J Supercomput (2018) 74:1045–1060
https://doi.org/10.1007/s11227-016-1827-3

E2FS: an elastic storage system for cloud computing

Longbin Chen[1] · Meikang Qiu[1] · Jeungeun Song[2] · Zenggang Xiong[3] · Houcine Hassan[4]

Published online: 27 August 2016

Abstract In cloud storage, replication technologies are essential to fault tolerance and high availability of data. While achieving the goal of high availability, replication brings extra number of active servers to the storage system. Extra active servers mean extra power consumption and capital expenditure. Furthermore, the lack of classification of data makes replication scheme fixed at the very beginning. This paper proposes an elastic and efficient file storage called E2FS for big data applications. E2FS can dynamically scale in/out the storage system based on real-time demands of big data applications. We adopt a novel replication scheme based on data blocks, which provides a fine-grained maintenance of the data in the storage system. E2FS analyzes features of data and makes dynamic replication decision to balance the cost

✉ Longbin Chen
longbin.chen@pace.edu

Meikang Qiu
mqiu@pace.edu

Jeungeun Song
jsong@hust.edu.cn

Zenggang Xiong
jkxxzg2003@163.com

Houcine Hassan
husein@disca.upv.es

1 Pace University, New York, USA

2 School of Computer Science and Technology, Huazhong University of Science and Technology, Hubei, China

3 School of Computer Science and Information Technology, Hubei Engineering University, Hubei, China

4 Universitat Politecnica de Valencia, Valencia, Spain

and performance of cloud storage. To evaluate the performance of proposed work, we implement a prototype of E2FS and compare it with HDFS. Our experiments show E2FS can outperform HDFS in elasticity while achieving guaranteed performance for big data applications.

Keywords Elastic replication · Data usage analysis · Cloud storage · Availability

1 Introduction

Cloud storage provides the primary support for big data applications. Large clusters of physical and virtual servers provide disk spaces to fulfill the demands of applications which deal with massive datasets [1]. Datacenters or cloud storage systems run two types of applications: external and internal applications. External applications work for cloud users directly. They allocate resources [2] for user applications and schedule tasks [3] for cloud consumers. Internal applications work for the datacenter itself. They determine the location of data copies, migrate virtual machine, and recover lost data during node failure.

Big data applications have high demands for back-end storage systems because of their volume, variety, velocity, and veracity [4,5]. Large-scale storage systems such GFS and HDFS have been designed, implemented, and used at the early stage of cloud computing. However, the initial rationales of those distributed storage were not fully targeted at big data applications. Although these storage systems have been widely used in real world for many years, there is always room for improvement of current big data storage. An important issue is to consider the trade-off between cost and performance.

In cloud computing, storage systems are designed to be scalable and elastic. During peak hours, cloud storage needs to scale out to meet the increasing demands of cloud users. Scaling out means adding more server machines to the storage system. During off-peak hours, a number of servers is shutting down gracefully to reduce power consumption and other costs. It is a trade-off between cost and performance. By scaling out, cloud storage can provide high throughput and sufficient storage or computing resources to clients. However, as a result, scaling out largely increases the power consumption, capital expenditure, and operation expenditure.

In addition to providing on-demands resources and services, cloud computing is notable for high availability of user data. In the era of cloud computing, more and more users choose clouds as their primary backup of important data, such as photos, text documents, movies, and even software [6]. One major reason is the guaranteed availability of data storage in clouds.

Replication surely improves the availability of cloud storage, but the extra pains are also coming along with it. Assume a cloud storage with replication factor of 3, which means there are two extra copies of all data existing in the datacenter. The extra cost on disk storage is tremendous in a large-scale cluster with thousands of nodes. Moreover, the use of replication schemes also brings the issue of data consistency and other operational expenditure.

In this paper, we present E2FS, an elastic storage system for large datasets. E2FS aims at minimizing the cost in datacenter while ensuring high availability and performance for cloud users. To achieve the optimal balance between cost and performance, E2FS uses a technique called data classification. The key idea is to classify data files stored in cloud storage systems. Data classification is common in many cloud applications especially data-intensive applications. But many cloud storage systems, such as HDFS and GFS, are not aware of the difference of data files in their replication schemes. Data classification of E2FS attaches tags to each data file based on its historical activities. These tags can be used to assign priorities to large data files. E2FS dynamically adjusts replication schemes for different data files and re-sizes the storage system accordingly to reduce energy consumption. E2FS also supports replication schemes with dedicated block distribution like CopySets [7]. Data blocks or chunks are basic unit of storage in datacenters. Instead of random replication, we compute the reliability of servers with many other factors and place data blocks for high availability and throughput. Another novel idea of E2FS is its awareness of network performance in the datacenter.

The main contributions of the paper include: (1) a data classification for replication proposed on cloud storage. (2) A model introduced to calculate availability in aware of network failure. (3) A fine-grained block distribution proposed to ensure high availability with minimum space cost. The rest of the paper is organized as follows: Sect. 2 motivates the work. Section 3 presents the proposed algorithms. Section 4 shows the experiments and results. Section 5 lists work related to this paper. Section 6 concludes the paper.

