An Access-Oriented Placement Strategy with Online Erasure Coding in Memory Stores

Shuang Wang
Wuhan Polytechnic
China
wangshuang@whut.edu.cn

Jian Luo
Wuhan University
China
luojian_edu@126.com

Yunfei Li*
Rajamangala University of Technology Tawan-Ok
Thailand
yunfei.li@rmutto.ac.th

Abstract

Objects in memory stores are becoming industry solutions for high-availability applications. These systems routinely encounter the challenges of popularity skew and server failures, which result in severe load imbalance across servers and degraded I/O performance. To solve the above problems, this paper proposes an adaptive placement scheme based on load awareness with erasure encoding for read-intensive clusters (RS-APS) and adopts the lease mechanism to optimize the data migration process. RS-APS places in-memory data and repositions them by (i) analyzing access patterns, (ii) perceiving servers' performance, and (iii) utilizing a lease mechanism during migration. Experimental results show that compared to other placement schemes, RS-APS can increase the load balancing by more than 10X under the situation of data intensive access. The mean latency under RS-APS can be reduced by more than 2x compared to the two static schemes, i.e., the configuration-dependent placement scheme (RS-LPS) and hashing placement scheme (RS-HPS). The read latency of RS-APS with the lease mechanism can be reduced by 16.15% compared with the enhanced RS-LPS (RS-LPS+) and system throughput can be improved to 1.19x. Moreover, RS-APS exhibits obvious advantages for the scalability of storage clusters.

CCS Concepts

• **Computing methodologies** → Modeling and simulation; Simulation evaluation.

Keywords

RS Coding, Distributed In Memory System, Availability, Scalability

ACM Reference Format:
Shuang Wang, Jian Luo, and Yunfei Li. 2024. An Access-Oriented Placement Strategy with Online Erasure Coding in Memory Stores. In *2024 9th International Conference on Intelligent Information Processing (ICIIP 2024), November 21, 22, 2024, Bucharest, Romania.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3696952.3696974

*Corresponding author.

ICIIP 2024, November 21, 22, 2024, Bucharest, Romania

ACM ISBN 979-8-4007-1807-6/24/11
https://doi.org/10.1145/3696952.3696974

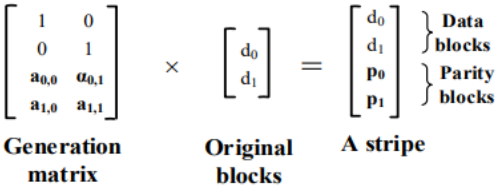

Figure 1: Encoding process of (4, 2) RS codes

1 Introduction

The increasing prominence of big data has intensified the challenges faced by time-sensitive applications such as online transaction processing (OLTP) and high-performance computing (HPC). OLTP systems must cope with rapidly expanding transaction volumes, while HPC applications frequently involve repeated processing of large data sets in scientific computing. Traditional disk I/O access latency is insufficient for ensuring time efficiency, making the adoption of memory storage technology inevitable. In-memory NoSQL databases like Redis, MemepiC, and MongoDB provide low-latency data access, while memory processing engines such as Spark, HANA, and Exalytics deliver high-throughput data analysis. For example, Spark introduces coarse-grained elastic distributed data sets (RDDs) to support distributed in-memory computation, and achieves task execution speeds up to 100 times faster than Hadoop. Facebook's Memcached stores data in-memory to speed up dynamic web applications by alleviating database load. Clearly, in-memory storage systems have become crucial for these applications.

However, the volatile nature of memory storage presents reliability challenges. Redundancy mechanisms are essential to ensure service during faults or system upgrades. Common redundancy schemes include: (i) persistent storage, reloading data from persistent media such as disks; (ii) replication, using multiple replicas to tolerate errors; and (iii) erasure coding, reconstructing invalid objects from surviving in-memory objects. High-speed networks now outperform local disk read performance, enabling most memory redundancy schemes to bypass disk I/O. Erasure coding enhances storage efficiency, offering higher fault tolerance with less storage space. RS code is a widely used erasure coding technique characterized by their maximum distance separable (MDS). property, (k + r, k) RS coding multiplies k data blocks by k * r redundancy matrix to calculate r parity blocks.

Figure 1 shows the generation process of parity blocks in RS coding.

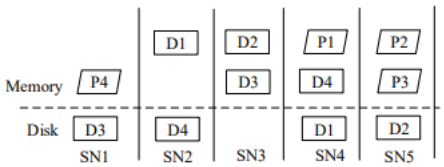

Figure 2: Layout of (4, 2) RS in memory and on disk

Read-intensive clusters often experience skewed data access distribution, where a small portion of data is frequently accessed while most data is rarely accessed. Ignoring access skew can lead to load imbalance, negatively impacting resource utilization and overall system performance. This study addresses the data placement schemes for erasure-coded in-memory clusters, focusing on access skew distribution and load imbalance. We propose RS-APS, a scheme where high-performance nodes handle more client requests. RS-APS prioritizes placing new in-memory blocks on low-load nodes and migrating hot objects to high-bandwidth nodes. By periodically evaluating data popularity and node performance, RS-APS dynamically adjusts load distribution to maximize resource utilization. Additionally, RS-APS uses a lease mechanism during migration to prevent bandwidth resource waste.

