EAFR: An Energy-Efficient Adaptive File Replication System In Data-Intensive Clusters

Yuhua Lin and Haiying Shen
Department of Electrical and Computer Engineering
Clemson University, Clemson, South Carolina 29634
{yuhual, shenh}@clemson.edu

***Abstract*—In data intensive clusters, a large amount of files are stored, processed and transferred simultaneously. To increase the data availability, some file systems create and store three replicas for each file in randomly selected servers across different racks. However, they neglect the file heterogeneity and server heterogeneity, which can be leveraged to further enhance data availability and file system efficiency (in terms of replication delay and request response delay). As files have heterogeneous popularities, a rigid number of three replicas may not provide immediate response to an excessive number of read requests to hot files, and waste resources (including energy) for replicas of cold files that have few read requests. Also, servers are heterogeneous in network bandwidth, hardware configuration and capacity (i.e., the maximal number of service requests that can be supported simultaneously), it is crucial to select replica servers to ensure low replication delay and request response delay. In this paper, we propose an Energy-Efficient Adaptive File Replication System (EAFR), which incorporates three components. It is adaptive to time-varying file popularities to achieve a good tradeoff between data availability and efficiency. Higher popularity of a file leads to more replicas and vice versa. Also, to achieve energy efficiency, servers are classified into hot servers and cold servers with different energy consumption, and cold files are stored in cold servers. Further, EAFR selects a server with sufficient capacity (including network bandwidth and capacity) to hold a replica. Experimental results on a real-world cluster show the effectiveness of EAFR in reducing file read latency, replication time, and power consumption in large clusters.**

***Index Terms*—Energy-efficient, Data-intensive clusters, File replication, Replica placement**

I. Introduction

Data intensive computing has been gaining popularity rapidly, and the storage and server demands from computing workloads have been growing exponentially. File storage systems are an indispensable component for data-intensive clusters. Various file systems have been developed, such as Hadoop Distributed File System (HDFS) [1], Oracle's Lustre [2], Parallel Virtual file system (PVFS) [3], and Ceph [4]. In order to enhance data availability, these file systems create a fixed number of replicas for each file and store the replicas in randomly selected servers across different racks. Then, concurrent I/O requests to the same file are spread across several servers rather than a single one. This uniform replication policy has three advantages. First, it avoids the hazard of single point of failure; the failure of a particular node does not make the data stored in itself unavailable. Second, clients can read the files from nearby servers. Third, it distributes file requests across the replicas, and thus achieves good load balancing.

However, this replication policy neglects the file and server heterogeneity, which can be leveraged to further enhance data availability and file system efficiency. First, the files in a large cluster exhibit wide disparity in popularity. For example, the data in HDFS can be classified into four categories according to their access patterns and popularity [1], [5]: hot data, cooled data, cold data and normal data. For cold data that is rarely requested, too many replicas may not improve file availability, but instead lead to unnecessary storage cost. Therefore, in order to improve replica efficiency, we should increase the replication factor (i.e., the number of replicas of a file) of hot data to guarantee data availability and load balance, and reduce the replication factor of cold data to save the storage cost.

Second, energy consumption contributes a significant portion of management cost for datacenters [6]. The energy cost of equipment during its lifetime is comparable to the initial equipment purchase price [7]. Existing file systems randomly select servers in each rack to replicate data (called replica destinations), but do not consider selecting replica destinations to reduce energy consumption. The file replication system actually can reduce energy consumption based on file popularity heterogeneity. We can separate the cluster into hot servers with high CPU utilization (i.e., high power consumption) and cold servers with low CPU utilization (i.e., low power consumption), and place the replicas of popular data in hot servers, which provide high performance, and place the replicas of cold data in cold servers as data backup.

Third, the random selection of replica destinations neglects server heterogeneity (i.e., different servers vary in network capacities and data request handling capacities). The writes due to creating replicas in production clusters at Facebook and Microsoft account for almost half of all cross-rack traffic [8]. Though the network inside clusters is generally underutilized, there exist some bottleneck links resulting from the network usage imbalance [9]. As the traffic in multi-tenant datacenters is not controlled, the traffic congestion of bottleneck links leads to performance degradation inside clusters [10]. If a large number of replicas are written to the same server simultaneously, the server may run out of network capacity and data request handling capacity. Thus, it is important to choose replica destinations to steer replica transfers away from network bottlenecks and overloaded servers.

A number of important challenges need to be overcome to achieve the aforementioned goals in file replication systems. First, the replication factor of each file should be dedicated

978-1-4799-9964-4/15/$31.00 ©2015 IEEE

assigned based on the request rate and availability of the file. Second, we need to maintain data availability when reducing energy consumption. Third, in order to avoid network bottlenecks, we need to effectively identify overloaded servers. In this paper, we propose an Energy-Efficient Adaptive File Replication System (EAFR), which incorporates three components. 1) It is adaptive to the time-varying file popularities to achieve a good tradeoff between data availability and efficiency. Higher popularity of a file on overloaded servers leads to more replicas and vice versa. 2) To achieve energy efficiency, servers are classified into hot servers and cold servers with different energy consumption, and hot/cold files are stored in hot/cold servers, respectively. 3) It selects servers with sufficient capacity (including network bandwidth and capacity) as replica destinations. The remainder of the paper is organized as follows. Section II gives an overview on the related work. Section III introduces the design of EAFR. The performance evaluation is presented in Section IV. Section V concludes the paper with remarks on future work.

