Implementing EAFR:An Energy-Efficient Adaptive File Replication Policy for Hadoop Distributed File System

Nurani Saoda
University of Virginia
ns8nf@virginia.edu

ABSTRACT

Hadoop Distributed File System (HDFS) has been widely used for storing large-scale data sets at high transmission bandwidth in data-intensive applications. To ensure high data availability, fault-tolerance and effective load balance among massively parallel servers, HDFS implements a static replication and block placement policy by default. However, the parallel operations in application layer and concurrent request handling characteristics of servers result in varying access rate for the stored files. Consequently, maintaining a static replication factor for every files may fail to provide acceptable performance in terms of request response time and replication delay for hot data and waste storage and energy to store cold data. Furthermore, the default block placement policy of HDFS neglect server heterogeneity in terms of network bandwidth and concurrent request serving capacity. In this paper, we implement an Energy-Efficient File Replication policy (EAFR) for HDFS which provides a dynamic replication strategy based on the varying file access pattern over time and an efficient block placement policy by modeling the server's storage capacity and block transmission speed. EAFR also provides a new storage policy consisting of hot,cold and standby servers which aims to reduce the energy consumption of a large-scale cluster. The goal of this paper is to achieve similar results for EAFR when integrated with HDFS as the prior work on EAFR. Experiments done in the paper show that EAFR in HDFS achieves lower read latency, replication time, power consumption, storage consumption and better load balance than the default HDFS and the results are consistent with prior work.

1. INTRODUCTION

With the increase of large-scale and data intensive applications over the past few decades, cloud and cluster computing system require more storage, I/O capacity and highly fault-tolerant system. Such large-scale cloud storage system exploits a large number of commodity hardware storages to make a single integrated storage to achieve high reliability and data availability in case of any outages such as network failure or hardware malfunction. Distributed file system such as Hadoop Distributed File System(HDFS)[1], Google File System (GFS)[2] have been developed in recent years to provide high fault-tolerant and more reliable storage system for data intensive frameworks. To provide high data availability and fault-tolerance in a massive system, HDFS adopts a static replication policy.

Static replication policy in HDFS divides each individual file into a number of fixed-size block except the last one and replicates them into three servers. For the block placement policy, HDFS maintains rack awareness which ensures the placement of at least one replica to a different node than others.The default replication and replica placement policy gives HDFS resilience to counter system and network failure. However, this static replication scheme fail to take into account the skewness of file popularity and server heterogeneity which can be leveraged to make the system more efficient.

Firstly,the parallel characteristics of application and data intensive computation make some data blocks highly accessed over time. According to the data access pattern, the data in HDFS can be classified into hot data, cold data, cooled data and normal data[4][1]. The data that have high concurrent request and high access per unit time are called hot data, the hot data that are no longer popular are classified as cooled data, the unpopular data are cold data and the rest are normal data This disparity in data access pattern over time make the naive replication policy of HDFS incompatible to provide faster service and energy efficient service.

Secondly, the default block placement of HDFS depends on random selection of datanodes while maintaining rack awareness policy[3] which does not take into account the capability of the server at handling requests and its storage capacity. Considering storage capacity and transmission bandwidth of the datanode to model the block placement policy can provide faster read and write operation and achieve better load balance across large cluster.

Thirdly, the default replication policy waste resources over storing the less accessed files which keeps all the servers active all the time. As a result, keeping three replicas for such files might not improve significant performance but can cause wastage of storage and energy.

Therefore, to make HDFS more efficient in terms of service time, replication completion time and power consumption, EAFR introduces an energy-efficient replication system. First, to integrate EAFR to HDFS the data access pattern is translated into complex event processing (CEP)[5] by analyzing HDFS audit logs. By processing the events, EAFR tags the files as hot or cold files and changes the replication

factor of the files to ensure enough replicas for hot files and a minimum number of replicas for cold files. Second, EAFR distinguishes the servers as hot, cold and standby server to maintain data availability. Third, EAFR implements a new block placement policy by sorting the available servers according to their remaining storage capacity and block transmission speed.

The contribution of the paper can be described as follows: 1) It introduces a new replication policy for HDFS by implementing EAFR. 2) It brings a new block placement policy for HDFS through the concept of EAFR. 3) It implements a new storage model for HDFS.

The rest of the paper is arranged as follows: section 2 gives an overview of related works, section 3 describes the architecture and principles of EAFR, section 4 is dedicated to analyze the experimental results and section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

2.1 Background

HDFS is a block-structured distributed file system designed to be built on commodity hardware. The high fault-tolerance achieved by HDFS makes it unique from other distributed file systems. HDFS stores files by partitioning each files into a sequence of blocks of same size except the last one. The default block size is set to 64 MB. Blocks are replicated among the machines to achieve higher data availability, fault tolerance and load balancing. The replication is configurable per file and can be changed after the creation of the file. By defualt, the replication factor is three.