We summarize our contributions as follows.

We present a decentralized layout like mirroring RAID-5 for RS-encoded in-memory storage system to guarantee data availability and reliability.

We quantify the heat level of in-memory data blocks and the bandwidth level of storage nodes, and apply efficient division standards for blocks' heat and nodes' bandwidth.

We propose RS-APS and its optimization on a distributed system prototype. Optimized RS- APS achieves load balancing best among various schemes, and improves system throughput and reduce the response latency.

We introduce lease mechanism to optimize the migration process in RS-APS, realizing tempo- rary random replicas for different heat-level blocks, to reduce response latency.

We conduct discrete-event simulations based on YCSB [4], and compare RS-APS and other schemes for various parameter choices. Experiments show that optimized RS-APS performs better on system's scalability.

2 DESIGN OF RS-APS

The data layout formed in the storage system, taking (4,2) RS as an example, is shown in Figure 2. When loaded into memory for processing, data blocks are organized into Reed-Solomon stripes, with disk blocks and in-memory copies stored on separate nodes. the distributed storage system achieves the load balance, improves fault tolerance, and optimizes memory usage across the cluster. RS-APS places data by periodically monitoring nodes' performance and load characteristics, timely adjusting the distribution of access pressure of each node in the storage cluster, to improve the overall response performance of the system.

2.1 Process of Data Placement

The system periodically evaluates the performance of nodes and the heat of blocks in memory. Whole data placement process is divided into two subprocesses according to the time of data placement.

Writing new encoded data into memory. When new encoded blocks are written to memory—either during data loading from disk to cluster memory or from user write requests, these blocks are placed on high-performance nodes distinct from those holding the on-disk copies. The encoding node retains the received data blocks in memory until k data blocks are collected and encoded. It then keeps one parity block and distributes the remaining parity blocks to other high-performance nodes.

Adjusting the location of data in memory. This involves migrating data, As the system runs, the "heat" (access frequency) of these blocks changes, necessitating periodic adjustments to balance the load. Hot data is migrated to high-performance nodes to mitigate load imbalance. The process addresses two main challenges: determining which data blocks to migrate and selecting the source and destination nodes, i.e., low-performance nodes that send hot data and high-performance nodes that receive them, respectively. To adjust the load distribution adaptively, our design considers the law of data access and adopts a simple but practical way to determine the candidate nodes to participate in the migration task.

Selection of candidate data blocks to be migrated. To select hot data, quantitative methods of cumulative statistics are used. When periodically evaluating the popularity of data blocks in memory, the access frequency of each block is counted, and the blocks are sorted by counts in order from high to low. The top data blocks, whose total number of visits accounts for 80% of the total visits thus far, are included in the range of candidate blocks. For example, the total visits of the top k data blocks is 80% of all blocks visits, i.e. $\sum_{m} Visits(i)$, where m is the total number of distinct data blocks in memory. In addition, the data blocks evaluated as high heat level will be migrated only when they are located at the nodes with low performance.

Selection of candidate nodes to participate in migration tasks. To prevent performance deterioration, the system unloads high-access nodes by redirecting requests to less burdened nodes. Nodes are evaluated using cumulative access statistics consistent with blocks. and the nodes are ranked by requests in descending order. The $N0$ nodes, whose accumulative visits occupy 80% of the total number of system visits, are selected as the nodes of low performance. Prioritizing to migrate those hot candidate blocks on the toughest nodes during each cycle can help improve system performance quickly. We need to detect whether candidate data blocks are in the memory of these toughest nodes.

The nodes with high performance in the system are appointed as data block destinations, which are referred to as candidate nodes, and the number T opK of high-performance nodes is set by Equation 1).To prevent all target blocks from migrating to the highest-performance node, hot blocks are distributed evenly across the candidate node set, from highest to lowest performance. Here, N is the total number of storage nodes in the cluster, N0 represents the low-performance nodes evaluated similarly to candidate hot

blocks, and T opK denotes the top K high-performance nodes.

$$TopK = \begin{cases} N/2 & if \frac{N}{2} \leq N - N_0 \\ N - N_0 & otherwise \end{cases} \quad (1)$$

2.2 Optimization Design

In the process of data deletion, the "lazy deletion" technique is often adopted to prevent I/O delay caused by frequent loading of data from disk to memory due to cache misses. Similarly, "lazy migration" delays the deletion of data blocks from source nodes. When blocks are migrated, the source nodes keep copies in a buffer queue for a period, only deleting them after a lease period ends. This technique of sacrificing space to save time can reduce traffic overhead by delaying the deletion of data blocks in source nodes.