II. Related work

File replication is a common strategy to improve data reliability and availability in large clusters. HDFS [1], Lustre [2] and PVFS [3] maintain a constant number of replicas for each file, and replicas of the same file are placed in randomly selected servers. Many methods [11]–[13] have been proposed to improve the replication policy for different purposes. CDRM [11] adjusts the replication factor to maintain a required availability for each file under server failures based on the relationship between file availability and replication factor when servers have a certain probability to fail. Scarlett [12] aims to speed up the jobs by increasing the replication factor in the MapReduce systems. It is an off-line system that studies the file access patterns, and computes a replication factor for each file with a replication budget for load balance. In order to improve data locality in the MapReduce systems, DARE [13] replicates remote data into the local node when a map task processes data from remote nodes. Unlike the above replication works, EAFR aims to improve the data availability with the consideration of file popularity and file storage system efficiency.

Hedera [14] aims to maximize aggregate network utilization by collecting flow information from constituent switches. It studies the traffic demands and routing flows, and instructs switches to re-route traffic accordingly. Orchestra [15] studies the short-term traffic, then incorporate scheduling policies such as multipath routing and transfer priority at the transfer level to improve network performance inside clusters. These schedulers are based on the constraint that the traffic sources and destinations are already fixed, EAFR flexibly selects the servers with available network capacity to avoid network bottlenecks.

Energy-conservation in large-scale datacenters has drawn considerable research attention. Some studies [16], [17] aim to reduce the power costs by dynamically transitioning the servers to a sleeping state in datacenters. of each file is stored

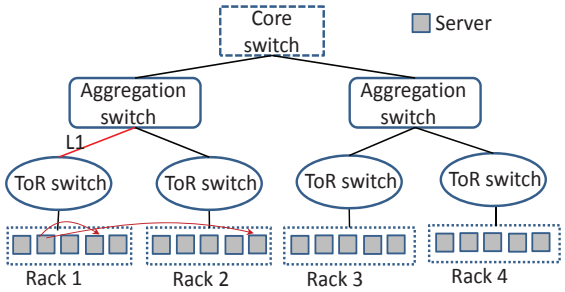


Fig. 1: Architecture of hierarchical storage system.

on active servers to provide service. GreenHDFS [18] trades performance for energy saving by logically separating the Hadoop cluster into hot and cold zones. Cold zone keeps low power consumption but provide less critical response for file accesses (i.e., long latency); while hot zone consumes more power and has strict performance requirements. Different from these works, EAFR considers file popularity when allocating file replicas in order to save energy.

III. System Design

A. Background of File Replication

A typical cluster file system uses a hierarchical storage architecture, as shown in Figure 1. The bottom layer consists of a set of storage servers, where the files (aka objects or blocks) are stored. In order to guarantee data availability in face of network failure or hardware damage, cluster file system makes multiple replicas for each file [12]. A replication factor (r_i) and a fault-tolerance factor (π_i) for file (f_i) are predefined in the system, which ensure that each file f_i has r_i replicas and the replicas are distributed in more than ($\pi_i < r_i$) fault domains (i.e. racks). Typically, HDFS uses $r_i = 3$ and $\pi_i = 1$, i.e., each file is stored in three servers across at least two racks. The red arrows in Figure 1 shows an example of the distributed writes when storing a file with $r_i = 3$ and $\pi_i = 1$. This replication systems neglect the heterogeneity in file popularity. As a result, copying files to only three servers is insufficient to meet the stringent response requirement for hot files but wastes resources for cold files.

On top of the servers are ToR switches, which are located within the Ethernet to aggregate the connectivity of all servers. The ToR switches maintain connections to the rest of network through aggregation switches, on top of which is the core switch. In this network architecture, the link capacity between switches is bounded by hardware limitations (e.g., NIC speed). Though link utilizations inside clusters are generally low and stable, there exist network congestions due to skewed link utilization [9]. For example, link L1 (marked in red) in Figure 1 may become a bottleneck if the there are many writes to Rack 1. Current replication policy does not consider link utilization when transmitting file replicas. Also, current clusters must keep all servers running constantly to guarantee file availability, which is very costly in power consumption. The files are skewed in popularity, and many files rarely get

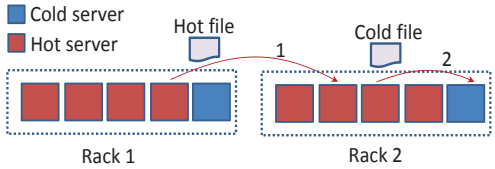


Fig. 2: An overview of EAFR.

accessed during their lifetime [13]. Thus, we can save power and management expense by putting servers storing the cold files in a "powernap" state. We summarize the shortcomings of current file replication methods as follows:

- A fixed number of replicas for each file is insufficient to provide quick file read for hot files while wastes resources for storing replicas of cold files.
- Random selection of replica destinations requires keeping all servers active to ensure data availability, which however wastes power consumption.
- As the random selection of replica destinations does not consider destination bandwidth and request handling capacity, network congestions may occur due to capacity limitation of some links and server may become overloaded by data requests.