HDFS follows a master/slave architecture which consists of a single namenode working as a masternode and multiple datanodes serving as slaves. Namenode and datanodes are softwares that runs on thousands of machines in a large cluster. The namenode maintains the file system namespace and controls access requests made by clients. It stores all the metadata of the file system with directory structures. Namenode also keeps the block ids and block locations of each files. Datanodes are responsible for actually storing the blocks and serves read and write request to clients. It also supervises the block placement and replica creation upon direction from namenode. Namenode receives heartbeat and block report periodically from the datanode. When a client requests access to a file in HDFS, namenode provides the client a list of datanodes that has a copy of the requested file. If the client requests a read operation, the datanodes with the corresponding file serves the request. When a write request is made by the client, the namenode provides the information of the datanode where the data can be written. The client will write to the datanode and the datanode will ask the namenode where the replicas of the files should be placed and then it creates a replication pipeline among the datanodes which are going to store the replicas. The replication is done in a pipeline manner among the datanodes.

To achieve fault-tolerance HDFS has a rack-aware policy that distributes replicas among multiple racks. While placing the first replica, HDFS chooses node from a local rack as destination. For the second replica, HDFS selects a node from a remote rack and the third replica is placed on a different node of the same remote rack. This placement policy cuts inter-rack write traffic in HDFS.

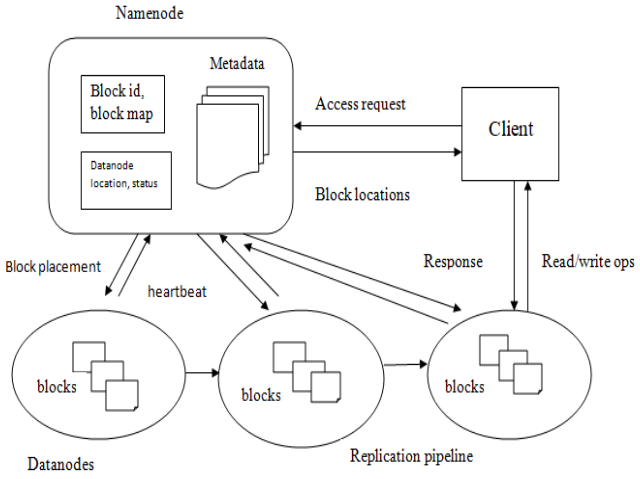


Figure 1: HDFS architecture

However, Studies have shown that the popularity of each block varies with time and not all data blocks have same popularity among clients which depends on different workloads. The default replication policy of HDFS is not efficient for such applications as it gives all the data blocks same preference by replicating them all three times.

2.2 Related Work

In data replication, the research has been fundamentally focused on two specific domains: efficient replication policy and replica placement policy.

A Cost-effective Dynamic Replication Management Scheme called CDRM[6] proposed a cost effective replication framework for cloud storage system which builds a model based on the relationship between replication factor and the expected file availability. The replica placement policy takes into consideration the blocking probability of each datanode, memory capacity, network bandwidth before placing a replica in a datanode. But this method does not consider the heterogeneous nature of file popularity in a large-scale cloud storage.

Another prediction-based off-line replication system proposed by Ananthanarayanan et al. named Scarlett[11] replicates popular files using the historic record of data access pattern. To ensure minimum interference within submitted jobs, it enforces a storage budget limit and data compression. Scarlett achieves better load balance in cluster by spreading replicas to avoid hotspots but it does not specify the replica placement policy succinctly.Also, the concept of aging to replace old replicas might cause some cold data to be lost.

Elastic replication management system (ERMS)[4] models the storage systems in HDFS as active/standby model and analyzes the data access using complex event processing. It classifies the data into hot and cold data and changes the replication factor accordingly using a technique called Condor. ERMS adopts erasure coding to free up disk space by erasing unpopular data.