RS-APS periodically assesses block heat levels and node bandwidth. Overloaded nodes have their hot blocks migrated to high-performance nodes. Realistic workload statistics show that block heat levels remain stable for a period, which may cause blocks to move back to their original nodes in subsequent cycles. This round-trip migration of frequently accessed hot data wastes network bandwidth.

Based on this analysis, storage nodes maintain a queue for temporary copies in local memory, labeled as migrated, and kept until their lease expires (Lease means the period for blocks in the buffer queue to live, measured by the number of time window Tw). If blocks return to their original nodes within the lease period, the metadata server updates the node information, and the original node revalidates the blocks. This saves network traffic and reduces response delays, as destination nodes can immediately access reserved blocks. Expired blocks are removed from the buffer, avoiding unnecessary traffic and ensuring timely client responses.

3 IMPLEMENTATION OF RS-APS

3.1 Overall Workflow

Figure 3 shows overall flowchart of the RS-APS scheme. The coordinator manages metadata information of the storage architecture, responsible for the task of making new stripes and adjusting old in-memory stripes. and the coordinator periodically updates the node's bandwidth level and blocks' heat level according to users' access records. Table 1 displays the meanings of symbols in the paper. In actual production clusters, available bandwidth levels of nodes and access heat levels of blocks are coarsely divided into high and low levels (see Section 2.1), described by numbers 1 and 2, which can meet the requirements of most applications. Of course, it can also be divided into finer granularity according to the specific workloads.

As clients' access requests flow into a distributed storage system, a new data block will be written to the memory of nodes with low value BL. New data blocks in memory come from those loaded from disk and those written by users. In addition, storage nodes periodically adjust existing memory stripes. If the heat level of the data blocks does not match the bandwidth level of the nodes where the data blocks are located, the migration process will be triggered. As shown in Figure 4, a new data block D5 is loaded from the disk of node SN2 to the memory of node SN3 with BL-1. Node SN1 with BL-2 migrates D7 of HL-1 to node SN4 with BL-1. It is worth noting

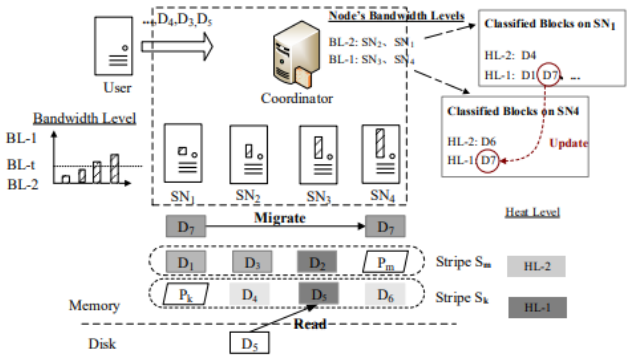

Figure 3: Overall flowchart of the RS-APS scheme.

that when hot blocks are migrated, the coordinator only needs to update the metadata information of in-memory blocks and does not need to recalculate the parity blocks in the stripe.

3.2 Procedures Implemented in Detail

3.2.1 Main Procedure. The encoding node computes *r* parity blocks, one parity block is retained by itself, and the remaining *r-1* parity blocks are sent to other nodes. The placement module also updates the access frequency of the blocks and the number of tasks received by the storage node during the system running process, seen in Algorithm 1

3.2.2 Subprocedure for Lazy Migration. See function Migration, during the migration process, we first spot these blocks with the highest heat and then check whether they are in low-performance nodes. If so, then the blocks are migrated from the source node to a high-performance node; otherwise, no migration tasks are needed. Furthermore, for source nodes, we set the lease period to those block copies labeled deleted to achieve lazy migration. These replicas will remain in the source node's memory until the lease expires.

4 Experiments

4.1 Experimental Environment

The cluster consists of one coordinator node, one proxy node, and 10 storage nodes interconnected via a Cisco Catalyst 2960-S gigabit switch. The coordinator node and agent node are Dell R720, equipped with a dual-core.

Xeon E5-2609 2.4 GHz CPU, 32 GB DDR3 memory, and IB 10 gigabit network adapters. Each storage node is equipped with an Intel E5800 3.2 GHz CPU, 3 GB DDR3, and an enterprise-class SATA2.0 disk WD1003FBYX. All storage nodes run Linux kernel 2.6.28.10 (x86_64). Coordinator nodes and proxy nodes run the Linux kernel 4.0-31-generic(x86_64).

4.2 Experimental Prototype.

To simulate the load in real life more realistically, YCSB is used to compose the trace file that conforms to the Zipf distribution. Here, we set the Zipf parameter to 0.8 and send a total of 10 W read requests made of 1K different blocks. To fairly test the performance

Table 1: Symbols and Descriptions.

