TEA: A Traffic-efficient Erasure-coded Archival Scheme for In-memory Stores

Bin Xu
binxu@hust.edu.cn
Huazhong University of Sci.& Tech.
Wuhan, Hubei, China

Jianzhong Huang*
Qiang Cao*
Huazhong University of Sci.& Tech.
Wuhan, Hubei, China

Xiao Qin
xqin@auburn.edu
Auburn University
Auburn, AL 36849, USA

ABSTRACT

To achieve good trade-off between access performance and memory efficiency, it is appropriate to adopt replication and erasure coding to keep popular and unpopular in-memory datasets, respectively. An issue of redundancy transition from replication to erasure coding (a.k.a., *erasure-coded archival*) should be addressed for unpopular in-memory datasets, since caching workloads exhibit long-tail distributions and most in-memory data are unpopular.

In this paper, we propose an encoding-oriented replica placement policy - *ERP* - by incorporating an interleaved declustering mechanism, and design a traffic-efficient erasure-coded archival schemes -*TEA* - for ERP-powered in-memory stores. With ERP in place, TEA embraces three salient features: (i) it alleviates cross-rack traffic raised by retrieving data-block replicas, (ii) it improves rack-level load balancing by distributing replicas via load-aware primary-rack-selection approach, and (iii) it mitigates block-relocation operations launched to sustain rack-level fault-tolerance. The empirical results show that TEA not only brings forth lower cross-rack traffic than four candidate encoding schemes, but also exhibits superb archival-throughput and rack-level-balancing performance. In particular, TEA accelerates archival throughput by at least 70.8%; and improves rack-level load-balancing by a factor of more than 1.58x relative to the four competitors.

CCS CONCEPTS

• **Information systems** → *Distributed storage*; • **Computer systems organization** → *Redundancy*.

KEYWORDS

In-memory store, Erasure encoding, Replication, Archival

ACM Reference Format:
Bin Xu, Jianzhong Huang, Qiang Cao, and Xiao Qin. 2019. TEA: A Traffic-efficient Erasure-coded Archival Scheme for In-memory Stores. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337826

*Jianzhong Huang (hjzh@hust.edu.cn) and Qiang Cao (caoqiang@hust.edu.cn) are the joint corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPP 2019, August 5–8, 2019, Kyoto, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337826

1 INTRODUCTION

1.1 Motivation

The following three aspects motivate us to delve in the development of an erasure-coded archival scheme for in-memory stores.

Aspect #1 –*low access latency in in-memory stores.* We are in an era of data-driven business world. For example, data-intensive analytics has become indispensable since enterprises want to gain insights to products, services and marketing strategies from increasing volumes of data. Commonly, a data-intensive application is supported by a cluster consisting of hundreds of nodes and petabytes of data. It is a technology trend to constitute an in-memory store upon the cluster to achieve low-latency performance. A traditional case is that Facebook leverages Memcached as a building block to construct a distributed key-value store facilitating the world's largest social network [19].

Aspect #2 –*demands for redundancy strategies.* Since volatile DRAM only maintains data while it is powered, existing in-memory stores accomplish memory-level fault-tolerance by applying replication and/or erasure coding. Replication is a simple yet effective redundancy scheme. For instance, Repcached [15] and Bigmemory [28] keep two replicas in memory among nodes. Compared to the replication, erasure coding achieves higher space efficiency defined as a ratio of user data to the combination of user data and redundancy data [12]. Unsurprisingly, space-efficient erasure codes are also adopted by in-memory stores, e.g., EC-cache [21], Cocytus [31], MemEC [30], and Ring [24].

Aspect #3 –*necessity of erasure-coded archival.* An analysis of traces collected from Facebook's Memcached deployment shows that caching workloads exhibit long-tail distributions, in which a small percentage of keys appeared in most of the requests whereas most keys repeated only a handful of times [3]. Therefore, it is not economical to employ a single redundancy scheme for the entire in-memory data (i.e., metadata, key and value in in-memory key-value stores) within the data lifetime. Nowadays, some key-value stores (e.g., Memcached in Facebook [19], Ring [24]) adopt a hybrid redundancy strategy, where replication is applied to popular data, while erasure codes are employed for unpopular data.

Generally, newly-loaded data is kept in a replication manner to support high access parallelism. Since most of the new data are infrequently accessed, it is wise to encode unpopular data replicas using erasure codes to achieve high space efficiency. We refer to such an encoding process as '*erasure-coded archival*'.

1.2 Challenges and Strategies

We face two challenges during the course of designing an erasure-coded archival scheme for in-memory stores.

Challenge 1: How to reduce cross-rack traffic while performing erasure-encoding operations? Cross-rack bandwidth is usually scarce compared to intra-rack one; ideally, cross-rack traffic should be eliminated. If data replicas are randomly distributed among multiple racks, then subsequent sequential encoding will retrieve required blocks via cross-rack transfer. Therefore, it makes sense to elaborately distribute data-block replicas.

Challenge 2: How to holistically consider locality and flexibility in stripe construction? Non-sequential striping potentially enables flexible stripe construction for unpopular blocks. To exploit locality, we have to restrict candidate members in a stripe to a small subset of unpopular data blocks and; therefore, the flexibility of stripe construction is worsened. In this paper, locality is taken into account while employing non-sequential striping.

It is arguably true that the replica placement is critical to reliability, access locality, load balancing, and the like. A typical example is the rack-aware data placement in HDFS [4]. To accomplish highly-efficient erasure encoding operations for unpopular replicas, it is expected to introduce an encoding-oriented placement for replicas, which satisfies the following three requirements. (i) Rack-level fault tolerance should be guaranteed for data-block replicas prior to encoding. (ii) The primary copies of a group of sequential data blocks should be placed to a rack; thus, blocks in the rack can construct a complete stripe during encoding. (iii) Cross-rack traffic should be minimized after encoding, because encoded blocks may be re-distributed to sustain rack-level fault tolerance.

To address the above requirements and challenges, we propose an Encoding-oriented Replica Placement policy - ***ERP***, in which an *Interleaved Declustering* mechanism [10] is introduced to distribute in-memory data-block replicas among racks. Furthermore, we elaborately design a Traffic-efficient Erasure-coded Archival scheme - ***TEA*** - for ERP-powered in-memory stores. The TEA scheme has the following three salient features. (i) It alleviates cross-rack traffic raised by retrieving required data-block replicas. (ii) It improves rack-level load balancing by distributing replicas via load-aware primary-rack-selection approach. (iii) It mitigates block-relocation operations launched to sustain rack-level and node-level fault-tolerance.

1.3 Contributions

This work makes three main contributions:

- We devise an encoding-oriented replica placement policy (i.e., ERP) for data-block replicas in in-memory stores. ERP enables subsequent erasure-encoding operations to alleviate cross-rack traffic while sustaining rack-level fault tolerance.
- We design a traffic-efficient erasure-coded archival scheme (i.e., TEA) for ERP-powered in-memory stores. TEA addresses both cross-rack-traffic and rack-level-load-balancing issues in encoding-rack-designation and encoded-block-relocation stages.
- We implement a proof-of-concept prototype, where TEA and four candidate encoding schemes are quantitatively evaluated. The experiments illustrate that the TEA scheme not only brings forth lower cross-rack traffic than the other four competitors, but also exhibits superb archival-throughput and rack-level load-balancing performance.

1.4 Roadmap

The rest of the paper is organized as follows. Section 2 outlines both erasure-coded archival techniques and schemes. ERP policy and TEA scheme are detailed in Sections 3 and 4, respectively. We present quantitative performance evaluation in Section 5. Section 6 concludes this paper.