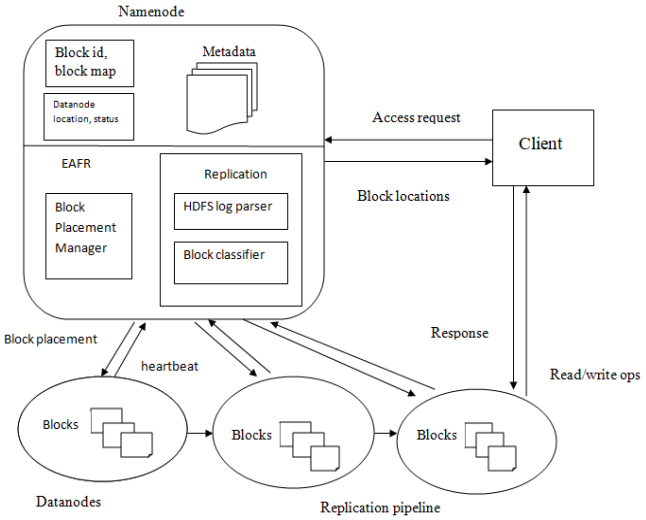


Figure 2: EAFR as an integrated component of HDFS

Adaptive replication management (ARM) [10] uses supervised learning to predict the file popularity and hence, builds a prediction-based replication strategy accordingly. ARM achieves high data availability by implementing enhancement techniques for locality metrics. This model can dynamically change replication factor in runtime.However, this method does not consider the storage or network capacity.

Another block replication technique called Aurora[8] proposed a dynamic block placement algorithm to ensure machine and rack-level reliability and proper load balance.First, it replicated the popular data blocks in large number in comparison with less popular data blocks. Then, It implements an optimal local search framework for block placement that achieves a minimum data availability and a good balancing among the servers.But this method does not take into account server heterogeneity while placing the replicas.

Dynamic replication strategy (DRS)[12] uses improved Markov model to distinguish between hot and cold in HDFS.DRS is comprised of a dynamic replica adjustment strategy (DARS) and a homogeneous replica placement strategy (HRPS). DARS computes transition probability matrix by defining the states of the files as Markov states. Based on the initial state and the transition matrix, DRS decided the number of replicas for each files. HRPS achieves uniform load distribution by keeping relevant data on same rack or node. However, this policy does not take into account the energy consumption of the servers.

By analyzing some of the relevant work, we found that the existing replication strategies does not focus significantly on the network bandwidth and the server capacity when dynamically increasing or decreasing replication factor for different type of data in the system.

3. ENERGY-EFFICIENT FILE REPLICATION SYSTEM IN HDFS

3.1 System Architecture Overview

Figure 2. shows the system architecture of our model to implement EAFR[13] in HDFS. We chose to analyze HDFS audit logs to get the necessary metrics[13] to implement the replication policy proposed by EAFR.Our method is designed to analyze HDFS file access pattern with the help of the unit called HDFS log parser. HDFS log parser analyzes the dynamic audit logs of HDFS for the clusters and translates them into metrics needed to classify the data as hot or cold. The metrics collected by the HDFS log parser are the number of file access count for each file and a file popularity map for all the files stored HDFS. The popularity metrics is defined as the number of access per unit time. The log parser keeps record of the file access count and uses it to classify the files as hot or cold and assigns a tag for the classified data. Block classifier interprets the data collected by the log parser and dynamically changes the replication factor for files and creates more replicas for hot files. This model automatically fits for cooled data and normal data as it removes the extra replicas when a hot data turn into a cooled data. This automatic handling is achieved by choosing a suitable time window for the log parser to build the file popularity map.

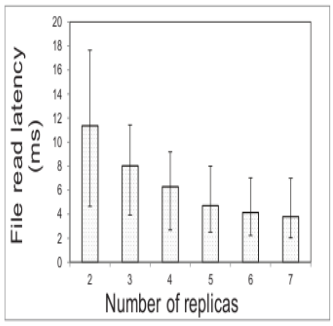
Also. EAFR proposes a storage model to quarantine the highly accessed data from the rarely accessed to minimize the power consumption in large clusters. The hot, standby and cold server storage model takes into account the file popularity map and chooses the hot/standby server for hot replicas and cold servers to store the cold replicas

3.2 A Brief Overview of Replication and Block Placement Policy of EAFR

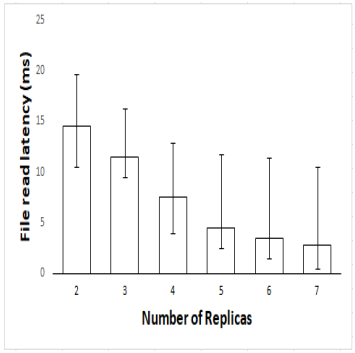
The EAFR model for HDFS can be sectored into two major components: Replication Manager and Block Placement Manager. The model designed to implement EAFR in HDFS is our own implementation idea which is completely different from the methods used to implement EAFR in [13]