The goal of EAFR is to cope with these problems and provide an effective and energy-efficient file replication strategy. In this paper, if a file is striped into multiple blocks, we treat each block as an independent file.

B. Energy-efficient and Popularity-adaptive File Replication

In large data-intensive clusters, most popular files are generally small in size, while large files seldom get read [19]. Therefore, replicating and migrating popular files is relatively light in storage and bandwidth cost. Taking advantage of this characteristic in clusters, EAFR increases the number of replicas of popular files in order to boost their availability and reduces the number of replicas of cold files in order to save resources. Figure 2 shows an overview of EAFR. EAFR divides servers into hot servers and cold servers: hot servers consume more power and provides prompt response for file requests; while cold servers stay in sleeping mode with 0% CPU utilization and low energy consumption. Each file f_i must have r replicas in all servers and $\epsilon < r$ replicas in hot servers, where r and ϵ are pre-defined numbers to guarantee file availability under server failures. For a file with a high visit rate, EAFR creates an extra replica and places it to a hot server, which is shown in step 1. The new replica is used to balance the read requests of a hot file among servers where the file replicas are stored. For a file with a low visit rate (i.e., a cold file), EAFR reduces the number of replicas of the file in the hot servers if it is larger than ϵ. That is, a replica in a hot server is either deleted or migrated to a cold server in order to save the power consumption, which is shown in step 2. This operation does not affect the availability of the cold file as it rarely gets read. In the following, we introduce how to set hot servers and cold servers (Section III-B1), how to identify hot files and cold files based on their visit rates (Section III-B2), and the details of the energy-efficient and popularity-adaptive file replication algorithm (Section III-B3). Table I shows the notations used in this paper.

TABLE I: Table of important notations.

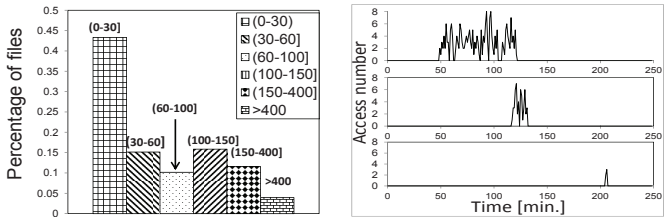
f_i: File i	s_j: Server j
r_i: # of replicas	π_i: Fault-tolerance factor
a_{ij}: Replica j for f_i	v_i: Total # of reads for file f_i
v_{ij}: # of reads for replica a_{ij}	
c_{c_j}: Service capacity of server j	
ϕ_j: Remaining capacity of server j	
b_j: Size of file j	
c_{s_i}: Storage capacity of server i	
∇: Remaining storage threshold	
τ_u: Upper bound for total # of reads of hot file	
σ_u: Upper bound for total # of reads of hot replica	
τ_l: Lower bound for total # of reads of cold file	
σ_l: Lower bound for total # of reads of cold replica	
V_t: Transmission speed in time window t	
w_{t_j}: Weight of selecting server j based on transmission rate	
w_{r_j}: Weight of selecting server based on remaining capacity	
p_j: Chance of selecting server j based on overall evaluation	
ϵ: minimum # of replicas stored in hot servers	

1) Different Types of Servers Based On Energy Consumption: Transitioning servers to an inactive, low power sleep /standby state (i.e., scale-down) is a technique to conserve energy. It trades energy consumption with server performance (e.g., CPU utilization). Table II shows the power consumption characteristics of the HP ProLiant ML110 G5 server at different server performances represented by server CPU utilization [20]. In EAFR, we define three types of servers: hot servers, cold servers and standby servers. A hot server runs at the active state, i.e., with CPU utilization greater than 0. A cold server is in the sleeping state with 0 CPU utilization and inactive DRAM and disks and it does not serve any file read request. A standby server is a temporary hot server that will be transitioned to a cold server. To maintain the consistency of stored files, cold servers wake up periodically (e.g., once a week) and check for file consistency. When there is a rack failure or a server failure inside clusters, cold servers storing the lost files will be turned on and become hot servers.

TABLE II: Energy consumption for different CPU utilizations [20].

CPU Utilization	0%	20%	40%	60%	80%	100%
Power (in Watts)	93.7	101	110	121	129	135
Server status	cold	hot	hot	hot	hot	hot

Writing a file to a cold server needs to wake up the server, which consumes more energy and may offset the benefit of sleeping [21]. Also, it creates excessive latency to transition a server from sleeping mode to active mode and thus delays the write operation. Therefore, we use a standby server to collect all cold files and turn into a cold server when its storage is full. A standby server still serves file requests as hot servers do.



(a) Percentage of files with different number of reads.

(b) The number of file reads over time.

Fig. 3: Trace analysis on file read pattern.

