An Ensemble of Replication and Erasure Codes for Cloud File Systems

Yadi Ma
University of Wisconsin
Madison, WI, USA

Thyaga Nandagopal
Bell Labs, Alcatel-Lucent
Murray Hill, NJ, USA

Krishna P. N. Puttaswamy
Bell Labs, Alcatel-Lucent
Murray Hill, NJ, USA

Suman Banerjee
University of Wisconsin
Madison, WI, USA

***Abstract*—Geographically distributed storage is an important method of ensuring high data availability in cloud computing and storage systems. With the increasing demand for moving file systems to the cloud, current methods of providing such enterprise-grade resiliency are very inefficient. For example, replication based methods incur large storage cost though they provide low access latencies. While erasure coded schemes reduce storage cost, they are associated with large access latencies and high bandwidth cost. In this paper, we propose a novel scheme named CAROM, an ensemble of replication and erasure codes, to provide resiliency in cloud file systems with high efficiency. While maintaining the same consistency semantics seen in today's cloud file systems, CAROM provides the benefit of low bandwidth cost, low storage cost, and low access latencies. We perform a large-scale evaluation using real-world file system traces and demonstrate that CAROM outperforms replication based schemes in storage cost by up to 60% and erasure coded schemes in bandwidth cost by up to 43%, while maintaining low access latencies close to those in replication based schemes.**

I. Introduction

High data availability is an important property that most cloud storage services such as Amazon S3, and Windows Azure offer today. For instance, Amazon S3 offers four 9's of availability. Today's cloud storage services provide these availability SLAs (Service Level Agreements) by replicating full copies [1], [2], [3] of a data object several times over geographically diverse data centers [2] (referred to as M-way replication in the literature, or *AllReplica* in this paper). When a copy of the data object in a data center is inaccessible, due to various transient failures, such as memory or disk overloads, network congestion, equipment failure or upgrades, or unplanned outages [4], [5], [6], the user is transparently redirected to another copy of the data to maintain high availability. Current storage services such as Amazon S3 and Google FS provide such resiliency against two failures [1] [1], [2] using this AllReplica model. However, with increasing number of users and enterprises moving to the cloud, the cost of replicating and keeping the huge amount of data consistent across multiple locations will only increase, driving up the bandwidth and storage costs. This necessitates a low-cost solution that can maintain high availability as well as low latency access times to this data object.

To reduce storage cost, cloud file systems (CFSs) are transitioning from replication to erasure codes [7]. Erasure codes have been employed by peer-to-peer (P2P) storage systems to address the same problem in the past, in order to

[1]To tolerate two simultaneous failures, a total of three copies of each data item are kept using replication model.

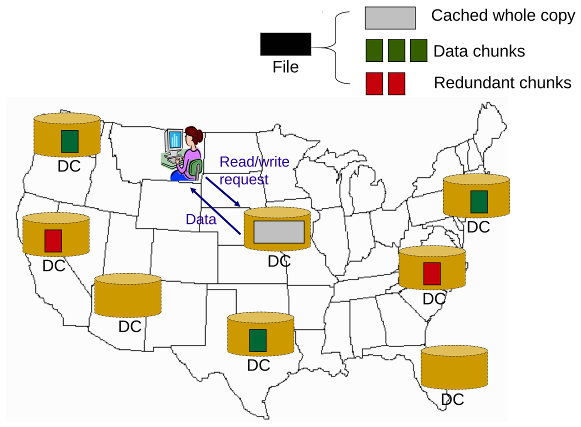


Fig. 1. An overview of our approach for cloud file systems. A file is divided into m data chunks and k redundant chunks using an erasure code. In this example, $m = 3$ and $k = 2$. Each chunk is stored in a different data center, which is called a backup data center. A copy of the whole file is cached temporarily in a primary data center upon write requests, in order to serve subsequent read and write requests. Upon a read request, the requested blocks can also be cached to serve subsequent reads of the same blocks.

provide high-availability and at the same time reduce storage cost. The key idea behind the solution is to use different encoding schemes to divide an object into multiple data chunks (say m data chunks), and k redundant chunks are calculated based on the data chunks to tolerate up to k failures. These chunks, both data chunks and redundant chunks, are stored in different locations [8], [9], [10]. Only m out of $m + k$ chunks are necessary to reconstruct the object. This scheme (referred to as *AllCode* thereafter) is efficient because instead of storing whole copies, it stores chunks and thus reduces the overall cost of storage. However, the problem is that it requires reconstructing the entire object and updating all the redundant chunks upon write requests. When erasure codes are applied to CFSs with both read and write workloads, they lead to significantly high access latencies and large bandwidth overhead between data centers.

As shown in Figure 1 (the details of our proposed scheme shown in Figure 1 will be described later), a cloud provider might have deployed eight data centers (DCs) in different locations across the United States. The problem we are addressing is: *Given a number of geographically diverse data centers, how to provide a low latency and lower cost storage mechanism for file systems in the cloud while maintaining the same consistency semantics seen in today's CFSs?*

Unfortunately, neither AllReplica nor AllCode, deployed in current systems, are best suited for this purpose, as mentioned

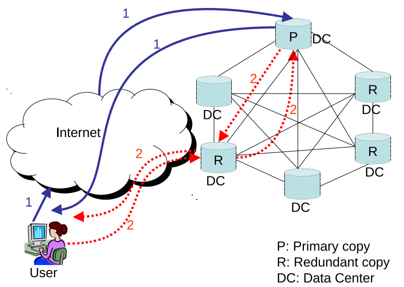


Fig. 2. Different methods to access files in a cloud file system.

before. Specifically, we want to provide *k-resiliency*, defined as data is accessible even in the event of simultaneous failures of k different DCs (data centers). Note that a single failure in our model refers to when a DC is completely inaccessible to the outside world. In reality, data stored in a particular DC is internally replicated (on different networks in the same DC) to tolerate equipment failures. A failure in our model could be due to either a network unreachability, or some other set of events that make all copies of the data in a DC inaccessible. In addition, we would like to provide an access latency close to that of the AllReplica scheme, and storage effectiveness close to that of the AllCode scheme.

