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

## Yuhua Lin, Member, IEEE and Haiying Shen, Senior Member, IEEE

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. 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. EAFR then selects a server with sufficient capacity (including network bandwidth and capacity) to hold a replica. To further improve the performance of EAFR, we propose a dynamic transmission rate adjustment strategy to prevent potential incast congestion when replicating a file to a server, a networkaware data node selection strategy to reduce file read latency, and a load-aware replica maintenance strategy to guickly create file replicas under replica node failures. Experimental results on a real-world cluster show the effectiveness of EAFR and proposed strategies in reducing file read latency, replication time, and power consumption in large clusters.

Index Terms—Data-intensive clusters, file replication, replica placement, energy-efficient

## 1 INTRODUCTION

ATA 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 these file systems, concurrent I/O requests to the same file can be spread across several servers rather than a single one. 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], [6]: 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 [7], [8], [9]. The energy cost of equipment during its lifetime is comparable to the initial equipment purchase price [10]. 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

Y. Lin is with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634. E-mail: yuhual@g.clemson.edu.

H. Shen is with the Department of Computer Science, University of Virginia, Charlottesville, VA 22904. E-mail: hs6ms@virginia.edu.

Manuscript received 23 Nov. 2015; revised 10 Aug. 2016; accepted 25 Sept. 2016. Date of publication 27 Sept. 2016; date of current version 15 Mar. 2017. Recommended for acceptance by P. Sadayappan.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2016.2613989

<sup>1045-9219 © 2016</sup> IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications\_standards/publications/rights/index.html for more information.

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 [11]. Though the network inside clusters is generally underutilized, there exist some bottleneck links resulting from the network usage imbalance [12], [13]. As the traffic in multi-tenant datacenters is not controlled, the traffic congestion of bottleneck links leads to performance degradation inside clusters [14], [15], [16]. 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 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 and dynamically change the transmission rate to prevent network congestion. 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. We further propose three strategies to improve the performance of EAFR. First, when replicating a file to a server, EAFR dynamically tunes the transmission rate to prevent potential incast congestion. Second, when a compute node needs to read a file, EAFR uses a network-aware data node selection strategy to reduce file read latency. Third, when replica node failure occurs, EAFR uses a load-aware replica maintenance strategy to quickly create file replicas in other nodes.

The remainder of the paper is organized as follows. Section 2 gives an overview on the related work. Section 3 introduces the design of EAFR and our proposed strategies. The performance evaluation is presented in Section 4. Section 5 concludes the paper with remarks on future work.

## 2 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 [17], [18], [19] have been proposed to improve the replication policy for different purposes. CDRM [17] adjusts the replication factor to maintain a required reliability for each file under server failures based on the relationship between file reliability and replication factor when servers have a certain probability to fail. Scarlett [18] 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 [19] replicates remote data into the local node when a map task processes data from remote nodes. It also applies a replication budge to limit the amount of replicas to save storage resource. Unlike the above replication works, EAFR aims to improve the data availability with the consideration of file popularity and file storage system efficiency.

Network bottleneck [12], [13], [15], [16] is critical issues in data-intensive clusters but are neglected in previous file replication methods. Hedera [20] 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 [21] studies the short-term traffic, then incorporates 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 [22], [23] aim to reduce the power costs by dynamically transitioning the servers to a sleeping state in datacenters. Works in [24], [25], [26] exploit the opportunities of geographical load balancing to minimize energy cost in data centers. Recent research [27], [28], [29] proposes maintaining a minimal subset of nodes that are guaranteed to be on, and put other nodes to sleeping mode. This strategy ensures that a primary replica of each file is stored on active servers to provide service to file requests; however, it does not consider file popularity and replicas of popular file are also stored on inactive nodes. This generates a large number of replicas for popular files in order to ensure the file's immediate availability, and it suffers from degraded write-performance as the writes need to be executed on all servers storing the file replicas. GreenHDFS [30], [31], [32] 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. It then uses data classification policies to place data onto a suitable temperature zone, that is, data that is frequently accessed by Hadoop framework is placed in hot zone, while unpopular data is place in cold zone. Autoplacer [33] identifies top-k objects that generate most remote operations (i.e., hotspots) for each node of the system, and optimizes the placement of hotspots to minimize the inter-node communication. Schism [34] represents a database and its workload using a graph, where tuples are represented by nodes and transactions are represented by edges. It then uses graph partitioning algorithms to minimize the number of multi-sited transactions. Different from these works, EAFR considers file popularity when allocating file replicas in order to save energy.



Fig. 1. Architecture of hierarchical storage system.

## **3** System Design

#### 3.1 Background of File Replication

A typical cluster file system uses a hierarchical storage architecture, as shown in Fig. 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 reliability in face of network failure or hardware damage, cluster file system makes multiple replicas for each file [18]. A replication factor ( $r_i$ ) and a fault-tolerance factor ( $\pi_i$ ) 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. When one rack suffers from a failure, the file is still available. The red arrows in Fig. 1 shows an example of the distributed writes when storing a file with  $r_i = 3$  and  $\pi_i = 1$ . By creating a constant number of replicas for each file, current replication systems neglect the heterogeneity in file popularity. Some hot files attract a large amount of read requests from clients, while some cold files attract few visits. 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 [12]. For example, link L1 (marked in red) in Fig. 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 accessed during their lifetime [19]. 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.



Fig. 2. An overview of EAFR: placing cold files in servers with low power consumption and hot files in servers with high 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.

## 3.2 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 [35]. 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. Fig. 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 percent 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  $\epsilon$  are pre-defined numbers to guarantee file reliability 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  $\epsilon$ . 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 3.2.1), how to identify hot files and cold files based on their visit rates (Section 3.2.2), and the details of the energy-efficient and popularity-adaptive file replication algorithm (Section 3.2.3). Table 1 shows the notations used in this paper.