2 Motivations

With the explosive growing of big data market, cloud storage systems face the critical challenge of maintaining large unruly datasets [8]. Because of the variety and volume of datasets, the desire for new elastic storage system is increasing. Traditional storage systems such HDFS and GFS have proved their success in supporting large-scale distributed computing. Yahoo reported great computing performance of Hadoop on HDFS. GFS is also greatly helpful for storing billions of web pages everyday. However, there is still room for improvement in cloud storage due to the following reasons: (1) the diversity of user data in this era. (2) Various user patterns on read and write. (3) The development of network virtualization technologies.

The diversity of user data is already quite different from what it is 5 years ago. The type of data varies from texts such as tweets and blogs to images and videos. Multimedia data such as images, audios, and videos have caused the major traffic of the Internet. In 2014, video traffic was 64 percent of all consumer Internet traffic and will reach 80 percent in 2019 [9]. Users tend to upload more and more files to cloud storage. Table 1 lists the storage service provided by major cloud companies [10]. Almost every vendor provides free storage space of several Gigabytes. Assume each user uses 50 % of the free space in average, the number will be 2.5 to 7.5 GB per user. In cloud storage, three-way replication is widely used. It means three copies of the data are stored across the datacenter to keep high availability of data files for clients.

Table 1 Major cloud storage services

Provider	Free storage	File size	Pricing
OneDrive	5 GB	10 GB	\$2/*M* for 50 GB
Dropbox	2 GB	10 GB	\$10/*M* for 1 TB
GoogleDrive	15 GB	5 TB	\$2/*M* 100 GB, \$10/M 1 TB
Box	10 GB	25 GB	\$10/*M* for 100 GB

Hence, if cloud storage providers adopt three-way replication, they will have to use extra 5 to 15 GB disk space for each user to ensure high availability. The number is even more astonishing if we consider individual users with paid plans and enterprise users.

2.1 Block-based replication

Replication can ensure high availability in cloud storage [11]. Usually, we use a replication factor R to indicate the number of copies of the same data in cloud storage. Besides the replication factor R, there is another important factor: block size. In large-scale storage systems, data are stored as files of several terabytes or larger. To improve efficiency, large files are split into equal-sized blocks. For example, the default block size in HDFS is 64 MB. Block size defines the granularity of storage. In some replication schemes, blocks are treated as the target of replication instead of whole files. Such designs make the storage system more flexible and effective [12].

Here, we introduce a term: distribution rate d. d works as a key factor in this paper and it reflects how data blocks are distributed across the storage system. In a storage system with N nodes and R as the replication factor. A large data file is divided into M blocks. We assume that N is far greater than M. We calculate d as: $d = D/M$, where D is the number of separately stored blocks and M is number of blocks for a file. For example, $N = 1000$, $R = 3$, and $M = 30$. If all data blocks of a copy are stored in the same replica node, the distribution rate d is $1/M$; if all data blocks are stored on M replica nodes separately, then d is 100 %. We adopt the factor in our node recovery scheme. Hence, when d is 100 %, the number of replica nodes involved in this replication is: $M \times R = 90$.

2.2 Cloud data usage analysis

Cloud data usage is the core parameter of the proposed work. Data usage means the access frequency of data after users upload it to cloud systems. Such data consist of several types. First, social media data such as photos, blogs, and videos from social media services. Facebook claims that it stores billions of pictures and millions of new pictures are uploaded to Facebook each day. Twitter also stores billions of tweets which can sum up to multiple TB. The other type is cloud storage service such as Google drive, Amazon S3, and DropBox. These services either provide free storage spaces of several GB or charge very little for large amount of disk space. Users can get extremely large storage with low-price plans. Individuals and small companies store documents, images, or even videos on such cloud storage while enterprise users purchase massive disk space for their large datasets (Fig. 1).

No matter which type of data, there is a usage rate for each of them. It is possible that a set of data is uploaded to cloud systems and never used for a long period of time, just like some data on personal computers. For such data, the read and write operations are relatively small compared to other data. We use the terms "hot" and "cold" to represent the popularity of user data. Hot data mean that the data are frequently used while cold data are the type of data rarely used or modified by users. We list some terms to measure the usage of cloud data in Table 2.

