Hindawi
Scientific Programming
Volume 2021, Article ID 9910942, 13 pages
https://doi.org/10.1155/2021/9910942

Research Article

ER-Store: A Hybrid Storage Mechanism with Erasure Coding and Replication in Distributed Database Systems

Zijian Li and Chuqiao Xiao

School of Software Engineering, East China Normal University, Shanghai 200062, China

Correspondence should be addressed to Zijian Li; 51194501053@stu.ecnu.edu.cn

Received 22 March 2021; Revised 12 July 2021; Accepted 26 July 2021; Published 10 September 2021

Academic Editor: Jiangbo Qian

In distributed database systems, as cluster scales grow, efficiency and availability become critical considerations. In a cluster, a common approach to high availability is using replication, but this is inefficient due to its low storage utilization. Erasure coding can provide data reliability while ensuring high storage utilization. However, due to the large number of coding and decoding operations required by the CPU, it is not suitable for some frequently updated data. In order to optimize the storage efficiency of the data in the distributed system without affecting the availability of the data, this paper proposes a data temperature recognition algorithm that can distinguish data tablets and divides data tablets into three types, cold, warm, and hot, according to the frequency of access. Combining three replicas and erasure coding technology, ER-store is proposed, a hybrid storage mechanism for different data types. At the same time, we combined the read-write separation architecture of the distributed database system to design the data temperature conversion cycle, which reduces the computational overhead caused by frequent updates of erasure coding technology. We have implemented this design on the CBase database system based on the read-write separation architecture, and the experimental results show that it can save 14.6%–18.3% of the storage space while meeting the efficient access performance of the system.

1. Introduction

With the rapid development of technologies such as big data, cloud computing, and the Internet of Things, as well as the diversification of data collection technologies, the amount of data has grown dramatically, which has made data storage methods biased towards distributed storage [1]. However, hardware failures and software upgrades of cheap storage devices often lead to storage node failures, which bring about data storage reliability problems [2, 3].

In order to ensure the reliability of the data, the common redundancy mechanism of distributed storage systems is the multiple replica strategy [4–6]. For example, Google's GFS [7] creates three replicas of each data chunk and ensures that all replicas of each chunk are not distributed on the same rack. Although this strategy of replication fully guarantees high availability of the data, it requires $M+1$ times storage space to tolerate M failures. This brings enormous storage overhead.

In recent years, with the improvement of CPU computing power and the slow development of storage, erasure coding has attracted people's interest as a new storage solution [8–11]. By dividing the original data object into blocks and encoding them to generate parity blocks, erasure coding can produce lower storage overhead while providing the same storage reliability. For example, in Facebook's warm storage system $F4$ [12], the system uses RS ($K=10$ and $M=4$) erasure coding technology to encode and store data. Compared with the traditional replication solution, this technology increases the fault tolerance of the system from 2 to 4 and reduces the storage overhead of the system by about $1.6X$. However, the complex erasure coding technology itself has many problems, such as a large amount of CPU computing overhead brought to the system when encoding and decoding data and the need to re-encode and calculate all the fragment contents when updating data. These problems not only increase the latency of data access requests but also cause excessive network resource consumption and hard disk I/O overhead. Therefore, erasure coding technology is mainly applied in distributed storage systems for storing videos, music, and archived data [13].

Additionally, with the continuous growth of cluster data, there are huge differences in the access frequency between data [14]. For example, in the field of social media, the access frequency of recently written data is much higher than that of the data produced long ago. These data are considered hot data. Through analysis, we can find that, as time goes by, the access frequency of data that are considered hot data will gradually decrease. When accessed only a few times a week, the data become warm. In the following months, if the frequency of data access continues to decrease, these data are called cold data [15].

Understanding the hot and cold laws of data and choosing appropriate storage strategies are the focus of research in the field of distributed storage. Zhang et al. [16] distinguished between hot and cold data by file size, using erasure coding for big-byte data and using three copies for small-byte data. Mao et al. [17] used the citations of data blocks to judge the popularity of data access. Their research showed that the higher the citations, the higher the popularity of data access.

Inspired by previous studies, we realize that, for data at different temperatures, we can combine the characteristics of replication and erasure coding to adopt corresponding redundant storage solutions for different data. For hot data, which are frequently accessed, replication strategy can be applied to ensure read-write performance; for warm data, a hybrid strategy of replication and erasure coding is used to balance storage efficiency and performance [18, 19]; for cold data that are rarely accessed, an erasure coding mechanism is used to improve storage utilization [20].

This article proposes a novel hybrid storage scheme on the CBase database [21]. The scheme divides data objects into tablets, using a data temperature differentiation algorithm based on Newton's cooling law [22]. By collecting the access frequency of the tablets, the system divides the temperature of the tablets. In order to ensure the storage efficiency of the system, different storage strategies are adopted for data at different temperatures. In addition, combined with the characteristics of CBase's read-write separation architecture, we propose two different data update algorithms.

In summary, the key contributions of this paper include the following:

(i) A data temperature differentiation algorithm based on Newton's cooling law, which can effectively distinguish the temperature of the tablets and perform dynamic data temperature conversion, is introduced

(ii) A hybrid storage mechanism is proposed to choose corresponding storage strategies for data at different temperatures, which improves the storage utilization of the system while ensuring the read-write performance of the system

(iii) A thorough evaluation is completed to confirm ER-store efficiency and availability

The rest of this article is organized as follows. In Section 2, we, respectively, introduce the characteristics and applicable scenarios of replication and erasure coding and introduce the infrastructure of the CBase database system. Section 3 describes the design of ER-store. In the design, we introduce how to select different storage strategies based on data temperature. In Section 4, we propose two different data update algorithms. In Section 5, we conduct experiments to compare the difference between ER-store and the three-replica mechanism in transaction processing performance and storage efficiency. Section 6 concludes the paper.

2. Background

2.1. Replication. Replication is a commonly used solution to provide high availability [23, 24]. By duplicating multiple copies of the stored data in a system, in the event of data loss, other copies can be used to provide services, and the surviving copies can be used to repair the data. The replication mechanism ensures high availability of the data in a distributed system.

The replication solution is simple to implement. By evenly distributing the copies on each node, we can not only improve the parallel read performance of the system but also achieve load balancing of the system. However, this is at the cost of high data redundancy, requiring $M + 1$ copies to tolerate the data loss of M. For example, by introducing three copies to tolerate the loss of two data blocks, the storage utilization rate of the system is only 33.3%.

2.2. Erasure Coding. As a redundant storage solution that can effectively improve storage utilization, erasure coding divides the original file into blocks and encodes the data blocks to generate parity blocks [25, 26]. After the data are lost, the parity blocks and the remaining data blocks are used to repair the data [27–29]. A common erasure coding is Reed–Solomon code (RS-code) [30]. RS-code is based on polynomial calculation performed on Galois field GF (2^w). We usually use two tuples (K and M) to represent the selection of RS-code parameters, where K refers to the number of data blocks in a stripe and M refers to the number of parity blocks in a stripe. The fault tolerance of the stripe is M. As shown in (1), RS-code uses the Vandermonde matrix and data blocks to perform matrix operations to obtain parity blocks.