A Cloud File System (CFS) workload is particularly challenging to cope with because of the consistency requirements of the file system in supporting applications such as users' desktop in the cloud, network file systems, email servers, document sharing portals, etc. These applications not only store data, but they also edit stored data and expect consistent response irrespective of where and when the data is accessed. Current CFSs [11] provide "replication-on-close" semantics, where all writes are kept at the local copy only, until the file is closed at which point replication occurs. We want to continue to provide the same consistency guarantees, but at a lower cost, lower latency and higher resiliency.

In order to maintain this tighter consistency guarantees without employing expensive distributed consensus algorithms, we use the model followed by currently deployed file systems [11], where all writes are redirected to a "primary" replica for a file. Figure 2 shows how data is accessed in today's CFS with read and write operations. Users could be directed to the primary copy (P) of the data item (solid path 1), or could be directed to the nearest replica (R) of the item (dotted path 2), if it exists. But, due to the consistency semantics needed for a highly-available CFS, all access to data needs to go through a consistent primary copy even if a replica is closer to the user (as shown in dotted path 2). This primary can be decided either at the granularity of a file or a user.

A. Our approach

In this paper, we evaluated several potential schemes for CFSs and proposed CAROM, Cache A Replica On Modification. As shown in Figure 1, CAROM combines the design of AllReplica and AllCode in a unique and novel way. In CAROM, when a file is stored, it is divided into m data chunks, and k redundant chunks are calculated using a erasure code, just as in the AllCode scheme. The $m+k$ chunks are stored in $m+k$ different data centers which are referred to as *backup data centers* in this paper. When a write operation is issued to the file, if the requested data is not cached, it is reconstructed by contacting any m backup data centers and then cached in the primary data center designed for the file until it is replaced out of cache; otherwise, if the file is cached, read and write operations are applied to the cached copy directly without the need to reconstruct the file.

CAROM is motivated by findings from prior work which concluded that reads and writes exhibit temporal locality [6], [12], [13] in file systems. Therefore the cached copy of a file covers a significant number of subsequent reads and writes, behaving like AllReplica. We explore various parameters involved in the design of this scheme, such as the size of the cache in each data center and the effect of caching on monetary cost, in more detail in later sections.

B. Summary of our contributions

In this paper, we show that CAROM outperforms replication as well as erasure coding solutions to provide k-resiliency at lower costs and latency. Our key contributions are:

- We identify the problem of providing low-cost, low-latency, k-resiliency with tight consistency semantics in cloud file systems.
- We propose CAROM, Cache A Replica On Modification, a bandwidth-, storage-, and latency-efficient resiliency scheme for cloud file systems.
- We analyze a trace from a deployed network file system obtained from two large-scale, enterprise-class file servers containing more than 35TB of data with nearly 4 million files. We use this trace to evaluate the performance of CAROM and compare it with a number of other schemes.
- Using trace-driven experiments, we show that while maintaining low access latencies close to replication based schemes, CAROM outperforms currently known replication schemes in storage cost by up to 60%, and it outperforms erasure code based schemes in bandwidth cost by up to 43%.

II. Background: Erasure coded storage

In a erasure coded storage system, a system of N disks is partitioned into m disks that hold data and k disks that hold coding information. The coding information is calculated from the data using an erasure code. An erasure code generally has two properties. First, it must be Maximum Distance Separable (MDS), which means if any k of the N disks fail, their contents can be recomputed from the m surviving disks. Once the disks are restored, the failed redundant disks may be recalculated. Second, it must by systematic, which means that the m data disks hold unencoded data. There are many MDS erasure codes that apply to storage system. Reed-Solomon codes are defined for all values of m and k and is one of the most well-known erasure codes [14].

Failure model of such a system is that of an erasure, which means when a device fails, it shuts down, and the system recognizes this shutting down. This is in contrast to an error, in which a device failure is manifested by storing and retrieving incorrect values.

It is worth noting that the computational overhead of erasure codes is negligible compared to the overhead of reading, writing or sending data. When writing data, the dominant factor is writing the erasure-coded chunks to disks, not calculating the codes. Similarly, when reading data, the dominant factor is reading the chunks from disks rather than decoding.

Though the schemes we present in this paper are independent of the specific erasure code used, we employ a general Reed-Solomon code in our evaluations. When a Reed-Solomon code is applied to a CFS across multiple data centers as shown in Figure 1, a data item is divided into multiple data chunks and redundant chunks are calculated accordingly, and each chunk is stored in a different data center.

III. Design overview

A cloud storage provider operates N data centers (DCs) across geographically diverse locations. A cloud provider's SLA will state that data will be available despite k location failures, where we use the term failure to indicate that *the copy within the data center is unreachable*. Recall that for an ideal enterprise-grade CFS, $k > 1$. We explore how we can meet this goal of k-resiliency, while reducing the cost of maintaining these copies, as well as the latency of accessing information in these copies.

A. Design considerations

a) Consistency: A key issue that needs to be considered in a CFS is the type of consistency. As described in Section I, we seek to achieve the same level of consistency provided in current cloud services, which is a *replication-on-close* consistency model. Here, as soon as a file is closed after a write, the next file open request will see the updated version of the file. In a cloud environment, this requires that the written parts of the file be updated before any other new file open requests are processed. This is very hard to accomplish in a distributed environment without incurring large delays. However, in our model, we assume that all requests are sent to a primary DC that is responsible for that file. Therefore, all file operations are essentially serialized via the primary DC, thus assuring such consistency can be achieved in the normal course of operation.