The Replication Manager consists of two handlers: HDFS log parser and block classifier. Firstly, to identify the file blocks as hot/cold data, EAFR analyzes the log files generated at the time of client requesting an access to a certain file. By counting the log access event for a specific file, HDFS log parser keeps track of the file count over time. The log parser works in a reasonable time window which is adjusted to not get trapped in local minima rather gather information overall. The time window was added as a configuration parameter for HDFS. Within the chosen time window, HDFS log parser generates a file popularity map for each of the file stored in the system. File popularity is calculated as the file count number per unit time. This map keeps on updating with time as the access pattern varies with time along days[7]. Secondly, after getting the metrics from the HDFS log parser, Block classifier classifies the file in the system as hot and cold files according to the condition specified in section III B-2. All the calculation to tag a block as hot or cold and dynamic change of replication factor satisfy the equations and conditions mentioned in [13].

Block Placement Manager of the integrated EAFR model is designed to take into acount server heterogeneity in terms of network bandwidth, transmission speed and server storage capacity. The manager builds a sorted tree of the available hot datanode candidates that have the potentiality to store a new replica for hot file. The sorting is done based on

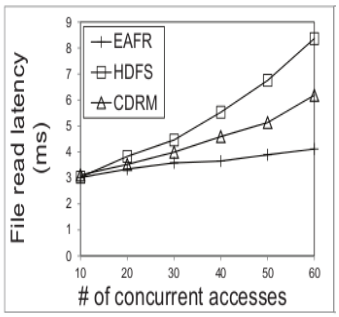


(a) File read response time from EAFR paper

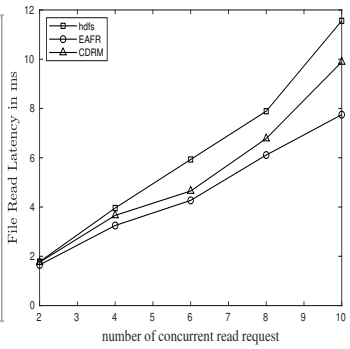


(b) File read response time with EAFR in HDFS

Figure 3: Performance under different number of replicas

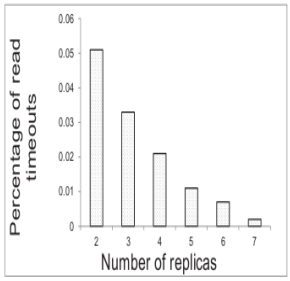


(a) File read response time in from EAFR paper

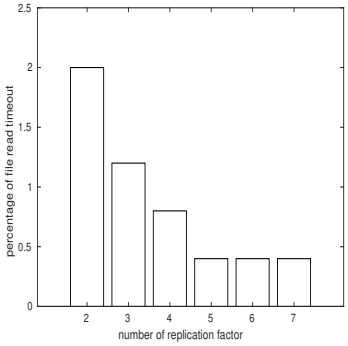


(b) File read response time with EAFR in hadoop

Figure 5: File read response time



(a) Percentage of file read timeouts from EAFR paper



(b) Percentage of file read timeouts with EAFR in HDFS

Figure 4: Performance under different number of replicas

the descending probability map assigned to candidate nodes. The candidate with high probability score is highly likely to be chosen as the destination for the new replica. The probability score is calculated from the order of the transmission speed and available storage capacity of the candidates. Firstly, EAFR builds a sorted tree for the candidates based on their transmission speed which is calculated over an exponential weighted moving average model. Secondly, EAFR builds another tree which consist of the sorted order of the candidates according to their available storage capacity. The candidate with higher position in either of the tree have a high probability to be selected as a destination for keeping the new replica for hot data blocks. For more detailed calculation the reader is encouraged to look at [13].

4. EXPERIMENT AND RESULT

This part of the section consists of two part: experiment and results. The experiment section will describe the environment to test EAFR in HDFS cluster and the results section will show the performance of EAFR when integrated into HDFS. The results are also compared with the results of [13] and it confirms that EAFR can achieve the same performance enhancement when integrated into Hadoop HDFS. Also, we compared EAFR in HDFS with default HDFS[3] and CDRM[6].

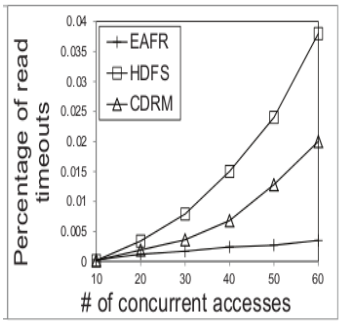
4.1 Experimental setup

We tested EAFR in the cluster of CS department at University of Virginia with one namenode and eight datanodes. The namenode and datanodes has the following configurations: Intel(R) Xeon(R) CPU E5-2660 v4 of 2Ghz, 32GB memory, 450GB disk capacity. The operating system used is Ubuntu 14.04 and the java version used is Oracle Java 8 update 161. The racks are connected with Gigabit ethernet processing.