2 RELATED WORK

2.1 Erasure-coded Archival Techniques

Replication and erasure coding are two main fault-tolerant techniques. Replication and erasure coding perform well in the aspects of access parallelism and space efficiency, respectively [26]. Apart from on-disk storage (e.g., GFS II [11], HDFS [5]), in-memory stores (e.g., Memcached in Facebook [19], Ring [24]) also adopt a hybrid redundancy strategy, in which replication is applied to newly created data whereas erasure coding is used to archive the same data once it becomes unpopular. Such a transition from replication to erasure coding is referred to as erasure-coded archival.

Among various erasure codes, Reed-Solomon (a.k.a., *RS*) codes become the most popular choices owing to the RS codes' maximum distance separable (MDS) feature as well as high level of fault tolerance [26]. RS codes accomplish parity generation by adopting simple linear combinations. Especially, as for (k+r,k) RS codes, r parity blocks $\{P_1, P_2, ..., P_r\}$ are generated by multiplying k data blocks $\{D_1, D_2, ..., D_k\}$ with a k×r *redundancy matrix*[18]. Both k data blocks and r associated parity blocks constitute a stripe, and (k+r,k) RS codes tolerate data loss of any r concurrent blocks.

Typically, the following four steps are involved in a (k+r,k) RS-coded archival operation. (i) An encoding node retrieves one replica of each of k data blocks from local and/or remote nodes; (ii) it computes r parity blocks from the k data blocks using (k+r,k) RS codes; (iii) it delivers r or r-1 parity blocks to other nodes (Note: it is of r-1 blocks when one parity block is kept by the encoding node); and (iv) only one replica is kept (i.e., not deleted) for each data block while deleting the other replicas.

To accomplish a r-node fault-tolerant distribution for in-memory stores, k data blocks and r parity blocks in a stripe are exclusively placed to the main memories of k+r nodes during the archival steps (iii) and (iv). In practice, a node may cache a data or parity block in a stripe and a data or parity block in another stripe. Usually, a data block is sealed from multiple small-sized objects in in-memory stores [31][30].

2.2 Erasure-coded Archival Schemes

Most of erasure-coded archival schemes are focused on distributed storage systems. For example, DP [14], RapidRAID [20], aHDFS [7], EAR [17], DSC [27], Sice [29], and to name just a few. Differently, our TEA scheme aims at in-memory stores.

DP is an erasure-coded data archival scheme for storage clusters, in which a chained-declustering layout is applied to organize Mirrored RAID-5 redundancy groups [14]. DP boosts data archival performance by leveraging the existence of replicas and employing a pipelined encoding process.

RapidRAID is a family of erasure codes that incorporate pipelined erasure coding to speed up archival processes [20]. Unlike RS codes,

Table I: Comparison of erasure-coded archival schemes

Scheme	System	Cross-rack traffic	Layout/Placement of data replicas	Encoding mode	Striping pattern	I/O balancing	
DP [14]	on-disk storage	×	Chained Declustering among nodes	Pipelined	Sequential	×	
RapidRAID [20]		×	place one copy on a random node	Pipelined	–	×	
aHDFS [7]		×	rack-aware placement in HDFS	Pipelined	Sequential	×	
Sice [29], DSC [27]		√	place one copy to a random rack	Centralized	Non-sequential	–	
EAR [17]		√	encoding-aware placement	Centralized	Sequential	√	random
ERP/TEA (ours)	in-memory store	√	Interleaved Declustering among racks	Centralized	Non-sequential	√	load-aware

RapidRAID codes are non-systematic. Therefore, decoding operations are required to service any request to blocks archived by RapidRAID codes, introducing extra data-access overhead.

An erasure-coded archival system called *aHDFS* was developed for HDFS [7]. A pipelined archival strategy is deployed in aHDFS to speed up the data archival process, where each merged single key-value pair is repeatedly delivered to a subsequent node's local key-value store until final parity blocks are generated.

EAR is an encoding-aware replication scheme for storage clusters using 3-way replication [17]. The primary goal of EAR is to enable later erasure encoding to eliminate cross-rack traffic and maintain rack-level fault tolerance. EAR keeps one replica of each data block in a core rack. Once a core rack has stored k distinct data blocks, the collection of k data blocks is sealed for later encoding, implying that EAR adopts sequential striping.

DSC is a dynamic stripe construction scheme applicable to the existing clustered file systems, where new coming files are stored in N-way replication [27]. DSC dynamically groups some data blocks into a stripe to avoid cross-rack traffic when transforming from N-way replication to erasure coding.

Similar to DSC, *Sice* adopts non-sequential striping to flexibly construct stripes for subsequent erasure encoding [29]. The basic idea of Sice is to construct stripes non-sequentially from a large number of blocks to reduce cross-rack traffic and co-located blocks. Sice is suitable for all existing replica-distribution methods such as the rack-aware replica placement in HDFS and the random replica placement (RRP) in HDFS-RAID.

Usually, cross-rack bandwidth is a scarce resource, because it is much lower than intra-rack bandwidth, and high cross-rack traffic can degrade front-end user performance [1][13]. Therefore, it is expected to reduce cross-rack traffic for back-end maintenance operations (e.g., erasure-coded archival). Among the aforementioned erasure-coded archival schemes, EAR, DSC, Sice and TEA take cross-rack traffic into account.

After encoding, RapidRAID, DP and aHDFS achieve node-level fault tolerance by exclusively placing k+r blocks in a stripe to k+r nodes for (k+r,k) erasure-coded schemes; whereas Sice, DSC, EAR, and TEA support both node-level and rack-level fault tolerance by further ensuring that k+r nodes are located in k+r distinct racks.

All the above-mentioned schemes accomplish erasure encoding in a parallel manner for all stripes. Different from RapidRAID, DP, and aHDFS adopting pipelined encoding for a stripe, Sice, DSC, EAR, and TEA employ centralized encoding for a stripe, where a specific node to fulfill encoding for a stripe by retrieving enough data blocks from remote nodes.

Among the above schemes, EAR and ERP are closely related, since they are focused on replica placement tailored for later encoding; and they keep the primary replica of a group of data blocks in the same rack. However, ERP is distinguished from EAR as follows. (i) To maintain load balancing, EAR randomly distribute replicas, while ERP delivers a group of primary replicas to a least-loaded rack. (ii) EAR stores the remaining replicas in other racks according to the set of maximum matching, while ERP places the secondary replicas using the interleaved declustering mechanism. (iii) EAR-powered encoding advocates for sequential striping, whereas ERP-powered encoding (i.e., TEA) applies non-sequential striping.

3 THE ERP POLICY

The ERP policy is devised to facilitate later encoding operations.

3.1 Preliminary Consideration

The ERP policy takes into account the following three factors.

Fault-tolerance: In-memory stores can achieve high availability via a single-fault-tolerant scheme since datasets are kept in memories for hours or days [3][24]. Some in-production in-memory solutions (e.g., Repcached [15], Bigmemory [28], etc.) keep two in-memory replicas among nodes. In this paper, 2-way replication is also applied to newly-loaded in-memory datasets.

Cross-rack Traffic: For each block to be kept in memories, if random-replication-like strategies are applied to determine the racks for block replicas, then it is unlikely for a rack to house at least one replica of k sequential blocks in a stripe. Thus, if sequential-striping is adopted and an encoding rack is arbitrarily designated for a stripe, then cross-rack traffic will occur.

Load Balancing: Random Replication or *RR* allows CFS to achieve good load balancing by distributing data over a uniformly random set of nodes [8]. However, the bandwidth factor is not covered in the DataNode selection for block placement. Blocks in a DataNode with low bandwidth may be accessed frequently, thereby leading to imbalanced node-level access load [23].

ERP places one replica of a collection of newly-loaded blocks to a specific rack, where the collection of replicas in the rack are assumed to be grouped into erasure-coded stripes. Such a replica placement helps to eliminate cross-rack traffic incurred by erasure encoding. Additionally, ERP selects primary racks based on the aggregate traffic of ToR switches, thus leveraging replica placement to accomplish effective cross-rack bandwidth usage and achieve rack-level I/O balancing.