In case of a failure between a file-close operation and a file-update operation, the written data is inaccessible. This behavior, which is present in current schemes, such as AllReplica and AllCode, can be avoided by using a journaling file system [15] that writes updates to a different DC. Given that the amount of written data at any given time is very small (as shown later), we can safely assume that such a file system can indeed be used in our model without compromising latency or storage any more than it would to an AllReplica or AllCode based CFS.

b) Data accesses: Analysis of two sets of data traces collected in [16], [17] shows that in file systems, bulk of data is not accessed at all, or is accessed after very long intervals. An analysis of the file system traces used in [16] shows that file sizes do not have an impact on whether a file is written after the initial store. This can be seen in Figure 3, where we plot the cumulative fraction of the file system that all files and unwritten files occupy, as a function of file sizes, from the data in [16]. Regardless of the file sizes, we observe that around two-third of the files stored are untouched (i.e., unwritten). This observation suggests it is not always necessary to store multiple copies of a file in its uncoded form, considering the high storage cost and infrequent writes.

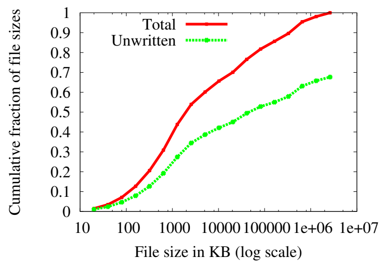


Fig. 3. Cumulative fraction of file system that total and unwritten files occupy vs. file sizes. Around two-third of stored files are not written.

In this paper, we use the traces from [17] to evaluate the schemes for CFSs. It is important to note that these traces only contain files that are accessed, and not the entire file-system. In reality, the savings of our proposed CAROM will be even larger over AllReplica than what we will see later in our experiments.

B. CAROM: Cache A Replica On Modification

Based on the design aspects presented, we now outline our strategy for providing low cost, low latency accesses along with resiliency in a CFS. We name this strategy *Cache A Replica On Modification* (CAROM). CAROM uses a combination of caching and erasure coding to accomplish our goals. We use erasure coding to save on storage, while we use caching to improve file access latency. Our contribution lies in our ability to merge these two policies effectively for any file-system regardless of its size and access patterns.

1) Motivation: CAROM is motivated by the observation that a large fraction of files remain unwritten in a file system, as shown in Figure 3. This means that erasure coding is acceptable because bandwidth-intensive reconstructions do not happen frequently. Therefore, in CAROM, instead of storing multiple replicates of a file as in AllReplica, we store erasure coded chunks to reduce storage cost. We duplicate AllCode scheme in this sense. CAROM is also motivated by observations from prior work which showed that file accesses are temporally related in file systems [6], [12], [13]. Thus in CAROM, in addition to the chunks, we occasionally cache a whole file (upon write requests) or partial file (upon read requests) to serve subsequent read and write requests, in order to reduce bandwidth costs incurred from frequent reconstructions of the file.

Since different coding algorithms have their pros and cons, we focus more on the architecture here without tying ourselves to a specific coding algorithm. We assume a generic $(m+k, m)$ systematic Reed-Solomon erasure code is used, as in AllCode scheme. Each of these $(m + k)$ chunks are stored on different nodes, which are data centers in our case, and any m chunks can reconstruct the original data. [2]

[2]Although any m chunks can reconstruct the original data, we generally prefer the m unencoded data chunks to reduce computational overhead. Other strategies could be to choose the closest m data centers, or we can select more than m data centers and choose the first m arrivals, as in [18], to reduce latency.

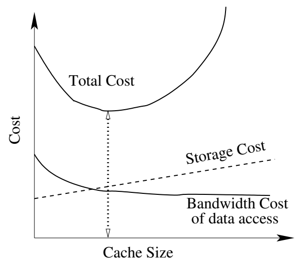


Fig. 4. Costs vs. cache size. Bandwidth cost is expected to decrease with the increase of cache size, while storage cost increases as cache size increases. Therefore, at some cache size, the total cost is minimized.

2) Caching: CAROM uses a block-level cache at each data center. "Cache" here refers to local file accesses that can be in a hard-disk or even DRAM at a local DC. Each DC acts as the primary DC for a set of files in the CFS. Whenever such files are being accessed, these accesses are sent to this primary DC, which retrieves/updates the referred blocks. Any block accessed for the first time is immediately cached. This is motivated by the findings in [17] which states that in a typical file system, the same block is often accessed in a very short span of time. Therefore caching will reduce access latency of successive operations on the same data block.

Recall that we store all files using a systematic erasure code. For a read request, we retrieve requested data blocks from unencoded data chunks and cache them in the primary DC. If a subsequent read request arrives for those cached blocks, then we use the cached data to serve this request, without having to retrieve from the data chunks.

Whenever a write operation is performed, we perform two operations in parallel: first, the written blocks are stored in the cache of the primary DC and the write is acknowledged to the user. Second, we reconstruct the entire file from any m out of $m + k$ chunks and update it in cache. Once a file-close operation is performed, the redundant chunks are recalculated using Reed-Solomon erasure code and the updated chunks are written across the other DCs. However, *we still keep the entire file in cache even after the update is complete*. This is because the same file can be accessed again.

We use the LRU (Least Recently Used) cache replacement algorithm in CAROM to evict old blocks from cache whenever the cache is full. The effectiveness of this scheme depends on two factors, the cache size, and the amount of data accessed in any given time window. We will explain in details below.

3) Cache size adaptation: While any amount of cache can definitely help in reducing access latency, the right size is important to reduce storage costs. Clearly, at one extreme we can cache the entire file system at each DC, which leads to huge storage costs. At the other extreme, keeping a very small cache leads to higher bandwidth costs incurred in fetching content from backup DCs to the primary DC. This behavior is illustrated in Figure 4, where the total cost curve is most likely a convex function. Please note that this figure is an illustration figure and it is not based on real data. Please refer to Figure 8 in section V-E1 for real data plots. The optimal cache size that minimizes the total cost is somewhere in the middle of these two extremes. Given that bandwidth pricing is structured to discourage its overuse, one can surmise that attempting to minimize the cost of bandwidth transfers will also maximize the effectiveness of caching. Our goal is to balance the trade-off between the cache size and its effectiveness.