We use visit per time period (VPT) to present the number of accesses to a certain data file during a give time period. It could be a minute, a hour, or even a day. Visit intervals represent the average time interval between two visits of the data file during a given time period. In this paper, the term VPH means the number of visits per hour. For example, a data file is accessed 200 times in an hour, then its VPH is 200. If a data file is visited at time $t1$ and re-visited at time $t2$, then the visit intervals of the data file is $t2 - t1$. In a time period N, we can calculate the average time interval between two visits as: $t = \text{avg}(t2 - t1, t3 - t2, \ldots, tn - tn - 1)$, where avg is the function to compute average numbers. The largest time interval can be expressed as:

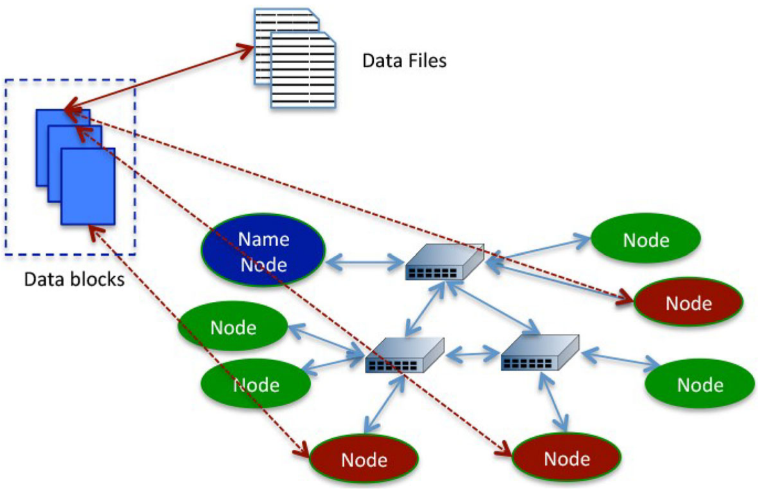

Fig. 1 Replication scheme

Table 2 Measurement of data usage

Name	Description
VPT	Visit per time period
AVT	Average visit time interval
LAVT	Largest visit time interval

$t = \max(t2 - t1, t3 - t2, \ldots, tn - tn - 1)$, where max is the function to get maximum number.

2.3 Elasticity of cloud storage

The key point of E2FS is to be elastic even when external demands are stable. Many cloud storage are designed to be elastic to only meet changing user demands. For example, if users require more hardware resources, the datacenter will power on more servers to meet their requirements of computation or storage. When the demands of resource declines, the datacenter turns down parts of its servers to reduce power and operational costs.

Such elasticity can guarantee the service level agreement (SLA) while utilizing hardware and software resources in datacenters. However, a major concern is the overhead of storage and transmission caused by coarse-grained replication schemes. As we mentioned previously, most cloud storage systems use random three-way replication. Three extra copies of data files are stored in the datacenter. As we calculated above, the price of ensuring high data availability is expensive. In cloud storage, with the classification of data files, we can adjust our replication schemes for different types of files.

In E2FS, we propose a dynamic scheme to scale out or in copies of data files based on the popularity of data files. Instead of giving two extra copies to all data, E2FS assigns different numbers of copy to various data based on their popularity. If a data file is visited frequently, the replication factor of the data will be three or higher. Meanwhile, if a data file is rarely used, its number of extra copy will shrink to 1 or 0.

With the analysis of user data, we will be able to perform our dynamic replication schemes. We profile user activities on data files. The method collects a user's visit history of data files in a E2FS storage system. E2FS maintains metadata for each user to record the access history of stored data. Based on the metadata, E2FS dynamically adjusts the number of replications for all data of the user.

3 Algorithms

In this section, we introduce our replication schemes. First, we explain our block-based replication scheme. Our scheme aims at distributing data blocks evenly across the cluster. Therefore, we can achieve maximum parallelism in reads. Second, we introduce our elastic replication adjustment, which adjusts the number of replicas in real-time based on the profiled user data patterns.

In traditional replication schemes, nodes or servers are selected randomly to store the copies of data blocks. Such approaches are not effective since the network topology of the storage system is not fully considered. Furthermore, the location of data blocks is not treated as a primary factor. In our replication scheme, we include network topology of the storage system and the distribution rate of blocks into our replication scheme to further improve the availability and efficiency of cloud storage. Our node recovery scheme also benefits from this replication design.

The selection of replica nodes is described in Algorithm 3.1. In this algorithm, we define a sub-network as: if servers connect to the same switch directly, then we consider them in the same sub-network. If servers connect with others through more than one switch, we consider them in different subnetworks. When a data file is uploaded to the storage system, it is split into equal-sized blocks (lines 1–2). The replication factor R in line 3 represents the number of copies will be existing in the storage system. During the replication, we consider the distribution rate of blocks. We term K to be the number of blocks which will be copied on the same node. K is calculated as: $K = M/D$, where M is the total number of blocks and D is the distribution rate. For each copy of data, we decide the replica node for every K blocks (lines 4–20).