3.2 Placement Policy

All racks in a data center are partitioned into multiple clusters, which are connected through Core switches [25]. Given a cluster of N+1 racks communicated via an EoR (a.k.a., End of Row) switch (see Fig. 1), we refer to one rack as *primary rack* and the rest of N racks as *replication racks*. For example, a data center is comprised of a total of 810 nodes housed in 45 racks (i.e., about 18 nodes per rack), which are divided into 3 clusters (i.e., about 15 racks per cluster).

Interleaved Declustering (ID) mechanism is originally proposed to dispatch data replicas over disks in RAIDs [10]. In this study, the ID mechanism tailored for RAIDs is introduced to organize data-block replicas among racks as follows. Primary copies of a group of N blocks $\{B_1, B_2, ..., B_N\}$ are exclusively placed on nodes in a selected primary rack, and all secondary copies of the N blocks are exclusively placed to N replication racks.

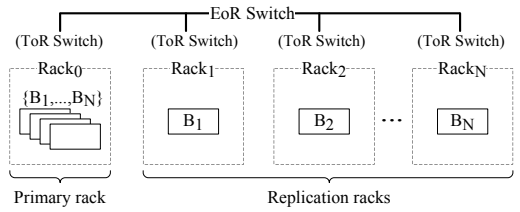


Fig. 1: 2-way replicas distribution for N blocks $\{B_1, B_2, ..., B_N\}$ among N+1 racks using interleaved declustering.

To archive unpopular data to conserve memory space, we advocate for a (k+r,k) erasure encoding of k unpopular data blocks residing in the same rack. Usually, parameter k is smaller than N. When multiple k-block groups share the same primary rack, the secondary copies of all the blocks are placed across the replication racks organized as a chain. More specifically, the secondary copies of the first group is assigned to the first k replication racks; the secondary copies of the second group should be placed to the next k replication racks. The above procedure is repeated until all the secondary copies are distributed over N replication racks like a chain. Algorithm 1 presents the pseudo-code of replica-distribution procedure, in which a primary rack is selected based on the aggregated traffic of ToR switches, and nodes in the primary rack are randomly chosen (see §3.3). Additionally, a 'P' flag is added to indicate the primary copy of each data block (see §4.3).

Algorithm 1: Distribute-block-replicas(B[ ])

Input:
$B[\,]$: a sequence of newly-loaded data blocks

```
1 while B[ ] is not empty do
2     Select a primary rack based on available ToR-switch bandwidth;
3     Place k sequential blocks {B[1], B[2], ..., B[k]} to memories of k
        randomly-chosen nodes in the primary rack;
4     Distribute each secondary copy of the k blocks over k out of N replication
        racks;
5     Add flag 'P' to each copy of the k blocks in the primary rack;
6     Remove the first k blocks from collection B[ ];
```

3.3 Selection of Racks and Nodes

Fig. 1 reveals that N blocks is placed to one primary rack, whereas only one block is residing in one replication rack. That is, the consumption of cross-rack bandwidth is dominated by primary replicas rather than secondary replicas in the replica-placement stage. Therefore, we solely consider the selection of primary racks. Once a primary rack is determined for k blocks, secondary copies of the k blocks will be distributed over k out of N replication racks (see Line 4 in Algorithm 1).

From the angle of effective network usage, it is desirable to enable each ToR switch to deliver similar aggregated traffic. As to homogeneous ToR switches, the available bandwidth of a rack's ToR-switch is high when the aggregated access load of requests issued to nodes in the rack is low. In other words, a rack is the least-loaded one if its ToR-switch delivers minimal aggregated traffic.

Similar to RRP [8], ERP places each primary block to a randomly-chosen node in a primary rack; the replica of a block is also dispatched to a randomly-chosen node in the corresponding replication rack. That is, nodes within a rack are randomly appointed for primary or secondary replicas.

4 THE TEA SCHEME

The TEA scheme aims to jointly consider fault tolerance, cross-rack traffic, and load balancing for ERP-powered in-memory stores.

4.1 Encoding Processes

Let us elaborate the erasure-coded archival process via an example. Suppose that (6, 4) RS codes are applied to k=4 unpopular data blocks $\{D_1, D_2, ..., D_4\}$, with 2-way replicas of the 4 blocks are kept to 5 racks abiding by the ERP policy (see Fig. 2(a)). Next, $Rack_0$ is nominated as the encoding rack since it keeps all the 4 blocks, and r=2 parity blocks $\{P_1, P_2\}$ are generated by a node (a.k.a., encoding node) in the encoding rack (see Fig. 2(b)). Finally, parity block P_2 is kept in the encoding node; parity block P_1 is migrated to another distinct rack (e.g., $Rack_5$); and the 4 data blocks in $Rack_0$ are discarded (i.e., marked as 'deleted'). In this way, k=4 data blocks and r=2 parity blocks in a stripe are exclusively stored to k+r=6 racks after encoding to tolerate any r=2 rack failures (see Fig. 2(c)).

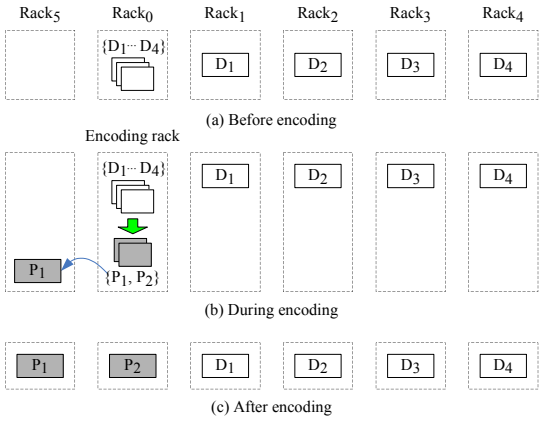


Fig. 2: An encoding process under ERP, with coding parameters k=4, and r=2

From the above process, we observe that, (1) no cross-rack traffic occurs during encoding since the primary copies of k=4 data blocks reside in $Rack_0$, and (2) there exist r=2 co-located parity blocks after encoding. Out of consideration for fault tolerance, r-1=1 parity block needs to be migrated to a different rack (e.g., $Rack_5$).

Three 'How to' issues are raised in an erasure-coded archival process. (i) How to distinguish unpopular blocks? (ii) How to construct stripes? (iii) How to guarantee availability for encoded blocks? TEA handles the above issues as follows. (i) Average access frequency serves as the popularity threshold. (ii) Non-sequential striping is deployed to construct stripes. (iii) Parity-block relocation is adopted to preserve fault tolerance.

4.2 Determining Unpopular Blocks

An in-memory store maintains a block list to globally manage all in-memory blocks. BlockID - a unique identifier - is assigned to an in-memory block, which is kept on a node with NodeID in a rack identified by RackID. Fig. 3 shows four types of metadata affiliated to a block, including location (RackID, NodeID), block status (Flag), number of accesses (NOAs), and timestamp (TS) of block placement. Flags 'P' and 'S' represent it is the primary and secondary copy of a block, respectively; flags 'U' and 'E' indicate it is a block in an uncoded and encoded stripe, respectively.

BlockID	RackID	NodeID	Flag	NOA	TimeStamp(TS)
0x00000011	0x1E	0x03	'P'	0x0011	2018-07-29 00:00:01.283
0x00000012	0x1E	0x04	'S'	0x0101	2018-07-29 00:02:01.002
0x00000022	0x10	0x01	'U'	0x0111	2018-07-29 00:02:03.012
...	...	...	...	...	...
0x000000e4	0x1F	0x02	'E'	0x0001	2018-07-29 00:03:02.123

Fig. 3: Diagram of structure 'Block Metadata'.