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & a_1^1 & \cdots & a_1^{K-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{M-1}^1 & \cdots & a_{M-1}^{K-1} \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_K \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_M \end{bmatrix}. \tag{1}$$

In the RS-code, the node where the data update occurs broadcasts the incremental data to the node where the parity block is located. As shown in (2), P_i' and D_i' are obtained after updating P_i and D_i, respectively. ΔD is the delta data obtained by the XOR operation of D_i' and D_i. A is the Vandermonde matrix in (1).

$$A * \begin{bmatrix} D_1 \\ \vdots \\ D_i' \\ \vdots \\ D_K \end{bmatrix} = A * \begin{bmatrix} D_1 \\ \vdots \\ D_i + \Delta D_i \\ \vdots \\ D_K \end{bmatrix} = \begin{bmatrix} P_1 \\ \vdots \\ P_i \\ \vdots \\ P_M \end{bmatrix} + \begin{bmatrix} 1 \\ \vdots \\ a_i^{i-1} \\ \vdots \\ P_{M-1}^{i-1} \end{bmatrix} * \Delta D_i = \begin{bmatrix} P_1' \\ \vdots \\ P_i' \\ \vdots \\ P_M' \end{bmatrix}. \quad (2)$$

2.3. CBase Database System. CBase database is a distributed relational database developed by Bank of Communications. As shown in Figure 1, CBase can be divided into four modules, RootServer (RS), UpdateServer (UPS), Chunk-Server (CS), and MergeServer (MS). In CBase, the data are divided into baseline data and incremental data according to the timeline. The baseline data are read-only. All modifications are updated to the incremental data, and the incremental data and baseline data are merged in the system through regular merging. The following briefly introduces the basic functions of the four modules:

RS: it manages all servers in the cluster, stores meta-information about the distribution of tablets, and manages copies. RS is generally divided into a primary node and a backup node, and strong data consistency [31] is ensured between the primary and standby nodes.

UPS: it is the only module in the cluster that can perform write operations, and it stores incremental data. UPS accepts update operations, writes them into memory, and periodically merges with baseline data. UPS is also divided into a primary node and a backup node. During deployment, the UPS process and the RS process often share a physical server.

CS: it stores baseline data. In CBase, the baseline data are sorted according to the primary key and are divided into data objects with roughly the same amount of data, called tablets. The baseline data are generally stored in the form of three replicas.

MS: it accepts and parses the sql queries and forwards the parsed sql query to CS and UPS.

3. The Hybrid Storage Mechanism

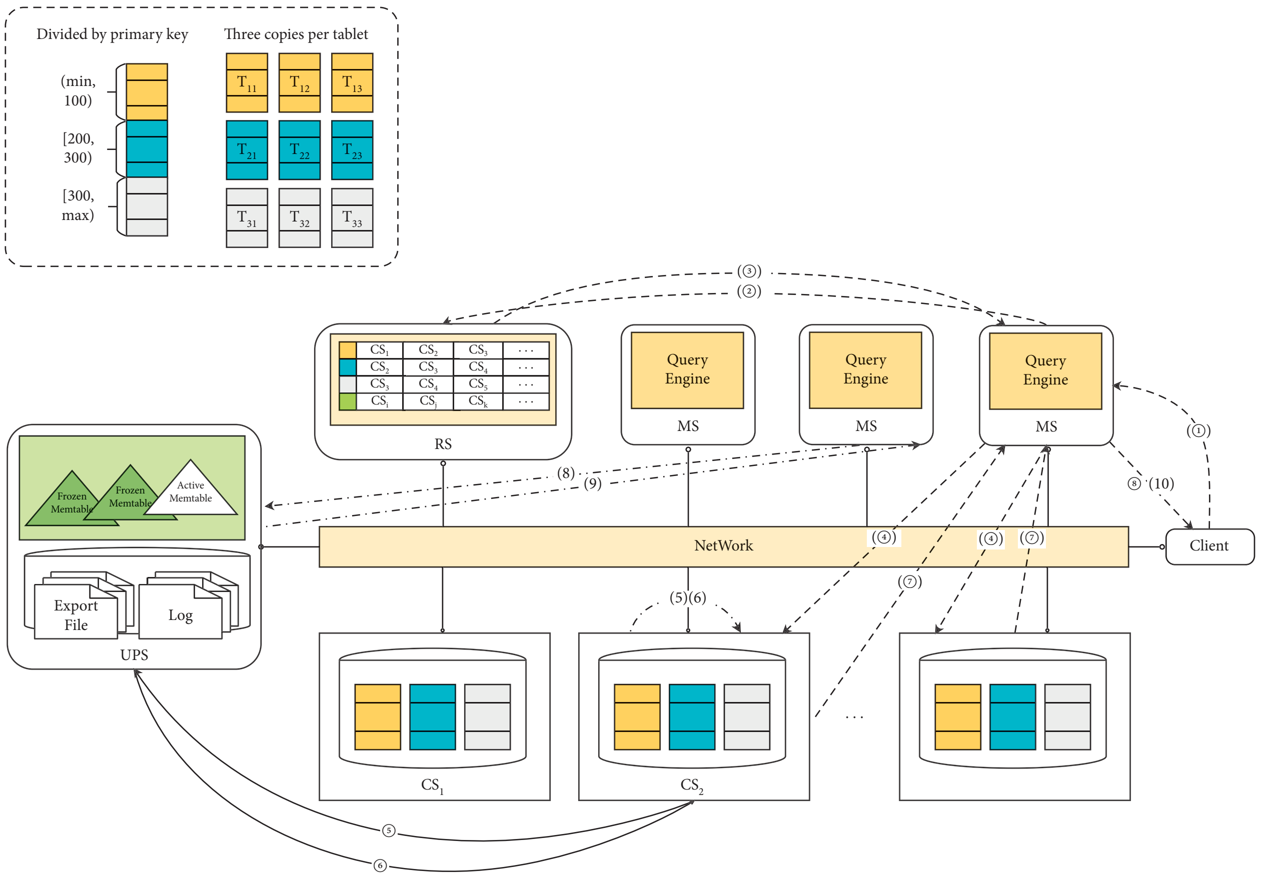
3.1. ER-Store Design. Combining the characteristics of multiple replicas and erasure coding technologies that we analyzed in Section 2, we designed an ER-store hybrid storage solution. Based on the CBase system architecture, this paper changes the original three-replica storage strategy and uses data temperature to distinguish each tablet. In ER-store, the hot tablets use the traditional three-replica strategy, and the cold tablets adopt the RS-code ($K = 4$ and $M = 2$) technology to improve the storage utilization of the system. For the warm tablet, we adopt a compromise scheme of two replicas and RS-code ($K = 4$ and $M = 1$) and find a tradeoff between the system's read-write performance and storage utilization. At the same time, taking the combination of baseline data and incremental data in the CBase system as a cycle, the data temperature differentiation algorithm is used to calculate the temperature of the tablets, and tablets' temperature is dynamically converted.

As shown in Figure 2, the design of ER-store does not change the overall architecture of CBase, but increases the tasks of each module and changes the data structure maintained.

The RS node is responsible for monitoring the status of each cluster and storing the metainformation of the tablets. As shown in Table 1, in order to add the data temperature recognition function, we modified the data structure of the original RootTable. The TS and t fields, respectively, identify the temperature status and value of the tablet; the SS identifies the redundant storage scheme of the tablet in the CS, and the RI records the location of the tablet and the data distribution of the code stripe where the tablet is located. RS determines the data temperature of each tablet through the access frequency of each tablet in the current period and the data temperature of each tablet in the previous period.

