values of LKP from 1 - 3 and multiple percentage of MNC from 0.1 - 0.9

6. CONCLUSION

In this research, we introduced a mechanism based on predictive probability graph approach of data prefetch that takes advantage of the parallelism of hybrid (multi levels) storage environment in order prefetch the valuable data from bottom level to the top one. This process is performed in parallel with the application data read requests. Valuable data blocks are those that are expected to be requested by the user's application very soon. This process helps the users' applications to find more of their future requests in the top level that is distinguished in its fast reading speed. We implemented our solution (i.e. PPDHSS) in a distributed environment where there exist several storage-side nodes of two levels storage hierarchy (i.e. SSs and HDs) and client (application) side nodes. PPDHSS reduces the running time in about 2% when using small size cache and 4% when using moderate size one.

ACKNOWLEDGEMENT

This research was discussed with my advisor Dr. Xiao Qin and his research group at Auburn University. System parameters were validated at his research laboratories. I would like to express my sincere gratitude for his support.

REFERENCES

1. Al Assaf, M. M., Jiang, X., Qin, X., Abid, M. R., Qiu, M., & Zhang, J. (2018). **Informed prefetching for distributed multi-level storage systems**. Journal of Signal Processing Systems, 90(4), 619-640. https://doi.org/10.1007/s11265-017-1277-z

2. Al Assaf, M. M. (2015). **Predictive Prefetching for Parallel Hybrid Storage Systems**. International Journal of Communications, Network and System Sciences, 8(05), 161.
3. Al Assaf, M. M., Jiang, X., Abid, M. R., & Qin, X. (2013). **Eco-storage: A hybrid storage system with energy-efficient informed prefetching.** Journal of Signal Processing Systems, 72(3), 165-180. https://doi.org/10.1007/s11265-013-0784-9
4. Jiang, X., Al Assaf, M. M., Zhang, J., Alghamdi, M. I., Ruan, X., Muzaffar, T., & Qin, X. (2013). **Thermal modeling of hybrid storage clusters.** Journal of Signal Processing Systems, 72(3), 181-196.
5. Al Assaf, M. M., Alghamdi, M. I., Jiang, X., Zhang, J., & Qin, X. (2012, August). **A pipelining approach to informed prefetching in distributed multi-level storage systems**. In 2012 IEEE 11th International Symposium on Network Computing and Applications (pp. 87-95). IEEE.
6. Griffioen, J., & Appleton, R. (1994, June). **Reducing File System Latency using a Predictive Approach.** In USENIX summer (pp. 197-207).
7. Li, C., Shen, K., & Papathanasiou, A. E. (2007, March). **Competitive prefetching for concurrent sequential I/O.** In Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (pp. 189-202). https://doi.org/10.1145/1272996.1273017
8. Domenech, J., Sahuquillo, J., Gil, J. A., & Pont, A. (2006, December). **The impact of the web prefetching architecture on the limits of reducing user's perceived latency.** In 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06) (pp. 740-744). IEEE.
9. Zhang, Z., Lee, K., Ma, X., & Zhou, Y. (2008, June). **Pfc: Transparent optimization of existing prefetching strategies for multi-level storage systems.** In 2008 The 28th International Conference on Distributed Computing Systems (pp. 740-751). IEEE.
10. Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2008). **Bigtable: A distributed storage system for structured data.** ACM Transactions on Computer Systems (TOCS), 26(2), 1-26.
11. Nijim, M. (2010, July). **Modelling speculative prefetching for hybrid storage systems.** In 2010 IEEE Fifth International Conference on Networking, Architecture, and Storage (pp. 143-151). IEEE. https://doi.org/10.1109/NAS.2010.27
12. Nijim, M., Zong, Z., Qin, X., & Nijim, Y. (2010, September). **Multi-layer prefetching for hybrid storage systems: algorithms, models, and evaluations.** In 2010 39th international conference on parallel processing workshops (pp. 44-49). IEEE.
13. Lewis, J., Alghamdi, M., Al Assaf, M., Ruan, X., Ding, Z., & Qin, X. (2010, December). **An automatic prefetching and caching system.** In International Performance Computing and Communications Conference (pp. 180-187). IEEE.
14. Erraissi, A., & Belangour, A. (2019) **Hadoop Storage Big Data Layer: Meta-Modeling of Key Concepts and Features.** International Journal of Advanced Trends in Computer Science and Engineering, 8, 646-53. https://doi.org/10.30534/ijatcse/2019/49832019
15. Al Assaf, Maen & Rodan, Ali & Qatawneh, Mohammad & Abid, Mohamed. (2015). **A Comparison Study between Informed and Predictive Prefetching Mechanisms for I/O Storage Systems.** Int. J. Communications, Network and System Sciences. 8.
16. LASR tracemachine01, doi: http://iotta.snia.org/traces/
17. Hadoop Archive Guide, doi: http://hadoop.apache.org/
18. Lim, Sang. (2019). **Classification and Big Data Usages for Industrial Applications.** International Journal of Advanced Trends in Computer Science and Engineering. 8. 1117-1122. https://doi.org/10.30534/ijatcse/2019/18842019
19. Ahamad, D., Akhtar, M., & Hameed, S. A. (2019). **A Review and Analysis of Big Data and MapReduce.** International Journal of Advanced Trends in Computer Science and Engineering, 8(1), 1–3. https://doi.org/10.30534/ijatcse/2019/01812019