2) Different Types of Files Based on Read Rates: In HDFS, more than 90% files exhibit a relatively short hotness lifespan (i.e., less than 3 days) and a significant portion of data is cold (i.e., never gets read) [22]. In order to justify the heterogeneity of file popularity in large clusters, we analyzed the file storage system trace data from Sandia National Laboratories [23], which records the number of file reads for 16,566 accessed files during 4 hour run. Figure 3(a) shows the percentage of files attracting different range of file reads. We see that about 43% files receive less than 30 reads and 4% files receive a large number of reads (i.e., >400). The results confirm that most of these files attract a small amount of file reads and hence do not need many replicas. Popular files constitute a small percentage of files, thus will not generate a large overhead by creating more replicas. We sorted the files by their number of reads within a 4 hour period, then identified files with the 99th, 50th, and 25th percentiles and plotted their read count over time in Figure 3(b) from the top to the bottom, respectively. These figures demonstrate the variation in file access pattern for files with different popularities over time. We see that the files tend to attract a relatively stable number of reads within a short period of time. Thus, extra replicas can be created to meet the frequent short-term read requests for hot files, and then are deleted when they become cold. Inspired by the observations of the previous works and the above analysis, we can group files into different categories based on popularity and perform different operations according to their popularity.

Current replication factor of f_i is r_i, and the replicas of f_i is denoted by vector $A_i = \{a_{i1}, a_{i2}, ...a_{ir_i}\}$. The number of reads for replica a_{ij} at time interval T is denoted by v_{ij}, and the total number of reads for file f_i is denoted by v_i: $v_i = \sum_{j=1}^{r_i} v_{ij}$. First of all, a hot file should have a large number of concurrent reads across all its replicas. We define a hot file as a file with average read rate per replica exceeds a pre-defined threshold (τ_u):

$$v_i/r_i > \tau_u. \tag{1}$$

Secondly, we also consider the read rate of individual replicas. In locality-aware file reads in large clusters, clients read nearby replicas, so a large amount of concurrent reads for a file may be drawn by some replicas, which also reflects the popularity of the file. Therefore, a hot file may gain high read rate in some of its replicas. Thus, if more than a certain fraction (denoted by γ) of a file's replicas attract an excessive number of reads (denoted by σ_u), we consider this file a hot file:

$$\sum_{j=1}^{r_i} I(v_{ij} > \sigma_u) > r_i\gamma_v \ (0 < \gamma_v < 1), \tag{2}$$

where $I(\cdot)$ is an indication function. If either Equation (1) or Equation (2) is satisfied, file f_i is a hot file represented by $H(f_i) = 1$. Similarly, we use Equation (3) and Equation (4) to determine if file f_i is a cold file. In the equations, τ_l is the lower bound of the number of reads per replica in T.

$$v_i/r_i < \tau_l. \tag{3}$$

Equation (3) shows that a cold file receives a small amount of concurrent reads across its replicas. A cold file waste storage space if the number of its replicas is large.

$$\sum_{j=1}^{r_i} I(v_{ij} < \sigma_l) > r_i\gamma_v \ (0 < \gamma_v < 1) \tag{4}$$

In Equation (4), σ_l denotes the lower bound for the number of reads that a file replica receives in T. If more than γ_v fraction of a file's replicas attract few reads, the file is potentially cold. As creating a replica consumes network traffic and CPU usage, and the cost of mistakenly deleting a file replica is expensive, so we adopt a conservative way to determine if a file is cold. Only when both Equation (3) and Equation (4) are satisfied, file f_i is considered a cold file, represented by $C(f_i) = 1$.

After a file is labeled with its popularity (i.e., hot file or cold file), EAFR adjusts the number of its replicas according to its popularity. The details are presented in Section III-B3.

3) Adaptive File Replication: We first define r and ϵ ($\epsilon < r$), which are the minimum number of replicas a file needs to maintain in all servers and in hot servers, respectively, to guarantee file availability.

Consider a large cluster consisting of: 1) p hot servers, which are denoted by a set HS=(hs_1, hs_2,..., hs_p); 2) q cold servers CS=(cs_1, cs_2,..., cs_q); and 3) w standby servers SS=(ss_1, ss_2,..., ss_w). For a file f_i with r_i replicas, we use a set S_i=(s_1, s_2,..., s_{r_i}) to represent the servers that store its replicas. For server s_j, we define its service capacity (c_{c_j}) as the maximum number of concurrent file reads it can support. We use h_j to denote the concurrent reads s_j receives. If $h_j/c_{c_j} > \tau_c$, where τ_c is a threshold (e.g., 0.8), server s_j is considered as overloaded; otherwise, it is a lightly loaded server. The remaining capacity of a lightly loaded server s_j is calculated by $\phi_j = c_{c_j} - h_j$, which indicates the number of additional file requests it can handle.

At time T, if file f_i is hot ($H(f_i) = 1$), EAFR examines the load status of all server in S_i=(s_1, s_2,..., s_{r_i}). An extra replica is needed for f_i if more than γ_s $(0 < \gamma_s < 1)$ fraction of these servers are overloaded, that is:

$$\sum_{s_j \in S_i} I(h_j/c_{c_j} > \tau_c) > r_i\gamma_s \ (0 < \gamma_s < 1). \tag{5}$$

When the inequality in Equation (5) is met, the current replica servers of f_i do not have enough capacity to handle an excessive number of file reads. EAFR then increases the number of replicas of f_i by 1. The new replica will be placed in a hot server, so that it can serve new incoming file requests.