The CS node stores the baseline data and divides the baseline data into tablets of equal size through the primary key. When the system goes online, CS starts to count the access frequency of each tablet. Before the data undergo temperature conversion, the access frequency information of the tablets is sent to RS. Then, RS uses the access frequency information of each tablet, combined with the temperature differentiation algorithm, to recalculate the temperature of each tablet. Subsequently, by comparing the original temperature with the tablets, RS generates a temperature conversion table. Then, RS notifies CS to copy and encode the tablets to be converted and stores each tablet on different nodes according to the new load balancing strategy. We attempt to adopt a regenerative code coding strategy for the cold tablets, which can effectively reduce the huge network IO that exists during data recovery due to the use of erasure coding technology systems. For the warm tablets, it is an intermediate state that allows the hot tablets and the cold tablets to perform smooth and dynamic conversion. It uses two copies and a parity block to provide the same fault tolerance while ensuring the concurrent read performance of the system.

The UPS node is still used to store and manage incremental data. All data update operations do not immediately change the baseline data, but are written to the memory table, which is called memtable in UPS. The UPS realizes the separation of baseline data and incremental data and updates the parity blocks when the baseline data and incremental data are merged, which can eliminate the write request and update delay caused by erasure coding technology and save the system CPU computing resources.



(①) The client sends a query/update request to MS.

(②)(③) MS obtains data distribution information from RS.

(④) MS sends a request to the relevant CS node.

⑤⑥ CS pulls incremental data from UPS.

⑦ CS returns the fused data to the MS.

⑧ MS merges query results and return to the user.

(5)(6) CS obtains baseline data.

(7) CS sends data to MS.

(8) MS submits update operation to UPS.

(9) After the transaction is submitted, UPS informs the MS.

(10) MS notifies the client.

- - - - → Query/Update

——→ Query

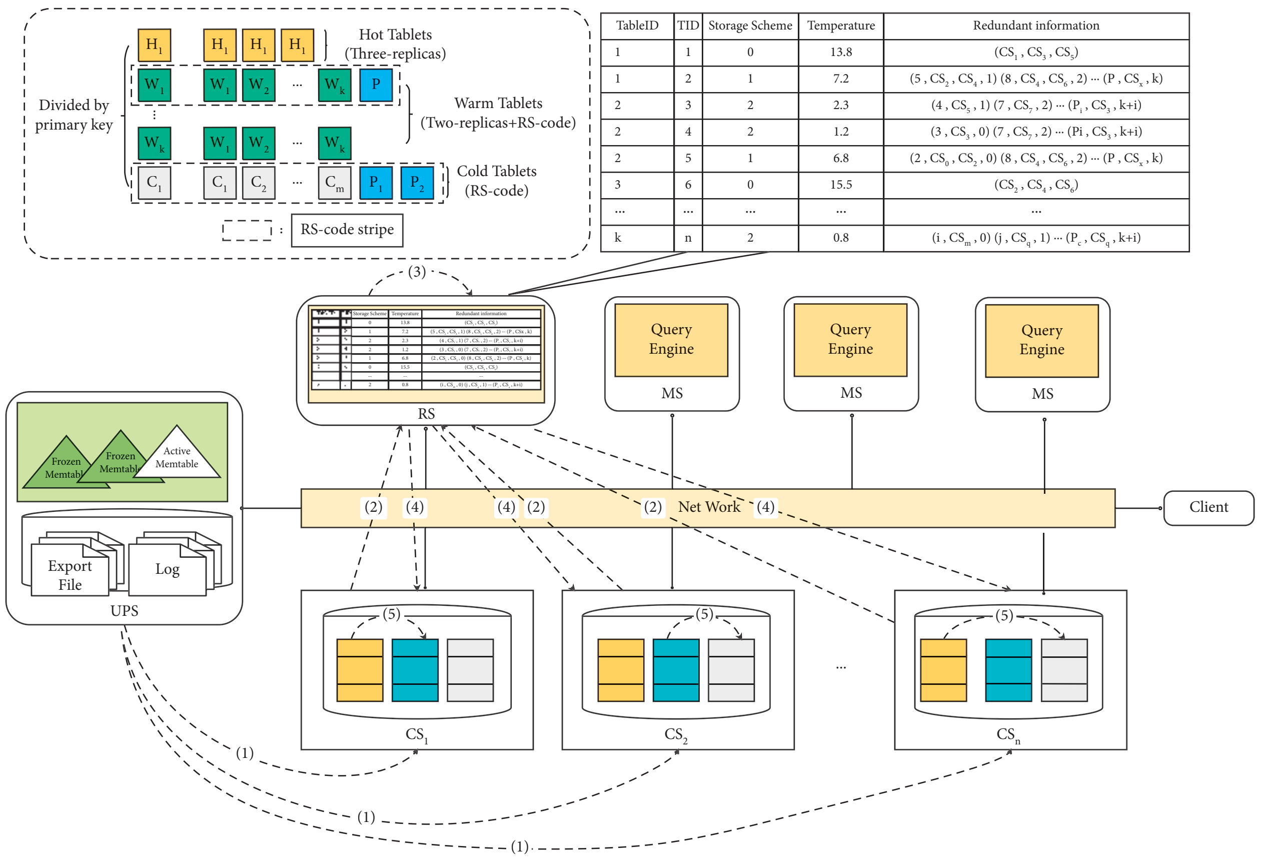
-·-·-·→ Update

Figure 1: CBase architecture. As shown in the upper left figure, original data are divided into tablets according to the primary key, and the three-replica strategy is used to store tablets. The middle part describes the query process and update process in CBase. The dotted line represents the common operation of query update. The solid line represents the data update operation, and the dashed solid line represents the data query operation.

The MS node is mainly responsible for the parsing and forwarding of sql queries. Because the cold tablets and the warm tablets are stored using erasure coding technology, the number of copies for processing data access requests is reduced, which reduces the transaction processing performance of the system. We have effectively solved this problem through reasonable data temperature distinction.

3.2. Data Temperature Recognition Algorithm. As mentioned previously, ER-store uses corresponding redundant storage strategies for data tablets of different temperatures. Therefore, the accurate determination of the tablet temperature and the dynamic conversion of the tablet temperature directly affect storage efficiency and system performance. In this section, we introduce the data temperature recognition algorithm.

At the beginning of the period T, each CS node counts the access frequency of the stored tablets. Before the start of the new period $T+1$, each CS node sends the access frequency information of each tablet in the period T to the RS. Then, RS updates the temperature of each tablet and resets the field t in the RootTable. After obtaining the new temperature of all the tablets, RS sorts all the tablets' temperatures and uses the temperature ratio threshold set by the system to redetermine the temperature type of each tablet. RS can obtain the data temperature conversion table (TCT) by comparing the new temperature type of each tablet with the original RootTable field TS. Subsequently, RS resets the field TS and modifies the redundant storage information of the corresponding tablet in the new period. In the storage information of each type of the tablet, $(\mathrm{CS}_i, \mathrm{CS}_j, \mathrm{CS}_k)$ represents the location information of CS nodes where each copy of the hot tablets exists. $(\mathrm{Tid}, \mathrm{CS}_m, \mathrm{CS}_n, K)$ represents



TableID	TID	Storage Scheme	Temperature	Redundant information
1	1	0	13.8	(CS_1, CS_3, CS_5)
1	2	1	7.2	$(5, CS_2, CS_4, 1)$ $(8, CS_4, CS_6, 2)$ ⋯ (P, CS_x, k)
2	3	2	2.3	$(4, CS_5, 1)$ $(7, CS_7, 2)$ ⋯ $(P_i, CS_3, k+i)$
2	4	2	1.2	$(3, CS_3, 0)$ $(7, CS_7, 2)$ ⋯ $(Pi, CS_3, k+i)$
2	5	1	6.8	$(2, CS_0, CS_2, 0)$ $(8, CS_4, CS_6, 2)$ ⋯ (P, CS_x, k)
3	6	0	15.5	(CS_2, CS_4, CS_6)
...	...	...	...	...
k	n	2	0.8	$(i, CS_m, 0)$ $(j, CS_q, 1)$ ⋯ $(P_c, CS_q, k+i)$

(1) CBase starts to merge regularly, and each CS node pulls incremental information from UPS.

