electronics

MDPI

Article

CBase-EC: Achieving Optimal Throughput-Storage Efficiency Trade-Off Using Erasure Codes

Chuqiao Xiao *, Yefeng Xia, Qian Zhang, Xueqing Gong * and Liyan Zhu

Software Engineering Institute, East China Normal University, Shanghai 200062, China; 51184501163@stu.ecnu.edu.cn (Y.X.); 52184501012@stu.ecnu.edu.cn (Q.Z.); 51184501093@stu.ecnu.edu.cn (L.Z.)
* Correspondence: 52184501011@stu.ecnu.edu.cn (C.X.); xqgong@sei.ecnu.edu.cn (X.G.)

Abstract: Many distributed database systems that guarantee high concurrency and scalability adopt read-write separation architecture. Simultaneously, these systems need to store massive amounts of data daily, requiring different mechanisms for storing and accessing data, such as hot and cold data access strategies. Unlike distributed storage systems, the distributed database splits a table into sub-tables or shards, and the request frequency of each sub-table is not the same within a specific time. Therefore, it is not only necessary to design hot-to-cold approaches to reduce storage overhead, but also cold-to-hot methods to ensure high concurrency of those systems. We present a new redundant strategy named CBase-EC, using erasure codes to trade the performances of transaction processing and storage efficiency for CBase database systems developed for financial scenarios of the Bank. Two algorithms are proposed: the hot-cold tablets (shards) recognition algorithm and the hot-cold dynamic conversion algorithm. Then we adopt two optimization approaches to improve CBase-EC performance. In the experiment, we compare CBase-EC with three-replicas in CBase. The experimental results show that although the transaction processing performance declined by no more than 6%, the storage efficiency increased by 18.4%.

Keywords: erasure codes; distributed database system; hot and cold separation; storage efficiency

Citation: Xiao, C.; Xia, Y.; Zhang, Q.; Gong, X.; Zhu, L. CBase-EC: Achieving Optimal Throughput-Storage Efficiency Trade-Off Using Erasure Codes. *Electronics* **2021**, *10*, 126. https://doi.org/10.3390/electronics10020126

Received: 12 November 2020
Accepted: 5 January 2021
Published: 8 January 2021

1. Introduction

With the increasing complexity of the Internet business model, various Distributed Database Management System (DDBMS) architectures are emerging and developing. Relational DDBMSs have always been adopted by the master/slaver read-write separation architecture adaptive for large-scale and highly concurrent business scenarios [1]. Moreover, DDBMSs store massive and various data on many commodity servers daily, and the data access mechanisms and redundancy strategies of different data need dividing [2]. In any distributed system, whether it is a DDBMS or a distributed storage system, system reliability needs to be guaranteed. The reliability of data storage is ensured in part by adopting redundancy in some form, such as simple replication or a more sophisticated erasure code (EC) strategy. Some DDBMSs always use the complete data backup strategy, also known as multi-replicas strategy, to guarantee high system reliability, but they have low storage utilization [3]. Nevertheless, due to the requirements of business and user data integrity, a large amount of data must be stored and cannot be deleted [4]. As time goes on, a significant fraction of data stored in DDBMSs is rarely accessed. These data are named cold data [5]. Cold data have been identified as the fastest-growing storage segment, with a 60% cumulative annual growth rate [6].

At present, to deal with the unavailability, or loss, of data caused by error failure, existing disk array storage systems and distributed storage systems often use EC strategies, which can tolerate broader classes of failure scenarios with less extra storage overhead [7]. Disk array storage, such as the redundant array of independent disks (RAID) organizes multiple independent storage devices (HDD, SSD) into a logically continuous storage space

to provide the system with larger storage space. In distributed storage systems, the design of erasure coding technology has great practical significance. These studies included the following aspects: trading off storage efficiency and repair bandwidth overhead, improving recovery rates, selecting the optimal data block storage location, and optimizing utilization of CPU resources.

However, data read and write performances of ECs are not as good as the multi-replicas strategies on DDBMSs. The ECs are not suitable for DDBMSs for two reasons from the perspective of query and update. First, transaction processing requires frequent data access in DDBMSs. In the read-write separation architecture, the multi-replicas mechanism can improve system throughput. Second, after completing the data update on a master node, only the log needs to be transmitted to other slaver backup nodes to the data asynchronous update. Although ECs are not as efficient as multi-replicas strategies in terms of data access and data updates, it can significantly reduce storage overhead and improve storage utilization [8].

The CBase (CBase Homepage: https://github.com/BankOfCommunications/CBASE.) is a high availability distributed relational database developed for financial scenarios. It realizes cross-record and cross-table transactions on hundreds of terabytes (TB) of data and hundreds of billions of records. The CBase adopts a distributed architecture with read-write separation based on OceanBase 0.4.2 (OceanBase Homepage: https://oceanbase.alipay.com/), and its redundancy strategy is the three-replicas. It divides the data into baseline data and incremental data and merges these data at a specific period. This system architecture is representative of the research prototype, so relevant technical work on CBase can be migrated and extended to other database systems.

Consequently, we trade off the transactions processing performance and storage efficiency, and propose a new redundancy strategy CBase-EC, which can recognize and dynamically convert the hot and cold tablets of CBase.

Our contributions:

1. We propose the redundancy strategy CBase-EC, which includes the hot and cold tablets recognition algorithm, the hot and cold tablets dynamic conversion algorithm and the load balance scheme.
2. The encoding and updating performances of CBase-EC based on locally repairable codes (LRC) have been reduced, and we have designed the LRC increment the update algorithm to optimize the updating process and have presented a heuristic algorithm to find local optimal Bitmatrix of LRC to improve encoding and decoding performances.
3. We use sysBench, a benchmark tool in the distributed database field, to compare the storage efficiency and transaction processing performance of CBase-EC with the original three-replicas strategy on the CBase database system. The experimental results show that the strategy presented in this paper has no significant decrease in transaction throughput, at most about 6%, but the storage efficiency improves by 18.4%.

The remainder of this paper is organized as follows. In Section 2, we introduce some basic concepts including the comparison between EC and the replication strategy and basic concepts about the CBase database system. In Sections 3 and 4, we propose the new redundancy strategy CBase-EC and introduce two optimization methods of encoding and updating. In Section 5, we report on our comparative experiments, mainly comparing the storage efficiency and transaction processing performances for new CBase with the original CBase. The conclusion and future works are in Section 6.