## 3.2.1 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 2 shows the power consumption characteristics of the HP ProLiant ML110 G5

TABLE 1 Table of Important Notations

- $f_i$ : File i
- $s_j$ : Server j
- $r_i$ : # of replicas
- $\pi_i$ : 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  $a_{ij}$
- $c_{c_i}$ : Service capacity of server j
- $b_i$ : Size of file j
- $\phi_i$ : Remaining capacity of server j
- $c_{s_i}$ : Storage capacity of server i
- $\nabla$ : Remaining storage threshold
- $\tau_u$ : Upper bound for total # of reads of hot file
- $\sigma_u$ : Upper bound for total # of reads of hot replica
- $\tau_l$ : Lower bound for total # of reads of cold file
- $\sigma_l$ : Lower bound for total # of reads of cold replica
- *V<sub>t</sub>*: Transmission speed in time window t
- $w_{t_j}$ : Weight of selecting server j based on transmission rate
- $w_{r_i}$ : Weight of selecting server based on remaining capacity
- $p_j$ : Chance of selecting server j based on overall evaluation
- *c*<sub>*b*</sub>: Bandwidth capacity of the destination node
- $B_a$ : Proportion of available bandwidth
- $\eta_1$ : Highly utilized network capacity threshold
- $\eta_2$ : Under-utilized network capacity threshold
- $\delta_i$ : Transmission rate adjust-up factor
- $\beta_i$ : Transmission rate adjust-down factor
- $\epsilon$ : minimum # of replicas stored in hot servers

server at different server performances represented by server CPU utilization [36]. Higher CPU utilization consumes more power, and when a server runs at 0 percent CPU utilization (e.g., in sleeping state), the power consumption is 93.7 Watts. 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.

As a cold server runs at smaller power consumption compared to a hot server, switching hot servers to cold mode can save energy. A cold server stores cold files with few read requests. Writing a file to a cold server needs to wake up the server, which consumes more energy and may offset the benefit of sleeping [37]. 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.

TABLE 2 Energy Consumption for Different CPU Utilizations in Watts [36]

CPU Utilization	0%	20%	40%	60%	80%	100%
HP ProLiant G5	93.7	101	110	121	129	135
Server status	cold	hot	hot	hot	hot	hot



(a) Percentage of files with different (b) The number of file reads over time. number of reads.

Fig. 3. Trace analysis on file read pattern.

#### 3.2.2 Different Types of Files Based on Read Rates

In HDFS, more than 90 percent files exhibit a relatively short hotness lifespan (i.e., less than three days) and a significant portion of data is cold (i.e., never gets read) [31]. In order to justify the heterogeneity of file popularity in large clusters, we analyzed the file storage system trace data from Sandia National Laboratories [38], which records the number of file reads for 16,566 accessed files during 4 hour run. Fig. 3a shows the percentage of files attracting different range of file reads. We see that about 43 percent files receive less than 30 reads and 4 percent 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 Fig. 3b 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 these 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. We present how to determine hot files and cold files below.

Current replication factor of  $f_i$  is  $r_i$ , and the replicas of  $f_i$  is denoted by vector  $A_i = \{a_{i1}, a_{i2}, \dots, a_{ir_i}\}$ . 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  $v_i$ , and

$$v_i = \sum_{j=1}^{r_i} v_{ij}.$$
 (1)

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 ( $\tau_u$ ):

$$v_i/r_i > \tau_u. \tag{2}$$

Second, 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  $\gamma$ ) of a file's replicas attract an excessive number of reads (denoted by  $\sigma_u$ ), we consider this file a hot file:

$$\sum_{j=1}^{r_i} I(v_{ij} > \sigma_u) > r_i \gamma_v \ (0 < \gamma_v < 1), \tag{3}$$

where  $I(\cdot)$  is an indication function, and I(A)=1 when the assertion *A* is satisfied. If either Equations (2) or (3) is satisfied, file  $f_i$  is a hot file represented by  $H(f_i) = 1$ .

Similarly, we use Equations (4) and (5) to determine if file  $f_i$  is a cold file. In the equations,  $\tau_l$  is the lower bound of the number of reads per replica in *T* 

$$v_i/r_i < \tau_l. \tag{4}$$

Equation (4) 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

$$\sum_{j=1}^{r_i} I(v_{ij} < \sigma_l) > r_i \gamma_v \ (0 < \gamma_v < 1)$$
(5)

In Equation (5),  $\sigma_l$  denotes the lower bound for the number of reads that a file replica receives in *T*. If more than  $\gamma_v$  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 Equations (4) and (5) 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 3.2.3.

#### 3.2.3 Adaptive File Replication

EAFR constantly monitors file popularity and adaptively tunes the number of replicas of a file based on whether it is hot or cold according to Section 3.2.2. If a file is a hot file and many of its replica servers are overloaded, EAFR creates more replicas for this file to reduce overload degree and increase file availability. If a file is a cold file, EAFR reduces its replicas or transfers its replicas to standby servers. We first define r to guarantee file reliability, which is the minimum number of replicas a file needs to maintain in all servers. We then define  $\epsilon$  to guarantee file reliability ( $\epsilon < r$ ), which is the minimum number of replicas a file needs to maintain in hot servers.

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  $h_j/c_{c_j} > \tau_{c_j}$  where  $\tau_c$  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  $\gamma_s$  ( $0 < \gamma_s < 1$ ) fraction of these servers are overloaded, that is:

$$\sum_{s_j \in S_i} I(h_j/c_{c_j} > \tau_c) > r_i \gamma_s \ (0 < \gamma_s < 1). \tag{6}$$

When the inequality in Equation (6) is met, the current replica servers of  $f_i$  do not have enough capacity to handle an excessive number of file reads. Then, EAFR increases the number of replicas of  $f_i$  by 1. Otherwise, the current replica servers of  $f_i$  can handle the file reads even though  $f_i$  is hot, and there is no need to increase the number of replicas of  $f_i$ . The new replica will be placed in a hot server, so that it can serve new incoming file requests. The details of selecting the replica destination for the replica is presented in Section (3.3).

#### Algorithm 1. Pseudo-Code of EAFR

1:	Determine the popularity of file $f_i$		
2:	<b>if</b> $H(f_i) = 1$ : //create one replica		
3:	Select $hs_i$ from the hot server pool; place replica in $hs_i$		
4:	end if		
5:	<b>if</b> $C(f_i) = 1$ : //reduce number of replica by one		
6:	if number of replicas $r_i > r$		
7:	Select $s_j$ according to Equation (7)		
8:	Delete the replica of $f_i$ in $hs_j$		
9:	else		
10:	if more than $\epsilon$ replicas of $f_i$ are stored in HS		
11:	Migrate one replica of $f_i$ from $hs_i$ to $ss_k$		
12:	if Equation (8) is satisfied for $ss_k$		
13:	$ss_k$ turns into a cold server		
14:	end if		
15:	end if		
16: end if			

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  $\epsilon$  replicas in hot servers in order to guarantee file reliability. 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  $\epsilon$ , 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:

$$s_j = \arg\min_{\substack{s_i \in S_i}} \{\phi_j\}.$$
(7)

Erasure coding [39] creates redundant data pieces from the actual data. When the actual data is lost, it enables the file system to reconstruct data fragments by using the forward error correction technique. Reducing the number of replicas for cold files will decrease file reliability. In order to maintain file reliability while reducing storage cost, we can use erasure coding to improve file reliability. However, erasure coding is not the focus of this paper.