Symbol	Description
Tw	Time window unit
N	Number of nodes in a storage cluster
blkSize	Block size
Sw	Sliding window to control request arrival rate
k, r	The number of data blocks and parity blocks in a stripe
SetSN	Node set of high performance
HL	Blocks' heat level in memory
BL	Nodes' available bandwidth level
SNBlk,m/d	The node whose memory/disk holds target blocks
iTw, jTw	Timestamp for blocks being migrated and expired respectively
kTw	Lease period measured by Tw
srcId,dstId	The ID for source node or destination node during migration
Srow	The rowth stripe

Algorithm 1 function Migration

```
1: function Migration(HL, BL)
2: ◃ HL: Blocks' heat level ◃BL: Nodes' bandwidth level
3: for Blk ∈ TopK in-memory blocks do
4: if H (Blk) ≠ BL(SNBlk,m) then
5: Setmig ← Blk
6: end if
7: end for
8: for Blk ∈ Setmig do
9: Select a node SNdstld from SetSN
10: if Blk is invalid on SN then
11: Send block Blk from node SNsrcld, to node SNdstld;
12: Set lease info for Blk on SNsrcld,
13: end if
14: Update mapping between block Blk and node SNdstI;
15: end for
16: end function
```

Algorithm 2 RS-APS

```
1: procedure RS-APS(Seq, H L, BL)
2: Initialze T imer
3: while Seq is not over do
4: if T imer ≥ Tw then
5: Update H L for in-memory blocks
6: Update BL for storage nodes
7: Migration(HL, BL)
8: Reset T imer
9: end if
10: if Requested block Blk is in memory then
11: Update(Fre (Blk), Fre (SNBlk,m))
12: else
13: if Blk is on disk then
14: Read Blk from disk in node SNBlk,
15: else
16: Write Blk to disk in a node selected from SetSN
17: end if
18: Select a destination node SN from SetS , with SN ≠ SNBlk,d ;
19: Send Blk to the memory of SN
20: Send Blk to encoding node ENrow of stripe Srow
21: Update(Fre (Blk), Fre (SNBlk,m))
22: end if
23: if EN row has received k data blocks in stripe Srow then
24: Generate r parity blocks
25: Keep a parity block in local memory
26: Dispatch other r − 1 parity blocks among remote memories
27: Discard the k data blocks
28: row ← row + 1
29: end if
30: end while
31: end procedure
```

of the five schemes, data are evenly distributed to disks of the cluster's nodes in the initial configuration, and the memory space is empty. In addition, to test the ability of the system to deal with fluctuating load and high pressure, the sliding window is introduced to refer to the maximum number of uncompleted requests thus far. The erasure encoding process is completed via the Jerasure code library to generate parity blocks. $(k + r , k)$ RS encoding is adopted to organize the in-memory data. By default, the configurations are as follows: in a stripe, the number of data blocks k is 4, the number of parity blocks r is 1, the number of nodes in the cluster N is 6, the block size is 16 KB, the sliding window is 12, and the time window Tw is 10 ms. The k in the kTw of the RS-APS+ scheme is 10.

The experiment tested the above placement schemes. The performance indicators are as follows: average response delay, system throughput, and system load imbalance. A response delay is measured depending on the time consumption between issuing requests and receiving data. The skew of the system's load is quantified by a skew factor, which can be calculated by formula (2). System throughput is the ratio of the number of responses and total time consumption. Each experiment was repeated 5 times, and the statistical results are the mean of 5 measurements. Besides, we evaluated the effectiveness of the optimized design through network traffic

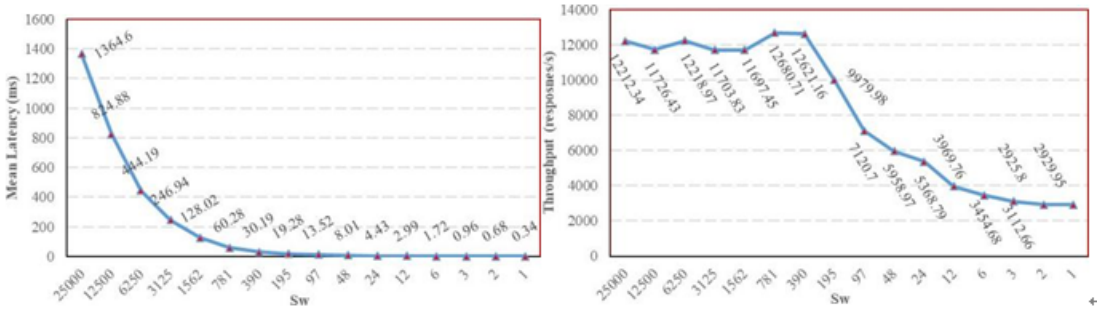

Figure 4: Simulation results for the network.