For example, if $R = 3$ and $D = 3$, for the first copy of data, we select three available nodes from the same sub-network of the original datanode. We assume there are at least three datanodes available in the sub-network. We distribute data blocks across all three datanodes. Each datanode has $M/3$ data blocks in it. For the second copy of the data, we select three available nodes from a different sub-network in the same rack. Similarly, we copy $M/3$ blocks on each datanode. After the replication, there are three copies of the data files in the storage system with a distribution rate of $3/M$.

```
Algorithm 3.1 Block-based replication scheme
Require: data file, node list, distribute number D, replication factor R
Ensure: replicated data blocks
 1: while data file is not null do
 2:   split file into N blocks
 3:   compute subnetworks for R replicas
 4:   for each replica subnet do
 5:     get M available nodes
 6:     if $M \geq D$ then
 7:       for each node i do
 8:         if $N \geq 0$ then
 9:           copy next N/D blocks to i
10:         end if
11:       end for
12:     else
13:       for each block j do
14:         $k = j \bmod M$
15:         copy j to node k
16:       end for
17:     end if
18:   end for
19: end while
```

In Algorithm 3.2, we explain our dynamic adjustment of replicas. During a time period T, we check metadata of each data file (line 2). We ignore files which have been adjusted recently (line 3). If a file's VPT is greater than the higher threshold VPTH and the file has fewer replicas than three, we increase the number of copies for that file (lines 4–7). On the contrary, if the file's VPT is less than the lower threshold VPTL and the number of copies is greater than two, we reduce copies for the file (lines 8–11). We perform the similar scheme on average visit per time period in lines 12–19. We increase or decrease the number of copies based the average visits of files.

```
Algorithm 3.2 Dynamic Replica Adjustment
Require: Data usage information D, time period T, threshold VPTH and VPTL threshold
    AVTH and AVTL
Ensure: Replication adjustment
 1: while During a time period T do
 2:   for Each file f in datanode do
 3:     if f is not in recent adjustment list then
 4:       if f' vpt greater than VPTH and f's replica less than 3 then
 5:         increase f's replica
 6:         add f to adj list
 7:       end if
 8:       if f's vpt less than VPTL and f's replica greater than 2 then
 9:         decrease f's replica
10:         add f to adj list
11:       end if
12:       if f's avt great than AVTH and f's replica less than 3 then
13:         increase f's replica
14:         add f to adj list
15:       end if
16:       if f's avt less than AVTL and f's replica greater than 2 then
17:         decrease f's replica
18:         add f to adj list
19:       end if
20:     end if
21:   end for
22: end while
```

4 Experiments

In this section, we evaluate the performance of our replication and recovery schemes on customized storage system. We examine read and write latency of our replication scheme. We also generate random user data access and classify the data based on their visit history.

4.1 Implementation

To evaluate the performance of our replication and node recovery schemes, we develop a customized cloud storage system which is similar to HDFS but with basic features only. The storage system has 16 nodes including 1 namenode and 15 datanodes. These nodes are virtual nodes running on physical machines.

Each node has limited disk storage and virtual network interfaces. Datanodes store data files in the form of blocks. The namenode stores and maintains metadata such as the location of data files, the mapping between data files and blocks, and the information of all copies of a data file. In our design, all metadata are stored in disk files. The information of a data file is kept as entries in a unique file on namenode; when data on datanodes changes, the related information on metadata files is also updated. The namenode is responsible for assigning datanodes to clients and selecting replica nodes for each data file.

We provide interfaces for client programs. Through the interfaces, clients can perform operations including upload, read, and write. When a client program uploads files from its local disk to the storage system. The namenode receives the request, and determines a datanode to store the file. The address of the datanode is returned to the client program. After that, the files are sent over TCP sockets to the datanode. Once the write operation completes, the namenode selects several datanodes to store the copies of the new data files. This process is transparent to the client.

4.2 Experimental setup

Our experiments were running on four physical machines. We develop prototype systems to create 16 virtual nodes on the four physical servers. We configure one virtual node to be namenode of the storage system, and the other 15 nodes to be datanodes. Each node has a CPU core of 2.5 GHz, 4 GB RAM, and 100 GB local hard disk space. The default block size is 32 MB. The namenode has manager program running on it, which maintains metadata of each datanode and takes charge of replication and recovery. We also have network controller running on namenode. The controller reads link data from switches and computes forwarding decisions at runtime.

To make our evaluation simple, we use files with size of 640 MB, 1.28 or 2.56 GB. Hence, the file size can be easily divided by block size. In the fat-tree topology, we assume each switch has eight ports. We set the bandwidth of edge switches to be 1 MB/s and the one of core switches to be 8 MB/s. By read operations, we mean downloading the full file from the storage system. For write operations, we upload the whole file and update all copies of the file instead of performing partial updates. To complete these operations, we write programs to simulate clients of the storage system. The client programs use interfaces to upload and download files to and from the storage system. As a comparison, we also set up HDFS on our virtual nodes.

4.3 Analysis of data usage

In this experiment, we profile data usage for our elastic replication. Figure 2 shows the number of visits per hour and Fig. 3 shows the maximum time intervals between two visits of the same data. Both parameters show the popularity of data in cloud storage. Visit-per-hour shows the frequency of data usage while visit-time-intervals reveal the idle time of data. In Fig. 2, we divide the number of visits into ten sections. The first section means the number of visits is greater than 1000 times per hour. The second section means the number of visits is between 900 to 1000 in an hour. The last section indicates the number of visits is less than 200 in an hour. As we observe, only 15 % of total data has a vph higher than 1000. More than 30 % of data is visited less than 200 times in an hour. While the other vph numbers all have percentages lower than 12 %.