In the case of  $r_i = r$ , if there are  $\epsilon$  replicas in hot servers, no action is performed; if more than  $\epsilon$  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 (7),



Fig. 4. Replication completion time for different servers.

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:

$$c_{s_i} - \sum_{j=1}^m b_j < \tau_s, \tag{8}$$

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  $\tau_s$  is the remaining storage threshold.

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.

#### 3.3 Replica Destination Selection

When a network link suffers from congestion, the consequence is reflected in long write latency. 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 [11] that monitors the network status (e.g., concurrent traffic, link utilization) and selects the links with light traffic [11]. 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. To verify this rationale, we conducted an experiment as below.

We randomly selected a server as the source and 20 destination servers in Palmetto Cluster in Clemson University [40]. The source replicated 20 files to each destination server at the rate of once every 15 seconds. The file size is 100 MB. We recorded the replication completion time on these servers, and showed the maximum, median and minimum replication completion time in Fig. 4. We can make two observations from the figure. First, the replication completion time towards different servers exhibits obvious variance. The replications towards some servers (e.g., servers 1, 5 and 8) have smaller replication completion times than those towards other servers, while replications towards some servers (e.g., servers 2, 4 and 9) generally take longer completion time. This observation justifies the necessity and motivation of allocating the new replica to a server that has good network condition. Second, each server shows relatively stable replication completion time with a small variance between the maximum and minimum completion time. Thus, when multiple replicas are transmitted to the same server within a short period, the completion time for these replications should be similar. Therefore, a server's recent transmission speed can be used to predict the transmission speed in the near future. EAFR does not need to look into the link utilization and monitor the network congestion status when allocating a new file replica. It selects the replica destination based on the transmission speed of recent files.

To more accurately estimate the transmission delay of the next file based on the delays of previous file transmissions, we use an exponentially weighted moving average (EWMA) [41]. Using T as a time window size, EAFR calculates the average file transmission speed from source server  $s_s$  to destination server  $s_d$  by sliding the time window. Then the transmission speed in the next time window (denoted by  $V_t$ ) is calculated by:

$$V_t = \alpha \times Y_t + (1 - \alpha) \times V_{t-1} (0 < \alpha < 1), \tag{9}$$

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.  $\alpha$  is a constant used to control the degree of weighting decrease; a larger value of  $\alpha$  discounts older observations faster. The weighting for each older EWMA data point decreases exponentially, but never reaches zero.

In addition to replica transmission latency, the replica destination must have enough storage capacity for new replicas. Also, as a new replica is created to serve an excessive amount of file requests, the replica should be placed in a server that has sufficient capacity to serve incoming file requests. Then, given a file from source server  $s_s$ , and a set of hot servers HS to place the new replica, EWMA 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, EWMA 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  $ID_t = \{hs^1, hs^2, \dots hs^m\}$ . EAFR also orders the candidates based on the decreasing order of their remaining capacities  $ID_r = \{hs^1, hs^2, \dots hs^m\}$ . 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 as

$$\frac{1}{j} / \sum_{k=1}^{m} \frac{1}{k}.$$
 (10)

The probability of selecting a server in the candidates is calculated by the weighted average of both of its  $w_t$  and  $w_c$ :

$$p = \beta \times w_t + (1 - \beta) \times w_c. \tag{11}$$

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:

$$y = \begin{cases} 1 & \text{if } x < p_1 \\ j & \text{if } p_{j-1} \le x < \sum_{k=1}^j p_k \text{ and } j > 1. \end{cases}$$
(12)

As we can see from Equation (12), the new replica is more likely to be allocated to a server with a high p value.

#### 3.4 Dynamic Transmission Rate Adjustment

TCP incast occurs when a number of files from multiple storage servers are being sent to a server concurrently [42], and network congestion is likely to arise on the receiver side when multiple connections contend for bandwidth resources. TCP incast congestion increases the packet drop rate and reduces the transmission throughput, thus degrading network performance. To avoid incast congestion, while transmitting a file replica to a new server, EAFR dynamically adjusts transmission rate to prevent incast congestion.

Each server has a TCP receive window [43], which is a limited size of buffer that prevents a fast sender from overflowing a slow receiver's buffer. In EAFR, a destination server monitors its available bandwidth by using a bandwidth estimation tool [44] to detect sudden throughput burstiness. When it notices that a congestion is likely to occur, it reduces its receive window in proportion to the extent of congestion and notifies senders to reduce their transmission rates. If the destination node has enough available bandwidth to support larger transmission rates, it increases the TCP receive window.

Algorithm 2. Pseudo-Code of Dynamic Transmission Rate Adjustment

- 1: Input: set of server *S* sending data through the link;
- 2: **Output:** adjust-down factor  $\beta_i$  and adjust-up factor  $\delta_i$  for each sender in *S*;
- 3: for each  $s_i \in S$  do:
- 4: calculate available bandwidth on the link:  $R_{c_b} = (c_b u_b)/c_b$
- 5: **if**  $R_{c_b} < \eta_l$  //network capacity is highly utilized
- 6: calculate adjust-down factor  $\beta_i = \frac{i}{\sum_{k=1}^{p'} k} \times (\eta_1 R_{c_b})$ 7: end if
- 8: if  $R_{c_b} > \eta_h$  //enough network capacity on the receiver side
- 9: calculate adjust-up factor  $\delta_i = \frac{i}{\sum_{k=1}^{p'} k} \times (R_{c_b} \eta_h)$
- 10: **end if**
- 11: record  $\beta_i$  and  $\delta_i$  for sender  $s_i$
- 12: end for

Assume the link bandwidth capacity of the destination node is  $c_b$  (which is determined by its NIC and system settings), and the total bandwidth utilized by all incoming traffic is  $u_b$ . We then define the proportion of available bandwidth  $u_b$  on that link as

$$R_{c_b} = (c_b - u_b)/c_b.$$
(13)