Many dynamic replication schemes (e.g., TWR [22], LALW [6]) apply average access frequency as a popularity threshold to find out popular files. In TWR, access frequency is measured via the NOAs during a specific time interval. In LALW, access frequency is gauged as a product of access weight and the NOAs, where different weights are assigned to accesses in different time intervals.

When it comes to erasure-coded archival, average access frequency can be used as the popularity threshold to pick out unpopular blocks. Since this study is focused on stripe construction rather than popularity estimation, the NOAs can be directly adopted to reflect the access frequency of a block. The average access frequency $\overline{AF}$ of all in-memory blocks can be calculated via Eq.(1).

$$\overline{\mathrm{AF}} = \frac{1}{|\mathcal{S}\mathrm{et_B}|} \sum_{\mathrm{B}\in\mathcal{S}\mathrm{et_B}} \mathrm{NOA(B)}, \tag{1}$$

where $\mathcal{S}et_B$ is a set of in-memory blocks, $|\mathcal{S}et_B|$ denotes the number of distinct blocks in $\mathcal{S}et_B$, and $NOA(B)$ indicates the total number of accesses for both primary and secondary replicas of block B during a time interval.

Algorithm 2 outlines the pseudo-code of the unpopular-blocks-determination process in TEA, where set $\mathcal{S}et_{up}^{[v]}$ contains unpopular primary blocks in rack $Rack_v$; only blocks that have been accessed less than $\overline{\mathrm{AF}}$ times are encoded to improve memory space utilization. That is, the average access frequency $\overline{\mathrm{AF}}$ is used as a threshold to distinguish unpopular blocks.

Algorithm 2: Determine-unpopular-blocks( )

Output:
$\mathcal{S}et_{up}^{[]}$: set of unpopular primary blocks

1 Compute $\overline{AF}$ using Eq.(1);
2 **for** $Rack_v \leftarrow Rack_0$ **to** $Rack_N$ **do**
3 Collect all blocks marked with flag 'P' in $Rack_v$ into $\mathcal{S}et_{up}^{[v]}$;
4 Find out popular blocks with AF exceeding $\overline{AF}$;
5 Remove popular blocks from $\mathcal{S}et_{up}^{[v]}$;
6 **return** $\mathcal{S}et_{up}^{[]}$;

4.3 Construction of Stripes

4.3.1 Non-sequential Striping. More often than not, objects kept in in-memory KV stores are small-sized (e.g., in the range of a few bytes to few kilobytes). When it comes to an erasure-coded KV in-memory store, a block is sealed from multiple small-sized objects (see Cocytus [31], MemEC [30]). Logically, there is not inherent correlation among blocks. Therefore, it is applicable to group any k blocks into a stripe in in-memory KV stores. That is, it is feasible for in-memory stores to deploy non-sequential striping.

As analyzed in §4.1, no cross-rack traffic occurs during encoding if the primary copies of k blocks reside in an identical rack. When it comes to stripe construction, it is expected to select candidate members of a stripe from unpopular primary blocks in one rack.

When one or more popular blocks in a set of sequential primary blocks are filtered out, the remaining unpopular blocks are non-sequential. If data blocks of stripes are composed of such unpopular blocks, non-sequential striping will be reached accordingly.

4.3.2 Spatial Locality. Spatial locality refers to the access of data elements within relatively close storage locations. It is probable that k chosen blocks in a stripe may reside on k different nodes and; therefore, the later data-block retrieval operations may cause k-1-block intra-rack traffic when one of nodes keeping the k blocks acts as an encoding node. If spatial locality is exploited by stripe construction, intra-rack traffic will be reduced because a stripe is constructed with k blocks from as few nodes as possible.

If a stripe is formed with k unpopular primary blocks on an identical node, then no intra-rack traffic will occur in the data-block-retrieval stage. In particular, all the unpopular primary data blocks in a specific rack are sorted based on the node identifier (i.e., $NodeID$) (see Line 2 in Algorithm 3) and; thus, two blocks or more in one node may be put into a stripe.

Example: After applying Algorithm 2 (i.e., Determine-unpopular-blocks( )), a set of unpopular primary blocks in a specific rack are determined, e.g., $\mathcal{S}et_{up}^{[v]}$ for rack $Rack_v$ ($v \in [1, N]$). Let D[i,j] be an unpopular primary block on node $Node_i$ ($1 \le i \le N_{node}$) in rack $Rack_v$, and its secondary replica is kept in rack $Rack_j$ ($j \in [0, N]$, and $j \neq v$). Fig. 4 illustrates a diagram of unpopular primary blocks in sorted set $\mathcal{S}et_{up}^{[1]}$, where blocks on the same node are listed on a row. It is noted that two distinct primary blocks kept on $Node_1$ in $Rack_1$ are denoted as 'D[1,3]', with their secondary replicas are kept in $Rack_3$. From the perspective of the entire in-memory store, it is prevalent that two and more different unpopular primary blocks reside on an identical node, and their replicas are kept within the same rack.

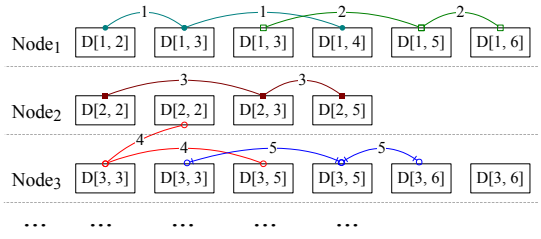


Fig. 4: Diagram of unpopular primary data blocks in sorted set $\mathcal{S}et_{up}^{[1]}$. The number in a curve represents the ordinal number of a stripe, which consists of k=3 data blocks.

After a k-block-selection operation is applied to blocks in the sorted set of $\mathcal{S}et_{up}^{[1]}$, a stripe may consist of blocks on one or multiple nodes. Case-1 *(on one node)*: For example, 1^{st} stripe is formed from k=3 blocks {D[1,2], D[1,3], D[1,4]} on node $Node_1$, which becomes the encoding node of this stripe. In this case, the encoding of the 1^{st} stripe induces no intra-rack traffic. Case-2 *(two or multiple nodes)*: For instance, 4^{th} stripe is generated from k=3 blocks {D[2,2], D[3,3], D[3,5]} on two nodes; $Node_3$ is elected as the encoding node of this stripe, because two blocks are kept by $Node_2$. In this case, merely k-2=1 block needs to be transfered to $Node_2$ in the data-block-retrieval stage, i.e., the corresponding intra-rack traffic is one block.

4.3.3 Algorithm. Algorithm 3 gives the pseudo code of TEA's stripe construction, where all the racks are polled in a round-robin manner. Unpopular primary blocks in a specific rack (e.g., $Rack_v$) are sorted in an ascending order of $NodeID$ to exploit spatial locality.

An iterative k-block-selection process is imposed on the sorted unpopular primary blocks until condition "replicas of blocks in $\mathcal{S}et_{up}^{[v]}$ reside in at least k racks" is violated. In the block-selection process, if k blocks chosen from the sorted set $\mathcal{S}et_{up}^{[v]}$ in a sequential order meet the requirement of "replicas of the k chosen blocks are hosted in k distinct racks", then the k chosen blocks will be grouped into a stripe. Among nodes keeping the k chosen blocks, one node keeping the most blocks in a stripe is nominated asfor the encoding node. After k blocks in $\mathcal{S}et_{up}^{[v]}$ are elected, flag 'P' of each block is marked as 'U', signifying that the k blocks will be used for the later encoding.