(2) Each CS node sends the access frequency information of each tablet to the RS.

(3) RS uses the temperature recognition algorithm to recalculate the temperature of each tablet and generates a temperature conversion table.

(4) RS sends the temperature conversion table to each CS node.

(5) Each CS node changes the storage scheme of tablets according to the temperature conversion table to complete the tablets temperature conversion.

FIGURE 2: The hybrid scheme of ER-store in CBase. As shown in the upper left figure, the data are divided into tablets according to the primary key. The three-replica strategy is used for the hot tablets; the two replicas + RS-code are used for the warm tablets, and the RS-code is used for the cold tablets. The upper right corner describes the new RootTable in RS after the introduction of ER-store. The middle part describes the calculation process of the tablet's temperatures and the conversion process of the storage scheme when the data are regularly merged in the ER-store.

the IDs of other tablets, the position information of the CS nodes of the copy, and the sequence number in the coding stripe where the warm tablet exists. $(\mathrm{Tid}, \mathrm{CS}_d, N)$ represents the position information of other tablets in the coding stripe where the cold tablet is located and the sequence number in the stripe. Algorithm 1 describes the details for recognizing the temperature of the data tablet.

Based on Newton's cooling law in the physics world, we designed the tablet temperature formula related to the creation time and access frequency in ER-store. Newton's cooling law is used to describe the change in temperature of an object over time in a physical environment. That is, the decreasing speed of the object's temperature is proportional to the difference between the ambient temperature and the object's temperature, as shown in the following:

$$\frac{dT(t)}{dt} = -k(T(t) - E). \quad (3)$$

The left side represents the decreasing speed of the object's temperature, the right side $T(t)$ represents the object's temperature, E represents the environmental temperature, and k represents the proportional coefficient of the object's temperature change speed and the object's environment temperature difference. Solving the differential equation, derivation (4) of Newton's cooling law can be obtained, and the object's temperature can be obtained by the following formula:

$$T(t) = (T_0 - E)e^{-kt} + E. \quad (4)$$

The application scenario of Newton's cooling law is aimed at the physical environment. In the design of ER-store, we modified it to distinguish the temperature of the data tablet. In the system, the temperature changes between the data tablets are independent and are determined by the creation time and access frequency of the data tablets. When

Table 1: New RootTable fields.

Name	Definitions
TableID	Table ID
PK	Primary key range
Tid	Tablet ID
SS (0 is 3 replicas, 1 is 2 replicas + EC, 2 is EC)	Storage scheme
t	Temperature
TS	Temperature status
RI	Redundant information
Tidsum	Tablet size
v	Baseline data version

the data tablets exist longer and the corresponding access frequency is lower, the temperature of the data tablet is lower. Therefore, when calculating the change of the tablet temperature with time, the ambient temperature E is ignored, and the proportional coefficient T_{heat}, which represents the incremental temperature of the tablet and the access frequency, is introduced. The temperature calculation formula of the data tablet is shown in (5); $T(t_n)$ represents the data temperature of the data tablet in the period n; α represents the cooling coefficient; and F_{n-1} represents the access frequency of the tablet in the $n-1$ period.

$$T(t_n) = T(t_{n-1})e^{-\alpha(t_n - t_{n-1})} + T_{\text{heat}} * F_{n-1}. \quad (5)$$

3.3. Data Temperature Conversion Algorithm. The temperature conversion of the data tablet occurs when the baseline data and incremental data are merged in the CBase system. As shown in Figure 2, after the RS obtains the data temperature conversion table in period T, the RS traverses the temperature conversion table. The table temperature is classified to generate a set of hot, warm, and cold tablets, and corresponding redundant storage information is generated according to the load balancing strategy. Then, RS sends the generated redundant storage information to each CS node. Each CS node copies and encodes the tablet according to the redundant storage information. After all node operations have been completed, the RS node is notified to update the RI field in the RootTable and complete the data temperature conversion of the tablet. The details of tablet temperature conversion are shown in Algorithm 2.

4. Regularly Updated

As previously mentioned, ER-store uses an erasure coding strategy to encode the warm data tablet and the cold data tablet and stores the encoded parity tablet and data tablet on the CS node. When the CBase system merges incremental data and baseline data, the data tablet is updated, and the parity block in the code stripe must be updated synchronously to meet data availability. In order to adapt to different update scenarios, we have designed two algorithms for updating the check block: re-encoding algorithm and incremental encoding algorithm.

$$\begin{aligned} P'_j &= \sum_{i=1}^{k-1} G_{ji}D_i + G_{jk}D'_k \\ &= \sum_{i=1}^{k-1} G_{ji}D_i + G_{jk} - G_{jk} + G_{jk}D'_k \\ &= \sum_{i=1}^{k} G_{ji}D_i + G_{jk}\Delta D_k \\ &= P_j + G_{jk}(D'_i - D_i), j \in [1, m]. \end{aligned} \quad (6)$$

4.1. Re-Encoding Algorithm. The encoding rules are used to read out the data tablets of the entire stripe and re-encode them to generate the latest parity tablets. This method is suitable for scenarios where most data tablets in the stripe are updated.

As shown in Figure 3, in the stripe coded with RS (k, m), the data tablet D_1 merges with the incremental update data from UPS to become D'_1 (1), and then the system reads out other data tablets $D_2, D_3, \ldots, D_k$ in the stripe from the corresponding node (2). Then, these tablets and D'_1 are used to regenerate m new parity tablets $P'_1, P'_2, \ldots, P'_n$ according to the encoding rules (3). Finally, D'_1 and m new parity tablets are stored on the corresponding node (4). In this process, a total of $k+m$ times of network I/O are consumed. These are $k-1$ times of hard disk reads and $m+1$ times of hard disk writes.

4.2. Incremental Encoding Algorithm. The second method to update the parity tablets is incremental update, which is suitable for scenarios where a small number of data tablets in a stripe are updated. It can be seen from (1) that $P_j = \sum_{i=1}^{k} G_{ji}D_i$; when the UPS updates data tablets D_k to D'_k, the formula can be derived to obtain the second update parity tablets (6). It can be seen from (6) that the value of the new parity tablet P'_j is only related to the old parity tablet P_j and the data tablet increment ΔD_k. Therefore, an incremental update method can be derived, which updates the parity fragments by calculating the incremental information of the data fragments.