 $R_{c_b}$  is an indicator of potential oversubscribed bandwidth for a destination node. EAFR has two thresholds  $\eta_l$  and  $\eta_h$ to determine whether a network link capacity is highly utilized or underutilized. When  $R_{c_b} < \eta_l$  (e.g.,  $\eta_l$ =0.2), the network capacity is highly utilized and thus the receive window needs to be reduced. Suppose the destination server is receiving traffic from a number of connections from a set of servers  $S = (s_1, s_2, \ldots, s_{p'})$ . These connections have different priorities based on the connection establishment times. Since we aim to reduce the transmission latency of each flow, the connections with older establishing times have higher priorities. The senders are ordered in descending order of the priorities of their connections. For a sender with priority *i*, its adjust-down factor  $\beta_i$  is define as

$$\beta_i = \frac{i}{\sum_{k=1}^{p'} k} \times (\eta_1 - R_{c_b}). \tag{14}$$

When  $R_{c_b} > \eta_h$  (e.g.,  $\eta_h$ =0.5), there is enough network capacity on the receiver side, then the receive window is increased for each connection to increase the transmission rate. The adjust-up factor for the server with priority *i* is  $\delta_i$  defined by

$$\delta_i = \frac{i}{\sum_{k=1}^{p'} k} \times (R_{c_b} - \eta_h). \tag{15}$$

After rate adjustment calculation, the destination node notifies the corresponding senders about the new transmission rates. Each sender then reduces its transmission rate by  $\beta_i$ times or increase its transmission rate by  $\delta_i$  times.

Algorithm 2 shows the pseudo-code of dynamic transmission rate adjustment. For each sender  $s_i$  which establishes connection with the receiver, Algorithm 2 first calculates the link's available bandwidth capacity (Line 4); when the network capacity is highly loaded, EAFR reduces the sender's transmission rate by  $\beta_i$ ; when the link's network capacity is lightly loaded, EAFR deliberately increases the sender's transmission rate by  $\delta_i$  accordingly (Lines 5-7). The computation complexity of Algorithm 2 is O(p), where p is the number of senders in the cluster. By dynamically adjusting the receive window in proportion to the extent of congestion, EAFR reduces the latency for file transmission in replications by avoiding incast congestions.

#### 3.5 Network-Aware Data Node Selection

Inside a cluster, user requests are processed in compute nodes and the compute nodes need to fetch files from data nodes where the requested files are stored. The file read latency is affected by two factors: 1) transmission delay between a compute node and a data node, and 2) queueing delay in a data node. As the intra-rack connection has much higher bandwidth than the cross-rack connection, choosing a data node in the same rack as the requester computer node to transmit its requested file generates shorter latency than choosing a data node in a different rack. Also, when a cluster consists of heterogeneous servers, the service capacities of servers may vary significantly. A high-capacity data node can finish reading a file stored in its local disk faster than low-capacity nodes, resulting in smaller queue size and queueing delay. Accordingly, we propose a networkaware data node selection strategy by considering the aforementioned factors, i.e., a compute node should fetch its requested file from a data node within the same rack and with a short queue size.

Suppose compute node  $s_j$  needs to read file  $f_i$  when processing a user request, and  $f_i$  has  $r_i$  replicas stored in a number of servers. We use set  $S_i=(s_1, s_2, ..., s_{r_{i-1}})$  to represent all hot servers and standby servers that store  $f_i$ 's replicas. We use  $h_k$ 

to denote the queue size of file read requests on server  $s_{k_{\ell}}$ which is the number of reads that  $s_k$  has received but has not been processed. Algorithm 3 shows the pseudo-code of selecting data nodes for compute node  $s_i$ . For each file  $f_i$  that  $s_i$ needs to read from data nodes, Algorithm 3 first identifies all hot servers and standby servers storing  $f_i$ 's replicas (Line 4). It then selects the data nodes within the same rack as  $s_i$  and puts them in a set  $S_i^1 = (s_1, s_2, \dots, s_{r_{i-1}^1})$  (Line 5). In order to reduce transmission delay between a compute node and a data node, we prefer intra-rack connection over cross-rack connection. Thus, if  $S_{i}^{1}$  is not empty, Algorithm 3 selects data node  $s_k$  with the minimum queue size from  $S_i^1$ :  $s_k = \operatorname{argmin}_{s_k \in S^1} \{h_k\}$  (Lines 6-7); otherwise, it chooses a data node with the minimum queue size from  $S_i$  (Lines 8-10). The computation complexity of Algorithm 3 is  $O(m \times \overline{r})$ , where m is the number of files needed to read and  $\overline{r}$  is the average number of replicas for these files.

Algorithm 3. Pseudo-Code of Network-Aware Data Node Selection for Compute Node  $s_i$ 

- 1: **Input:** set of file *F* needed to fetch from data nodes;
- 2: **Output:** select a data node to read each file in *F*;
- 3: for each  $f_i \in F$  do:
- 4: find servers storing  $f_i$ 's replicas  $S_i = (s_1, s_2, \dots, s_{r_{i-1}})$
- 5: select  $S_i^1$  from  $S_i$  that are in the same rack with  $s_j: S_i^1 = (s_1, s_2)$  $s_2, \ldots, s_{r_{i-1}^1})$
- if  $S_i^1 \neq null$  //find a data node within the same rack 6:
- 7: select a data node by  $s_k = \operatorname{argmin}_{s_k \in S_i^1} \{h_k\}$
- 8: else //find a data node within another rack
- 9: select a data node by  $s_k = \operatorname{argmin}_{s_k \in S_i} \{h_k\}$
- 10: end if
- 11. record  $s_k$  as the data node to read file  $f_i$
- 12: end for

#### Load-Aware Replica Maintenance Under Node 3.6 Failure

As each server stores a large number of files, when a server failure happens, we create these lost files in other servers in order to maintain the minimum number of replicas per file. Suppose all files stored in a failed server are represented by  $F=(f_1, f_2, \ldots, f_m)$ . For each file  $f_i$  in F, we make a replica from a non-failed server (called source server) and place it in another non-failed server (called destination server). In order to minimize the energy consumption and time for the recovery process, we consider two objectives. First, we try to avoid waking up cold servers as it consumes extra energy and may lead to long recovery time. Second, we try to balance the incast traffic load caused by the file replications on the destination servers, i.e., the number of replicas allocated to destination servers, in order to avoid incast congestion in the destination servers and hence constrain the recovery latency.