Algorithm 3: Construct-stripes($\mathcal{S}et_{up}^{[]}$)

Input:
$\mathcal{S}et_{up}^{[]}$: set of unpopular primary blocks

```
for Rack_v ← Rack_0 to Rack_N do
    Sort blocks in Set_up^[v] in an ascending order of NodeID;
    while replicas of blocks in Set_up^[v] reside in at least k racks do
        Choose k blocks from sorted Set_up^[v] in an sequential order, ensuring
          their replicas are hosted in k racks;
        Group the k chosen data blocks into a stripe;
        Designate one node keeping the most blocks of the k blocks in the
          stripe as its encoding node;
        Change the flag of each of the k blocks from 'P' to 'U', and remove the
          k blocks from Set_up^[v];
```

When flag 'P' is assigned to a block replica residing in a primary rack, encoding tasks are likely to be balanced across all the racks. If flag 'P' is not attached to a block, then the primary copy of a block and the secondary copy of another block in the same rack may be put into a stripe and; thus, only a part of racks will encode a majority of stripes when encoding racks are selected in the round-robin fashion.

After the stripe-construction stage, both data-block members and an encoding node are determined for each stripe. The encoding of a stripe is accomplished by one encoding rack in a time; multiple encoding nodes independently handle the encoding of multiple stripes. Given a stripe associated with k data blocks, an encoding node retrieves required data blocks from the other nodes within the same rack, followed by computing r parity blocks using (k+r,k) erasure codes in a centralized manner.

We observe from Algorithm 3 that after iterating over unpopular primary blocks to construct stripes, remaining unpopular primary blocks may not be filled into a stripe since the blocks do not satisfy the condition "replicas of blocks in $\mathcal{S}et_{up}^{[v]}$ reside in at least k racks". In this case, our TEA scheme puts such unpopular data blocks to next round of stripe construction because new unpopular data blocks are dynamically produced.

4.4 Block Relocation

After encoding a stripe, k unpopular primary data blocks in the stripe are discarded (i.e., marked as 'deleted'). Meanwhile, the flag of each secondary replica of the k blocks is changed from 'S' to 'E', indicating that it is kept in an erasure-coded form.

Fig. 2(b) reveals that an encoding node generates r corresponding parity blocks from k data blocks in a stripe. As a result, there are r co-located parity blocks in the encoding node. To tolerate concurrent r-rack and r-node failures, parity-block relocation should be carried out as follows. One parity block is still kept by the encoding node; the other r-1 parity blocks are migrated to r-1 racks different from both the encoding rack and racks keeping the secondary copies of the k data blocks. Consequently, it is deduced that the number N+1 of racks in a cluster should be greater than or equal to the number k+r of encoded blocks in a stripe. Thus, we have $N+1 \geq k+r$.

Rack-level fault tolerance is a vital feature for a large-scale distributed storage system, because the failure of a ToR switch may cause all the nodes in a rack out of contact. Nevertheless, rack failure is a rare event in real-world storage systems [11], it is adequate to support single-rack fault tolerance. Hence, it is unnecessary to migrate r-1 of r parity blocks from the encoding rack to r-1 different racks; rather, r-1 out of r parity blocks are moved from the encoding node to r-1 different nodes in the same rack to achieve r-node fault tolerance.

5 EXPERIMENTS

5.1 Experimental Environment

5.1.1 System Setup. We build a storage cluster, which consists of one client, one coordinator, one proxy, one regulator, and 36 storage nodes {SN_0, SN_1, ..., SN_{35}}. All nodes are connected via a Cisco Catalyst 2960-S series Switch. The client, coordinator, proxy and regulator nodes are Dell R720 servers, each of which is equipped with two Xeon® E5-2609 @2.40GHz CPUs, and 32GB DDR3 memory. Each storage node contains an Intel® i3-2120 @3.30GHz CPU, and 4GB DDR3 memory. We install the Ubuntu 16.04 (x86-64) with Linux Kernel 4.4.0-62-generic on all the machines.

5.1.2 Prototype Design. Fig. 5 depicts an application-level prototype for in-memory stores, where the coordinator node embraces the replica placement and stripe construction modules, and maintains all the metadata for blocks, stripes, nodes, and racks in the in-memory store; the regulator node or RN imitates the cross-rack settings; the proxy node redirects I/O requests issued by client nodes to the coordinator node; the client node replays the trace derived from the YCSB benchmark [9], simulating concurrent clients; each

storage node not only furnishes pre-allocated memory region to in-memory blocks, but also performs erasure-encoding operations.

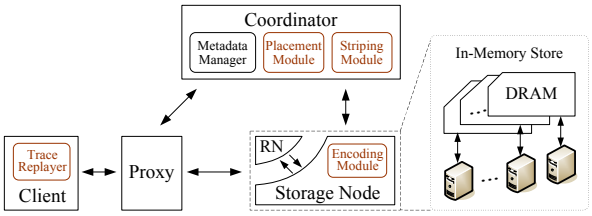


Fig. 5: An application-level prototype of encoding-powered in-memory stores.

The Linux traffic controlling tool or *TC* [2] is deployed in the regulator node to adjust its available bandwidth. Cross-rack communications are accomplished in a packet-forwarding manner, in which requested blocks are relayed to the regulator node and then forwarded to destination nodes.

Both the random and encoding-oriented replica placement strategies are supported by the replica placement module. The stripe construction module incorporates five erasure-coded archival schemes (i.e., TEA, DSC, Sice, EAR, and SS-RRP (see Table. II)), which govern storage nodes to handle erasure-encoding operation for unpopular in-memory data blocks.

The 'Encoding' module in storage nodes undertakes RS-encoding operations according to data-block-selection and encoding-node-designation decisions made by 'Striping' module hosted in the coordinator node. Intel's Intelligent Storage Acceleration Library (Intel ISA-L) is adopted to implement RS-encoding [16].

5.1.3 Testing Methodology. Table. II lists four categories of erasure-coded archival solutions, namely, (i) sequential striping for random replica placement (*SS-RRP*), (ii) non-sequential striping for random replica placement (*nSS-RRP*), (iii) sequential striping for encoding-oriented replica placement (*SS-ERP*), and (iv) non-sequential striping for encoding-oriented replica placement (*nSS-ERP*). DSC and Sice belong to the nSS-RRP category; EAR and TEA are classified into the SS-ERP and nSS-ERP categories, respectively.

Table II: Categories of erasure-coded archival solutions.

	Sequential Striping	Non-Sequential Striping
Random Replica Placement	SS-RRP	nSS-RRP (e.g., DSC/Sice)
Encoding-oriented Replica Placement	SS-ERP (e.g., EAR)	nSS-ERP (e.g., TEA)

To accomplish comparative tests, we designate SS-RRP, DSC, Sice and EAR as candidate encoding schemes. We use workloads generated from the YCSB benchmark [9] to evaluate TEA and the four candidate schemes. We focus on read-heavy workload, with GET/SET ratio is of 95:5. The number of unique data blocks is 5K; the YCSB client issues total 500K requests; and the access pattern follows a Zipf distribution with a high shape parameter μ.

In the validation tests, we set up the default configuration as follows. (i) encoding parameters 'k=6, and r=2' are applied unpopular blocks; (ii) the value of μ is 0.8; (iii) each block is of size 4KB; (iv) bandwidth over-subscription ratio is set to '10:1' for cross-rack communications [1]; (v) each rack contains three nodes; and (vi) an encoding operation is triggered when 100k requests are replayed.

Our primary metrics for performance comparisons are archival throughput, cross-rack traffic, and load balancing. Archival throughput is a rate at which stripes are produced. Cross-rack traffic is measured by a total number of blocks handled by cross-rack nodes. Just like node-level load imbalance measurement [21], rack-level load imbalance can be measured via an imbalance metric λ_{rack}, with λ_{rack}=(L_{max}-L_{avg})/L_{avg}, where L_{max} and L_{avg} are the maximal and average load among all the racks, respectively. Here, the load on a rack can be translated to the total number of blocks sent to/received by nodes in the rack. A small λ_{rack} value indicates good load balancing.