Figure 4 shows the process of ER-store using the incremental update method to update a data tablet in a stripe. After the CS node where the data tablet D_i is located obtains the incremental update data of tablet D_i from the UPS, the

```
Require:
  $T(t_{n-1})$, $F(t_{n-1})$ of each tablet, $T_{\text{heat}} = 1$, $\alpha$;
  Thresholds of each temperature-type tablet set by the system, $Td_{\text{hot}}$, $Td_{\text{warm}}$, $Td_{\text{cold}}$;
Ensure:
  The data temperature conversion table in $n$ period, TCT;
 (1) : Initialize Tidsum is the sum of tablets;
 (2) : Initialize Tid is the ID of the tablet;
 (3) : Initialize $t_{n-1}$ is the current period and $t_n$ is the new period;
 (4) : for Tid = 1 to Tidsum do
 (5) :    $T(t_n)_{\text{Tid}} = T(t_{n-1})_{\text{Tid}} * e^{-\alpha(t_n - t_{n-1})} + T_{\text{heat}} * F(t_{n-1})_{\text{Tid}}$
 (6) : end for
 (7) : Use any sorting algorithm to rank the tablet temperature in the descending order;
 (8) : $s_{\text{Tid}}$ is the sorted position of this tablet;
 (9) : $T_{\text{size}}$ is the size of TCT;
(10) : for $s_{\text{Tid}} = 1$ to $T_{\text{size}}$ do
(11) :    if $s_{\text{Tid}} \le Td_{\text{hot}} *$ Tidsum then
(12) :       $TS_{\text{tid}} = 0$;
(13) :       if $TS_{\text{Tid}} \ne SS_{\text{Tid}}$ then
(14) :          $TCT$.insert(Tid, $TS_{\text{Tid}}$);
(15) :       end if;
(16) :    else if $Td_{\text{hot}} *$ Tidsum $< s_{\text{Tid}} \le Td_{\text{warm}} *$ Tidsum then
(17) :       $TS_{\text{Tid}} = 1$;
(18) :       if $TS_{\text{Tid}} \ne SS_{\text{Tid}}$ then
(19) :          $TCT$.insert(Tid, $TS_{\text{Tid}}$);
(20) :       end if;
(21) :    else
(22) :      $TS_{\text{Tid}} = 2$
(23) :       if $TS_{\text{Tid}} \ne SS_{\text{Tid}}$ then
(24) :          $TCT$.insert(Tid, $TS_{\text{Tid}}$);
(25) :       end if;
(26) : end for
(27) : return TCT;
```

Algorithm 1: Data temperature recognition algorithm.

system reads the original data tablet D_i from the node and performs the XOR calculation with the update data D_i' to obtain the update increment ΔD_i. Then, the CS node sends ΔD_i to the nodes where all the parity tablets are located. The parity nodes read the original parity tablets, calculate the new parity tablets according to (6), and store the new parity tablets. Finally, the CS node stores D_i' and completes the parity tablet update process.

5. Evaluation and Comparisons

In this section, we conducted a series of performance tests on the CBase system that was applied to the ER-store storage mechanism to evaluate the relationship between storage utilization and transaction processing performance. In our experiment, all the tests were performed on a CBase cluster built by 10 PC servers. The node configuration in the cluster was composed of an 8-core 2.4 GHz (Intel(R) Xeon(R)) E5-2620 processor, 64 GB memory, a 2 TB disk, and a 10 Gbps network card. Each node ran Linux release 7.5.1804.

In the cluster, the CS module was deployed on each server, two servers were deployed with UPS and RS modules, and four were used to deploy MS. In addition, the hot tablets used the three-replica strategy storage, the warm tablets used the RE (4, 4, 1) hybrid storage scheme, and the cold tablets used the RS (4, 2) erasure coding scheme. As a modular multithreaded benchmarking tool, sysBench was used for testing in this experiment. In addition, according to the well-known 80/20 rule [32], in most data access scenarios, 80% of the data access of the system is concentrated on 20% of the data, we set the threshold of hot data to 20%, and for cold data, three sets of thresholds (20%, 40%, and 60%) were set to simulate complex data access scenarios.

5.1. Storage Efficiency. Our first experiment evaluated the disk consumption of the CBase system using the ER-store storage mechanism. We used three replicas and the ER-store mechanism to store the same data in the cluster. Figure 5(a) shows the disk consumption of different storage mechanisms under different numbers of records. It can be seen from the figure that when the original three-replica mechanism was used, the storage efficiency of the system was only 33.3%, and the ER-store mechanism could significantly improve the system's storage utilization. As the proportion of cold data increased to 60%, the storage efficiency of ER-store reached 51.3%, an increase of 18%. This means that compared with the traditional three-copy storage mechanism, the system saved 1.05 PB of disk space for every 1 PB of

```
Require:
  The data temperature conversion table, TCT;
  The set of hot tablets to be converted, hotTablets;
  The set of warm tablets to be converted, warmTablets;
  The set of cold tablets to be converted, coldTablets;
 (1)  ID is the sequence number of the tuple in the TCT;
 (2)  Tid is the tablet ID to be converted in the TCT;
 (3)  $T_{size}$ is the size of TCT;
 (4)  for Id = 1 to $T_{size}$ do
 (5)    if $TS_{tid}$ == 0 then
 (6)      hotTablets.insert(Tid);
 (7)    else if $TS_{tid}$ == 1 then
 (8)      warmTablets.insert(Tid);
 (9)    else
(10)      coldTablets.insert(Tid);
(11)  end for
(12)  while Tid ∈ hotTablets do
(13)    Use three replicas to back up the tablet;
(14)    CS notifies RS to update column RI and SS = 0 of the RootTable;
(15)  * use two replicas and Reed–Solomon to store redundant tablets; */
(16) : while each 4 tablets ∈ warmTablets do
(17) :   re _ init _ n _ n _ k(rs,4,4,1,tablets);
(18) :   CS notifies RS to update column RI and SS = 1 of the RootTable;
(19) :/* use Reed–Solomon to encode redundant tablets; */
(20) : while each 6 tablets ∈ coldTablets do
(21) :   rs _ init _ n _ k(rs,6,2,tablets);
(22) :   CS notifies RS to update column RI and SS = 2 of the RootTable;
```

Algorithm 2: Data temperature conversion algorithm.

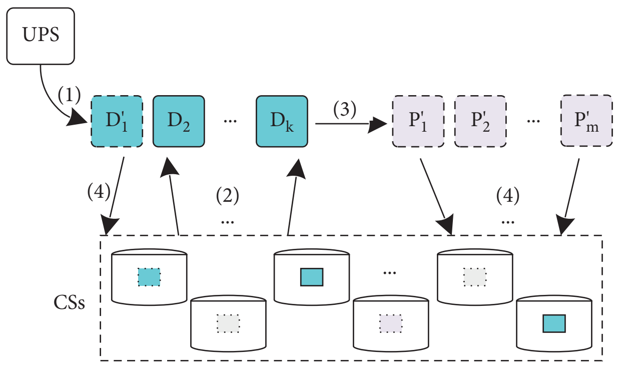


Figure 3: Re-encoding algorithm.

data stored. Therefore, the ER-store storage solution significantly improved the storage efficiency of the system.

In addition, as previously mentioned, ER-store uses the erasure code mechanism to store warm and cold data, while the erasure code uses encoding to achieve fault tolerances, which means that, as the system has higher requirements for fault tolerance, ER-store can improve the storage capacity more significantly. We verified this by setting different fault tolerances, as shown in Figure 5(b). When the fault tolerance was increased to 4, the three-replica mechanism consumed 4 PB of extra space for every 1 PB of data stored, while the ER-store consumed only 1.54 PB of extra space.