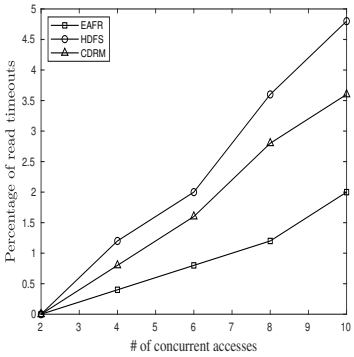
We implemented EAFR in Hadoop-2.8.1 for which we needed to modify Hadoop as the following: a) modify the default replication policy to dynamically change according to data popularity b) modify the block placement policy for the selection of hot servers 3) change the configuration parameter for hot/standby, cold storage.

To apply the replication manager model of EAFR in HDFS, we added a new class in HDFS. For the block placement manager, BlockPlacementPolicyDefault class was modified to choose the best candidate for replica placement according to the principles of EAFR. The whole modified code is around 400 lines. The default block placement policy works according to the same principle described in section 2.1. Instead of randomly choosing a destination for block placement, we added our algorithm described in section 3.2 as proposed in [13].

For the workload generation, we used the CTH trace data[14] and executed a trace-driven experiment to test the performance of EAFR in HDFS. This trace records 3,972,284 I/O calls on 16,566 files during about 4 hours in a large cluster with 3300 client size. We scaled down the trace data to 10000 I/O calls by choosing the first 10000 calls and generated 400 files. Among 400 files, the size of 200 files were set according to byte length of the trace data and the rest of files are within the size of 1KB to 1GB.The server capacity follows the normal distribution with a mean of 15 and variance of 10[13].Other system parameters were set as: $\tau_u = 20$, $\tau_l = 10$, $\sigma_u = 8$, $\sigma_l = 1$, r = 2 and $\varepsilon = 1$ where the notations have the similar interpretation used in [13] for ensuring understandability. We randomly selected 7 of datanodes as hot