5.2 Experimental Results

5.2.1 Validation tests. A group of experiments is conducted to examine the validness of the TEA scheme, where the above-mentioned default configurations are applied. Fig. 6 shows the experimental results in terms of archival throughput, cross-rack traffic and rack-level load balancing.

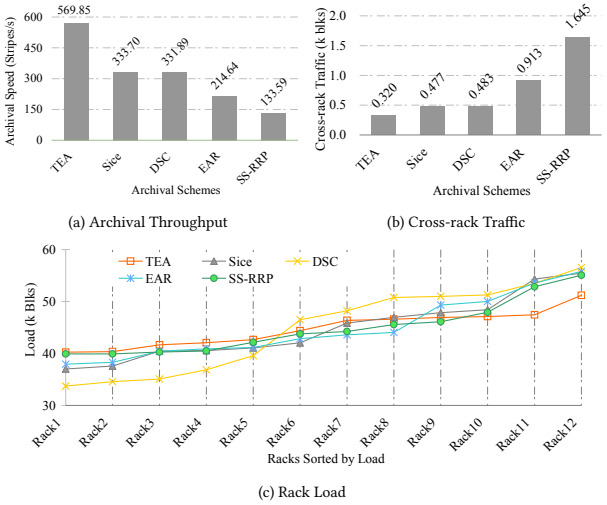


Fig. 6: Comparison of the five encoding schemes using default configurations, with parameters k, r, μ, R_{os}, S_{blk}, and N_{rack} are set to 6, 2, 0.8, '10:1', 4KB, and 12, respectively.

We draw two intriguing observations from Fig. 6(a). First, SS-RRP exhibits the lowest archival throughput among the five encoding schemes, because the sequential stripe construction greatly increases the number of cross-rack data-block retrievals for subsequent erasure encoding. Second, TEA outperforms the other four encoding schemes thanks to (1) the ERP policy facilitating the cross-rack block-retrieving process and (2) the locality-aware striping methods mitigating the intra-rack block-retrieving operations. In particular, TEA speed up the four encoding schemes by at least 70.8%.

Fig. 6(b) indicates that compared to non-sequential striping, sequential striping causes a larger amount of cross-rack traffic. Three indications are given as below. (i) SS-RRP causes the most cross-rack traffic for encoding operations among the five encoding schemes, since it not only ignores bandwidth over-subscription, but also employs sequential striping. (ii) Relative to SS-RRP, EAR creates less cross-rack traffic owing to it's encoding-aware replication scheme. (iii) In comparison with the other two non-sequential

striping schemes (i.e., Sice and DSC), the TEA scheme leads to less cross-rack traffic, because it is an ERP-powered encoding scheme, where the ERP policy potentially enhances the probability of choosing k unpopular data blocks for a stripe from one rack.

Fig. 6(c) presents the load distribution across racks in terms of the number of blocks accessed from each rack. It is obvious that the loads across racks in TEA are distributed more uniformly than the other encoding schemes, because TEA selects a primary rack according to the current loads of racks. The percent imbalance metric λ_{rack} for TEA, Sice, DSC, EAR, and SS-RRP in this experiment are 14.38%, 23.76%, 26.19%, 24.47%, and 22.77%, respectively. Therefore, TEA improves rack-level load-balancing by a factor of more than 1.58x relative to the four candidate encoding schemes.

5.2.2 Numbers of data blocks – k. Recall that a stripe in (k+r,k) RS codes consists of k data blocks. To examine the sensitivity of the encoding schemes to parameter k, we set the k value to 4, 6, and 8, respectively. Fig. 7 plots the archival-throughput, cross-rack-traffic, rack-level-balancing performance of the five encoding schemes with respect to the three k values.

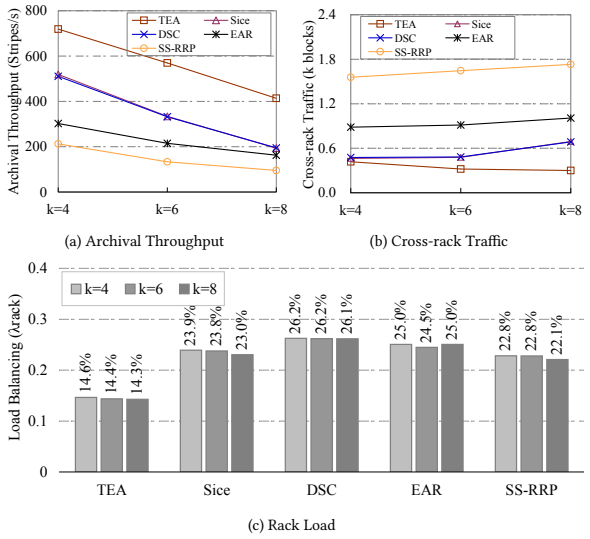


Fig. 7: Comparison of the five encoding schemes using default configurations, with parameter k is set to {6,8,10}.

We draw two observations from Fig. 7(a). First, given a specific encoding scheme, its archival throughput slumps when the value of parameter k increases. Because a large k value potentially reduces the probability of constructing a stripe from k unpopular blocks while satisfying the condition that 'replicas of blocks in $\mathcal{S}et_{up}^{[v]}$ must reside in at least k racks'. Second, the archival-throughput discrepancy between TEA and Sice (or DSC) becomes pronounced when parameter k grows, because the RRP policy is more likely to limit the efficiency of stripe construction than the ERP policy.

Total cross-rack traffic is the summation of the product of stripe number and per-stripe cross-rack traffic. The number of stripes drops when parameter k rises. Fig. 7(b) reveals that the cross-rack traffic caused by the four candidate encoding schemes marginally increases when parameter k grows up. The reason is two-fold. (i) As to sequential striping, a high k value potentially uplifts the probability of selecting k unpopular blocks from nodes on different racks. (ii) The RRP policy may reduce the probability of intra-rack unpopular-block retrievals used to construct a stripe, and this probability decreases when parameter k increases. Fig. 7(b) also shows that TEA causes less cross-rack traffic when parameter k grows up, the reason is that the ERP policy sightly raises the amount of per-stripe across-rack traffic.

We observe from Fig. 7(c) that TEA outperforms the four alternatives in terms of rack-level load-balancing for a specific k value. Additionally, given a specific encoding scheme, the fluctuation of rack-level load-balancing metric is rather small ($\Delta\lambda_{rack} < 4\%$) when parameter k changes. We ascribe such a gentle fluctuation to the fact that the rack-level loads usually are dominated by user accesses because back-end erasure-encoding operations on unpopular blocks is fewer than front-end user requests issued to popular blocks; the distribution of user accesses mainly depends on replica placements, which are established prior to encoding.

5.2.3 Different numbers of parity blocks – r. We examine the sensitivity of the encoding schemes to parameter r by setting r to 1, 2, and 3. Fig. 8 presents the archival-throughput, cross-rack-traffic, and rack-level-balancing performance of the five encoding schemes with respect to different r values.

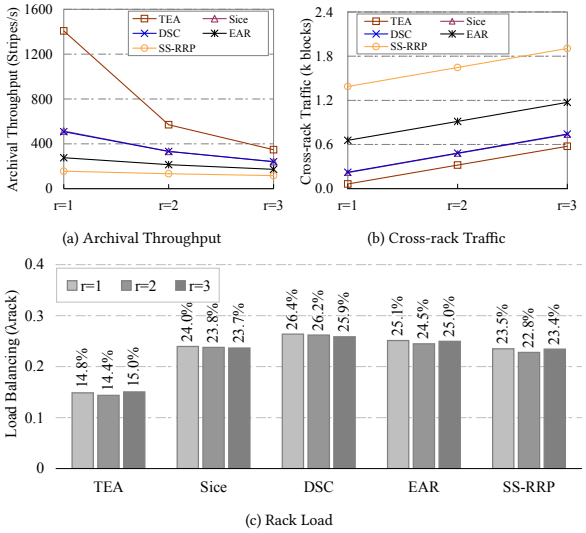


Fig. 8: Comparison of the five encoding schemes using default configurations, with parameter r is set to {1,2,3}.