Recall that each file has  $r_i$  replicas; at least  $\epsilon$  replicas are stored in different hot servers, other replicas are stored in standby servers and cold servers. A hot server runs at active state and serves file requests; a standby server is a temporary hot server that stores cold files, and it turns to a cold server when its storage is fully utilized; and a cold server stores cold files, and it is in sleeping mode and does not serve file requests.

To achieve the first objective of avoiding rebooting the cold servers from the sleeping mode, we first try to select source servers from hot servers and standby servers. A cold server is waken up and selected as the source server only when a file stored in the failed server does not have any replicas stored in hot servers or standby servers. Specifically, for a file  $f_i$  in a failed server, we first put all hot servers and standby servers that store  $f_i$  in a server set  $S_i = (s_1, s_2, ..., s_p)$ . In order to find a server that has the maximum available service capacity to support the file reading operation, we first sort servers in  $S_i$  in decreasing order of their available service capacities. Then, we choose the first server as the source server. If there are no hot servers or standby servers storing file  $f_i$  (i.e.,  $S_i$  is an empty set), we check the cold servers that store file  $f_i$  in the same manner and choose a cold server with the maximum available service capacity.

## Algorithm 4. Pseudo-Code of Load-Aware Replica Maintenance for a Failed Server

- 1: **Input:** set of files F, counter = 1;
- 2: **Output:** source server and destination server for files in *F*; 3: for each  $f_i \in F$  do:
- order hot servers and standby servers storing  $f_i$ 's replicas 4: by their remaining service capacities,  $S_i = (s_1, s_2, ..., s_p)$
- 5: select  $s_1$  as the source server
- if  $S_i$  is empty 6:
- 7: choose a cold server with the maximum available service capacity as the source server
- 8: end if
- 9: while true //select a destination server from DS
- 10: choose server  $ds_k$  with index equaled to *counter*
- if  $c_{s_i} \sum_{j=1}^m b_j \ge \tau_s$ , //enough storage 11:
- 12: select  $ds_k$  as the destination server
- 13: break
- else  $14 \cdot$
- 15: increase counter by 1
- 16: end if
- 17: end while
- 18: record  $f_i$ 's source server and destination server
- 19: end for

We use  $DS = (ds_1, ds_2, ..., ds_w)$  to denote a set of candidate destination servers, DS is determined based on the popularity of  $f_i$ . If  $f_i$  is a hot file, DS is a set of non-failed hot servers; if  $f_i$  is a cold file, DS is a set of non-failed standby servers. Here, we do not choose a cold server as the destination server in order to avoid waking up the cold server for file replication, which otherwise increases the recovery latency. To meet the second objective of balancing the incast traffic load of destination servers, we evenly place the replicas of all files of the failed server to DS by using a round robin [45] assignment method. We use a counter to record the index of the destination server candidates; counter increases from 1 to w in a circular manner, i.e., counter restarts from 1 after it reaches w. Assume counter = k and we need to select a destination server for file  $f_i$ . We first examine the server  $ds_k$  whose index in DS equals the value of *counter*. If  $ds_k$  has enough storage capacity calculated by Equation (8), it is selected as the destination server for  $f_i$ where we store the replica of  $f_i$ ; otherwise, we increase the value of *counter* by 1 until a server with enough storage capacity is detected. After we find a destination server for  $f_i$ , we add 1 to the *counter* and use it to identify the destination server for the next file  $f_{i+1}$ .

Algorithm 4 shows the pseudo-code of load-aware replica maintenance for a failed server, in which we choose a source server and a destination server for file  $f_i$  stored in the failed server. For each file  $f_i$  stored in the failed server, Algorithm 4 first orders all hot servers and standby servers storing  $f_i$ 's replicas by their remaining service capacities (Line 4); it then selects a hot server or standby server with the maximum available service capacity as the source server (Line 5); if no hot servers or standby servers that store file  $f_i$ , it checks all cold servers that store file  $f_i$  and chooses a cold server with the maximum available service capacity as the source server (Lines 6-8); Algorithm 4 continues to select a destination server for file  $f_i$  by using a round robin assignment method (Lines 9-17). The computation complexity of Algorithm 4 is  $O(m\overline{r}\log\overline{r})$ , where m is the number of files in the failed server and  $\overline{r}$  is the average number of replicas for these files.

## 4 PERFORMANCE EVALUATION

We conducted trace-driven experiments in a large-scale HPC cluster located in Clemson University's Palmetto Cluster [40] which has 1,978 nodes. We deployed EAFR on 300 servers evenly scattered in 10 racks. The storage capacities of these servers were randomly chosen from (250, 500, 750 GB) [46]. We compared EAFR with HDFS [1] and CDRM [17] that are similar to our work. In HDFS, every file has a fixed number of three replicas placed across different randomly selected servers. CDRM aims to deal with server failures. In our experiment, CDRM creates two replicas for each file initially, it increases the number of replicas to maintain the required file reliability 0.98 for server failure probability 0.1, and required file reliability 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 [38] 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 3,300 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 (100 KB, 10 GB). The server capacity follows the normal distribution [1] with a mean of 15 and variance of 10. When the number of concurrent file reads is larger than a server's service capacity, new coming file requests will be put into a waiting queue until the server has available capacity. The remaining storage threshold  $\nabla$  was set to 10 GB; other system parameters were set as:  $\tau_u$ =20,  $\tau_l$ =10,  $\sigma_u$ =8,  $\sigma_l$ =1, r = 2and  $\epsilon$ =1. The power consumption for different types of servers was calculated based on Table 2. We randomly selected 70 percent of servers as hot servers, and 30 percent of servers function as standby servers. A standby server with full storage turns into a cold server. The experiment runs two days by repeatedly using the read rates from the trace data. We are interested in the following performance metrics:



Fig. 5. Performance under different number of replicas.

- *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 when 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 percent 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.
- *Recovery latency*: the time span from when the creation of file replicas in a failed server is initiated until all file replicas stored in the failed servers are recovered.