If $C(f_i) = 1$, f_i is cold and it draws few file reads. Then,

the number of f_i's replicas can be reduced in order to save the storage. The rule of replica reduction is to delete a replica in a hot server, while still maintaining at least ϵ replicas in hot servers in order to guarantee file availability. In the replica reduction stage, EAFR first checks the number of replicas for f_i, i.e., r_i. If $r_i > r$ and the number of replicas in hot servers is larger than the threshold ϵ, EAFR chooses the server with the least remaining capacity and deletes the replica of f_i from it. That is, the selected server s_j satisfies:

$$s_j = \arg\min_{s_j \in S_i} \{\phi_j\}. \tag{6}$$

In the case of $r_i = r$, if there are ϵ replicas in hot servers, no action is performed; if more than ϵ replicas are stored in hot servers, one replica is moved from a hot server to a cold server in order to save energy. EAFR selects a hot server s_j with the least remaining capacity according to Equation (6), and migrates the replica of f_i from s_j to a standby server. The standby server functions like hot server (i.e., it serves file requests) before turning to a cold server. Suppose the storage capacity of standby server s_i is c_{s_i}, if:

$$c_{s_i} - \sum_{j=1}^{m} b_j < \tau_s, \tag{7}$$

a standby server is ready to turn cold. b_j is the size of file j, m is the number of cold files that are currently stored in the standby server, and τ_s is the remaining storage threshold.

Algorithm 1 Pseudo-code of EAFR.

```
Determine the popularity of file f_i
if H(f_i) = 1:  //create one replica
    Select a hot server hs_j; place replica in hs_i
end if
if C(f_i) = 1:  //reduce number of replica by one
    if number of replicas r_i > r
        Select s_j according to Equation (6)
        Delete the replica of f_i in hs_j
    else
        if more than ε replicas of f_i are stored in HS
            Migrate one replica of f_i from hs_i to ss_k
            if Equation (7) is satisfied for ss_k
                ss_k turns into a cold server
            end if
        end if
end if
```

Algorithm 1 shows the pseudo-code of EAFR. This algorithm runs periodically to adaptively tune the number of replicas for each file. When a new replica of a file is created, hot servers that do not store the file's replica are candidates to be the new replica holders. In order to reduce the replication completion time and balance server load, EAFR selects the replica destination by considering both the expected transmission rate and server workload status.

C. Replica Destination Selection

In order to complete the file replication within a short time, the connection from source server to destination server should have high transmission rate. For this purpose, we can use an existing method [8] that monitors the network status (e.g., concurrent traffic, link utilization) and selects the links with light traffic [8]. However, such a monitoring method is complicated and requires additional monitoring overhead for large clusters. EAFR estimates a server's network condition based on recent completion time of transmitting a file to the server. This method is based on the rationale that the recent replication completion time can be an indicator of the server's network condition.

We use the exponentially weighted moving average (EWMA) [24] model to estimate the transmission delay of the next file based on the delays of previous file transmissions. Using T as a time window size, EAFR calculates the average file transmission speed from source server s_s to destination servers_d by sliding the time window. Then the transmission speed in the next time window (denoted by V_t) is calculated by:

$$V_t = \alpha \times Y_t + (1 - \alpha) \times V_{t-1} (0 < \alpha < 1), \tag{8}$$

where V_{t-1} is the estimated transmission speed in time window $t-1$, and Y_t represents the actual average transmission speed at time t. α is a constant used to control the degree of weighting decrease.

In addition to short replica transmission latency, the replica should be placed in a server that has sufficient capacity to serve incoming file requests. Given a file from source server s_s, and a set of hot servers HS to place the new replica, EAFR selects a replica destination $s_d \in HS$ such that transmitting the file replica from s_s to s_d takes a short time and s_d has a high service capacity and enough storage capacity. For this purpose, EAFR first selects candidates from all hot servers HS that have enough storage space for this file replica.

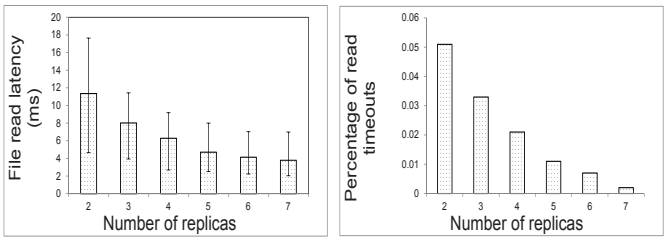
EAFR calculates the expected transmission speed from s_s to all candidate servers, then orders the candidates based on the decreasing order of the transmission delays $ID_t = \{hs^1, hs^2, ...hs^m\}$. EAFR also orders the candidates based on the decreasing order of their remaining capacities $ID_r = \{hs^1, hs^2, ...hs^m\}$. A server having a faster transmission speed or a higher remaining capacity should have a higher probability to be selected. We use w_t and w_c to denote these two probabilities for a server. The probability of the j^{th} server in these two ordered lists can be calculated by: $\frac{1}{j} / \sum_{k=1}^{m} \frac{1}{k}$. The probability of selecting a server in the candidates is calculated by the weighted average of both of its w_t and w_c:

$$p = \beta \times w_t + (1 - \beta) \times w_c \tag{9}$$

We use vector $P = (p_1, p_2, ...p_m)$ to record all probabilities of selecting the candidate servers. Then, s_s selects the destination server based on P. The selection process first generates a random value x within the range of [0, $\sum_{k=1}^{m} p_k$], then server with order y in the list P is selected by:

$$y = \begin{cases} 1 & \text{if } x < p_1 \\ j & \text{if } p_{j-1} \le x < \sum_{k=1}^{j} p_k \text{ and } j > 1 \end{cases} \tag{10}$$

As we can see from Equation (10), the new replica is more likely to be allocated to a server with a high p value.