In Fig. 3, we examine the maximum time intervals between two visits of the same data. We divide the time intervals into ten sections. The smallest interval is less than 100 s, and the largest interval is over 1350 s (roughly 22 min). This experiment shows the idle time of selected datasets. Only 8 % of the profiled data is visited twice or more within 1 and a half minute. About 9 % of the data is visited twice within 2.5 min. More than 20 % of the profiled data is never visited within 22 min.

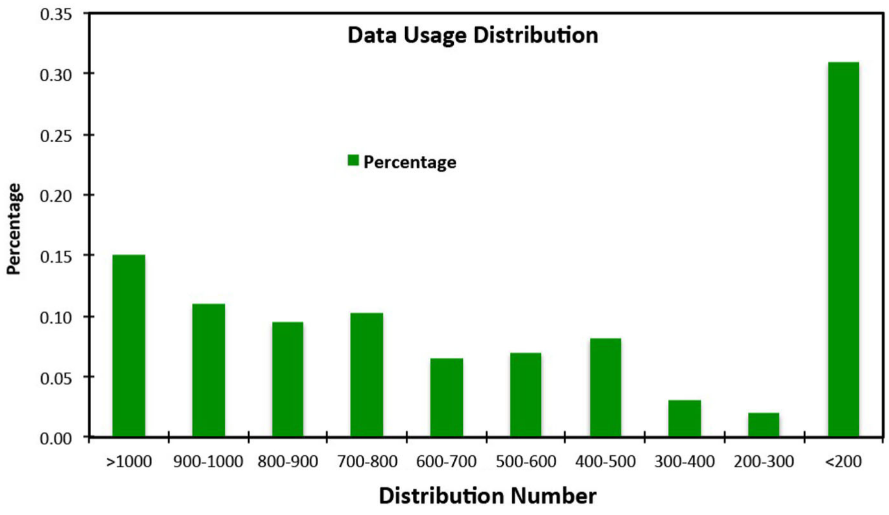

Fig. 2 Visit per hour

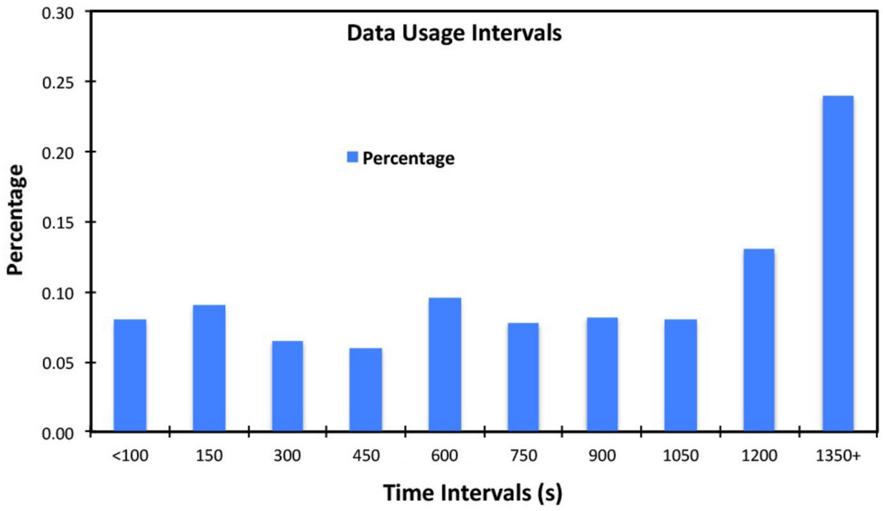

Fig. 3 Visit time intervals

4.4 Latency analysis

In this evaluation, we focus on how read/write latency is affected by our replication scheme. We assume the read/write latency of single copy storage as baseline. We first analyze read latency of HDFS and E-replica scheme. We use replication number as the tuning factor to examine the effect of E-replica scheme. The replication number R varies from 1 to 15. When $R = 1$, only a single copy of the data exists in the storage system. In this case, when a client requests for the data file, only one datanode is available for the read operation.

Hence, the maximum speed of the read depends on the only TCP link between the datanode and the client. When $R = 15$, the data are fully replicated across the storage system. Every datanode has a copy of the data. Hence, the read latency depends on the total bandwidth of the client's network.

However, in traditional storage system, copies of data files are not fully used. Figure 4 shows that even if the replication factor grows, it has little effect on the read latency of HDFS storage. In contrast, our scheme makes full use of all replications when a client requests for the file. Through computation, our scheme returns the client with connections to multiple datanodes. Each datanode has a copy of the file. Our design allows different blocks of the file to be transmitted to the client over simultaneous connections. Therefore, the read performance can be fully utilized.