#### 4.1 Experimental Results for Overall Performance

#### 4.1.1 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.

Number of Replicas. We first study the effectiveness of creating extra file replicas for hot files in reducing the file read response time. We selected 20 random files, varied the number of replicas for each file, and generated 60 concurrent read requests towards each file. Fig. 5a 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, i.e., when the number of replicas for each file increases from 2 to 7, the median read response time drops from about 11 to 4 ms. This is due to the reason that when there are only three replicas allocated in three individual servers, large numbers of concurrent read requests are flooded to the same server, and some read requests need to wait if the server capacity is already fully occupied by requests. However, when more replicas are created in different servers, more server capacity can be utilized to serve the read requests. Thus, concurrent read requests are forwarded to different servers and are less likely to contend for server capacity. We define 10 ms as the required latency threshold, and record the



Fig. 6. Performance under different # of concurrent accesses.

percentage of file read requests that are served past the required latency. Fig. 5b 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 Fig. 5a. That is to say, creating more replicas for hot files can prevent resource contention between excessive number of synchronous requests. Figs. 5a and 5b prove the rationale of *EAFR* that increasing the replicas of hot files can shorten the read response time and increase data availability.

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. Fig. 6a 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 the response latency increases as the number of concurrent reads increases. This is because servers can serve a limited number of requests at a time and new file requests must wait in queues until the servers have available capacity. We also 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. Thus, a large number of concurrent file requests are not likely to overload the servers and wait for response. As CDRM does not consider file popularity in replication, file requests towards hot files still need to contend for server capacity. Fig. 6b shows



Fig. 7. Performance under different access arrival rates.



(a) Replication latency for files of vari- (b) Energy consumption per day. ous sizes.

Fig. 8. Replication latency and energy consumption.

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 Fig. 6a.

Access Arrival Rate. Access arrival rate is defined as the number of file requests generated in the system in each second. In order to investigate the performance of EAFR under different workload distributions, we varied the file read arrival rates by changing the time interval between two consecutive reads in the trace data (e.g., reduce the time interval between two successive reads to increase the file read rate). Fig. 7a shows the average file read response latency with different arrival rates. We can see that as the access arrival rate grows from 100 to 300 reads per second, HDFS and CDRM both rise quickly. This is due to the reason that a limited number of replicas are insufficient to serve large number of read requests, and as the access arrival rate gets higher, more requests are likely to stay in waiting queue. EAFR adaptively increases the number of replicas for hot files, thus produces less file read response latency than HDFS and CDRM due to same reasons as in Fig. 6a. Fig. 7b shows the percentage of read timeouts with different arrival rates. We see that EAFR produces fewer read timeouts than HDFS and CDRM for the same reasons as in Fig. 6b. As applications (such as some web-based applications) deployed on large clusters need to provide prompt service to their clients, and the above figures show the effectiveness of EAFR in reducing file read latency and providing high quality support for time-sensitive applications.

## 4.1.2 Replication Completion Time

We grouped the files with the same size (ranging from 1 to 10,000 MB) together and calculated the average replication latency of each group of files. Fig. 8a shows the replication completion time for different file groups. We also set the replication completion time of HDFS as base and plot the ratio of other methods' replication completion time over the base in the embedded figure. We see that replication operations can be completed with short latency for small files due to the high-speed network connections on Palmetto clusters. However, the replication completion time grows rapidly for files with large sizes. EAFR speeds up the file replication especially for large files, and the improvement reaches about 30 percent when the file size is 10,000 MB. 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, i.e., file replicas are more likely to be allocated to servers



Fig. 9. Load balance status.

with good network condition. Also, it dynamically adjusts the file transmission rate during replication process in order to prevent incast congestion on the receiver side, thus reducing transmission latency.

#### 4.1.3 Energy Efficiency

We examined the effectiveness of *EAFR* in reducing energy consumption. We set the power consumption of different genre of servers according to Table 2. Fig. 8b 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 150 kWh per day in a cluster consisting of 300 servers. Given a fixed number of servers in the cluster, EAFR aims to allocate popular files to servers that are guaranteed to be on (hot servers), and it 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.

## 4.1.4 Load Balance Status

It is crucial to constrain the workloads of servers under their capacities (i.e., achieving load balance), which help reduce file read response latency. Server utilization is an indicator of how balance the file requests are distributed among servers in the system. 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. Fig. 9a plots the 1st percentile, median and 99th percentile of server utilization of different methods. 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 and reduces the number of replicas for cold files. Also, it creates new replicas in servers with the highest remaining capacity with a high probability. As the workloads are better balanced in *EAFR*, it can effectively prevent the servers storing hot files from becoming overloaded, and file requests are less likely to be blocked. We then tested the performance of EAFR under different workload distributions by varying the file read arrival rates using the same method as in Section 4.1.1. Fig. 9b shows the percentage of overloaded servers during the experiment in the system. We see that the percentage of overloaded servers rises gradually with increased read arrival rates for all methods, as more server capacity is consumed to serve read requests. EAFR maintains the least percentage of overloaded servers due to the same reason as in Fig. 9b.

#### 4.1.5 Overhead

Fig. 10a 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 two replicas in the system, and small amount of extra replicas are created for hot files to meet the shortterm intensive read requests. In *HDFS*, keeping a fixed number of three replicas consumes more storage resource than *EAFR*. *CDRM* maintains two replicas for each file initially, and increases the number of replicas to meet the required file reliability, so it demands more memory consumption than *EAFR*.

When a file is modified, each replica of the file should be updated in order to maintain consistency, and 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, until the servers are switched from sleeping mode to active mode. More precisely, the cold servers are woken up once per week to check for file updates. Whenever a file replica is dirty (i.e., not updated), a write operation is performed to synchronize the replica. 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). Fig. 10b shows the 1st percentile,





Fig. 11. Replication latency for files of various sizes.

median and 99th percentile of maintenance overhead for all methods. We see *EAFR* displays substantially smaller median, 1st percentile and 99th percentile 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. Second, the replicas in cold servers in *EAFR* do not need updates when the servers are in sleeping mode. Third, *EAFR* tries to reduce network congestions in file replication, which may also help reduce the updating latency. As a result, *EAFR* produces relatively low maintenance overhead.

#### 4.1.6 Server Failure Resilience

We tested EAFR's resilience to server failures though it is not *EAFR*'s objective. Each server has a failure probability  $\lambda$ , and when all servers storing a file's replicas fail, requests for this file fail. We measured the file reliability as the percentage of available files among all files stored in the system, i.e., the percentage of successful read requests. A good file system in cluster should provide high file reliability to clients. Fig. 10d 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 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 three 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. Fig. 10d 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 reason that only two replicas are maintained for most files. CDRM increases the number of file replicas to maintain a required file reliability, so it provides high file reliability under different server failure probabilities.

#### 4.2 Experimental Results for Enhancement Strategies

In the following, we show the effectiveness of each of our proposed strategies for enhancement: i) dynamic transmission rate adjustment strategy, ii) network-aware data node selection strategy and iii) load-aware replica maintenance strategy. In the following figures, we use *EAFR/B* to denote the basic EAFR without any enhancement strategies.