A simple heuristic is: based on the needs and access patterns of users, find an acceptable cache size that is large enough but does not cost much in storage costs, and use it as the default for a CFS. However, this requires careful fine-tuning given that access patterns can change everyday as well as evolve over time.

We therefore, propose an elegant adaptation method that takes advantage of the potential convex nature of the total cost. An important aspect of this adaptation algorithm is that it constantly probes for the optimal cache size that minimizes cost. It is described in Algorithm 1 below, where δ is initialized to 1 GB and T is set to 1 day in our evaluation, while both parameters can be configured by cloud operators.

Algorithm 1 Adaptive Cache Sizing in CAROM

```
1:  $\delta = 1$ GB, oldCacheSize = 0, currentCacheSize = $\delta$
2:  $T = 1$ day, $\tau = 0$, currentTime $t = 0$
3:  At time $0, T, 2T, 3T, \ldots$, run Cache_Adapt()
4:
5:  function Cache_Adapt ()
6:  for Every file access at $t \in [\tau, \tau + T)$ do
7:      Compute Storage and Transfer cost with currentCacheSize
8:      Compute Storage and Transfer cost with oldCacheSize
9:  end for
10: S = currentCacheSize
11: if cost(currentCacheSize) $>$ cost(oldCacheSize) then
12:     if currentCacheSize $>$ oldCacheSize then
13:         Reduce currentCacheSize by $\delta$
14:     else
15:         Increase currentCacheSize by $\delta$
16:     end if
17: else
18:     if currentCacheSize $>$ oldCacheSize then
19:         Increase currentCacheSize by $\delta$
20:     else
21:         Reduce currentCacheSize by $\delta$
22:     end if
23: end if
24: oldCacheSize = S
25: $\tau = \tau + T$
```

As is fairly evident from Lines 11-23 of this pseudo-code, we try to seek and then stay within 1GB of the optimal cache size, assuming the costs are similar over consecutive days. Whenever the costs vary a lot over successive days, the algorithm will try to adapt as quickly as it can. The cloud operator can run this algorithm more frequently once large changes in cost are detected. In our evaluation, however, the access patterns do not vary so wildly to necessitate such dynamic reconfiguration of update intervals.

C. Resilient schemes for cloud file systems

In this paper, we evaluated existing schemes AllReplica and AllCode, our proposed scheme CAROM, and another scheme we named Hybrid. Hybrid is also based on erasure codes. The

Scheme	# of primary copy	# of backups
AllReplica	1 (full copy)	k (full copies)
Hybrid	1 (full copy))	$m+k$ (chunks)
AllCode	0	$m+k$ (chunks)
CAROM	0 (or 1 full copy if cached)	$m+k$ (chunks)

TABLE I
SUMMARY OF SCHEMES EVALUATED.

difference between Hybrid and CAROM is that in Hybrid, instead of caching a copy of a whole file temporarily (as in CAROM upon writes), a copy of the file is kept permanently in its primary DC. We compare CAROM with Hybrid to show that caching in CAROM is effective in reducing storage cost without sacrificing much on bandwidth cost, as shown later in Section V.

Table I summarizes the different k-resiliency schemes described in this paper for a quick reference. Next we describe how each scheme handles basic read and write operations in a CFS.

AllReplica: In this scheme, given any value of k, we store $k+1$ full copies of each file in the system. One copy is stored in a primary DC and k other copies are stored in k backup DCs. The primary DC serves read and write requests directly. Upon write operations, k backup DCs are updated on file-close.

AllCode: In this scheme, all files are split using a $(m+k, m)$ erasure code, and $m+k$ chunks are stored in $m+k$ backup DCs. Upon a read request, if a full copy of the file has already been reconstructed after file-open, the primary DC serves the read directly. Otherwise, the primary DC contacts other backup DCs to get requested data and then it serves read. For the first write operation after file-open, the primary DC contacts other $m-1$ backup DCs to reconstruct the whole file while it performs write on the file. The full copy is kept until file-close and subsequent read/write can be served directly from this copy until file closes. The primary DC updates the modified data chunks and all the redundant chunks on file-close.

Hybrid: In Hybrid, we store one complete primary copy of the file, and the backup copies are split using a $(m+k, m)$ erasure code. The basic idea here is that the primary replica with full copy of the file will, by default, answer the read/write queries from users. The chunks are mainly used to provide resiliency. In Hybrid, read and write requests are satisfied by the primary copy as in AllReplica. Upon write requests, all modified data chunks plus redundant chunks are updated on file-close as in AllCode.

CAROM: To serve a read request, if the requested data exists in the cache of its primary DC, the primary DC serves the request directly. Otherwise, it behaves the same as AllCode, except that the requested data will be cached in the primary DC. Upon a write request, if the requested file resides in the cache of the primary DC, the primary DC performs write without the need to reconstruct the whole file. Otherwise, the file is reconstructed as in AllCode while processing the write. All the modified chunks are updated on file-close also as in AllCode. However, different from AllCode, the reconstructed copy is stored in the cache even after file-close in CAROM (until file is replaced by the LRU algorithm).

IV. TRACE DETAILS

To evaluate our schemes, we use the CIFS network trace collected from two larger-scale, enterprise-class file servers deployed in the NetApp corporate headquarters. This trace was collected in 2007, and is described in full detail in [17]. One file server is deployed in the corporate data center that hosts data used by over 1000 marketing, sales, and finance employees, and the other is deployed in the engineering data center used by over 500 engineering employees [17], [19]. We refer to these workloads as Corporate (or Corp) and Engineering (or Eng), respectively. This trace is representative of a mid-sized enterprise file system, and it is collected over a period of two months for Corporate and three months for Engineering. While the trace had complete details of I/O activities, as collected at Ethernet layer, we parsed the trace and extracted the following details of I/O operations: <timestamp, I/O command, I/O size, File name>.