2. Background

In this section, we introduce the technical terms of EC and compare the EC with the replication strategy. Then we briefly describe the query and update process of the CBase database system.

2.1. Comparison of Replication and Erasure Codes (EC)

Figure 1a illustrates key terminology related to EC. A file is divided into equal units, which are called *data blocks*. We divide each k of those blocks into a group, and for each set of k data blocks, use any EC algorithm to obtain $n-k$ *parity blocks*. The set consists of both data and parity blocks called a *block stripe*. The data and parity blocks belonging to a stripe are placed on different disks or nodes in a distributed storage system. If a disk/node reliability is p, while those systems use EC, the system reliability is the following formula:

$$1-\sum_{i=0}^{k-1}\binom{n}{i}p^i(1-p)^{n-i}. \tag{1}$$

In a distributed database, the system generally uses a partition algorithm to split large tables into approximately equal partitions called *shards* or *sub-tables*. The shards are called tablets in CBase. If they use the multi-replicas strategy, each tablet generates multiple redundant tablets. The unreliability of the system means that the original data cannot be recovered, that is any C_k^i types of data blocks are lost and the redundant blocks are completely lost, and only a part of all the data blocks of the remaining $(k-i)$ types of blocks exists. The system's reliability based on the replication strategy is shown as follows:

$$1-\sum_{i=1}^{k}\binom{k}{i}(1-p)^{\frac{in}{k}}\left[\sum_{j=0}^{\frac{n}{k}-1}\binom{\frac{n}{k}}{j}(1-p)^j p^{\frac{n}{k}-j}\right]^{k-i}. \tag{2}$$

Comparing Equations (1) and (2), Figure 1b shows the reliability of these two formulas changing with redundancy $r=\frac{n}{k}$.

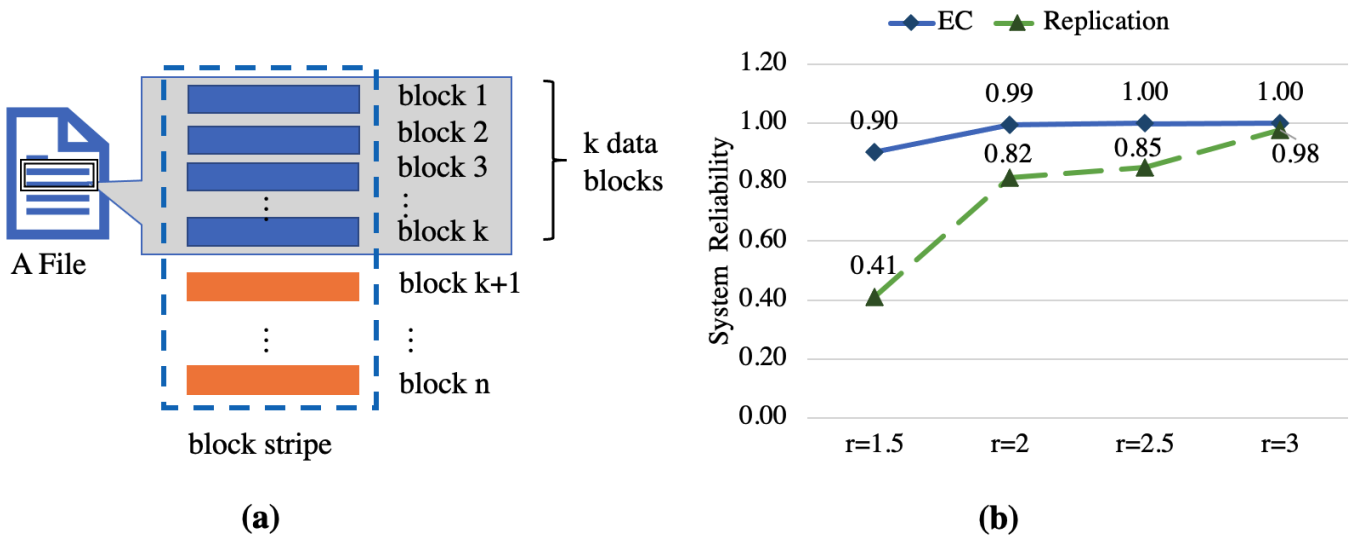

Figure 1. (**a**) Terminology of Erasure Code (EC). In the blue dashed box is a stripe consisting of k blue data blocks and $n-k$ orange parity blocks. (**b**) System reliability of EC and replication while reliability $p=0.8$. The reliability of the replication strategy is lower than EC while r is smaller, and EC can guarantee more than 90% reliability in any case.

2.2. CBase Database System

The CBase is a distributed relational database which adopts shared-nothing and read-write separation architecture based on open-source OceanBase 0.4.2.

As shown in Figure 2, there are four types of modules in the CBase: RootServer (RS), UpdateServer (UPS), ChunkServer (CS) and MergeServer (MS). RS is responsible for all modules in management and stores a RootTable of metadata that records the range of primary keys and locations for each tablet. The CS node stores baseline data and their replicas, which are snapshots of data sorted by the primary key. UPS buffers the incremental data use the MemTable structure in memory, that as an LSM structure can optimize for

frequent updates (insert, update and delete operations) [9]. Once the active MemTable reaches the set threshold, it becomes a frozen MemTable that is immutable and UPS makes a new active MemTable. Incremental data are regularly merged with the baseline data over a period of time on CS nodes. MS receives query/update requests and forwards them to the corresponding CSs and UPS, and then merges the results back to the client. The query process is executed as depicted in Figure 2.

1. The client sends a query request to a MS node.
2. MS asks RS to get the distribution of data from RootTable and sends the request to CS nodes where the baseline data are located.
3. Each tablet stored on CS nodes involved in the request gets its incremental data from the active MemTable and frozen MemTables on UPS and returns results to MS.
4. MS merges the results that come from CS nodes.
5. MS returns the results to the client.

The execution processes of the updates differ from the query requests. Shown also in Figure 2, after parsing the request to obtain an execution plan, MS asks CSs to get baseline data, and then sends the data and execution plan to UPS. UPS executes the updates and returns the results to MS. Finally, MS returns results to the client.