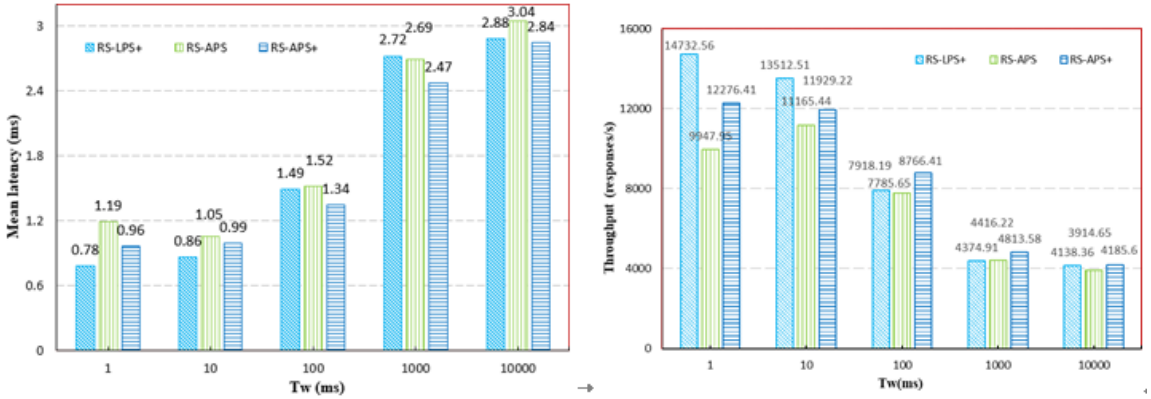

Figure 5: Comparison of response delay and system throughput for three dynamic placement schemes with different *Tw*.

overhead.

$$\lambda = \frac{L_{max} - L_{avg}}{L_{avg}} \times 100 \quad (2)$$

4.3 Experimental Results and Analysis

4.3.1 Impacts of Sliding Window Sw. To simulate the arrival rate of requests to the system, the sliding window mechanism in TCP network transmission is introduced, which is defined as the difference between the number of requests sent thus far and the number of requests completed thus far. The effect of sliding window *Sw* on the response latency and system throughput is studied for RS-LPS. Figure 4 shows the system throughput and average user response delay under different *Sw*.

When the sliding window *Sw* is gradually decreased exponentially from 2.5 W to 2, the access delay for erasure-coded in-memory systems also decreases gradually from 1.36 s to less than 0.4 ms. As the access pressure on the storage system decreases, the access delay queue of storage nodes shortens, resulting in reduced waiting time for requests. This allows the storage system to process access requests quickly, significantly reducing access delay.

However, reducing the sliding window value exponentially does not lead to a monotonic change in system throughput. When *Sw* is between 2.5 W and 390, the system throughput remains basically unchanged. Below 390, the throughput decreases with decreasing sliding window size, but not exponentially. Since system throughput depends on both task arrival rate and task execution time, improving either can enhance system throughput. For the scheme RS-LPS, some nodes are designated for storing encoded data only, which concentrates response pressure on a subset of nodes. *Sw* is set to 12 in subsequent experiments, as the current system configuration can handle such access pressure by adjusting load distribution through policies.

4.3.2 Impacts of time window Tw. Figure 5 shows the average user response time and system throughput of the three dynamic schemes under the conditions $Sw = 12$, $N = 6$, $K = 4$, $r = 1$, and $blkSize = 16KB$.

When *Tw* is set to 1 ms and 10 ms, RS-LPS+ exhibits slightly better user response performance compared to RS-APS+ and RS-APS, with smaller average user response delays. The advantage of RS-LPS+ is attributed to its placement of new blocks on high-performance nodes during stripe generation, fortunately forming even distribution of blocks with various heat levels in the cluster without migration cost. However, as *Tw* increases, the advantage diminishes. For example, at 10 ms, RS-LPS+ reduces average response delay by 18.1% and 13.13% compared to RS-APS and RS-APS+, respectively. But as *Tw* further increases, RS-APS+ performs best among the three, and RS-LPS+ and RS-APS exhibit similar performance.