(a) Performance under different # of (b) Performance under different access concurrent accesses. arrival rates.

Fig. 12. File read response time.

## 4.2.1 Effectiveness of Dynamic Transmission Rate Adjustment Strategy

Fig. 11 shows the replication completion time for different file groups with and without the dynamic transmission rate adjustment strategy (denoted by *EAFR-rate* and *EAFR/B*). We set the replication completion time of *EAFR/B* as base and plot the ratio of *EAFR-rate* and *EAFR/B*'s replication completion time over the base. We see that *EAFR-rate* effectively reduces replication time compared to *EAFR/B*. The reason is that *EAFR-rate* dynamically adjusts the senders' transmission rates based on the receiver's bandwidth consumption, which can prevent incast congestion on the receiver side. As a result, the receiver is not likely to be congested and file replication operations can be completed within short latency.

## 4.2.2 Effectiveness of Network-Aware Data Node Selection Strategy

We denote EAFR with and without applying the proposed network-aware data node selection strategy by *EAFR-net* and *EAFR/B*, respectively. In *EAFR/B*, a compute node fetches its requested file from a randomly selected data node among the data nodes storing the file's replicas.

Figs. 12a and 12b show the average file read response time with different number of concurrent reads and different access arrival rates, respectively. We see that the response latency increases as the number of concurrent reads increases due to the same reason as in Fig. 6a. We also see that EAFR-net reduces the file read response latency. A compute node in EAFR-net tends to fetch files from data nodes within the same rack as the requester computer nodes to minimize the file transmission time, and from data nodes with small queue sizes to reduce the queueing delay. Therefore, a compute node can finish reading a file within shorter latency in EAFR-net than that in EAFR/B. We also notice that the reduction in response latency becomes larger when there are a larger number of concurrent reads and access arrival rates in the system. This is because when the number of concurrent reads and access arrival rates increase, servers in *EAFR/B* are more likely to be overloaded as the read requests are assigned to servers without considering their queue sizes, which leads to file read response time increase. On the other hand, EAFR-net aims to assign read requests to servers with small queue sizes, which reduces the file read latency compared to EAFR/B. Figs. 13a and 13b show the percentage of file read timeouts with different number of concurrent reads and different access arrival rates, respectively. We see that EAFR-net reduces the percentage of file



(a) Performance under different # of (b) Performance under different access concurrent accesses. arrival rates.

#### Fig. 13. Percentage of file read timeouts.

read timeouts due to the same reason as explained in Figs. 12a and 12b. *EAFR-net* aims to minimize file read response time by letting a compute node to fetch files from data nodes within the same rack and from data nodes with small queue sizes; while *EAFR/B* randomly assigns read requests to data nodes, which results in high percentage of file read timeouts. Figs. 12a, 12b, 13a, and 13b show the effectiveness of our proposed network-aware data node selection strategy in reducing file read response time.

## 4.2.3 Effectiveness of Load-Aware Replica Maintenance Strategy

We denote EAFR with and without applying the proposed load-aware replica maintenance strategy by EAFR-load and EAFR/B, respectively. In this experiment, we randomly selected a number of servers as failed servers every 30 minutes and recovered all file replicas stored in each failed server. We then recorded the average recovery latency. EAFR/B randomly selects a source server for a file's replica without considering server capacities, and also randomly selects a destination server with enough storage capacity to place a file's replica without balancing the number of replicas stored in each destination server to constrain the incast network load. Fig. 14 shows the average replica recovery latency for a various number of failed servers. We see that as the number of failed servers increases, both EAFR-load and *EAFR/B* generate longer replica recovery latency. The reason lies in that more failed servers lead to the replication of a larger number of file replicas. As more files are transmitted from source servers to the identified destination servers, these transmissions need to compete for limited bandwidth capacity and thus lead to longer transmission delay. We also see that EAFR-load improves EAFR/B by generating shorter recovery latency. In EAFR-load, we aim to select servers with the maximum remaining service capacity as source servers, so file can be read from source servers with short latency. Also, compared to EAFR/B, EAFR-load can balance the load



Fig. 14. Server maintenance latency for various number of failed servers.

(i.e., the number of replicas allocated) of destination servers as it aims to evenly allocate file replicas to all destination servers, which effectively prevent incast congestion in destination servers and generate shorter file transmission time. As a result, *EAFR-load* generates shorter recovery latency.

## 5 CONCLUSIONS

The popularity of data-intensive clusters places demands for file systems such as short file read latency and low power consumption. File replication is an effective method to enhance data availability, reduce read latency and power consumption. However, replication methods in current file systems cannot meet these demands well. 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 percent CPU utilization to save power. EAFR selects servers with sufficient capacities to place new replicas to shorten replication completion time and avoid overloading servers. EAFR also has a transmission rate adaptation strategy to further prevent potential incast congestion, a network-aware data node selection strategy to reduce file read latency and a load-aware replica maintenance strategy to maintain a certain number of replicas upon server failures. Experimental results from a real-world large cluster show the effectiveness of EAFR and the proposed strategies 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.

#### ACKNOWLEDGMENTS

This research was supported in part by US National Science Foundation grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of ICCCN 2015 [47].

#### REFERENCES

- K. Shvachko, K. Hairong, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.
- [2] Lustre File System. (2015). [Online]. Available: http://www.lustre.org, Accessed on: Nov. 2015.
- [3] Version 2 parallel virtual file system. (2015). [Online]. Available: http://www.pvfs.org, Accessed on: Nov. 2015.
- [4] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Operating Syst. Des. Implementation*, 2006, pp. 307–320.
- [5] Z. Cheng, et al., "ERMS: An elastic replication management system for HDFS," in *Proc. CLUSTER Workshops*, 2012, pp. 32–40.
- [6] Q. Chen, J. Yao, and Z. Xiao, "Libra: Lightweight data skew mitigation in MapReduce," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2520–2533, Sep. 2014.
- [7] A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, pp. 28–28.
- [8] J. Luo, L. Rao, and X. Liu, "Temporal load balancing with service delay guarantees for data center energy cost optimization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 775–784, Mar. 2014.