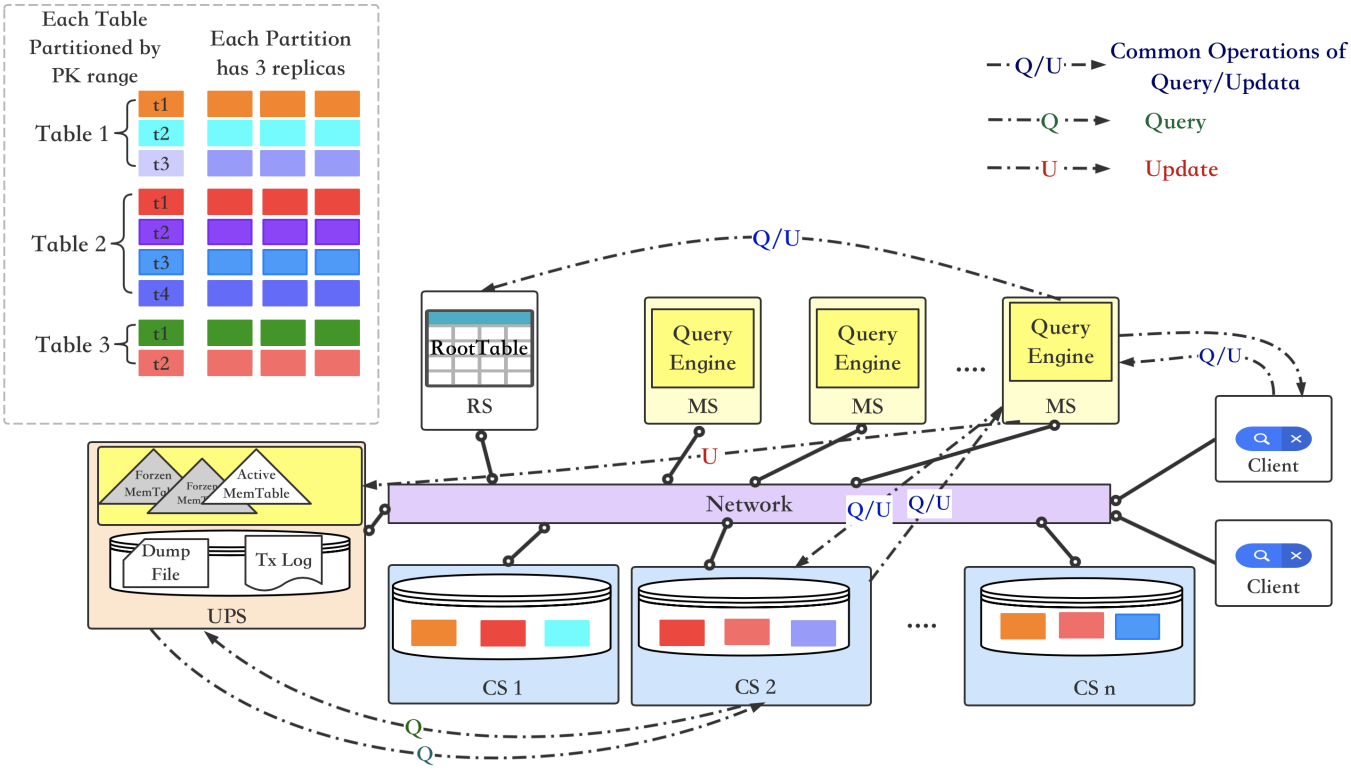

Figure 2. CBase Architecture. The CBase system includes query and updates operations. The blue markers are the same part of these two operations, the green markers are query-specific operations, and the red markers are update-specific operations. The top left corner is a schematic diagram of the database tables divided by primary key range, with warm colors representing hot partitions and cold colors representing cold partitions. However, in the original CBase system, there is no distinction between hot and cold data, and the redundant backup strategy is the three-replicas.

3. Dynamic Conversion of Hot and Cold Data Storage

In this section, we introduce our new redundancy strategy CBase-EC for the CBase Database system in more detail. CBase-EC includes hot and cold tablets recognition stage on RS node and dynamic conversion of hot and cold tablets stage on CS nodes. As shown in Figure 2, when the incremental data of the MemTable reaches the upper threshold, CBase merges the incremental data with baseline data. Additionally, it triggers daily merges at an

idle time every night. The new conversion period $T(n)$ of the CBase-EC strategy will start after the daily merges of the last period $T(n-1)$ are completed.

3.1. Hot and Cold Tablets Recognition

At the beginning of the new period T, the RS node obtains the access frequency of each tablet from the HashTable in all CS nodes. The key of the HashTable is the *Tbid* of the tablet stored in that CS node, and the value is the update and query frequency of this tablet. As shown in Figure 2, CS node can get *Tbid* from the query or update request, and then add 1 to the value of this *Tbid* in HashTable. Then RS calculates new temperatures and updates the result of the t column of the RootTable. RS rejudges the hot and cold state of all tablets according to the hot data ratio R_{hot} (default 20%), and updates the result of the *HC* column. Finally, RS can obtain the *cold*2*hot Tbid* set and the *hot*2*cold Tbid* set by judging whether the *HC* and r columns' values are equal. The modified contents of the RootTable is shown in Table 1. The *RI* column uses (CS_i, CS_j, CS_k) store locations of hot tablet copies and stores all tablets' information $(Tbid, CS_x, num)$ of the tablet-stripe except this cold tablet. The detailed description of the hot and cold tablets recognition processes as shown in Algorithm 1.

Table 1. New RootTable Structure.

Name	Definitions
TableID	Table id
PK	Primary Key Range
Tbid	Tablet id
HC (0 is hot, 1 is cold)	Hot and Cold state
r (0 is 3-replicas, 1 is EC)	Redundancy Strategy (last period)
t	Temperature
RI	Redundant Information
size	Tablet Size
v	Baseline data version

Algorithm 1: Hot and cold tablets recognition algorithm

Input: $T(t_{n-1})$, $F(t_{n-1})$of each tablet, $T_{heat} = 1$, α, $R_{hot} = 20\%$
Output: *cold*2*hot-Tbid* set, *hot*2*cold-Tbid* set

```
Initialize Tbidmax is the sum of tablets;
Initialize Tbid is the id of this tablet;
Initialize t_n is current time and t_{n-1} is the end time of last period;
while Tbid ≤ Tbidmax do
    T(t_n)_Tbid = T(t_{n-1})_Tbid e^{-α(t_n - t_{n-1})} + T_heat × F(t_{n-1})_Tbid;
Use any sorting algorithm to rank the tablet temperature in descending order;
s_Tbid is sorted position of this tablet;
if s_Tbid ≤ R_hot × Tbidmax then
    HC_Tbid = 1;
    if HC_Tbid ≠ r_Tbid then
        cold2hot-Tbid.insert(Tbid);
else
    HC_Tbid = 0;
    if HC_Tbid ≠ r_Tbid then
        hot2cold-Tbid.insert(Tbid);
        Add another Tbid in the same tablet stripe into hot2cold-Tbid set;
```

Based on the understanding of Newton's cooling law (Newton's law of cooling: https://en.wikipedia.org/wiki/Newton%27s_law_of_cooling) in the field of natural sci-

ence, we designed and implemented the tablet temperature formula. Newton's cooling law as shown in Formula (3).