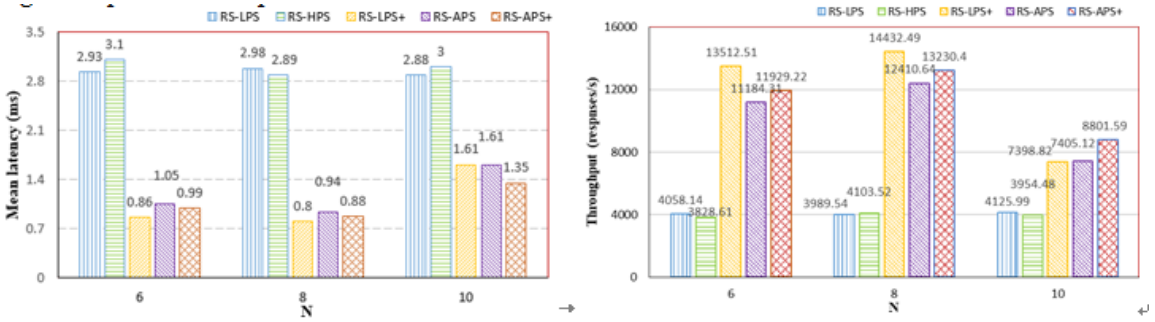

Figure 6: Comparison of response delay and system throughput for placement schemes with different *N*.

Similarly, when *Tw* is set to 1 ms and 10 ms, RS-LPS+ shows smaller average user response delays than RS-APS and RS-APS+, resulting in slightly higher overall system throughput. For instance, at 1 ms, RS-LPS+ achieves 1.48x and 1.2x throughput compared to RS-APS and RS-APS+, respectively. However, as *Tw* increases, RS-LPS+'s advantage diminishes. At 10 ms, RS-LPS+ achieves 1.21x and 1.13x throughput compared to the other two schemes, respectively. Even at 100 ms, RS-APS+ outperforms RS-LPS+ by 1.11x in system throughput.

As *Tw* continue to enlarge, the performance of RS-LPS+, RS-APS, and RS-APS+ becomes similar to the static placement scheme RS-LPS. When *Sw* is set to 10 s or larger, the system experiences fewer than 2 time windows, leading to delayed migration and uneven load distribution, eventually resembling RS-LPS. Moreover, the system performance does not consistently improve as *Tw* shrinks. Setting smaller *Tw* allows the system to adjust load distribution more frequently, but it also increases the total number of migrations, wasting cluster bandwidth.

4.3.3 Impacts of Storage Nodes N.. In terms of system performance (see Figure 6), dynamic placement schemes significantly outperform static schemes. RS-LPS+ performs best with 6 and 8 storage nodes, showing average user response delay less than a third of RS-LPS and RS-HPS, and system throughput over three times higher. At 10 nodes, RS-APS+ has the smallest response delay and highest throughput, with 16.15% smaller delay and 1.19x higher throughput compared to RS-LPS+ and RS-APS.

Regarding scalability, except for RS-LPS, all schemes show improvements when increasing storage nodes from 6 to 8. At 8 nodes, RS-HPS, RS-LPS+, RS-APS, and RS-APS+ have user response delays 6.77%, 6.98%, 10.48%, and 11.11% less than at 6 nodes, respectively. Throughput also increases by 1.072x, 1.068x, 1.11x, and 1.11x. However, with 10 nodes, RS-HPS remains nearly unchanged, while RS-LPS+, RS-APS, and RS-APS+ see significant degradation, with response delays increasing by 101.25%, 71.28%, and 53.41%, respectively. The less performance decline in RS-APS+ and RS-APS compared to RS-LPS indicates the effectiveness of dynamically migrating hot data blocks to high- performance nodes. In RS-APS+, the performance decline is least severe due to the lease mechanism, which allows multiple high-performance nodes to handle hotspots.

From the perspective of data block and load distribution (see Figure 7), RS-LPS shows extreme unevenness because it selects only k + r nodes for stripe block placement. RS-HPS, using circular

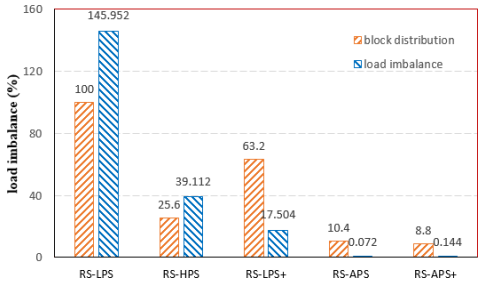

Figure 7: Imbalance rate for block distribution and response distribution when the number of storage nodes is 8.

placement, achieves more uniform block distribution. The skewing factor ensures that in-memory and on-disk copies are on different nodes. RS-LPS+ has a very uneven block distribution with a 63.2% inclination factor, but the load distribution is balanced at 17.5%, due to heat diversity among blocks. For RS-APS and RS-APS+, block distribution statistics are completed at the trace's end, and their migration processes help achieve load balance.

4.3.4 Impacts of Data Blocks k in A Stripe. In this cluster of 10 storage nodes, the system performance is minimally affected by the number of data blocks *k* (see Figure 8). For *k* values of 4, 6, and 8, the average user response delay and system throughput of each scheme almost remain unchanged. In RS-LPS, data blocks are evenly distributed to the top $k + r$ nodes, which may still result in centralized distribution of hot data blocks to a few nodes. In RS-HPS, data block distribution remains uniform regardless of *k*. For dynamic placement schemes RS-LPS+, RS-APS, and RS-APS+, the remains uniform regardless of *k*. For dynamic placement schemes RS-LPS+, RS-APS, and RS-APS+, the formation process of new stripes in memory has little impact due to the read-intensive load. RS-APS+ performs the best, followed by RS-APS, with system throughput reaching 1.21x that of RS-LPS+.