However, it is not practical to have high replication factor. There are some reasons: (1) waste of disk space. When we have high replication factor in storage systems, it means that too many copies of the same file exist in the system. Large amounts of disk space are used to store same files. (2) Replication increases write cost. In cloud storage, synchronization is a major challenge. Every copy of a data file needs to be synchronized correctly when the file is modified. Large replication factor means large numbers of write operations in the storage system. The cost of maintenance increases significantly. Figure 5 shows the impact of replication on write latency for HDFS and E-replica. We assume the write latency of a single copy is N. When the file is fully replicated, the write operations increase up to 10 times of the single copy replication. Therefore, in practice, a replication factor of three is widely agreed due to its balance in availability and write performance.

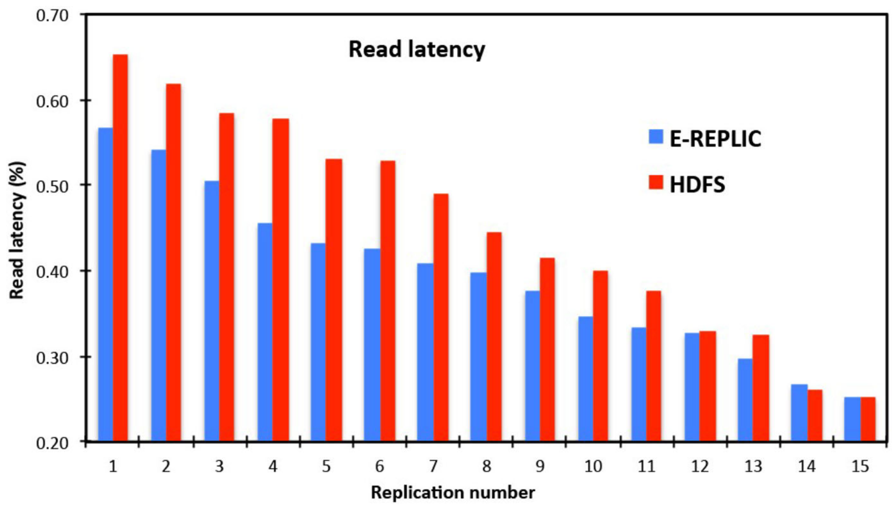

Fig. 4 Read latency analysis

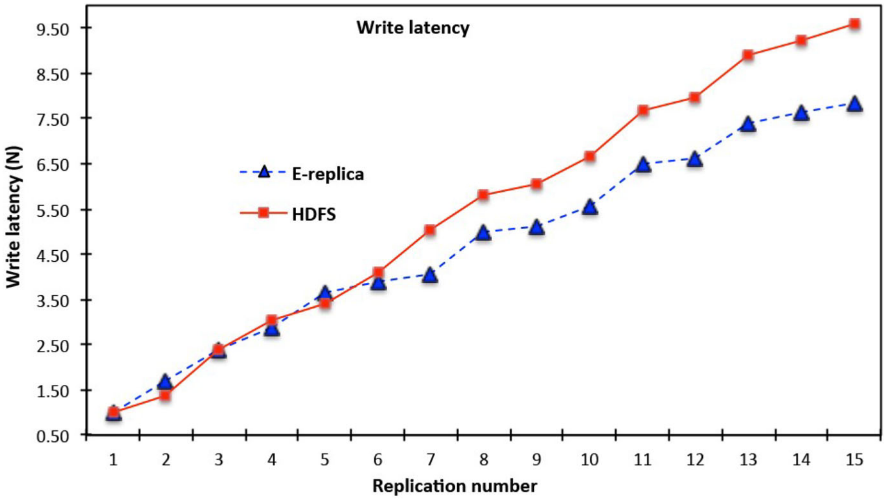

Fig. 5 Write latency analysis

4.5 Block distribution

Block distribution rate is a critical factor to increase data read and recovery performance. Block distribution can determine the parallelism of all data read operations, which means how many copies of data can be accessed simultaneously from multiple datanodes. Block distribution rate is defined based on the number of blocks for a data file. If a data file is split into N blocks, its block distribution rate ranges from $1/N, 2/N, \ldots, 1$. The term $1/N$ means that all blocks of the copies of file are stored on the same replica node. For example, if two extra copies of the file are kept in the storage system, then all blocks of each copy are stored on two datanodes. Therefore, the total number of datanodes attached to the file is three. If the rate is one, it means every block of the file is copied on a separate datanode. Assume the nu MBer of extra copies is also two.

For the first copy, N data blocks of the copy are stored on N different nodes. The scheme is repeated on the second copy, where another N datanode is selected. Therefore, in this case, the nu MBer of node related to the file is $2N + 1$. When a user tries to read a data file with block rate N, the ideal read latency can be calculated as: $t = s/(b + p \times n \times b)$. Term t is the total read latency, s is the size of the data file, b is the average bandwidth of nodes, p represents the percentage of using replicas in parallel, and n means the nu MBer of replicas.