	Corporate	Engineering
Trace start date	Sep 20, 2007	Aug 10, 2007
Trace end date	Nov 23, 2007	Nov 14, 2007
Days	65	97
Total number of files (million)	1.51	2.29
Total size of files (TB)	1.58	33.49
Total number of reads (million)	30.46	48.82
Total number of writes (million)	9.38	22.65

TABLE II
HIGH LEVEL STATISTICS OF NETAPP TRACE USED FOR OUR EVALUATION.

Table II shows the statistics associated with these workloads. We processed a total of 3.8 million unique files representing over 35 TB of data. Engineering traces were collected for 50% longer duration than those of Corporate, and the number of files reflects this. The total size of the Engineering files is significantly larger (more than 30 times), which is due to the larger average sizes of the files. The number of reads per day, however, is roughly equal to 0.5 million in both traces. In the case of Corporate, this corresponds to about 500 reads per user per day, which is about 2MB of reads on an average (using 4KB block size). The number for Engineering roughly averages to around 4MB of reads per user. In terms of writes, Engineering traces have more data written per user, around 2 MB per user per day. The key observation is that an erasure-coded scheme, such as AllCode or CAROM, can easily handle the processing of this level of data read/written with negligible computational cost.

Given the low average utilization of the file systems seen in this representative trace set, consolidating the file systems' users and moving to cloud has significant financial incentives to the companies, further supporting the premise of this paper.

While the scale of cloud is growing, a larger cloud environment can be treated as consisting of multiple instances of such a trace. We believe this trace is quite representative and our results will apply evenly across the growth in number of files, file sizes, number of accesses, and CFS access distributions.

We used the parsed trace to understand the duration for which the files are open and how far apart in time two open requests to the same file are. This helps us to understand the performance of the replicate-on-close model. Since file open durations were already plotted in [17], we only plot the duration between opens in this paper. From [17], files are

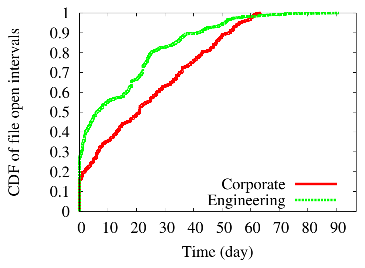


Fig. 5. Cumulative distribution of open intervals for the entire trace. Time between two consecutive opens of the same file is generally large.

opened for very short intervals. In fact, most files are for less than 50 seconds, while nearly 80% of the files are opened for about 1 second.

Figure 5 shows the cumulative distribution of the time intervals between two consecutive opens of same files. Clearly, time intervals between consecutive file opens of same files are generally quite large. Only 18% of the total file opens in Corporate and 33% of the opens in Engineering are within one day of previous open. Intuitively, this large interval may limit the effectiveness of caching in CAROM. However, we will show in Section V that CAROM still leads to remarkable improvement in the financial costs.

It is important to point out two unique aspects of this trace: (a) many reads initiated by clients are served by client-side caching, and are not captured by this trace, and (b) this trace only contains data of files that were accessed, and does not tell us anything about the fraction of the file-system that was not accessed at all during the measurement period. The former aspect reduces the effectiveness of caching somewhat, while the latter aspect significantly reduces the storage cost of AllReplica scheme. Thus, the performance improvements of CAROM over competing schemes shown in the next section are a worst-case scenario for CAROM. In practice, we expect the savings to be even larger than what are reported here.

V. Evaluation

A. Environment and metrics

Our main goal is to compare CAROM resiliency scheme with AllReplica, Hybrid and AllCode in terms of: (a) storage cost in storing the file systems, (b) total bandwidth consumed in storing, reading, and writing the file systems, and (c) total monetary cost of migrating and operating these file systems on cloud.

In the following experiments, we vary k from 1 to 3 in order to understand the performance of the four schemes with different resiliency. In each of these experiments, unless otherwise stated, the total number of data centers N is fixed to 10, and the number of chunks that a file is divided into, m, is set to 5.

When a file is opened for the first time, depending on the resiliency scheme, a number of data centers are chosen for the file to store replicas and/or coded chunks. Each time a file is closed after performing writes, these replicas and/or chunks are updated to keep data consistent. We compare the four schemes described in Section III-C in terms of storage-time, bandwidth and monetary cost.

B. Storage-time

Assume we have a file trace containing a set of files F, and the trace has a start time of u and finish time v. Suppose the total storage of a file f is $s_f(t)$ at time t. We define storage-time of the trace to be:

$$\sum_{f \in F} \sum_{t=u}^{v} s_f(t)$$

This metric is useful because it can be directly translated to monetary cost based on the storage cost per-GB-per-hour, as priced by Amazon and other cloud providers.

Every time a file is closed after a write access, we update the storage-time of the file. Figure 6 shows the storage-time over the entire trace period for Corporate and Engineering, respectively. The clear winners here are AllCode and CAROM. AllCode essentially has only $(1 + k/m)$ overhead for storing the files. The performance of CAROM is very close to that of AllCode. CAROM incurs an extra storage cost for caches. However, since cache size is small compared to the file system size (as shown later in Section V-E), this extra cost is insignificant. AllReplica incurs an overhead of k times data stored. This can be seen from the linear increase with k for AllReplica, while AllCode and CAROM grow much slower with increasing k. Hybrid outperforms AllReplica for $k > 1$, since in Hybrid, always one copy and $m+k$ chunks are stored, instead of k copies as in AllReplica. For $k = 1$, Hybrid has an extra overhead of k chunks for each file stored compared to AllReplica.

For $k = 2$, CAROM saves more than 50% (53%) in storage-time compared to AllReplica, and more than 40% (42%) compared to Hybrid. The percentage of savings compared to AllReplica is as high as 60% for $k = 3$.