(a) File read response time. (b) Percentage of file read timeouts.

Fig. 4: Performance under different number of replicas.

IV. Performance Evaluation

We conducted trace-driven experiments in a large-scale HPC cluster located in Clemson University's Palmetto Cluster [25] which has 771 8-core nodes. We deployed EAFR on 300 scattered servers. The storage capacities of these servers were randomly chosen from (250GB, 500GB, 750GB) [26]. We compared EAFR with HDFS [1] and CDRM [11] that are similar to our work. In HDFS, every file has a fixed number of 3 replicas placed across different randomly selected servers. CDRM aims to deal with server failures. In our experiment, CDRM creates 2 replicas for each file initially, it increases the number of replicas to maintain the required file availability 0.98 for server failure probability 0.1, and required file availability 0.8 for server failure probability 0.2. CDRM allocates the newly created replica to the server with the least concurrent reads.

Unless otherwise specified, the distributions of file reads and writes follow those in the CTH trace data [23] and each file read request was forwarded to a randomly selected server that owns the replica of the requested file. 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 created 50,000 files and randomly placed them on the servers. The sizes of 16,566 files were set according to the CTH trace data, and the sizes of other files were chosen from the range (100KB, 10GB). The server capacity follows the normal distribution [1] with a mean of 15 and variance of 10. The remaining storage threshold ∇ was set to 10GB; other system parameters were set as: τ_u=20, τ_l=10, σ_u=8, σ_l=1, $r = 2$ and ϵ=1. We randomly selected 70% of servers as hot servers, and 30% of servers function as standby servers. A standby server with full storage turns into a cold server. The experiment runs 2 days by repeatedly using the read rates from the trace data. We are interested in the following performance metrics:

- *File read latency*: the latency from a user requests a file until the user receives a response from the server.
- *Replication latency*: the latency from a file replication is initiated until the replication operation is finished.
- *Energy cost*: the server power consumption in kilowatt hour (kWh) in each day.
- *Load balance status* including: 1) *server utilization*, which is defined as the ratio of the number of concurrent file requests a server is serving over the server's capacity; and 2) *percentage of overloaded servers* that are defined as the servers with more than 80% utilization.
- *Memory consumption*: the storage usage to store all file replicas (including the original copy) in the system.
- *Maintenance overhead*. An update's maintenance overhead is defined as the product of the latency of this update and the update message size. A file's maintenance overhead is the sum of the maintenance overheads of the updates on all of its replicas.

A. File Read Response Latency

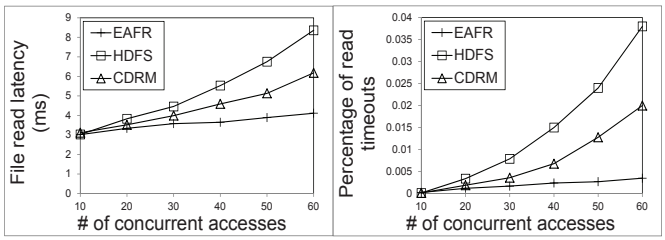
When a hot file attracts a large amount of concurrent reads, some file requests may contend for server capacity and network bandwidth, and hence suffer from response latency.

1) Number of replicas: We selected 20 random files, varied the number of replicas for each file, and generated 60 concurrent read requests towards each file. Figure 4(a) shows the 1st percentile, median and 99th percentile read response time when each file has a different number of replicas. We see that more replicas lead to decreased read response time. This is due to the reason that when more replicas are created in different servers, more server capacity can be utilized to serve the read requests. We define 10ms as the required latency threshold, and record the percentage of file read requests that are served past the required latency. Figure 4(b) shows the percentage of file read timeouts when a different number of replicas are created for each file. We see that the percentage of read timeouts drops gradually when more replicas are created for each file for the same reason as in Figure 4(a).

2) Number of concurrent read requests: We then varied the number of concurrent read requests by replacing one read in the trace data by x reads. x is varied from 10 to 60 increasing by 10 in each step. Figure 5(a) shows the average file read response time with different number of concurrent reads to the same file (i.e., x) in the system. We see that *CDRM* yields less latency than *HDFS*. This is because *CDRM* chooses the server with the least workload as the replica destination, then the server storing the new file replica is likely to have enough capacity to serve file requests. *HDFS* randomly selects replica destination, which may not have enough capacity to handle requests. Thus, *HDFS* incurs longer latency than other two methods as read requests are likely to wait for server response. *EAFR* produces the least read latency because it adaptively increases the number of replicas for hot files, and the new replicas share the read workload of hot files. As *CDRM* does not consider file popularity in replication, file requests towards hot files still need to contend for server capacity. Figure 5(b) shows the percentage of file read timeouts versus the number of concurrent reads. We see that the result follows *HDFS*>*CDRM*>*EAFR* for the same reasons as in Figure 5(a).