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

On the left side of the equation, the temperature of the object decreases with time. On the right side, $T(t)$ is the current temperature of the object. E is the ambient temperature, and k is the proportional coefficient between the temperature change rate of the object and the ambient temperature difference. By solving the differential equation, Formula (4) of Newton's cooling law can be derived, as shown in Formula (4).

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

We give the theorem of hot and cold tablets temperatures accordingly.

Theorem 1. *Assuming that the average temperature of all tablets in the distributed database system is the ambient temperature, and each tablet is independent of each other, there is no mutual influence of temperature and geographic location. The temperature of a tablet is only related to its creation time and access frequency. The temperature and access frequency of a tablet in the last period is* $T(t_{n-1})$ *and* $F(t_{n-1})$*;* T_{heat} *is the increment temperature, and* α *is cooling coefficient. Hence, the tablet's current temperature is as follows:*

$$T(t_n) = T(t_{n-1})e^{-\alpha(t_n - t_{n-1})} + T_{heat} \times F(t_{n-1}). \tag{5}$$

In CBase, when a new tablet is created, it will default to hot tablet with an initial value of the median of the temperature of all hot tablet.

3.2. Conversion Strategy of Hot and Cold Tablets

There are four types of conversion, namely **continuous hot**, **continuous cold**, **cold2hot** and **hot2cold**. If the tablet is continuous hot, and only needs to complete the daily merges and metadata update. If the tablet is continuous cold and needs to determine whether there is a stripe failure (failure is another tablet in the same stripe convert cold to hot). If there is no stripe failure, update the current tablet and parity tablets. Otherwise, add this tablet to the *hot*2*cold-Tbid* set.

After completing the collection of the *cold*2*hot-Tbid* set and the *hot*2*cold-Tbid* set, RS traverses the new RootTable to accomplish conversion of the tablet in *cold*2*hot-Tbid* set. As shown in Figure 3, RS notifies the CS nodes about these cold tablets' locations and informs them to adopt a three-replicas backup mechanism. After all cold tablet replicas have been generated, the relevant CS nodes send backup meta-information to RS and update column RI and change column $r = 0$.

For the tablet of *hot*2*cold-Tbid* set, we use locally repairable codes (LRCs) [10] to encode tablets in fine granularity. LRC based on Reed-Solomon [11] with the Vandermonde matrix can effectively reduce the network I/O, the amount of data and computation, and the data recovery time. LRC groups all coding blocks. The local parity blocks are generated within each group, and the global parity blocks are generated within the whole strip.

Although LRC is an EC strategy that trades off storage space and repair bandwidth overhead, its storage overhead is much less than the three-replicas strategy. LRC represents by triplets (k, m, n), where k is the number of original data blocks; m is the number of local parity blocks which is equal to the number of the group, and n is the number of global parity blocks. Since each tablet divides into many data blocks according to the size of its physical structure, we use LRC encoding k blocks with the same block id in multiple cold tablets. The encoding process is shown in the red dashed box in Figure 3, and the fine-grained coding scheme in Figure 4. Block stripe 0 is each cold tablets' metadata information. This fine-grained coding scheme has two advantages. First, when a tablet in a tablet stripe is lost and needs to be recovered, all block stripes can parallel encode and

decode. Second, it can prevent the whole tablet stripe from being recalculated when only a small number of data blocks are updated.

The detailed process of the dynamic conversion of hot and cold tablets can be seen in Algorithm 2. After compiling and storing all cold tablets, all CS nodes rebuild the local HashTable in local memory. *Tbid* is hash key and the access frequency of the tablet is the hash value. Then those CS nodes that participate in LRC encoding send relevant redundancy metadata to RS, and RS updates column RI, and changes column $r = 1$.