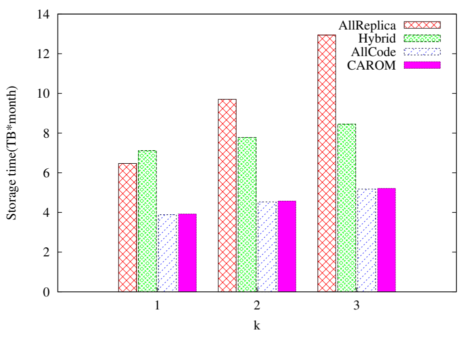
C. Bandwidth cost

In the default operational scenario, a file can be accessed at any given time for either a read, or a write. A file access operation is associated with a cost, which is expressed as network traffic in a distributed storage system. We evaluate the cost of such file accesses in terms of the total amount of data read or written to serve all file access requests.

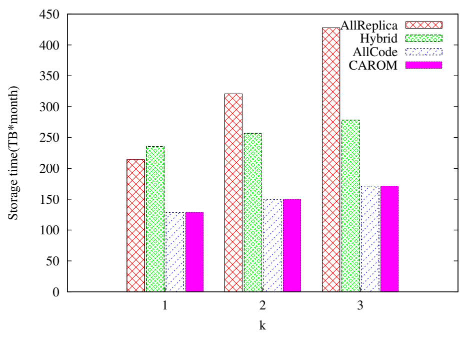
Under this scenario, a read access is served from a primary replica of the data item in AllReplica and Hybrid, while in AllCode, the primary DC has to get the requested data from its fragments in backup DCs [3]. CAROM behaves similar to AllCode unless the requested data is already cached. If the requested data is already cached in its primary DC, the read request is served directly from the primary, saving the cost of getting data from chunks as in AllCode.

Similarly, a write request is served from a primary replica in AllReplica and Hybrid. While in AllCode, the whole file needs to be reconstructed upon the first write request after file-open, and the whole file is kept until file-close. CAROM outperforms AllCode since once a file is reconstructed, it might be kept in cache even after the file is closed. Thus, it saves the re-assemble cost if the file is already in cache. Figure 7 shows the total data transferred as a result of file accesses summarized in Table II for both Corporate and Engineering workloads.

[3]This is true as long as a systematic code is being used.



(a) Corporate



(b) Engineering

Fig. 6. Storage-time (in TB*month) during the whole trace periods (In CAROM, cache size for Corp and Eng is 2GB and 5GB, respectively).

An interesting observation from the graph is the sharp increase in the data transfer overhead of AllReplica, with the increase of value k. For $k = 1$, AllReplica outperforms CAROM, but for values of $k = 2$ and $k = 3$, that are currently in use by major cloud providers, the overhead of AllReplica exceeds that of CAROM. When we compare CAROM with AllCode, for $k = 2$, we see 32% and 40% savings in the amount of data transferred for Corp and Eng, respectively.

It is worth noting that the bandwidth cost of CAROM is only slightly higher than that of Hybrid, which indicates that most read/write requests are served by cached copies. This proves the effectiveness of caching in CAROM.

D. Monetary cost

Furthermore, we compared the four schemes in terms of their monetary costs. Specifically, we compare the cost in US dollar of storing and keeping the files in a cloud file system under different schemes, using the prices of resources from Amazon EBS [20] as on July 13, 2012. Table III shows a summary of these prices.

Storage pricing	\$0.10 per GB-month
Request pricing	\$0.10 per 1 million I/O requests
Data transfer pricing	\$0.000 per GB for data transfer in \$0.120 per GB for data transfer out

TABLE III
AMAZON EBS PRICING OF RESOURCES AS OF JULY 13, 2012 [20]

The storage cost of EBS has two components: (a) the cost of storing the data for a month in terms of GBs, and (b) the cost of I/O operation, and the cost of the bandwidth while transferring data in and out of the data centers. We use these prices to simplify the comparison. Note, however, that these storage prices are the prices charged by Amazon to the customers. In reality, the storage cost incurred by data center operators will be lower.

Table IV summarizes the monetary cost of providing k-resiliency CFS for our trace with various values of k and four different schemes, where the column for CAROM is highlighted in gray. Clearly, AllCode and CAROM outperform AllReplica and Hybrid in terms of storage cost. While CAROM incurs slightly larger storage cost than AllCode, it has lower bandwidth cost. Overall, CAROM is the best scheme in term of total monetary cost. CAROM saves up to 59% over AllReplica, 41% over Hybrid and 10% over AllCode, respectively. We observe that CAROM works across all file-system sizes, while AllCode performs as good as CAROM for very large file systems where storage dominates read-writes. With more and more file systems moving to cloud, we expect that the savings of CAROM in bandwidth cost will be even more for read/write intensive workloads.

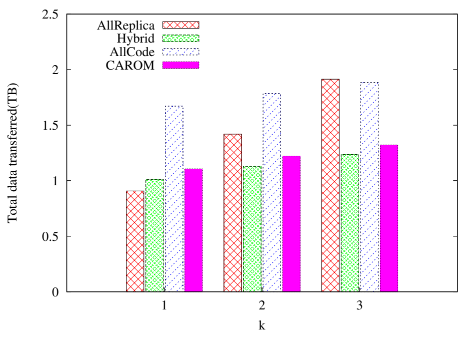
Trace	k	Type	AllReplica	Hybrid	AllCode	CAROM
Corp	1	Storage	663	728	398	402
		Transfer	116	128	209	140
	2	Storage	994	797	464	468
		Transfer	178	143	223	154
	3	Storage	1325	865	530	534
		Transfer	239	158	236	166
Eng	1	Storage	21904	24094	13143	13159
		Transfer	268	317	576	332
	2	Storage	32856	26289	15333	15349
		Transfer	412	365	622	380
	3	Storage	43808	28483	17524	17540
		Transfer	557	412	670	429

TABLE IV
MONETARY COST IN \$ FOR STORING AND ACCESSING THE FILE SYSTEMS OF THE TRACES ($m = 5$, CACHE SIZE IN CAROM IS 2GB FOR CORP AND 5G FOR ENG).