5.2. Performance. The second experiment tested the performance difference of the ER-store hybrid storage pattern for different data access scenarios with different numbers of concurrent threads. We set a fixed number of access operations and simulated two different data access scenarios: uniform distribution and Pareto distribution. In the uniform distribution scenario, all access operations were allowed to access each data tablet uniformly. According to the 80/20 rule, in a real application scenario, it is impossible for the data access operations to access each tablet evenly. Thus, we simulated the Pareto distribution scenario: let most access operations focus on a small number of hot data. We used

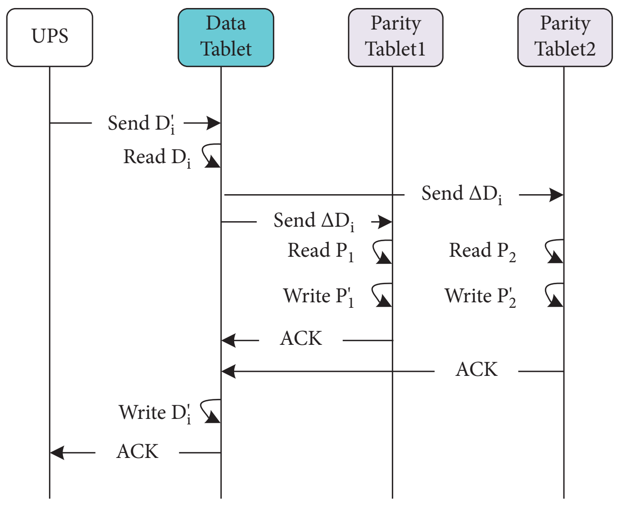
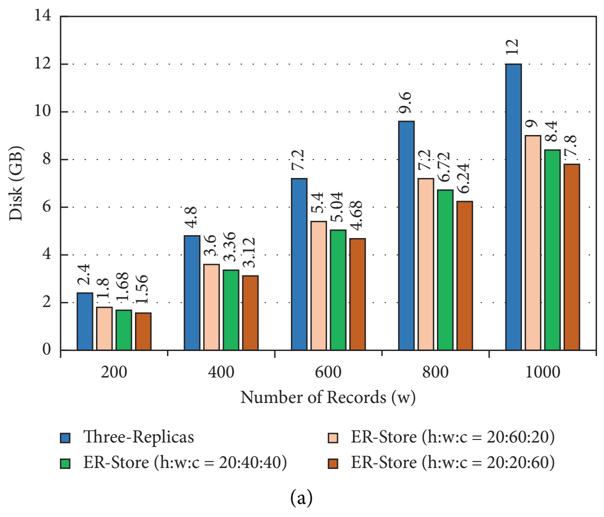


Figure 4: Incremental encoding algorithm.



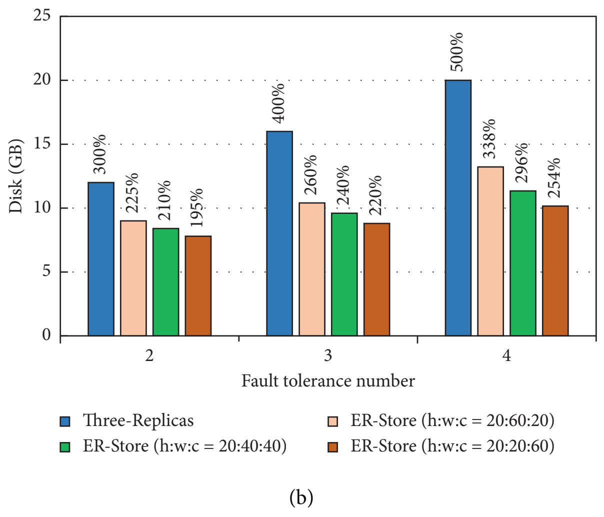


Figure 5: Disk consumption of different storage schemes. (a) Disk consumption under different record numbers for tolerating two failures. (b) Disk consumption for different fault tolerance numbers. The more the failures to be tolerated, the more the storage space ER-store will save.

transactions per second (TPS) and average access latency as metrics to measure system performance. We conducted a total of six experiments and took the average value as the final result.

The experimental results are shown in Figure 6. In the uniform distribution scenario, the performance of ER-store is significantly reduced compared to the three-replica mechanism by the two metrics of throughput rate and access latency. This may be because the system operations access each tablet uniformly, and ER-store used the RS erasure code mechanism to handle both warm and cold data. Each warm tablet replica number is reduced by 1, and the number of replicas of each cold tablet is reduced by 2. The number of replica in the system that can respond to access operations decreases by 33%–47%, which results in performance degradation. In a real data access scenario that conforms to the 80/20 rule, the system performance is not significantly affected because ER-store uses a data temperature algorithm to perform temperature partitioning of the tablets so that most of the access operations fall on the hot data with three copies and the warm data with two copies. Experiments have proven that ER-store can significantly save storage space under real data access scenarios conforming to the 80/20 rule while guaranteeing essential system performance.

Although we can determine the storage method of the data in the new cycle according to the data temperature of the previous cycle, there is still a possibility that a small part of the cold data will be accessed frequently. Therefore, we also designed related experiments to explore the impact of this sudden change in data temperature on system performance. In the experiment, the data temperature threshold is set to 2 : 2:6, the number of oltp threads is set to 60, and a fixed number of access requests is set. We let the access request to access the corresponding data in a Pareto