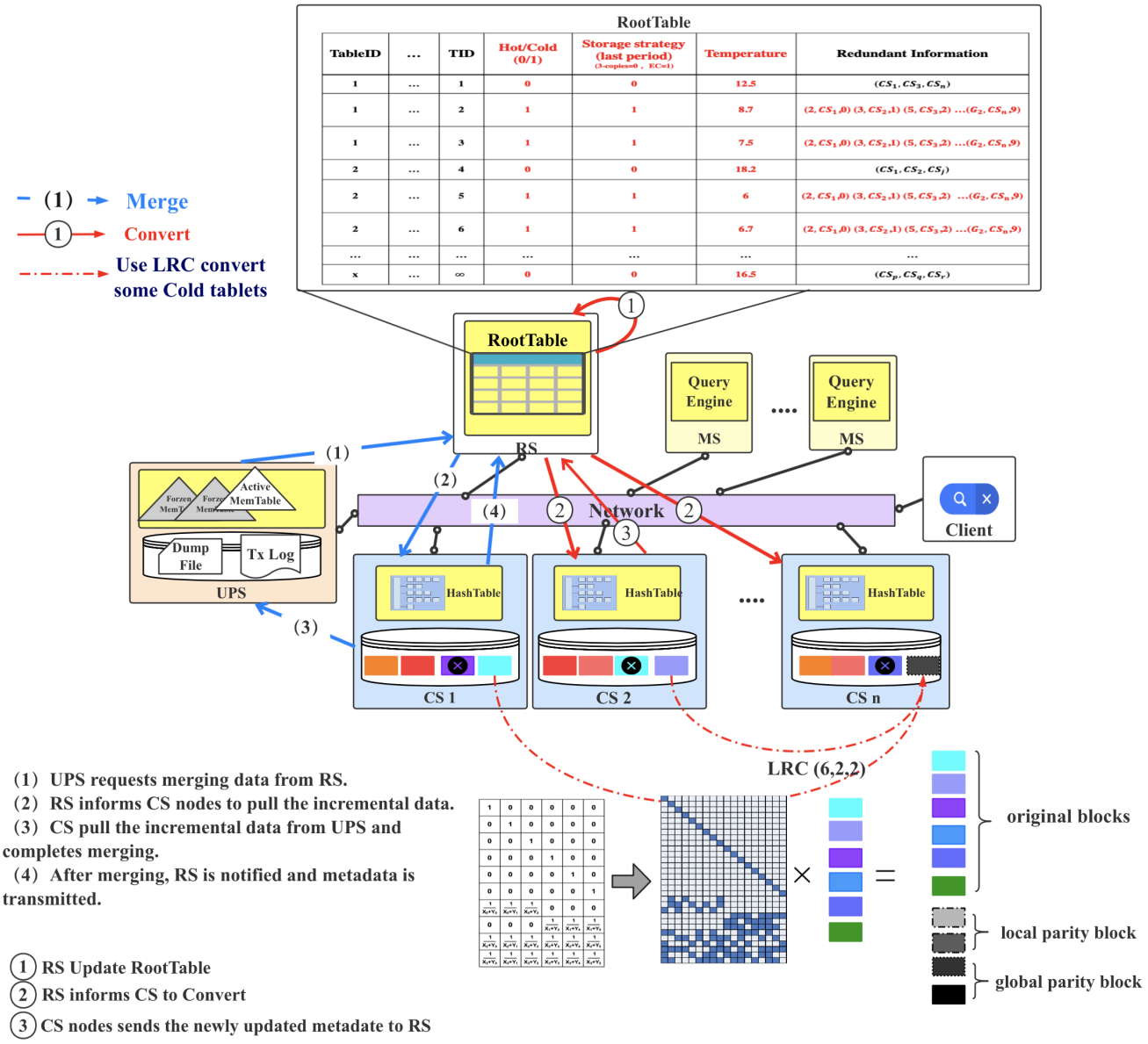

Figure 3. Merge and convert operations based on CBase-EC strategy on CBase. The blue line represents the process of daily merges. The red line indicates the dynamic conversion process of hot and cold tablets storage after completed merging. The RootTable shows the added column information in red color. The bitmatrix representation of locally repairable code (6,2,2) in the lower right corner of this figure.

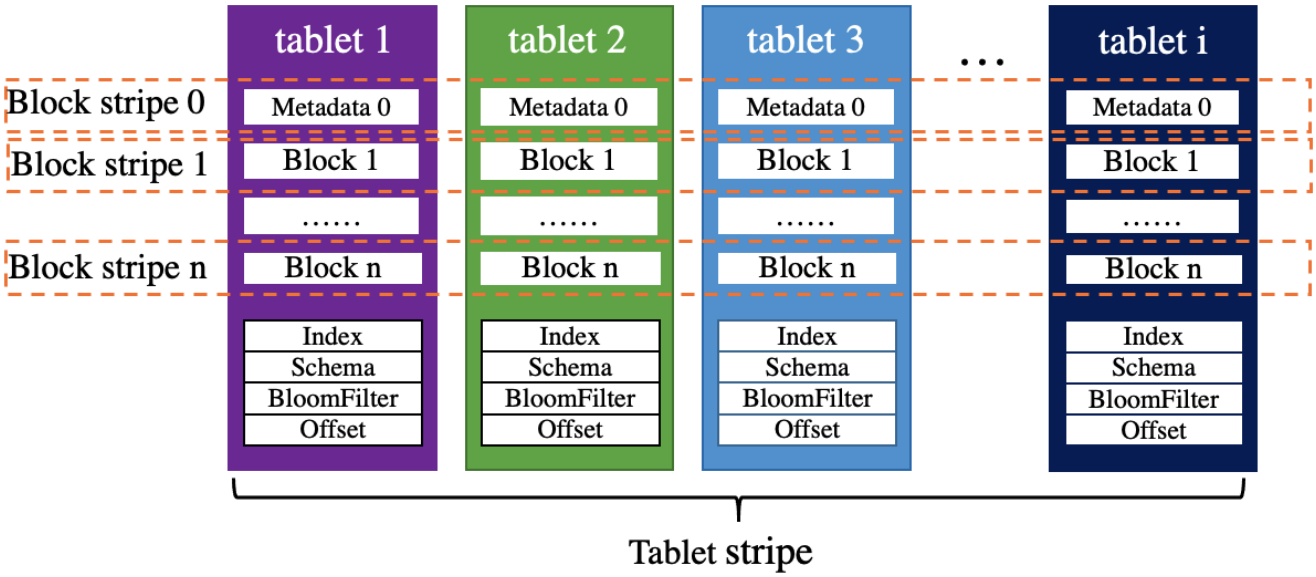

Figure 4. Cold tablet fine-grained coding scheme based on locally repairable code (LRC) code. Stripe 0 consists of the metadata of each tablet. Stripes 1–n consist of the block blocks in the tablet. Other data in the cold tablet are not encoded and can be quickly generated if needed.

Algorithm 2: Hot and cold tablets conversion algorithm

Input: *cold2hot-Tbid* set, *hot2cold-Tbid* set;
Initialize $Tbid_{c2h}$ is the tablet id in *cold2hot-Tbid* set;
Initialize $Tbid_{h2c}$ is the tablet id in *hot2cold-Tbid* set;
while $Tbid_{c2h} \in$ *cold2hot-Tbid* **do**
    Use three-replicas backup $Tbid_{c2h}$;
    cold2hot-Tbid.erase($Tbid_{c2h}$);
    CS notify RS to update column RI and $r = 0$ of the RootTable;
/* IF *hot2cold-Tbid*.size%6 != 0,use Reed-Solomen encode redundant tablets;*/
while *each 6 tablets* $Tbid_{h2c} \in$ *hot2cold-Tbid* **do**
    lrc_init_n(lrc,2,(uint8_t[ ]){3,3},4); // use LRC(6,2,2) encode cold tablets
    hot2cold-Tbid.erase($Tbid_{h2c}$);
    CS notify RS to update column RI and $r = 1$ of the RootTable;