In this experiment, the size of data files is 10.24 GB in total and they consist of documents, videos, and images. Since we have limited the nu MBer of nodes to 16, the nu MBer of data blocks of each file might be much larger than 10. Therefore, we use the distribution rate range from $1/10, 2/10, \ldots, 10/10$ since ten is the maximum nu MBer of possible replica. Similar to previous experiments, we use three-way unmodified HDFS as baseline for comparison. Figure 6 shows the read latency of data files. When only blocks are distributed to one or two extra nodes, the read latency is similar to

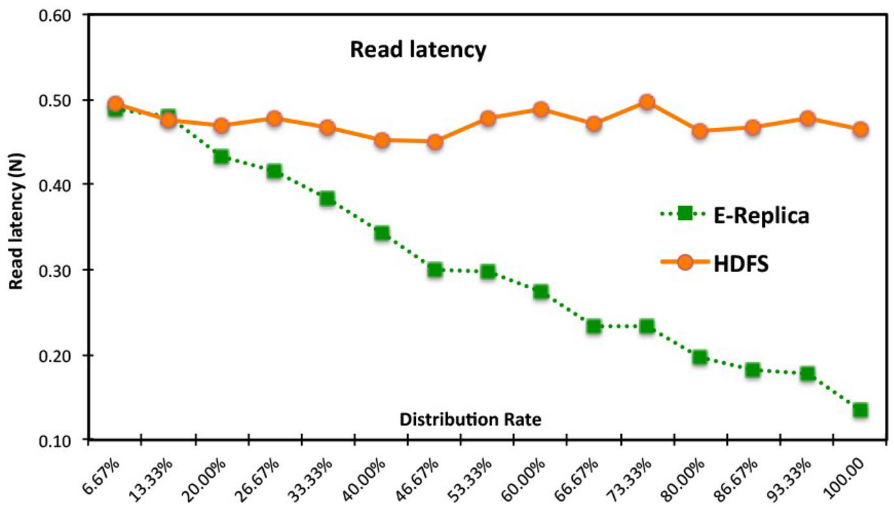

Fig. 6 Read latency with distribution rate analysis

HDFS. Read latency reduces significantly as three or more extra nodes are used in parallel. When data blocks are fully distributed across ten nodes, the total latency is around 13 % of the baseline nu MBer. Therefore, read operations can benefit largely from block distribution.

In the previous experiments, we randomly generate node failure in the simulated storage system. For simplicity, we assume only one node crashes at a time and no node crashes during the recovery of another node. If a node fails, the total time to recover data on that node is calculated as: $t = \text{tc} + \text{ts}$, where tc is the time used to choose a new node and ts is the time for all data to be sent to the new node.

5 Related works

SpringFS [13] is a new elastic storage system that fills the space between state-of-the-art designs in the trade-off among agility, elasticity, and performance. SpringFSs data layout and offoading/migration policies adapt to workload demands and minimize the data redistribution cleanup work needed for elastic resizing, greatly increasing agility relative to the best previous elastic storage designs. As a result, SpringFS can satisfy the time-varying performance demands of real environments with many fewer machine hours. Such agility provides an important building block for resource-efficient data-intensive computing (a.k.a. Big Data) in multi-purpose clouds with competing demands for server resources.

In an analysis of Facebook messaging [14], the authors present a multilayer study of the Facebook Messages stack, which is based on HBase and HDFS. We collect and analyze HDFS traces to identify potential improvements, which we then evaluate via simulation. Messages represent a new HDFS workload: whereas HDFS was built to store very large files and receive mostly sequential I/O, 90 % of files are smaller than 15 MB and I/O is highly random. We find hot data are too large to easily fit in RAM and cold data are too large to easily fit in flash; however, cost simulations show that adding a small flash tier improves performance more than equivalent spending on RAM or disks. HBases-layered design offers simplicity, but at the cost of performance; our simulations show that network I/O can be halved if compaction bypasses the replication layer. Finally, although messages are read dominated, several features of the stack (i.e., logging, compaction, replication, and caching) amplify write I/O, causing writes to dominate disk I/O.

Multi-tiered storage made up of heterogeneous devices are raising new challenges in allocating throughput fairly among concurrent clients. The fundamental problem is finding an appropriate balance between fairness to the clients and maximizing system utilization. In the work of Wang and Varman [15], the authors cast the problem within the broader framework of fair allocation for multiple resources. They present a new allocation model BAA based on the notion of per-device bottleneck sets. Clients bottlenecked on the same device receive throughputs in proportion to their fair shares, while allocation ratios between clients in different bottleneck sets are chosen to maximize system utilization. They show formally that BAA satisfies fairness properties of Envy Freedom and Sharing Incentive. They evaluated the performance of our method using both simulation and implementation on a Linux platform. The experimental results show that their method can provide both high efficiency and fairness.