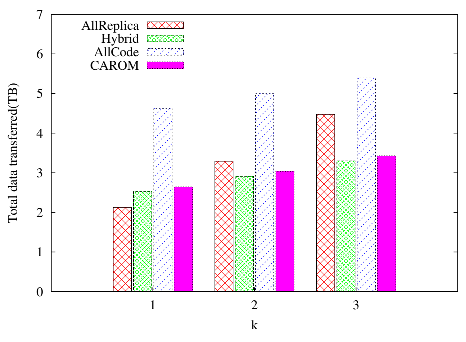
E. Discussion

1) Cache size vs. monetary cost: All of the results reported so far assume that CAROM uses a cache size of 2GB for Corp and 5GB for Eng. We now present the rationale behind these choices. We evaluated the overall cost of CAROM by sampling over different cache sizes at each DC. The resultant plot is shown in Figure 8 for both traces. It can be seen that the cost is minimized when the cache size is 2GB and 5GB for the Corp and Eng respectively. This is not necessarily the optimum, since this value is set to be the same at all DCs. Each DC might have a different optimal cache size, depending on the access patterns of its files. However, the numbers shown here are an upper bound on the optimum cost. It also confirms our estimate that the cost curve is convex in nature as a function of cache size (Figure 4).

2) Performance of cache adaptation: We evaluated our cache adaptation algorithm presented in Algorithm 1, and compared the costs derived from this algorithm to those from the optimal cache above. Note that the adaptation algorithm lets the cache at each DC adapt independently. We present the cost in Table V along with the average cache size in brackets for each trace. We can see that the Adaptive Caching scheme works very well in reaching the optimal cost.

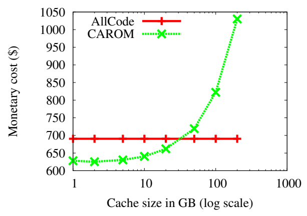


(a) Corporate

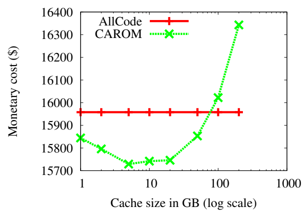


(b) Engineering

Fig. 7. The total amount of data read or written in response to file access requests (reads and writes) (In CAROM, cache size for Corp and Eng is 2GB and 5GB, respectively).



(a) Corporate



(b) Engineering

Fig. 8. Overall monetary cost of Corp and Eng for different cache sizes (k=2). CAROM can save up to 10% in monetary cost compared to AllCode.

	Corporate	Engineering
Optimal Caching	\$625 (2 GB)	\$15729 (5 GB)
Adaptive Caching	\$628 (1.76 GB)	\$15811 (1.95 GB)

TABLE V
MONETARY COST OF OPTIMAL CACHING AND ADAPTIVE CACHING.

3) Access latency: Table VI shows the cache hit ratios of CAROM for read and write accesses. In each open-close cycle of a file, we only count cache hits and misses for the read requests of the file before any write request, and for the first write request to a data block. The subsequent reads/writes will be served from the copy in the primary DC. We can see that the hit ratios for both Corp and Eng are very high, while those of Corp are even higher.

We convert this data into the access latency of a data block at the primary DC. A cache hit implies that data is available at the local disk in the primary, while a cache miss implies that data has to be fetched from a backup DC elsewhere, which takes longer. In AllCode scheme, all data has to be fetched from the backup DCs, while in AllReplica data is always available at the local disk. Remote disk access latencies can range anywhere from 20 milliseconds (for nearby DCs) to up to 100 milliseconds (for DCs across continents) among different DCs belonging to the same CSP [21]. We use the notation (A, B) to denote local disk access latency (A) and remote disk access latency (B) in units of milli-seconds. We present the access latencies for read operations in Table VI with different values of (A, B). Note that a well-designed implementation of AllCode and CAROM can parallelize write and data fetch to provide the same write access latency as that of AllReplica. From the table, we can see that CAROM provides read access latencies that are close to those of AllReplica (local disk access latencies), rather than those of AllCode (remote disk access latencies).

		Corporate	Engineering
Write	cache hit ratio	91.32%	83.40%
Read	cache hit ratio	96.43%	87.96%
	latency (10,20)	10.3ms	11.2ms
	latency (10,50)	11.4ms	14.8ms
	latency (10,100)	13.2ms	20.8ms

TABLE VI
CACHE HIT RATIO AND LATENCY FOR READ ACCESSES (K=2, CACHE SIZE FOR CORP AND ENG IS 2GB AND 5GB RESPECTIVELY).

4) Fault Tolerance: Though we do not have access to real traces with failure information, we expect CAROM to outperform AllCode. Erasure codes, especially MDS codes such as Reed-Solomon codes, require to retrieve m chunks to repair one single chunk. While in CAROM, our preliminary results show that in case of transient failures, the cached copy could serve a significant percentage of read or write requests without the need to retrieve the chunks, thus reducing bandwidth cost incurred by failures in the data centers.

VI. Related work

The simplest erasure code is the parity code, which protects against a single failure. There are other well-known erasure codes such as the Reed-Solomon code, part of a large class of optimal erasure codes called MDS codes and LDPC codes [14]. These can be used to design k-resilient coding schemes suited to CFS applications. A code minimizing the repair bandwidth when a chunk gets erased is called a minimum-bandwidth regenerating (MBR) code [22]. In [23], the construction of MBR codes with arbitrary parameters has been proposed. In [24], the authors introduce a new family of regenerating codes for the storage cloud, namely DRESS codes, which is highly optimized for recovery. In [7], a new class of rotated Reed-Solomon codes is defined which performs degraded reads more efficiently. While they describe algorithms to do the coding, the architecture of the system and the replication policies will have to be the one that we propose here in order to conserve bandwidth. In that sense, our work is complementary to these kind of work. More efficient erasure codes might reduce the bandwidth overhead further, but our contributions are independent of, and can co-exist with, the recovery schemes used by any coding algorithm.