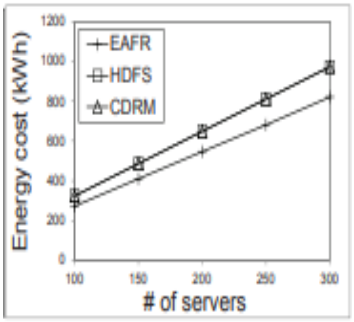


(a) Percentage of file read timeouts from EAFR paper

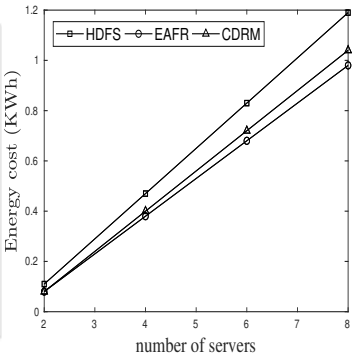


(b) Percentage of file read timeouts with EAFR in Hadoop

Figure 6: File read timeouts

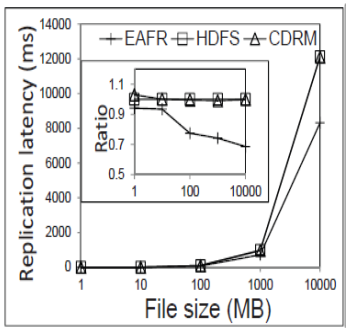


(a) Energy consumption from EAFR paper

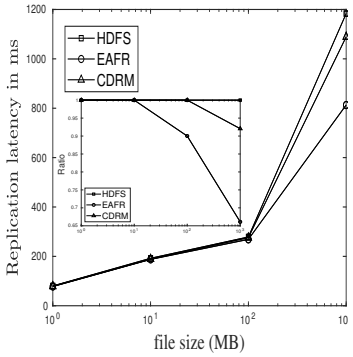


(b) Energy consumption of one hour with EAFR in HDFS

Figure 8: Energy consumption

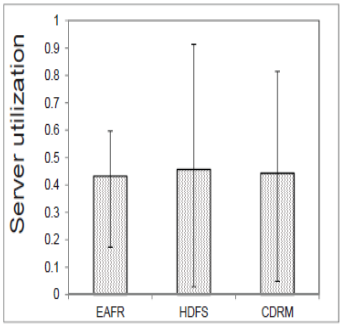


(a) Replication latency for various sizes from EAFR paper

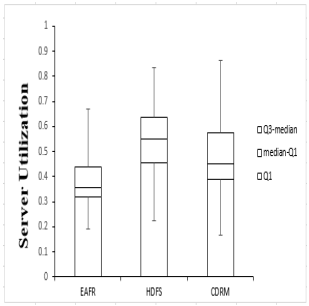


(b) Replication latency for various sizes with EAFR in HDFS

Figure 7: Replication Latency



(a) Server Utilization from EAFR pape



(b) Server Utilization with EAFR in HDFS

Figure 9: Load balance status

servers, and 3 of datanodes as archival servers by changing hadoop configuration parameter. The trace was conducted for 1 hour and the time interval of the HDFS log parser was set to .5 hour. The time difference between I/O calls follows the timing pattern of the trace data. The timing window for the moving average model in the block placement was set to 20 min. The rest of the configuration parameter was set according to [13].

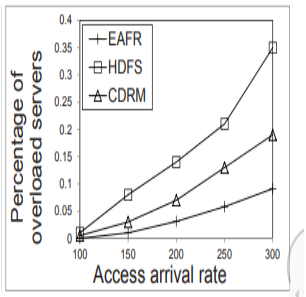
4.2 Results

The goal of this subsection is to repeat the experiments done in [13] and compare the results our EAFR model in HDFS with the prior work in [13].Also, We compare EAFR integrated in HDFS with default HDFS and CDRM.

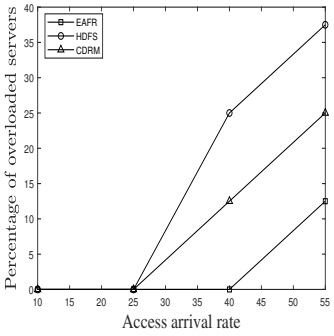
4.2.1 File Read Response Latency

1) Number of replicas: From the 400 files, we selected 20 random files and generated 5 concurrent read request for each of them. Figure 3b) shows the 1st percentile, median and 99th percentile read response time as we increased the number of replicas from 2 to 7 with EAFR in HDFS. Since, EAFR will be dynamically increasing the replication factor, it is worth analyzing the performance in terms of read latency. We can see from figure 3b) that the file read latency decreases as the increased number of replica provides better service of the read request from the server. We set the threshold of 10 ms for read timeout and calculated the percentage of read timeouts as we increased the number of replicas for the files.From figure 4b) The percentage of read timeout also decreases as expected. From figure 3a) and figure 4a) we can see that the results are similar in changing trend with the results of [13] without implementation in HDFS.

2. Number of concurrent read requests: To measure the read latency under concurrent request, we varied one read in the trace data from 2 to 10 concurrent reads and calculated the average read latency for various number of concurrent read request. We can see from figure 5a) and 5b) that as the number of concurrent request increases the read latency is increased as the server needs to serve more read requests. From figure 5b) we can observe that with EAFR in HDFS achieves lower read latency than the default HDFS and CDRM. EAFR in HDFS has lower read latency as it adaptively increases the number of replicas of hot files which are accessed more frequently. HDFS shows worst performance as it cannot change the replication factor adaptively for hot files and randomly selects a server as a destination. CDRM outperforms HDFS as takes into account block pop-