3.3. Load Balancing Scheme

After CBase has completed the tablets' storage according to the CBase-EC, RS starts to calculate the performance load and storage load of each CS node. In the original CBase that the system uses, the number of tablets on each CS node represents the load of this node, and the load is proportional to the sum of tablets. Nevertheless, CBase-EC changes the system's load balancing. Therefore, we use a new algorithm to calculate the load status of the CBase and CS nodes. The formula is as follows (6). $\bar{T}$ is the average temperature of all tablets, which represents the performance load degree of a CS node. R_{hot} is the proportion of hot tablets to total tablets. Thomas E. Nisonger proposed the "80/20 Rule", also termed a Pareto distribution or Pareto principle which means 80% of the data access of the system is concentrated in 20% of the data. The weight $(1 - R_{hot})$ is assigned to the sum of hot tablets and R_{hot} is assigned to the sum of cold tablets. $T_{average}$ is the average temperature of the CBase, which can be calculated from the average of $\bar{T}$.

$$\bar{T} = \frac{(1 - R_{hot}) \times \sum_{i=1}^{m} T_{hot} + R_{hot} \times \sum_{i=m+1}^{n} T_{cold}}{n}. \quad (6)$$

When RS finds $|\bar{T} - T_{average}| \geq \varepsilon$ (ε indicates the floating range of $T_{average}$), then RS uses a greedy algorithm to reduce the $\bar{T}$ of this CS node by subtracting the highest temperature

tablet until the temperature of this CS node falls to the normal temperature range. Next, RS will inform this CS node to send the hottest tablet to the lower temperature of CS node. CBase-EC iteratively calculates the $\bar{T}$ of each node in turn until all CS nodes are balanced. For the storage load of each CS, RS balances the storage load by counting the number of total tablets of each CS node in the RootTable.

4. Optimization Schemes of CBase-EC in Encoding, Decoding and Updating Processes

In this section, we adopt some optimization methods to improve CBase-EC performances when dealing with the cold tablets. Because CBase is a database system, it has frequent update operations. In Section 4.1, we introduce the incremental encoding algorithm to update local and global parity blocks encoded by LRC. In Section 4.2, we transform the multiplication operation in the finite field into exclusive OR (XOR) operation to reduce encoding or decoding calculation complexity.

4.1. Update Optimization

There are two updated schemes for ECs [12]. One is recoding by recalculating all data blocks in the stripe. This process will consume a lot of bandwidth and I/O disk overhead, which is only applicable to the scenario where a large number of data updates occur in all blocks. The other method is incremental coding, which calculates the increment data between the update data block and the original data block. Because update the new parity blocks only encodes the incremental data, which can effectively reduce the repair bandwidth overhead. However, this method only applies to the scenario where part of the data block is updated. The detailed analysis is shown in the Algorithm 3. When a cold tablet needs updating, only the local parity tablet in the group and the global parity tablet need to be re-calculated.

Algorithm 3: LRC increment update algorithm

Input: $t_i^{'}$ is update by tablet t_i, $i \in \{1, 2, ...k\}$;
Initialize $p_{x,y}$ is the y local parity tablet of x group;
Initialize g_j is the global parity tablet, $j \in \{1, 2, ...(n-k)\}$;
while $i \in \{1, 2, ...k\}$ **do**
    CS_i receives $t_i^{'}$;
    CS_i read t_i from local disk ;
    /* Assume there are r local parity tablets in a group. */ CS notify RS to update column RI and $r = 0$ of the RootTable;
    while $y \in \{1, 2, ...r\}$ **do**
        Generate $\Delta p_{x,y}^{'}$ with $t_i^{'} - t_i$;
    while $j \in \{1, ...(n-k)\}$ **do**
        Generate $\Delta g'_j$ with $t_i^{'} - t_i$;
    CS update $t_i^{'}$ to local disk;
while $y \in \{1, 2, ...r\}$ **do**
    CS_i dispatch $\Delta p_{x,y}^{'}$ to CS_{pxy} directly;
    CS_{pxy} read $p_{x,y}$ from local disk, and let $p_{x,y}^{'} = p_{x,y} + \Delta p_{x,y}^{'}$;
    CS_{pxy} update $p_{x,y}^{'}$ to local disk;
while $j \in \{1, ...(n-k)\}$ **do**
    CS_i dispatch $\Delta g'_j$ to CS_{gj} directly;
    CS_{pxy} read g_j from local disk, and let $g_j^{'} = g_j + \Delta g_j^{'}$;
    CS_{pxy} update $g_j^{'}$ to local disk;

4.2. Bitmatrix Normalization

All EC calculations are in a finite field $GF(2^w)$, when $w = 8$, each element $e \in GF(2^w)$ is a byte. In [13], they described a $1 \times w$ row vector V or a $w \times w$ matrix M which can be represented as an element over $GF(2^w)$. For any $e \in GF(2^w)$, use M(e) as the matrix whose i^{th} $(0 \leq i)$ column is V($e \times z^{i-1}$); M(1) is the identity matrix and M(0) is the all-zero matrix. When $w = 3$, these symbols can be converted into the representation of 3×3 bitmatrix as shown in Figure 5. The bitmatrix of e = 4 in which the 1th column is 100, the 2th column is $4 \times z = 4 \times 2 = 011$ and the 3th column is $4 \times z^2 = 4 \times 4 = 110$. The number of 1's (ones) in the bitmatrix means the number of XOR operations in encoding and o is the average number of ones per row in the matrix.

In LRC, they adopted a classical Vandermonde construct generate matrix, but the complexity of an $n \times n$ Vandermonde inversion is $O(n^3)$ while Cauchy matrix inversion is $O(n^2)$. If we choose a different X and Y will get different o. Hence, we use a Cauchy matrix as generator matrix and find the local optimal bitmatrix which has a lower number of XORs [14]. The detailed Algorithm 4 is as follows. While this does not guarantee an optimal number of ones, it typically generates a good matrix.

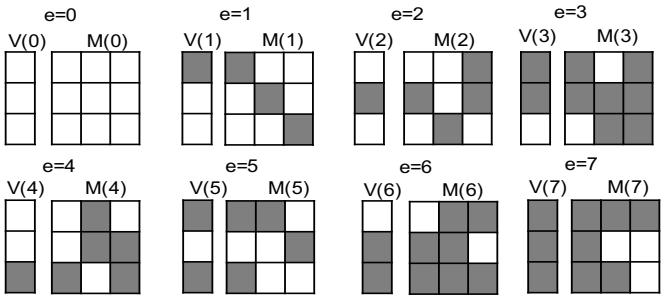

Figure 5. The bitmatrix representation of all elements over the finite field $GF(2^3)$. Gray represents binary 1, white represents binary 0.