4.3.5 Impacts of Lease Peroid kTw for RS-APS+. Figure 9 shows the impact of lazy migration (see 3.2.2) on system performance, where $kT\,w$ is set to 1, 5, 10, 100, and 1000 time windows. As shown in Figures 9(a) and 9(b), average latency decreases and system throughput increases with longer leases, although the effect size is smaller. Lazy migration effectively reduces response delay by keeping blocks in

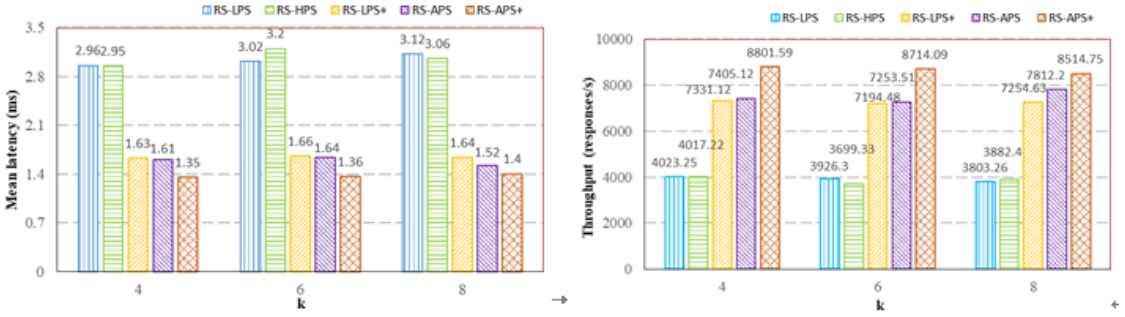

Figure 8: Comparison of response delay and system throughput for three dynamic placement schemes with different *k*.

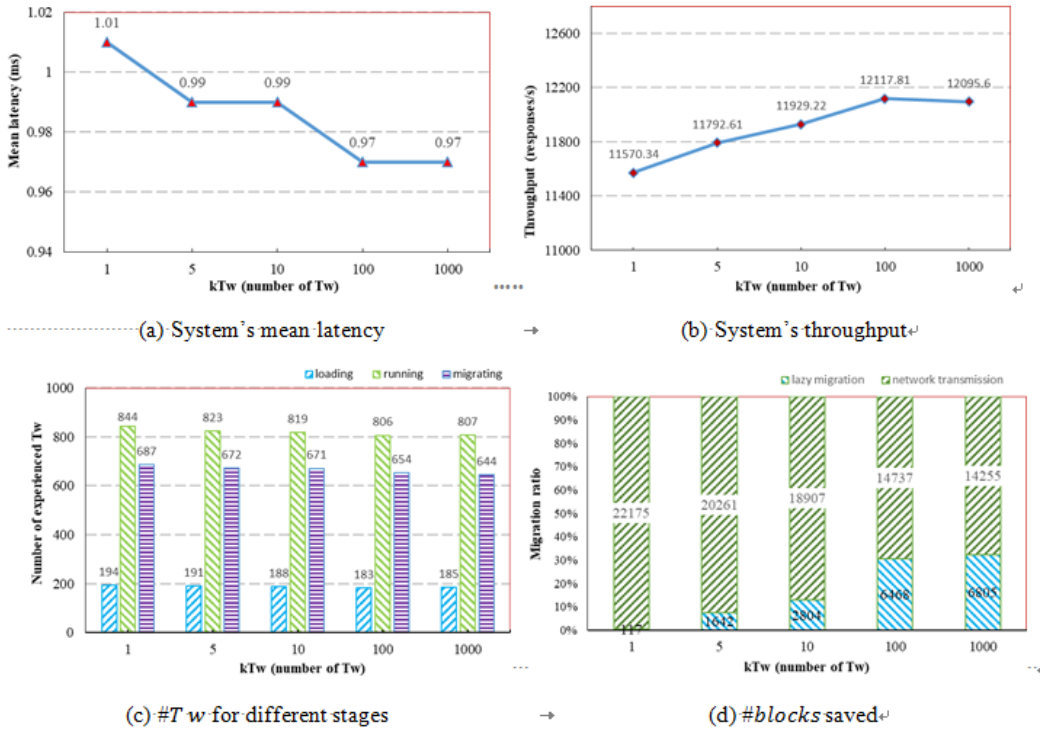

Figure 9: Characteristics of RS-APS+ as *kT w* changes.

source nodes longer, mitigating blocking requests caused by redirection. Additionally, Figure 9(d) indicates that increasing the lease from 1 Tw to 1000 Tw saves 1%, 7.5%, 12.9%, 30.5% and 32.3% of network transmission blocks, respectively. When *Tw* is 10 ms, blocks with a 1000 Tw lease always reside in source nodes' memory, aiding in faster response during redirection. The performance of RS-APS+ is close to RS-APS when the lease is 1 Tw, as blocks are retained in source nodes for a short time before deletion. Figure 9(c) shows that the number of Tw experienced during loading, running, and migrating processes remains similar as the lease increases. Lazy migration helps save network traffic with minimal impact on the experienced periods for these processes.