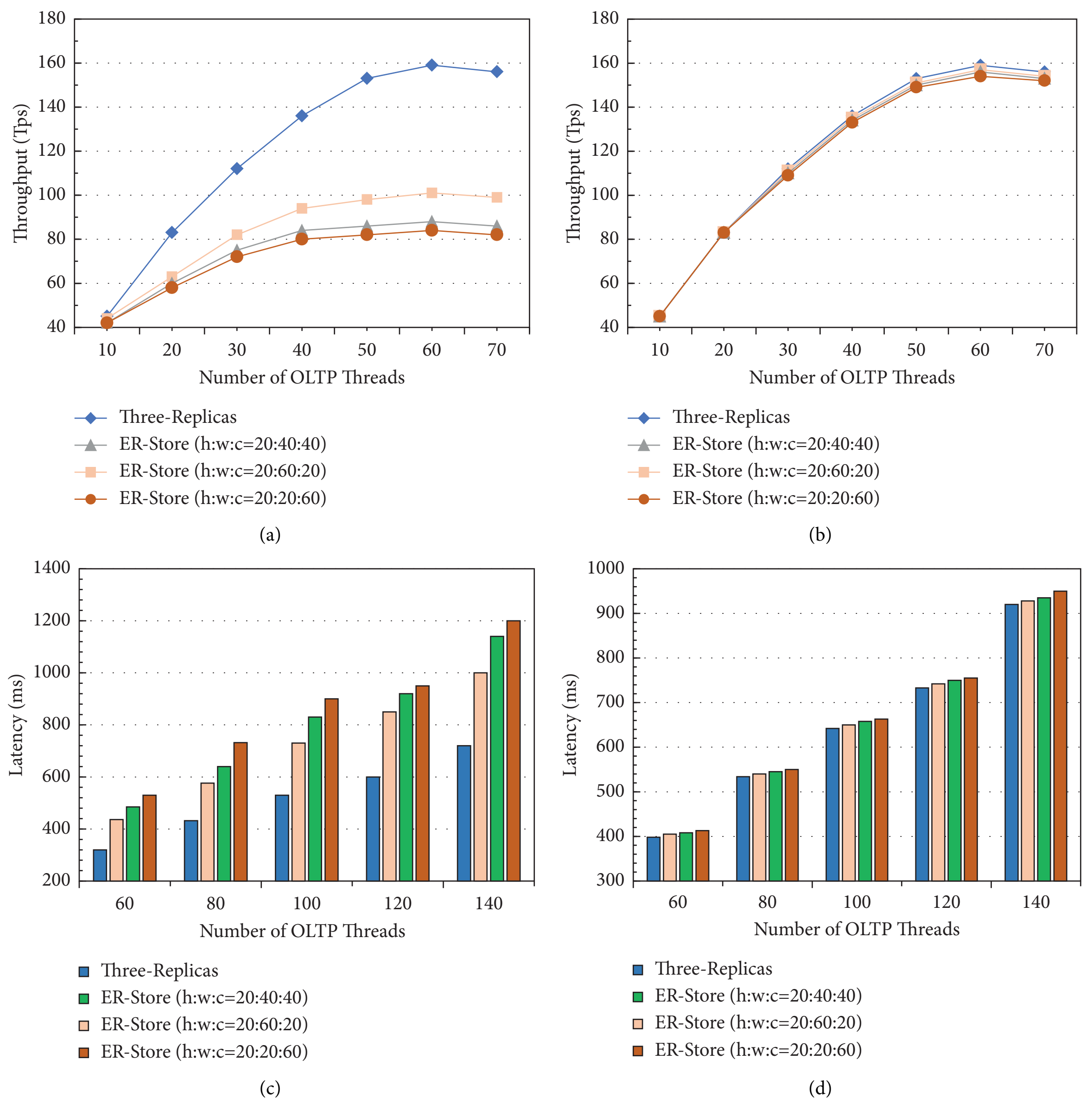


Figure 6: Comparison of the concurrent transaction processing performance between three replicas and ER-store. (a) Uniform distribution. (b) 80/20 distribution. (c) Uniform distribution. (d) 80/20 distribution.

distribution manner. In the unstable experiment, we transfer 10% of the requests to access some cold tablets at time t', causing the temperature of these tablets to rise. The result is shown in Figure 7; when the access request of the cold tablet increases, the throughput of the system decreases because there is only one data copy that accepts data access.

In order to optimize the overall performance of the system and verify the influence of the ratio of the three temperature thresholds of hot, warm, and cold on the system performance, we also perform related experiments. By setting a fixed number of access requests, the temperature threshold of the hot data is set to 20% according to the 80/20 rule, and the change of system performance is observed by changing the threshold of warm data. We evaluate the overall performance of the system from two indicators of storage efficiency and access performance. As shown in Figure 8, the data access performance and the warm data threshold are positively correlated, and the data storage efficiency and the warm data threshold are negatively correlated. Therefore, in real application scenarios, we can adjust the temperature threshold ratio of the system according to our business needs to find a balance between storage efficiency and access performance.

5.3. Update Efficiency. The third experiment evaluated the performance of two data update algorithms in the ER-store storage model in the Pareto distribution scenario. In the experiment, we set the size of the data tablet to 16 KB and wrote 500,000 update transactions to the UPS node in advance. After the update transactions were written, we manually triggered the periodic merge mechanism of the

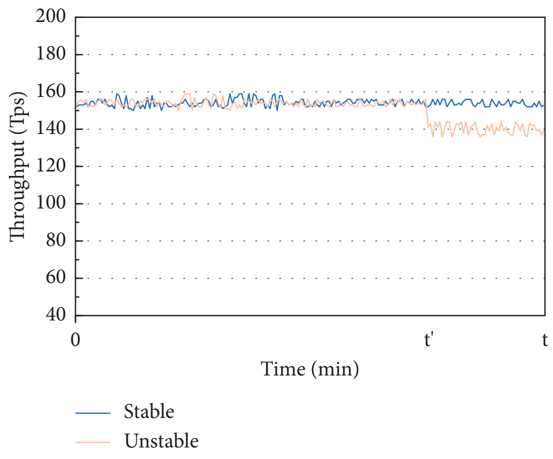


Figure 7: Performance impact of data mutation.

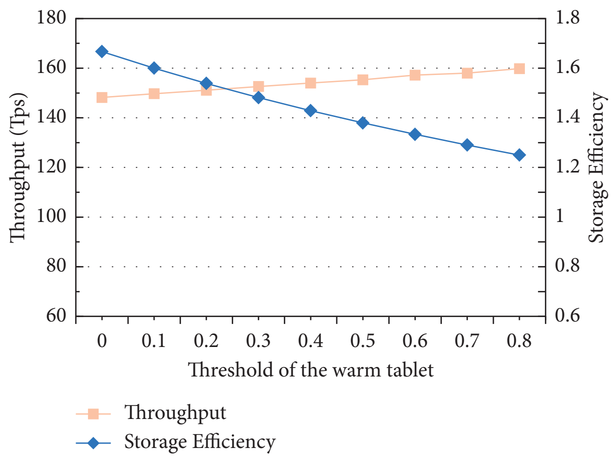


Figure 8: System performance changes with warm data threshold.

CBase database and let each CS node pull its own update data increment from the UPS. After all CS nodes completed the update operation, we evaluated the performance of the two update algorithms by counting the time consumed to complete the update. In addition, in the experiment, we also tested the performance of the two algorithms under different update scenarios by varying the number of coded stripe update tablets. As Figure 9 shows, in the RS encoding stripe with the data block of 4, the incremental encoding algorithm performed better than the recoding algorithm when the number of update tablets was less than 3. As the number of update tablets increased, the incremental encoding algorithm was inferior to the recoding algorithm.

5.4. Recovery Efficiency. The fourth experiment evaluated the performance of the system to recover data in case of node failure in the ER-store storage mode. In the experiment, we prepared 10 sysBench data tables containing 100,000 records and distributed the data evenly on each node according to the load balancing mechanism, with an average data size of 8.2 GB per node. We evaluate the recovery efficiency of ER-store under different node failure scenarios by killing different numbers of CS processes at the same time.

As shown in Figure 10, the ER-store was inferior to the three-replica mechanism in data recovery performance. This is likely because the erasure code mechanism involves a large number of encode operations and a large number of network IO in data recovery. At the same time, we found that the ER-store had higher data recovery efficiency when the warm data threshold was larger. This is likely because the warm data were stored in a hybrid storage mode of replication and erasure codes, and the data recovery performance was better than cold data that are based on erasure codes.

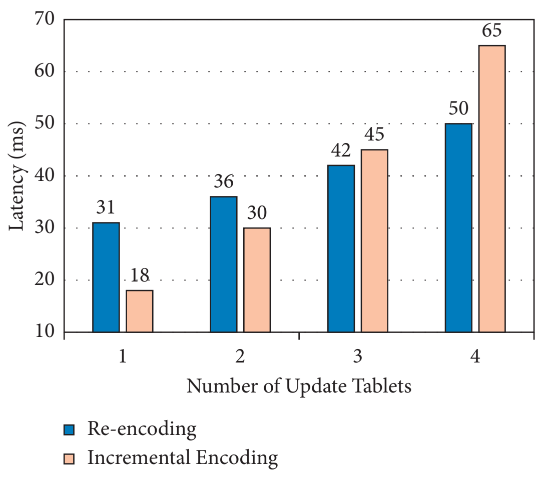


FIGURE 9: Performance comparison of two update algorithms.