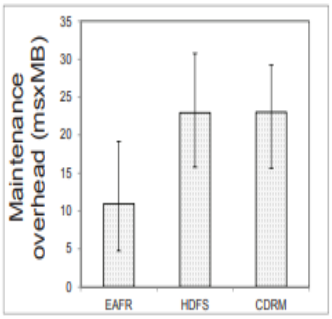


(a) Percentage of overloaded servers

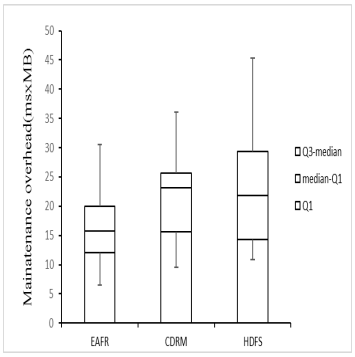


(b) Percentage of overloaded servers with EAFR in HDFS

Figure 10: Percentage of overloaded servers

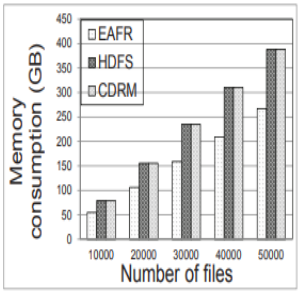


(a) Maintenance overhead from EAFR paper

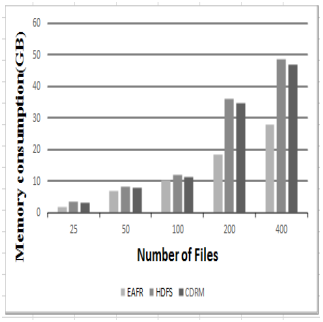


(b) Maintenance overhead with EAFR in HDFS

Figure 13: Maintenance overhead

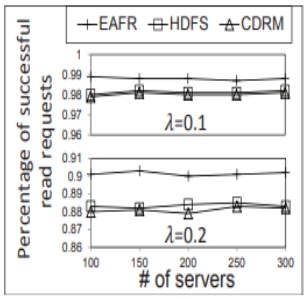


(a) Memory Consumption from EAFR pape

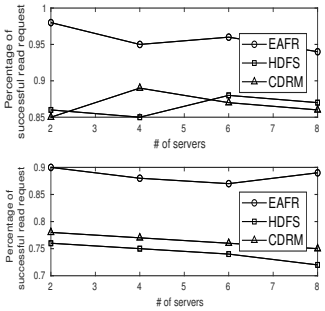


(b) Memory Consumption with EAFR in HDFS

Figure 12: Memory Consumption



(a) File availability when r=3 from EAFR pape



(b) File availability when r=3 with EAFR in HDFS(upper figure,λ = .1, lower figure,λ = .2)

Figure 14: File availability

ularity and blocking probability of servers to place replicas. Both the figures match the result of EAFR without HDFS[13]. Figure 6b) shows the average percentage of file read timeouts and EAFR outperforms other methods for the reasons specified above.

4.2.2 *Replication Completion Time*

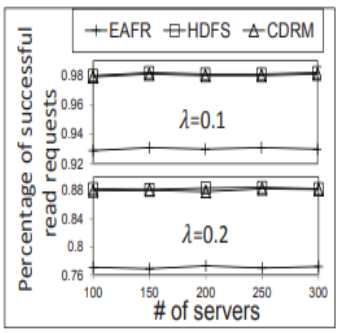
Files of same sizes are grouped from 1MB to 1GB and the average replication completion time was measured. As file size increases, the replication latency increases as it takes more number of blocks to spread among the destination datanodes. Figure 7b) shows that EAFR produces lower replication latency as the file size goes up. This is because EAFR takes into account the history of block transmission over a time period and chooses the faster servers as destinations. We also plotted the ratio of the replication latency with the replication latency of default HDFS as base. Also, the corresponding result of [13] is similar with the result we have found.

4.2.3 *Energy Efficiency*

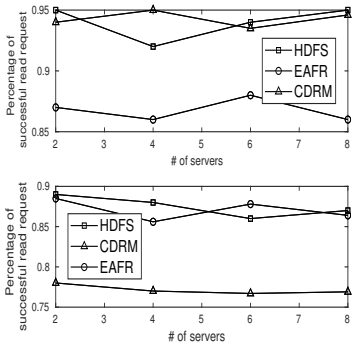
We measured the power consumption of 8 datanodes using the power chart for different CPU utilization in [15]. We computed the average CPU utilization of each server over one continuous hour. The energy consumption of one hour is shown in kWh in figure 8b) as the number of datanodes were increased from 2 to 10. We conducted the same experiment for each methods. We can see from figure 8b) EAFR in HDFS consumes much less energy as it implements the servers as hot/standby and cold servers. Figure 8a) shows the power consumption of EAFR in the experiments done in [13].

4.2.4 *Load Balance Status*

We took 10 utilization values at a frequency of once per minute and the plotted the values in figure 9b). Server utilization value is defined as the ratio of the number of concurrent request a server is serving to the server capacity. The server is considered overloaded if the utilization value is over .8 . For the figure 10b), we plotted the percentage of overloaded servers for the three methods with respect to access arrival rate in ms. As EAFR considers the storage capacity of the servers when choosing the replica for a hot data it achieves better load balance. CDRM performs better than HDFS as it takes into consideration the blocking probability of the server and creates a tree structure to search for the best candidate, whereas HDFS does not consider node performance while placing replicas. The blocking probability is defined as the probability that a new request will be blocked



(a) File availability when r=2 from EAFR paper



(b) File availability when r=2 with EAFR in HDFS(upper figure,λ = .1, lower figure, λ = .2)

Figure 15: File availability

by every datanode when the number of concurrent requests it can handle exceeds a pre-defined threshold. Figure 9a) and 10a) show the result of EAFR without HDFS as in [13]. If we compare our result with [13] we see that we got steeper curves as we have only 8 datanode in contrast to the experiment of [13] where they had 300 nodes.