Algorithm 4: Find the local optimal bitmatrix of LRC

Input: Use Cauchy matrix construct LRC generate matrix G;
Initialize o_i is the number of ones in this row. i is row, j is column;
for $j = 0$ *to* $k - 1$ **do**
  for $i = k$ *to* $n - 1$ **do**
    $G[i, j] = G[i, j] / G[0, j]$;
for $i = k + 1$ *to* $n - 1$ **do**
  for $j = 0$ *to* $k - 1$ **do**
    for $jj = 0$ *to* $k - 1$ **do**
      $G[i, jj] = G[i, jj] / G[i, j]$;
    Count o_i of this new row;
  Choose the row with min o_i as new i row;

5. Experiments

In this section, we adopted our redundancy strategy CBase-EC in the CBase cluster and compared it with the original three-replicas strategy. The experimental database cluster ran with 10 PC servers. Each server was running on Linux Release 7.5.18 and was equipped with 8-core Intel (R) Xeon (R) CPU e5-2620 V3@2.40 GHz, 64 GB RAM, 2 TB disk capacity, and the network transmission speed was 10 Gbps. The CBase was configured with one UPS node, one RS node, and one MS node. All servers were CS nodes. We measured the system transaction processing performances by sysBench, which is a standard open-source database benchmark that supports a user-defined workload. In the schema of sysBench, each record has a primary key id and three columns, k is the column of integer, and the

remaining two columns are *c* with a random string of 120 bytes and *pad* with a random string of 60 bytes.

5.1. Online Transaction Processing (OLTP) Performances

In this experiment, two tables containing three million records were used in the data set. Each transaction contained five single-point primary key query operations, one range query operation that continuously scanned 100 records according to the primary key, and the data update, insert and delete operation was set to 1. All experiments were run six times, and the experimental results were averaged.

Firstly, we fixed the data access requests that were evenly distributed across all the tablets and then tested the transactions per second (TPS) performance of CBase between three-replicas and CBase-EC when R_{hot} = 100% and R_{hot} = 20%. Secondly, we designed data requests to conform to the 80/20 distribution, where 80% of the data access was concentrated on 20% hot tablets. Finally, we introduced the average response time of transactions when the data access met the 80/20 distribution. All experiments results are shown in Figure 6.

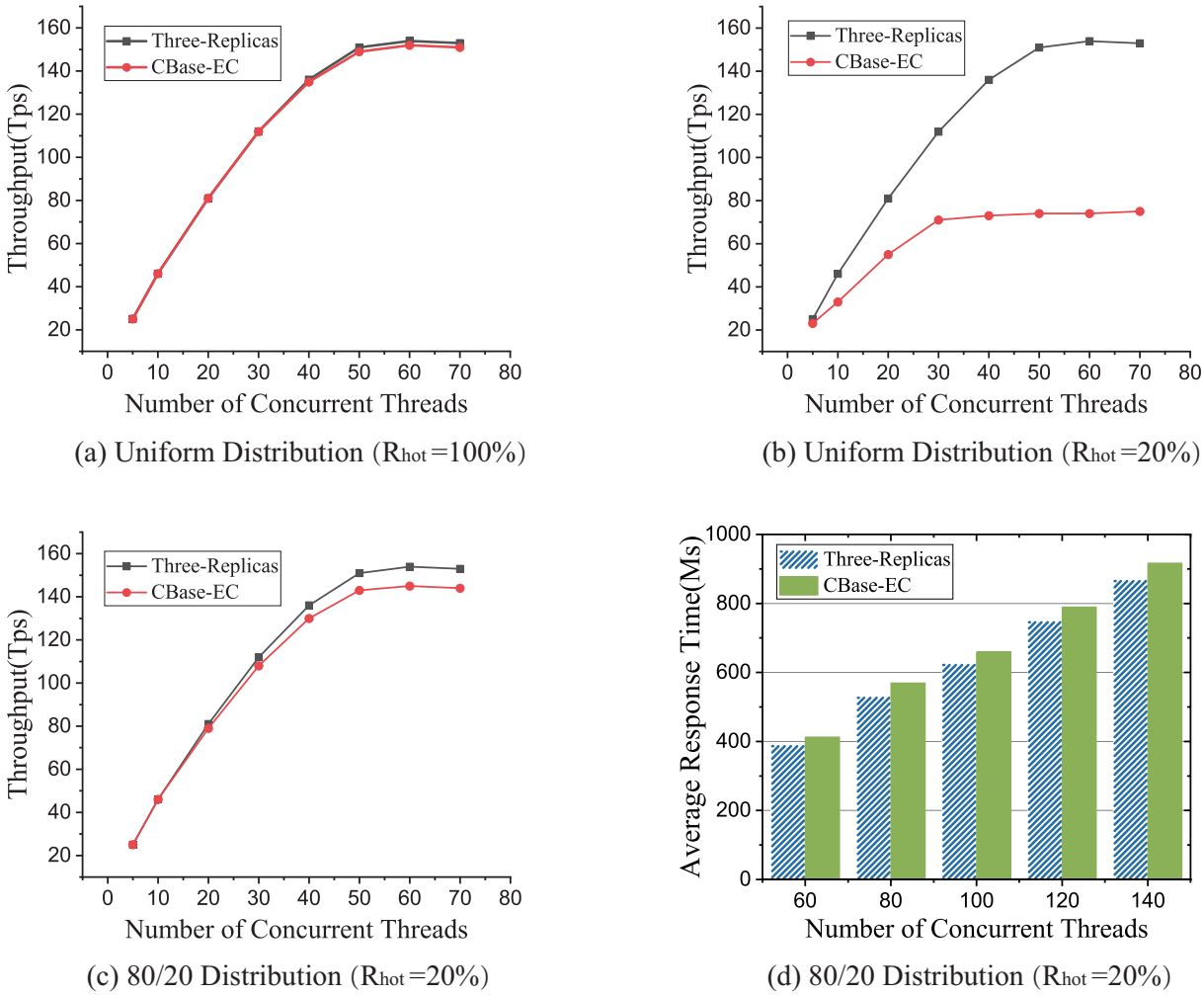

Figure 6. Comparison of Online Transaction Processing (OLTP) performance of CBase based on three-replicas and CBase-EC strategies. Uniform Distribution mean that data access requests are uniform across all tablets.The 80/20 distribution means that 80% of the data access are concentrated on 20% of the hot tablets. R_{hot} refers to the percentage of hot data ratio.