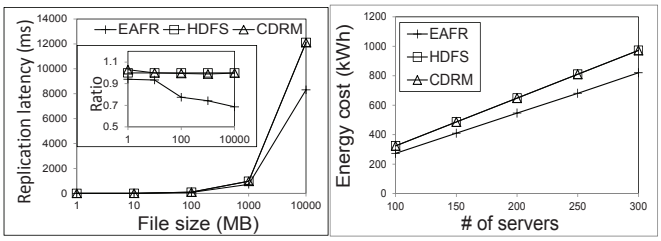
B. Replication Completion Time

We grouped the files with the same size (ranging from 1MB to 10,000MB) together and calculated the average replication latency of each group of files. Figure 6(a) shows the replication completion time for different file groups. We also set the replication completion time of *HDFS* as base and plot the ratio



(a) File read response time. (b) Percentage of file read timeouts.

Fig. 5: Performance under different # of concurrent accesses.



(a) Replication latency for files of various sizes. (b) Energy consumption per day.

Fig. 6: Replication latency and energy consumption.

of other methods' replication completion time over the base in the embedded figure. We see that the replication completion time grows rapidly for files with large sizes. *EAFR* speeds up the file replication especially for large files. This is because *EAFR* predicts the transmission speed based on previous file transmission experience and selects the server with a high transmission rate with high probability.

C. Energy Efficiency

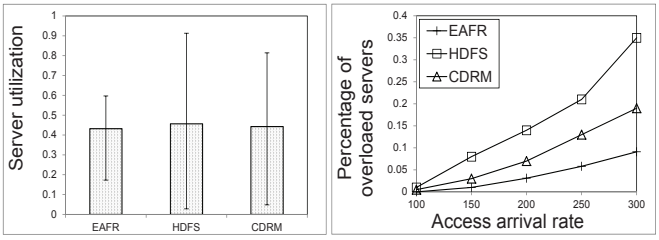
We set the power consumption of different genre of servers according to Table II. Figure 6(b) shows the total amount of energy consumption per day for different methods when various number of servers are used in the cluster. Due to the adoption of cold servers to store cold files that are rarely read by clients, *EAFR* manages to reduce the power consumption by more than 150kWh per day in a cluster consisting of 300 servers. *EAFR* stores some replicas of cold files in cold servers (in sleeping mode), which results in substantial power saving. It is worth noting that while the adoption of cold servers can reduce the energy consumption in large cluster, performance of the cluster in serving file requests is not compromised, which is demonstrated in the previous figures.

D. Load Balance Status

For each server, we sampled 10 utilization values within 10 minutes at a frequency of once per minute, then selected the highest value as the server's utilization to report. Figure 7(a) plots the 1st percentile, median and 99th percentile of server utilizations. We see that *EAFR* achieves better load balance than *CDRM* and *HDFS* with a smaller 99th percentile value and a larger 1st percentile value. *EAFR* adaptively increases the number of replicas for hot files to serve excessive file requests. Also, it creates new replicas in servers with the highest remaining capacity with a high probability. Figure 7(b) shows the percentage of overloaded servers. We see that *EAFR* maintains the least percentage of overloaded servers due to the same reason as in Figure 7(a).

E. Overhead

Figure 8(a) shows the memory consumption of different methods when a various number of original files are stored in the system. We see that *EAFR* has lower memory consumption than other two methods because cold files only maintain 2 replicas in the system. In *HDFS*, keeping a fixed number of 3



(a) Server utilization. (b) Percentage of overloaded servers.

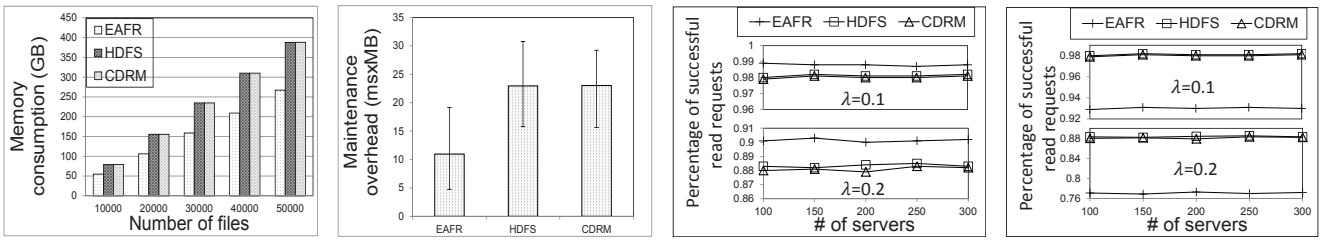
Fig. 7: Load balance status.

replicas consumes more storage resource than *EAFR*. *CDRM* maintains 2 replicas for each file initially, and increases the number of replicas to meet the required file availability, so it demands more memory consumption than *EAFR*.