In the work of Cidon et al. [16], the authors presented Tiered Replication, a replication technique that automatically places the *n*th replica on a separate cluster, while minimizing the probability of data loss under correlated failures, by minimizing the nu MBer of copysets. Tiered Replication improves the cluster-wide MTTF by a factor of 20,000 compared to random replication, without increasing the storage capacity. Tiered replication supports additional data placement constraints required by the storage designer, such as rack awareness and chain replication assignments, and can dynamically adapt when nodes join and leave the cluster. An implementation of Tiered Replication on HyperDex, a key-value storage system, demonstrates that it incurs a small performance overhead.

Bowers et al. [17] introduced High-Availability and Integrity Layer (HAIL), a distributed cryptographic system that allows a set of servers to prove to a client that a stored file is intact and retrievable. HAIL strengthens, formally unifies, and streamlines distinct approaches from the cryptographic and distributed system communities. Proofs in HAIL are efficiently computable by servers and highly compact, typically tens or hundreds of bytes, irrespective of file size. HAIL cryptographically verifies and reactively reallocates file shares. It is robust against an active, mobile adversary, i.e., one that may progressively corrupt the full set of servers. They propose a strong, formal adversarial model for HAIL, and rigorous analysis and parameter choices. They show how HAIL improves on the security and efficiency of existing tools, such as Proofs of Retrievability (PORs) deployed on individual servers. They also report on a prototype implementation.

6 Conclusion

High availability and performance are always the major concerns of cloud storage systems. While replication is widely used as a scheme to improve availability and throughput of storage systems, some important features of replication are still not explored. Data classification and block-based replication are promising in improving the efficiency of cloud storage. In the paper, we introduced a block-based replication and dynamic replica adjustment. Our block-based replication used block distribution as a key factor to improve effectiveness of replication schemes. We profiled and classified user data based on their visit history. We also implemented a prototype system and performed experiments. Our experiments proved the effectiveness of our dynamic replication adjustment compared to traditional three-way replication.

Acknowledgements This work is supported by NSF CNS-1457506 and NSF CNS-1359557.

References

1. Chen M, Hai J, Wen Y, Leung VC (2013) Enabling technologies for future data center networking: a primer. IEEE Netw 27(4):8–15
2. Li J, Qiu M, Niu J, Gao W, Zong Z, Qin X (2010) Feedback dynamic algorithms for preemptable job scheduling in cloud systems. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, DC, USA, pp 561–564
3. Dai W, Qiu M (2015) Energy optimization with dynamic task scheduling mobile cloud computing. Syst J IEEE PP(99):1–10
4. Chen M, Mao S, Zhang Y, Leung VC (2014) Big data: related technologies, challenges and future prospects. Springer Briefs in Computer Science
5. Zhang Y, Chen M, Mao S, Hu L, Leung VC (2014) Cap: Community activity prediction based on big data analysis. IEEE Netw 28(4):52–57
6. Chen M, Hao Y, Li Y, Lai C, Wu D (2015) On the computation offloading at ad hoc cloudlet: architecture and service modes. IEEE Commun Mag 53(6):18–24
7. Cidon A et al (2013) Copysets: reducing the frequency of data loss in cloud storage. In: USENIX Annual Technical Conference 2013 (USENIXATC 13). San Jose, pp 37–48
8. Qiu M, Ming Z (2013) Informer homed routing fault tolerance mechanism for wireless sensor networks. J Syst Archit 59(4):260–270
9. CISCO (2014) Cisco Visual Networking Index: Forecast and Methodology, 2014–2019 White Paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html. Accessed 18 Feb 2016
10. CNET (2013) Cloud storage comparison. http://www.cnet.com/how-to/onedrive-dropbox-google-drive-and-box-which-cloud-storage-service-is-right-for-you/. Accessed 18 Feb 2016
11. Gai K, Qiu M (2015) Dynamic Energy-aware Cloudlet-based Mobile Cloud Computing Model for Green Computing. J Netw Comput Appl 59:46–54
12. Wu G, Qiu M (2013) A decentralized approach for mining event correlations in dis- tributed system monitoring. J Parallel Distrib Comput 73(3):330–340
13. Xu L et al (2014) SpringFS: bridging agility and performance in elastic distributed storage. In: Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14). Santa Clara, CA, pp 243–255
14. Harter T et al (2014) Analysis of hdfs under hbase: A facebook messages case study. In: Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14), pp 199–212
15. Wang H, Varman P (2014) Balancing fairness and effciency in tiered storage systems with bottleneck-aware allocation. In: Proceedings of the 12th USENIX Conferenceon File and Storage Technologies (FAST 14), pp 229–242

16. Cidon A et al (2015) Tiered replication: a cost-effective alternative to full cluster geo-replication. In: 2015 USENIX Annual Technical Conference (USENIX ATC 15), pp 31–43
17. Bowers KD, Juels A, Oprea A (2009) Hail: a high-availability and integrity layer for cloud storage. In: Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM, New York, pp 187–198