Fig. 8(a) unveils that regardless of the r value, TEA outperforms the other four schemes in terms of archival throughput; this trend is consistent with that reported in §5.2.1. The archival throughput of each tested scheme drops along with the increasing r value, because at least r-1 parity blocks should be migrated to nodes across racks to achieve r-rack fault tolerance; and only when parity-block relocation operations are fulfilled can stripe consistency be guaranteed. The results reveal that the archival-throughput discrepancy between the TEA scheme and the competitive encoding schemes becomes small when the r value rises, because the percent discrepancy of total archival time (i.e., time spent to data-block retrieving, encoding, and parity-block relocation) shrinks when the parity-relocation time is prolonged.

Fig. 8(b) shows that in all the encoding schemes, cross-rack traffic for encoding operations becomes heavy when the r value increases because at least r-1 parity blocks should be migrated from an encoding node to nodes across racks. In each encoding scheme, the cross-rack-traffic discrepancy between 'r=1' and 'r=2' is almost equal to that between 'r=2' and 'r=3', because the incremental cross-rack traffic is predominantly caused by parity-block relocation operations.

Fig. 8(c) demonstrates that parameter r has a slight effect on the rack-level load-balancing performance. Such load-balancing performance trend is similar to that in the k-parameter case (see §5.2.2), because the distribution of front-end user accesses is dominated by the layout of data-block replicas.

5.2.4 Over-subscription ratios – R_{os}. Large clusters usually exhibit bandwidth over-subscription ratios (R_{os}) ranging from 5:1 to 20:1 or even higher [1]. It is expected that both cross-rack-traffic and rack-level-balancing performance are insensitive to parameter R_{os}, because the factor of bandwidth over-subscription is not considered by the archival steps (i.e., block retrieval, encoding, and block relocation). Nevertheless, parameter R_{os} may affect the archival-throughput performance of the five encoding schemes as it captures the cross-rack per-node bandwidths. Fig. 9 shows the archival-throughput performance of the five encoding schemes, where R_{os} is set to '5:1', '10:1', and '20:1', respectively.

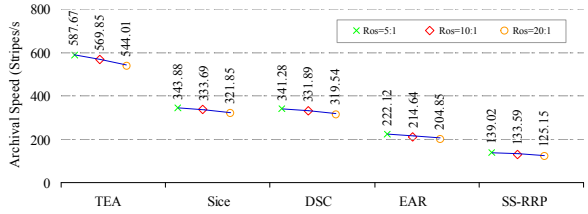


Fig. 9: Comparison of the five encoding schemes using default configurations, with parameter R_{os} is set to {'5:1', '10:1', '20:1'}.

We observe from Fig. 9 that the archival throughput of each encoding scheme slightly drops along with the increasing R_{os} value. Because a large value of R_{os} means low cross-rack per-node bandwidth, which results in long transmission time spent in both data-block retrieval and parity-block relocation.

5.2.5 Sizes of data-block – S_{blk}. The size S_{blk} of an in-memory block tends to affect the access performance of an in-memory store [31][30][24]. To examine the sensitivity of the encoding schemes to parameter S_{blk}, we adopt the S_{blk} collection of {1kB, 2kB, 4kB, 8kB, 16KB}. Block size has nothing to do with replica placements and stripe construction; thus, it has no impact on cross-rack traffic and rack-level-balancing performance. Fig. 10 only illustrates the archival throughput with respect to the size S_{blk} of block.

From Fig. 10 we obtain that the archival throughput of each encoding scheme drops along with the increasing parameter S_{blk}, because large blocks give rise to long transfer time spent in data-block retrieval and parity-block relocation. However, the archival throughput is not inversely proportional to the block size. From the viewpoint of throughput in terms of the amount of data being archived per second, the network bandwidth is under-utilized for small blocks. The under utilization is caused by the fact that

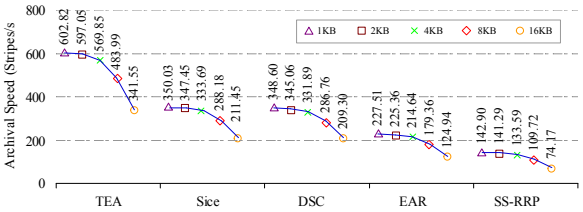


Fig. 10: Comparison of the five encoding schemes using default configurations, with parameter S_{blk} is set to {1KB, 2KB, 4KB, 8KB, 16KB}.

the archival time of a stripe is subject to data transmission and processing (e.g., across-address-space copying), and the processing overhead accounts for a great percentage of archival time of small blocks.

5.2.6 Distribution parameters – μ. A small value of Zipf parameter μ means a lowly-skewed popularity. It is necessary to examine the effect of parameter μ, because encoding schemes are applied to unpopular blocks. Fig. 11 plots the archival-throughput, cross-rack-traffic, and rack-level-balancing performance of the five encoding schemes, with μ is set to 0.7, 0.8, 0.9, and 0.99, respectively.

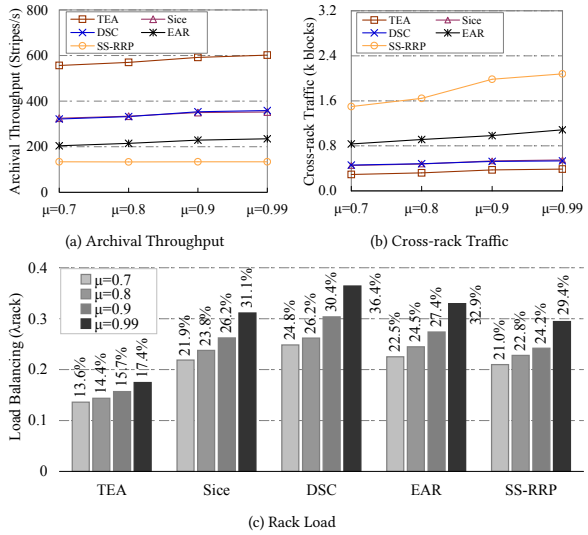


Fig. 11: Comparison of the five encoding schemes using default configurations, with parameter μ is set to {0,7, 0.8, 0.9, 0.99}.

Fig. 11(a) reveals that different from SS-RRP, the other four encoding schemes show high archival-throughput performance in the case of large μ. Because a larger μ value leads to more unpopular blocks; an encoding-oriented replica placement policy or a non-sequential striping method helps to produce more stripes.

A general trend plotted in Fig. 11(b) is that the cross-rack traffic in each encoding scheme goes up with the increasing μ value, because a larger μ results in more unpopular blocks for stripes, and more stripes in turn cause heavy cross-rack traffic. The results also show that incremental cross-rack traffic incurred by EAR and SS-RRP is larger than those of the other encoding schemes, because compared to the non-sequential striping method, the sequential one has a higher probability of selecting k unpopular blocks from nodes on different racks, namely, cross-rack traffic.

Fig. 11(c) illustrates that, (i) given a certain value of parameter μ, TEA exhibits better rack-level load-balancing than the other four encoding schemes; (ii) as to a specific encoding scheme, the rack-level-balancing performance deteriorates when μ increases, because the majority of accesses are centered around popular blocks, the number of which drops when parameter μ grows.

5.3 A Summary of Observations

From the above tests, we draw some observations as follows.