When a file is modified, the update of file is accomplished by performing a write operation to synchronize each of its replica. In *EAFR*, as cold servers do not serve file read requests, when a file is updated, the writes need not be sent to its replicas stored in cold servers immediately. Instead, the updates of replicas in cold servers are postponed, and the cold servers are woken up once per week to check for file updates. We generated the updates of files from the trace data, and defined a file's maintenance overhead as the product of total amount of latency (in ms) to send writes to all its replicas multiplied by the size of the file (in MB). Figure 8(b) shows the 1st percentile, median and 99th percentile of maintenance overhead for all methods. We see *EAFR* displays substantially smaller maintenance overhead than the other two methods due to three reasons. First of all, *EAFR* creates a smaller number of replicas for cold files compared to *CDRM* and *HDFS*, thus, fewer writes are needed if a cold file needs to be updated. Secondly, the replicas in cold servers in *EAFR* do not need updates when the servers are in sleeping mode. Thirdly, *EAFR* tries to reduce network congestions in file replication, which may also help reduce the updating latency.

F. Server Failure Resilience

We tested *EAFR*'s resilience to server failures though it is not *EAFR*'s objective. Each server has a failure probability λ, and when all servers storing a file's replicas fail, requests for this file fail. Figure 8(c) shows the percentage of successful read requests when $\lambda = 0.2$ and $\lambda = 0.1$, and the minimum number of replicas in *EAFR* is 3. We see that *EAFR* achieves



(a) Memory consumption. (b) Maintenance overhead. (c) File availability when r=3. (d) File availability when r=2.

Fig. 8: Overhead and file availability.

the highest percentage of successful file requests. This is because *EAFR* creates extra replicas for hot files, which in turn increase the percentage of successful requests of hot files in server failures. *HDFS* keeps a fixed number of 3 replicas for each file and achieves lower percentage of successful requests than *EAFR*. *CDRM* stops increasing the file replicas when the percentage is higher than 0.8. Figure 8(d) shows the percentage of successful read requests when the minimum number of replicas in *EAFR* is 2. We see that *EAFR* provides relatively lower percentage of successful read requests than the other two methods due to the same reason in Figure 8(c).

V. Conclusions

The popularity of data-intensive clusters places demands for file systems such as short file read latency and low power consumption. In this paper, we propose EAFR to reduce file read latency, power consumption and replication completion latency. EAFR adaptively increases the number of replicas for hot files to alleviate intensive file request loads, and thus reduce the file read latency, and also decreases the number of replicas for cold files without compromising their read efficiency. Some replicas of cold files with few accesses are transferred to cold servers with 0% CPU utilization to save power. EAFR also has a server selection strategy to shorten replication completion time and avoid overloading servers. Experimental results from a real-world large cluster show the effectiveness of EAFR in meeting the demands of file systems in large clusters. In the future, we will study increasing data locality in replica placement, and determining the optimal number of cold servers to maximize energy saving without compromising the file read efficiency.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, and Microsoft Research Faculty Fellowship 8300751.

References

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

[2] Lustre File System. http://www.lustre.org, [Accessed in March 2015].

[3] Version 2 Parallel Virtual File System. http://www.pvfs.org, [Accessed in March 2015].

[4] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of OSDI*, 2006.

[5] 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.

[6] A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Proc. of USENIX*, 2009.

[7] Y. Chen, A. Ganapathi, A. Fox, R. Katz, and DPatterson. Statistical workloads for energy efficient mapreduce. In *Technical report, UC, Berkeley*, 2010.

[8] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *Proc. of SIGCOMM*, 2013.

[9] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. *Proc.of SIGCOMM*, 2009.

[10] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. Maltz, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. In *Proc. of SIGCOMM*, 2012.

[11] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng. Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster. In *Proc. of CLUSTER*, 2010.

[12] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Proc. of EuroSys*, 2011.

[13] L. Abad and Y. Luand R. Campbell. Dare: Adaptive data replication for efficient cluster scheduling. In *Proc. of CLUSTER*, 2011.

[14] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. of NSDI*, 2010.

[15] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proc. of SIGCOMM*, 2011.

[16] M. Lin, A. Wierman, L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *Proc. of INFOCOM*, 2011.

[17] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely. Data centers power reduction: A two time scale approach for delay tolerant workloads. In *Proc. of INFOCOM*, 2012.

[18] R. Kaushik, T. Abdelzaher, R. Egashira, and K. Nahrstedt. Predictive data and energy management in greenhdfs. In *Proc. of IGCC*, 2011.

[19] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of www client-based traces. Technical report 1995-010, boston univ., 1995.

[20] 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.

[21] J. Liu, F. Zhao, X. Liu, and W. He. Challenges towards elastic power management in internet data centers. In *Proc. of ICDCS*, 2009.

[22] R. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In *Proc. of CloudCom*, 2010.

[23] Sandia CTH trace data. http://www.cs.sandia.gov/Scalable_IO/SNL_Trace_Data/, [Accessed in March 2015].

[24] J. Lucas and M. Saccucci. Exponentially weighted moving average control schemes: Properties and enhancements. *Technometrics*, 32(1):1–29, 1990.

[25] Palmetto Cluster. http://citi.clemson.edu/palmetto/index.html, [Accessed in March 2015].

[26] ThinkServer. http://shop.lenovo.com/us/en/servers/, [Accessed in March 2015].