4.2.5 Overhead

In figure 12b) we compare the memory consumption of EAFR in HDFS with the other two methods and we can see that EAFR outperforms both the method as it reduces the number of replicas for a cold file and keeps a minimum of 2 replicas. Figure 13b) shows the maintenance overhead of the three methods. Maintenance overhead is defined as the product of the time needed to send writes to all its replicas multiplied by the size of the file. HDFS with EAFR achieves lower maintenance overhead than default HDFS and CDRM because EAFR has less number of replicas for cold files which reduces the number of overall writes.

4.2.6 Server Failure Resilience

In figure 14b) we evaluate the performance of EAFR in HDFS when the servers has a failure probability of λ = .1 and λ = .2. The failure probability was used a configurable parameter for testing. We set the threshold for successful read to 100 ms which is higher than the specified value of 10ms in [13]. We chose to increase the threshold to achieve a comparable setting as [13] since the inter rack switch speed in our cluster is slower than [13] which had infinite bandwidth capacity. For a minimum replication factor of 3 , EAFR can serve more successful read request than CDRM and default HDFS. But for a minimum replication factor of 2, EAFR does not show good performance as it decreases the number of replication which result in higher read timeout. Figure 13a) and 14a) shows the result from [13] for EAFR without HDFS. As in our experiment we had,less files (400 instead of 50000), we got more sparse data points than [13] where they had 50000 files.

5. CONCLUSION

For data-intensive and work-load heavy clusters, high data availability can be ensured for frequently accessed hot data while storage is saved for cold data by adjusting replication factors.We tested the performance of EAFR in HDFS which dynamically adjusts replication factor based on block popularity and showed that it reduces read latency, replication latency, energy consumption and achieves better load balancing and server utilization than default HDFS and CDRM. All the results are consistent with the prior work on EAFR without HDFS based environment.

6. REFERENCES

[1] K. Shvachko, K. Hairong, S. Radia, and R. Chansler. The hadoop distributed file system, *In Proc. of MSST, 2010.*

[2] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, The Google File System, *Proceedings of 19th ACM Symposium on Operating Systems Principles(SOSP 2003)*, New York, USA, October, 2003.

[3] HDFS Architecture Guide http://hadoop.apache.org/common/docs/stable/hdfs$_d$*esign.html*

[4] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, and G. Guan. Erms: An elastic replication management system for hdfs. *In Proc. of CLUSTER Workshops, 2012.*

[5] S. Cheng, Z. Cheng, Z. Luan, and D. Qian, NEPnet: A scalable monitoring system for anomaly detection of network service, *in Proceedings of the 7th International Conference on Network and Service Management (CNSM 11), 2011, pp. 1-5.*

[6] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster. *in Cluster Computing (CLUSTER), 2010 IEEE International Conference on,* Sept 2010, pp. 188–196

[7] C. L. Abad, Y. Lu, and R. H. Campbell, Dare: Adaptive data replication for efficient cluster scheduling. *in CLUSTER. IEEE,*2011, pp. 159–168.

[8] Q. Zhang, S. Q. Zhang, A. Leon-Garcia, and R. Boutaba, Aurora: adaptive block replication in distributed file systems,. *in Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems,* Columbus, USA, 2015

[9] W. Dai, I. Ibrahim and M. Bassiouni, A New Replica Placement Policy for Hadoop Distributed File System *in Proceedings of the 2016 IEEE 2nd International Conference on High Performance and Smart Computing,*, pp. 262-267, New York, USA, April 2016.

[10] Dinh-Mao Bui, Shujaat Hussain, Eui-Nam Huh, Sungyoung Lee. Adaptive Replication Management in HDFS based on Supervised Learning. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2016.*

[11] Ananthanarayanan, G., Agarwal, S., Kandula, S., Greenberg, A., Stoica, I., Harlan, D., Harris, E. (2011, April). Scarlett: coping with skewed content popularity in mapreduce clusters. *In Proceedings of the sixth conference on Computer systems (pp. 287-300).* ACM

[12] Qu, K., Meng, L., Yang, Y. (2016, August). A dynamic replica strategy based on Markov model for hadoop distributed file system (HDFS). *In Cloud Computing and Intelligence Systems (CCIS), 20164*th International Conference on (pp. 337-342). IEEE.

[13] Y. Lin and H. Shen. Eafr: An energy-efficient adaptive file replication system in data-intensive clusters. *In Proc. of ICCCN,*,2005.

[14] Sandia CTH trace data. http://www.cs.sandia.gov/ScalableIO/SNLTra ceData/

[15] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. CCPE, 24(13):1397–1420, 2012.