There is wide-spread literature about the use of resiliency schemes in P2P storage systems. In [8], the authors quantified the resiliency and durability gained by erasure codes over replication in P2P systems and showed that erasure coded system can be many orders of magnitude better than replicated system in terms of storage, bandwidth and disk seeks. In [10], the authors modeled the behavior of a P2P system as a Markov Chain and concluded that for a P2P system with low availability of peers, replication is better than using erasure code. On the contrary, in [9], the authors presented that coding wins for low peer availability, while replication wins when peer availability is high. In [9], the authors consider a read-only workload with a cached copy of the data item plus multiple erasure-coded chunks for each data object. The paper makes several points against the use of coding which might be true for a read-only workload in a P2P system, but it is not true in a cloud-based file system as the workload includes both reads and writes.

The authors in [25] proposed another class of erasure codes, called Hierarchical Codes, aiming to reduce the network traffic caused by maintenance. While the repair cost is on average smaller compared to erasure codes, the worst-case repair bandwidth is the same as that of conventional erasure codes. In [26], the authors implemented Random Linear Regenerating Codes and found that these can result in a significant reduction of the communication overhead at the expense of storage and computation cost. However, their low encoding/decoding rates make them impractical in situations where data items have to be written. Overall, the existing P2P literature is mainly focused on read-only workload, which is the main reason existing P2P solutions will not work for cloud file systems.

In [27], the authors look into a setting where either only M-way replication is used or only erasure coding is used. Then they explore different trade-offs with different data center applications. In [18], the authors target append-only distributed file systems and implement erasure coding as a complementary technique to full data replication in Windows Azure Storage. Our focus is read-write file systems and we implement erasure coding with caching (caching decisions are based on file read/write requests).

Several recent measurement studies have studied file systems and networked file systems in detail. In [16], the authors conducted a five year study of a large amount of Windows PC file systems. They observed that file sizes are growing, and around 33% of the files in these file systems are written at least once. A recent measurement study [17] of a network file system has made similar observations. They note that the write-read ratio observed is higher than prior studies, due to more caching on clients that absorbs the reads.

VII. Conclusion

With an increase in the benefits of cloud computing, more and more users and enterprises are moving to the cloud, pushing critical systems such as file systems and applications atop the cloud. These applications not only demand high availability, but also require the consistency semantics provided by today's file systems running on end-users' machines. In this paper we propose CAROM for cloud based file systems which novelly combines the deployed replication schemes and erasure coded schemes. CAROM is as responsive, consistent, and resilient as the deployed replication schemes, and yet is much more efficient in terms of storage and bandwidth costs. We evaluate CAROM and compare it with existing schemes using real-world file system traces containing more than 35TB of data with nearly 4 million files. Our results show that CAROM significantly outperforms existing schemes in efficiently using the cloud resources to support k-resiliency and also leads to significant monetary benefits to the provider of such an enterprise-grade file system.

References

[1] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SOSP*, 2003.

[2] Amazon, "Amazon simple storage service faqs," http://aws.amazon.com/s3/faqs/.

[3] Gluster, "Gluster virtual storage appliance," http://www.gluster.com/products/public-cloud/rightscale/.

[4] R. Miller, "Major data center outages of 2009," http://www.datacenterknowledge.com/archives/2009/12/16/major-data-center-outages-of-2009/ .

[5] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, "Overdriver: Handling memory overload in an oversubscribed cloud," in *Proceedings Virtual Execution Environments (VEE)*, 2011.

[6] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron, "Everest: Scaling down peak loads through i/o off-loading," in *OSDI*, 2008.

[7] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing i/o for recovery and degraded reads," in *FAST*, 2012.

[8] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *IPTPS*, 2002.

[9] R. Rodrigues and B. Liskov, "High availability in dhts: Erasure coding vs. replication," in *IPTPS*, 2005.

[10] G. Utard and A. Vernois, "Data durability in peer to peer storage systems," in *CGRID*, 2004.

[11] XtreemFS, "A cloud file system," http://www.xtreemfs.org.

[12] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: practical power management for enterprise storage," in *FAST*, 2008.

[13] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches," in *FAST*, 2010.

[14] T. Cover and J. Thomas, "Elements of information theory," in *Wiley Series in Telecommunications*, 2006.

[15] M. T. Jones, "Anatomy of linux journaling file systems," in *IBM DeveloperWorks*, 2008.

[16] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," in *FAST*, 2007.

[17] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *In Proceedings of Annual Technical Conference*, 2008.

[18] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *USENIX ATC*, 2012.

[19] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz, "Design implications for enterprise storage systems via multi-dimensional trace analysis," in *ACM SOSP*, 2011.

[20] "Amazon ec2 pricing," http://aws.amazon.com/ec2/pricing/.

[21] Verizon, "Internet protocol latency statistics," http://www.verizonbusiness.com/about/network/latency/.

[22] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," in *arXiv:1004.4438v1*, April 2010.

[23] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," 2012.

[24] S. Pawar, N. Noorshams, S. E. Rouayheb, and K. Ramchandran, "Dress codes for the storage cloud: Simple randomized constructions," in *Symposium on Information Theory (ISIT)*, 2011.

[25] A. Duminuco and E. Biersack, "Hierarchical codes: How to make erasure codes attractive for peer-to-peer storage systems," in *Proceedings of Peer-to-Peer Computing (P2P)*, 2008.

[26] A. Duminuco, "A practical study of regenrating codes for p2p backup systems," in *Proceedings of ICDCS*, 2009.

[27] Z. Zhang, A. Deshp, X. Ma, E. Thereska, and D. Narayanan, "Does erasure coding have a role to play in my data center?" in *Microsoft Research Technical Report MSR-TR-2010-52*, 2010.