- In all the tested scenarios, TEA outperforms the other four competitor encoding schemes in terms of archival throughput, cross-rack traffic, and rack-level load-balancing. The performance improvements offered by TEA are ascribed to the optimized replica placements and stripe construction for in-memory stores.
- The comparative tests confirm that the archival throughput of TEA is affected by all the parameters, including k, r, S_{blk}, R_{os}, and μ. In contrast, the cross-rack traffic of TEA is insensitive to parameters S_{blk} and R_{os}; the rack-level balancing of TEA is insensitive to parameters k, r, S_{blk}, and R_{os}.

6 CONCLUSIONS AND FUTURE WORK

This study investigates encoding schemes to address the issue of redundancy transition from replication to erasure encoding for unpopular in-memory datasets, with aiming at reducing cross-rack traffic. An encoding-oriented replica placement policy (i.e., ERP) is proposed to facilitate subsequent encoding operations, and a traffic-efficient erasure-coded archival scheme (i.e., TEA) is developed for ERP-powered in-memory stores to (1) alleviate cross-rack traffic aroused by data-block retrieval and to (2) mitigate block relocation invoked to sustain rack-level fault tolerance. The comparative experiments illustrate that TEA accomplishes higher archival-throughput, cross-rack-traffic, and rack-level-balancing performance than the four competitors. In particular, within a group of validation tests, TEA speeds up archival throughput by at least 70.8%, improves rack-level load-balancing by more than 1.58x relative to the four alternative encoding schemes.

The five encoding schemes are evaluated in a small-scale cluster (e.g., 15 racks and 3 nodes per-rack). In the future, we plan to investigate them in a large-scale store (e.g., ≥10 nodes per-rack, ≥20 racks) through extensive simulations.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation of China under Grants 61572209, 61872156, 61872413, 61821003, and 61702004. Xiao Qin's work is supported by the U.S. National Science Foundation under Grants CCF-0845257, CNS-0917137, and CCF-0742187, and the 111 Project under Grant B07038.

REFERENCES

[1] Faraz Ahmad, Srimat T Chakradhar, and Anand Raghunathan. 2014. ShuffleWatcher: Shuffle-aware Scheduling in Multi-tenant MapReduce Clusters.. In *Proceeding of the 2014 USENIX Annual Technical Conference (ATC'14)*. 1–12.

[2] Werner Almesberger. 2001. Linux Network Traffic Control-Implementation Overview.

[3] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review*. ACM, 53–64.

[4] Dhruba Borthakur. 2007. The hadoop distributed file system: Architecture and design. *Hadoop Project Website* 11 (2007), 21.

[5] D Borthakur, R Schmidt, R Vadali, S Chen, and P Kling. 2010. HDFS RAID. Technical Talk. Yahoo! Developer Network.

[6] Ruay-Shiung Chang and Hui-Ping Chang. 2008. A dynamic data replication strategy using access-weights in data grids. *The Journal of Supercomputing* 45, 3 (2008), 277–295.

[7] Yuanqi Chen, Yi Zhou, Shubbhi Taneja, Xiao Qin, and Jianzhong Huang. 2017. aHDFS: An Erasure-Coded Data Archival System for Hadoop Clusters. *IEEE Transactions on Parallel and Distributed Systems* 28, 11 (2017), 3060–3073.

[8] Asaf Cidon, Stephen M Rumble, Ryan Stutsman, Sachin Katti, John K Ousterhout, and Mendel Rosenblum. 2013. Copysets: Reducing the Frequency of Data Loss in Cloud Storage.. In *Usenix Annual Technical Conference*. 37–48.

[9] Brian F. Cooper, Adam Silberstein, Erwin Tam, and et al. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*. ACM, 143–154.

[10] George Copeland and Tom Keller. 1989. A Comparison of High-availability Media Recovery Techniques. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data (SIGMOD'89)*. ACM, 98–109.

[11] D. Ford, F. Labelle, F.I. Popovici, and et al. 2010. Availability in Globally Distributed Storage Systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*. USENIX, 61–74.

[12] James Lee Hafner. 2005. WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems. In *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies (FAST'05)*. USENIX, 211–224.

[13] Li-Yung Ho, Jan-Jan Wu, and Pangfeng Liu. 2011. Optimal algorithms for cross-rack communication optimization in MapReduce framework. In *Proceedings of 2011 IEEE International Conference onCloud Computing (CLOUD)*. IEEE, 420–427.

[14] Jianzhong Huang, Yanqun Wang, Xiao Qin, and Xianhai Liang. 2015. Exploiting pipelined encoding process to boost erasure-coded data archival. *IEEE Transactions on Parallel and Distributed Systems* 26, 11 (2015), 2984–2996.

[15] KLab Inc. [n. d.]. Repcached[Online]. http://repcached.lab.klab.org.

[16] Intel. [n. d.]. Intel® Storage Acceleration Library (Open Source Version). https://goo.gl/zkVl4N.

[17] Runhui Li, Yuchong Hu, and Patrick PC Lee. 2017. Enabling efficient and reliable transition from replication to erasure coding for clustered file systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2500–2513.

[18] MS Manasse, CA Thekkath, and A Silverberg. 2009. A Reed-solomon Code for Disk Storage, and Efficient Recovery Computations for Erasure-coded Disk Storage. *Proceeding in Informatics* (2009), 1–11.

[19] Rajesh Nishtala, Hans Fugal, Steven Grimm, and et al. 2013. Scaling Memcache at Facebook.. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (NSDI'13)*, Vol. 13. 385–398.

[20] Lluis Pamies-Juarez, Anwitaman Datta, and Frederique Oggier. 2013. RapidRAID: Pipelined erasure codes for fast data archival in distributed storage systems. In *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM'13)*. IEEE, 1294–1302.

[21] K. V. Rashmi, Mosharaf Chowdhury, Jack Kosaian, and et al. 2016. EC-Cache: Load-Balanced, Low-Latency Cluster Caching with Online Erasure Coding. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, 401–417.

[22] Qaisar Rasool, Jianzhong Li, and Shuo Zhang. 2009. Replica placement in multi-tier data grid. In *8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC'09)*. IEEE, 103–108.

[23] T. P. Shabeera and S. D. Madhu Kumar. 2013. Bandwidth-aware data placement scheme for Hadoop. In *Proceedings of the 2013 IEEE Recent Advances in Intelligent Computational Systems (RAICS'13)*. 64–67.

[24] Konstantin Taranov, Gustavo Alonso, and Torsten Hoefler. 2018. Fast and strongly-consistent per-item resilience in key-value stores. In *Proceedings of the 13th European Conference on Computer Systems (EuroSys'18)*. ACM, 39–53.

[25] A Vahdat. 2009. Scale and efficiency in data center networks. *UC San Diego* (2009).

[26] Hakim Weatherspoon and John D Kubiatowicz. 2002. Erasure Coding vs. Replication: A Quantitative Comparison. In *Peer-to-Peer Systems*. Springer, 328–337.

[27] Shuzhan Wei, Yongkun Li, Yinlong Xu, and Si Wu. 2017. DSC: Dynamic stripe construction for asynchronous encoding in clustered file system. In *IEEE Conf. on Computer Communications (INFOCOM'17)*. IEEE, 1–9.

[28] Daniel Wind. 2013. *Instant Effective Caching with Ehcache*. Packt Publishing Ltd.

[29] Yanwen Xie, Dan Feng, and Fang Wang. 2017. Non-Sequential Striping for Distributed Storage Systems with Different Redundancy Schemes. In *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 231–240.

[30] Matt M. T. Yiu, Helen H. W. Chan, and Patrick P. C. Lee. 2017. Erasure Coding for Small Objects in In-memory KV Storage. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR'17)*. ACM, Article 14, 12 pages.

[31] Heng Zhang, Mingkai Dong, and Haibo Chen. 2016. Efficient and Available In-memory KV-store with Hybrid Erasure Coding and Replication. In *Proceedings of the 14th Usenix Conf. on File and Storage Technologies*. USENIX, 167–180.