- [9] L. Mashayekhy, M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of MapReduce jobs for big data applications," *IEEE Trans. Parallel Distrib. Syst. TPDS*, vol. 26, no. 10, pp. 2720–2733, Oct. 2014.
- [10] Y. Chen, A. Ganapathi, A. Fox, R. Katz, and D. Patterson, "Statistical workloads for energy efficient mapreduce," Tech. Rep. UCB/EECS-2010-6, Univ. California, Berkeley, CA, USA, 2010.
- [11] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 231–242.
- [12] A. Greenberg, et al., "VL2: A scalable and flexible data center network," Proc. ACM SIGCOMM Conf. Data Commun., 2009, pp. 51–62.
- [13] W. Yu, Y. Wang, and X. Que, "Design and evaluation of networklevitated merge for hadoop acceleration," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 3, pp. 602–611, Mar. 2014.
- [14] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in Proc. SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun., 2012, pp. 431–442.
- [15] M. Bourguiba, K. Haddadou, I. El Korbi, and G. Pujolle, "Improving network I/O virtualization for cloud computing," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 3, pp. 673–681, Mar. 2014.
- [16] J. Zhang, F. Ren, L. Tang, and C. Lin, "Modeling and solving TCP incast problem in data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 478–491, Feb. 2015.
- [17] 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. IEEE Int. Conf. Cluster Comput.*, 2010, pp. 188–196.
- [18] G. Ananthanarayanan, et al., "Scarlett: Coping with skewed content popularity in MapReduce clusters," in Proc. IEEE 7th Int. Conf. Cloud Comput., 2011, pp. 408–415.
- [19] L. Abad, Y. Lu, and R. Campbell, "Dare: Adaptive data replication for efficient cluster scheduling," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2011, pp. 159–168.
- [20] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 19–19.
- [21] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 98–109.
  [22] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-siz-
- [22] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proc. INFOCOM*, 2011, pp. 1–10.
  [23] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data
- [23] 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. IEEE INFOCOM*, 2012, pp. 1431–1439.
- [24] Y. Guo, Y. Gong, Y. Fang, P. Khargonekar, and X. Geng, "Energy and network aware workload management for sustainable data centers with thermal storage," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 8, pp. 2030–2042, Aug. 2014.
- [25] H. Shao, L. Rao, Z. Wang, X. Liu, Z. Wang, and K. Ren, "Optimal load balancing and energy cost management for internet data centers in deregulated electricity markets," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 10, pp. 2659–2669, Oct. 2014.
- [26] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 1, pp. 200–211, Jan. 2014.
- [27] H. Amur, J. Cipar, V. Gupta, G. Ganger, M. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 217–228.
  [28] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of
- [28] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of hadoop clusters," *Operating Syst. Rev.*, vol. 44, no. 1, pp. 61–65, 2010.
- [29] J. Chase and R. Doyle, "Balance of power: Energy management for server clusters," in *Proc. 8th Workshop Hot Topics Operating Syst.*, 2001, pp. 163–165.
- [30] R. Kaushik, T. Abdelzaher, R. Egashira, and K. Nahrstedt, "Predictive data and energy management in green HDFS," in *Proc. Int. Green Comput. Conf. Workshops*, 2011, pp. 1–9.
- [31] 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. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, 2010, pp. 274–287.

- [32] R. Kaushik and M. Bhandarkar, "Greenhdfs: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster," in *Proc. Int. Conf. Power Aware Comput. Syst.*, 2010, pp. 1–9.
  [33] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues, "AUTOPLACER:
- [33] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues, "AUTOPLACER: Scalable self-tuning data placement in distributed key-value stores," in *Proc. 10th Int. Conf. Autonomic Comput.*, 2013, pp. 119–131.
- [34] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: A workload-driven approach to database replication and partitioning," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 48–57, 2010.
- [35] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW client-based traces," Boston Univ., Boston, MA, USA, Tech. Rep. 1995–010, 1995.
- [36] 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," *Concurrency Comput.: Practice Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.
  [37] J. Liu, F. Zhao, X. Liu, and W. He, "Challenges towards elastic MULTICE and MULTICE
- [37] J. Liu, F. Zhao, X. Liu, and W. He, "Challenges towards elastic power management in internet data centers," in *Proc. 29th IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, 2009, pp. 65–72.
- [38] Sandia CTH trace data. (2015). [Online]. Available: http://www. cs.sandia.gov/Scalable\_IO/ SNL\_Trace\_Data/, Accessed on: Nov. 2015.
- [39] C. Huang, et al., "Erasure coding in windows azure storage," in Proc. USENIX Conf. Annu. Tech. Conf., 2012, pp. 2–2.
- [40] Palmetto cluster. (2015). [Online]. Ávaialable: http://citi.clemson. edu/palmetto/index.html, Accessed on: Nov. 2015.
- [41] J. Lucas and M. Saccucci, "Exponentially weighted moving average control schemes: Properties and enhancements," *Technometrics*, vol. 32, no. 1, pp. 1–29, 1990.
- [42] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *Proc. Conf. Supercomputing*, 2004, Art. no. 53.
- [43] E. Krevat, et al., "On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems," in *Proc. Conf. Supercomputing*, 2007, pp. 1–4.
- [44] J. Strauss, D. Katabi, and M. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proc. 3rd ACM SIG-COMM Conf. Internet Measurement*, 2003, pp. 39–44.
- [45] R. Hendrickson, "The Facts on File Encyclopedia of Word and Phrase Origins," New York, NY, USA: Facts on File, 1997.
- [46] ThinkServer. (2015). [Online]. Available: http://shop.lenovo.com/ us/en/servers/, Accessed on: Nov. 2015.
- [47] Y. Lin and H. Shen, "EAFR: An energy-efficient adaptive file replication system in data-intensive clusters," in *Proc. 24th Int. Conf. Comput. Commun. Netw.*, 2015, pp. 1–8.



Yuhua Lin received the BS degree in software engineering and the MS degree in computer science from Sun Yat-sen University, China, in 2009 and 2012, respectively. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, Clemson University. His research interests focus on effective and economical content delivery strategy on the cloud. He is a member of the IEEE.



Haiying Shen received the BS degree in computer science and engineering from Tongji University, China, in 2000, and the MS and PhD degrees in computer engineering from Wayne State University, in 2004 and 2006, respectively. She is currently an associate professor in the Department of Computer Science, University of Virginia. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid

and cloud computing. She was the program co-chair for a number of international conferences and member of the program committees of many leading conferences. She is a Microsoft faculty fellow of 2010, a member of the ACM, and a senior member of the IEEE.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.