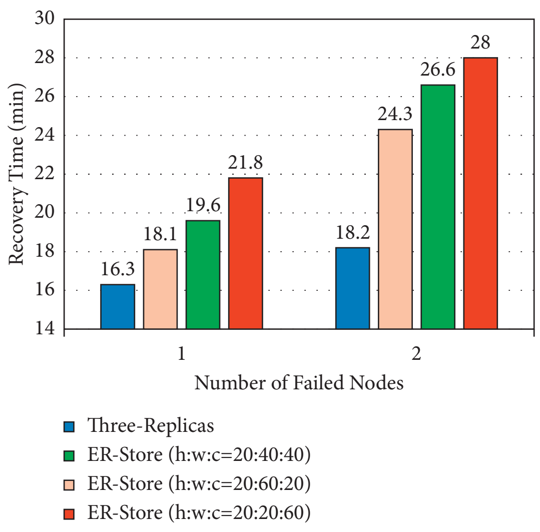


FIGURE 10: Comparison of the recovery performance between three -replicas and ER-Sstore.

6. Conclusions

Efficiency and reliability are two key features in distributed storage. This article has discussed the efficiency and reliability of the multireplicas and erasure code storage methods. We also understand the difference in access frequency between data and propose using data temperature to distinguish data. Through the division of data temperature, we have proposed an ER-store hybrid storage strategy, which combines the characteristics of replication and erasure coding to balance performance and storage utilization. We implemented this design on the CBase distributed database and conducted a series of experimental comparisons with the original three-replica mechanism of the system. Experimental results show that the ER-store mechanism can increase the system storage rate by 14.6%–18.3% while ensuring the performance of concurrent transaction processing.

Data Availability

The data used to support the findings of this study have not been made available because the data also form part of an ongoing study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded by the National Key Research and Development Project (Grant no. 2019YFB2102600) and National Natural Science Foundation (Grant nos. 61572194 and 61672233).

References

[1] M. Zwolenski and L. Weatherill, "The digital universe rich data and the increasing value of the Internet of Things," *Journal of Telecommunications and the Digital Economy*, vol. 2, no. 3, 2014.

[2] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[3] A. Roy, R. Das, H. Zeng, J. Bagga, and A. C. Snoeren, "Understanding the Limits of Passive Realtime Datacenter fault Detection and Localization," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2001–2014, 2019.

[4] B. Navin, M. Keith, B. S. Fred, and T. Sam, *The Primary-Backup Approach, Distributed Systems*pp. 199–216, ACM Press/Addison-Wesley Publishing Co, NY, USA, 2nd edition, 1993.

[5] R. V. Renesse and F. B. Schneider, "Chain replication for supporting high throughput and availability," in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6 (OSDI'04)*, pp. 91–104, USENIX Association, San Francisco, CA, USA, September 2004.

[6] J. Wang, H. Wu, and R. Wang, "A new reliability model in replication-based big data storage systems," *Journal of Parallel and Distributed Computing*, vol. 108, pp. 14–27, 2017.

[7] M. K. Mckusick and S. Quinlan, "GFS: Evolution on fast-forward," *ACM Queue*, vol. 7, no. 7, pp. 10–20, 2009.

[8] H. Andreas, M. Alan, and D Peter, "Glacier: highly durable, decentralized storage despite massive correlated failures," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*, pp. 143–158, USENIX Association, Boston, MA, USA, May 2005.

[9] H. Cheng, S. Huseyin, X. Yikang et al., "Erasure coding in windows azure storage," in *Proceedings of the 2012 USENIX conference on Annual Technical Conference (USENIX ATC'12)*, USENIX Association, Boston, MA, USA, June 2012.

[10] S. Maheswaran, A. Megasthenis, P. Dimitris et al., "XORing elephants: novel erasure codes for big data," *Proceedings of VLDB Endow*, vol. 6, pp. 325–336, 2013.

[11] K. V. Rashmi, N. B. Shah, K. Ramchandran, and P. V. Kumar, "Information-theoretically secure erasure codes for distributed storage," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1621–1646, 2018.

[12] M. Subramanian, L. Wyatt, and R. Sabyasachi, "F4: Facebook's warm BLOB storage system," in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation (OSDI'14)*, pp. 383–398, USENIX Association, Broomfield, CO, USA, October 2014.

[13] J. S. Plank, "Erasure Codes for Storage Systems: A Brief Primer," *Login:: The Magazine of USENIX & SAGE*, vol. 38, no. 6, pp. 44–50, 2013.

[14] T. Rebecca, M. Essam, and S. Marco, "E-store: fine-grained elastic partitioning for distributed transaction processing systems," *Proceedings of VLDB Endow*, vol. 3, pp. 245–256, 2014.

[15] S. Balakrishnan, R. Black, and A. Donnelly, "A building block for exascale cold data storage," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, pp. 351–365, Broomfield, CO, USA, October 2014.

[16] H. Zhang, M. Dong, and H. Chen, "Efficient and available in-memory KV-store with hybrid erasure coding and replication," in *Proc. of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, CA, USA, March 2016.

[17] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, "Read-performance optimization for deduplication-based storage systems in the cloud," *ACM Transactions on Storage*, vol. 10, no. 2, pp. 1–22, 2014.

[18] Y. Li, J. Zhou, W. Wang, and Y. Chen, "Re-store: reliable and efficient kv-store with erasure coding and replication," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–12, Albuquerque, NM, USA, September 2019.

[19] J. Li and B. Li, "Demand-aware erasure coding for distributed storage systems," *IEEE Transactions on Cloud Computing*, vol. 9, no. 2, pp. 532–545, 2021.

[20] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the Facebook warehouse cluster," in *Proceedings of the 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 13)*, San Jose, CA, USA, June 2013.

[21] "CBase Home Page," 2021, https://github.com/BankOfCommunications/CBASE.

[22] M. Vollmer, "Newton's law of cooling revisited," *European Journal of Physics*, vol. 30, no. 5, pp. 1063–1084, 2009.

[23] W. Yijie and L. Sijun, "Research and performance evaluation of data replication technology in distributed storage systems," *Computers & Mathematics with Applications*, vol. 51, no. 11, pp. 1625–1632, 2006.

[24] A. R. Nasibullin and B. A. Novikov, "Replication in distributed systems: Models, methods, and Protocols," *Programming and Computer Software*, vol. 46, no. 5, pp. 341–350, 2020.

[25] M. Gandelman and Y. Cassuto, "Treeplication: an erasure code for distributed full recovery under the random multiset channel," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3542–3556, 2021.

[26] Y. Hu, L. Cheng, Q. Yao et al., "Exploiting combined locality for wide-stripe erasure coding in distributed storage," in *Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 2021)*, CA, USA, February 2021.

[27] S. Mitra, R. Panta, M. R. Ra, and S. Bagchi, "Partial-parallel-repair(PPR): a distributed technique for repairing erasure coded storage," in *Proceedings of the 11st European Conference on Computer Systems (EUROSYS'16)*, London, UK, April 2016.

[28] R. Li, X. Li, P. Lee, and Q. Huang, "Repair pipelining for erasure-coded storage," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '17)*, pp. 567–579, Santa Clara, CA, USA, July 2017.

[29] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*, pp. 29–41, Cascais, Portugal, October 2011.

[30] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[31] P. Kathrin and R. Alexander, "Consistency and fault tolerance for erasure-coded distributed storage systems," in *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date (DIDC '12)*, pp. 23–32, Association for Computing Machinery, NY, USA, June 2012.

[32] M. Joseph, "J. Pareto principle home page," 2021.