5 RELATED WORK

Load Balancing: Various approaches address time-varying workloads, including redundancy choices and data migration. Scarlett [1] spreads files based on access patterns to avoid hotspots in MapReduce scenarios. NetRS [10] uses in-network replica selection for latency-critical data stores. EC-Cache [8] employs erasure coding for object storage achieve high availability. DistCache [7] leverage independent hash functions for multi-layer hierarchical cache allocation and the "power of two choices" for query routing, while NetCache [6] leverages programmable switches for on-path caching. SPORE [5] redistributes hot objects, and MBal [3] handles hotspots

with multi- phase mechanisms. RS-APS focuses on dynamic encoding and migration to improve efficiency and availability based on servers' load and popularity skewness in distributed storage system.

Erasure Coded Storage: Erasure coding is widely used in large-scale distributed systems. Some systems [2], [11], [9] employ erasure coding for highly available key-value stores. To reduce encoding overhead, researchers have designed new codes or exploited algebraic properties. Our work combines adaptive selective replication with dynamic data migration in erasure coded systems to adapt to changing workloads and access patterns.

6 CONCLUSIONS

In this work, we propose an adaptive placement scheme for in-memory erasure coded clusters to achieve high availability in large-scale distributed systems, crucial for time-sensitive applications in the big data era. Our scheme evaluates node bandwidth and data heat levels to place online encoded blocks on high-performance nodes. It introduces a migration procedure for hot data and a lease mechanism to reduce network overhead. Experimental results show enhancements in load balancing, scalability, and availability. Our scheme is also effective for read-intensive applications and suitable for large-scale production clusters. Our research contributes to the management of big data to some extent.

While our focus is on handling system heterogeneity due to external load fluctuations, production clusters also face heterogeneity from infrastructure configuration and background processes. Future work may involve refining evaluation techniques and studying adaptive adjustment techniques based on real-time cluster performance.

References

[1] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. 2011. Scarlett: coping with skewed content popularity in mapreduce clusters. In Proceedings of the sixth conference on Computer systems. 287–300.

[2] Haibo Chen, Heng Zhang, Mingkai Dong, Zhaoguo Wang, Yubin Xia, Haibing Guan, and Binyu Zang. 2017. Efficient and available in-memory KV-store with hybrid erasure coding and replication. ACM Transactions on Storage (TOS) 13, 3 (2017), 1–30.

[3] Yue Cheng, Aayush Gupta, and Ali R Butt. 2015. An in-memory object caching framework with adaptive load balancing. In Proceedings of the Tenth European Conference on Computer Systems. 1–16.

[4] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing. 143–154.

[5] Yu-Ju Hong and Mithuna Thottethodi. 2013. Understanding and mitigating the impact of load imbalance in the memory caching tier. In Proceedings of the 4th annual Symposium on Cloud Computing. 1–17.

[6] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. Netcache: Balancing key-value stores with fast in-network caching. In Proceedings of the 26th Symposium on Operating Systems Principles. 121–136.

[7] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. {DistCache}: Provable Load Balancing for {Large-Scale} Storage Systems with Distributed Caching. In 17th USENIX Conference on File and Storage Technologies (FAST 19). 143–157.

[8] KV Rashmi, Mosharaf Chowdhury, Jack Kosaian, Ion Stoica, and Kannan Ramchandran. 2016. {EC-Cache}:{Load- Balanced},{Low-Latency} Cluster Caching with Online Erasure Coding. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 401–417.

[9] Dipti Shankar, Xiaoyi Lu, and Dhabaleswar K Panda. 2017. High-performance and resilient key-value store with online erasure coding for big data workloads. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 527–537.

[10] Yi Su, Dan Feng, Yu Hua, Zhan Shi, and Tingwei Zhu. 2020. An In-Network Replica Selection Framework for Latency-Critical Distributed Data Stores. IEEE Transactions on Cloud Computing (2020).

[11] Matt MT Yiu, Helen HW Chan, and Patrick PC Lee. 2017. Erasure coding for small objects in in-memory kv storage. In Proceedings of the 10th ACM International Systems and Storage Conference. 1–12.