It can be seen from Figure 6a when the number of concurrent threads was less than 30, the TPS was exactly the same, because the CBase adopted three-replicas based on both strategies, but when the number of threads was more than 30, the TPS performance of

CBase-EC was reduced because CS nodes needed to constantly update the local HashTable and consumed a small amount of I/O and CPU resources. It can also be clearly seen from Figure 6b that when the cold tablets accounted for 80% of all tablets, with an increase in the number of threads, the performance of the CBase based on CBase-EC strategy quickly reached the bottleneck. Comparing the two kinds of strategies, it can be concluded that when the data access was uniform, the CBase was not suitable for deploying ECs, because the sum of transactions processed on cold-tablets was much larger than the sum of transactions on the hot tablets. From Figure 6c, we observed that when data access conformed to 80/20 distribution, the OLTP performance of CBase based on two strategies was basically the same, and the CBase performance loss based on CBase-EC strategy was about 5.3%. As can be seen from Figure 6d, with an increase in the number of threads, the average transaction response time of CBase based on three-replicas was about 6% lower than that of CBase based on CBase-EC. According to the above experiments, the adoption of the CBase-EC strategy will reduce OLTP performance within a limited range.

5.2. Storage Efficiency

An important index of redundancy strategy is storage efficiency, which is the ratio of original data size to redundant storage space. In Figure 7a, it can be observed that with an increasing number of table records, storage overhead based on CBase-EC was much lower than that based on three-replicas. Besides, as shown in Figure 7b we have experimented with the storage efficiency based on the change of $(1 - R_{hot})$. It can be concluded that if the CBase adopted three-replicas stores 1 PB data needed 3 PB actual storage capacity. Nevertheless, the maximum storage efficiency of CBase could reach 60% based on CBase-EC, which means that only 1.67 PB actual storage space was needed to store 1 PB data. Consequently, CBase-EC could effectively reduce the storage overhead of CBase.

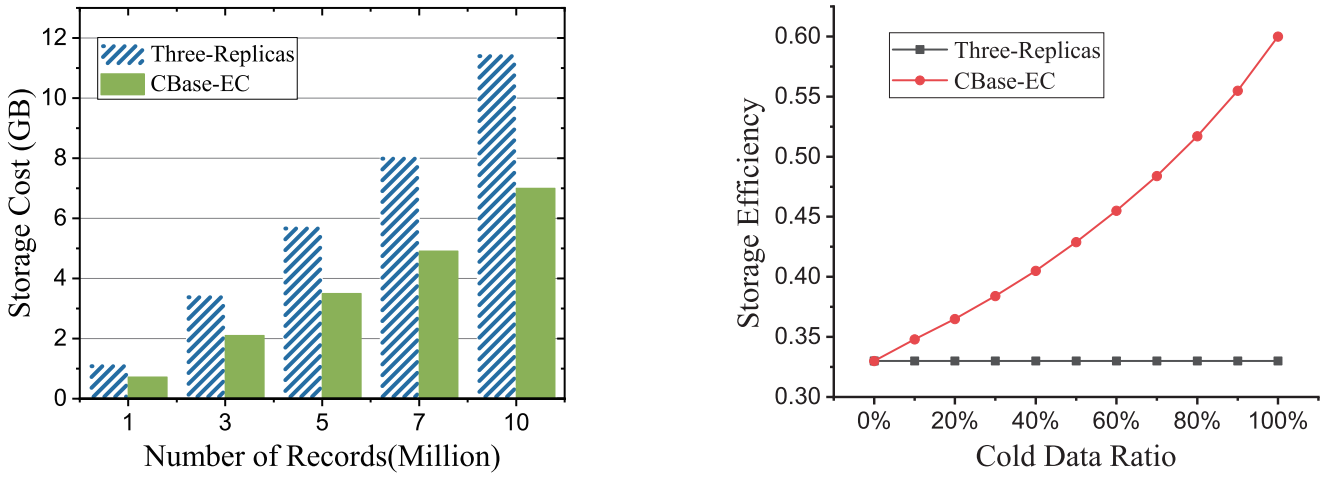

Figure 7. Comparison of storage efficiency of CBase based on three-replicas and CBase-EC strategies.

5.3. Encoding Performance Optimization

The number of XORs of the Vandermonde-LRC, the Original Cauchy-LRC, and the Optimized Bitmatrix-LRC, was tested by encoding performance optimization experiments. Then we calculated the improvement of the new optimization method over the baseline which was Vandermonde-LRC and the results are reported in Table 2. For rows 1–3, 4–6, and 7–9 in the table, we measured the performance improvement by fixing the number of local and global parity blocks, fixing the number of the local parity blocks only, and changing the number of local and global parity blocks at the same time. In the last row of the table, the average encoding performance increase was 16.18%, as measured by the

number of XORs reductions in all test parameters. Since each combination of X and Y was different, an entirely different generation matrix was constructed, and the degree of performance improvement was not stable as you can see from the last column.

Table 2. Total Number of XOR's.

(k, m, n)	Vandermonde-LRC	Original Cauchy-LRC	Optimized Bitmatrix-LRC	Reduction of XORs
(6,2,2)	317	474	219	30.9%
(8,2,2)	364	602	303	16.7%
(10,2,2)	555	768	389	29.9%
(6,2,4)	642	854	595	7.3%
(8,2,4)	816	1116	809	0.8%
(10,2,4)	1210	1448	1048	13.4%
(8,4,4)	956	1464	907	5.12%
(10,4,4)	1502	1952	1164	22.5%
(10,4,6)	2110	2632	1709	19%
Average	-	-	-	16.18%

5.4. Update Performance

In this experiment, the data block of each tablet was 4 KB and the transaction thread wrote 50,000 update transactions with different granularity. We tested average update time of recoding and LRC-incremental update coding under different update scenarios. As shown in Figure 8, taking LRC (6,2,2) as an example, when cold tablets with no more than $\frac{k}{2}$ in the same strip were updated, the performance of LRC-incremental update coding was higher than that of recoding.

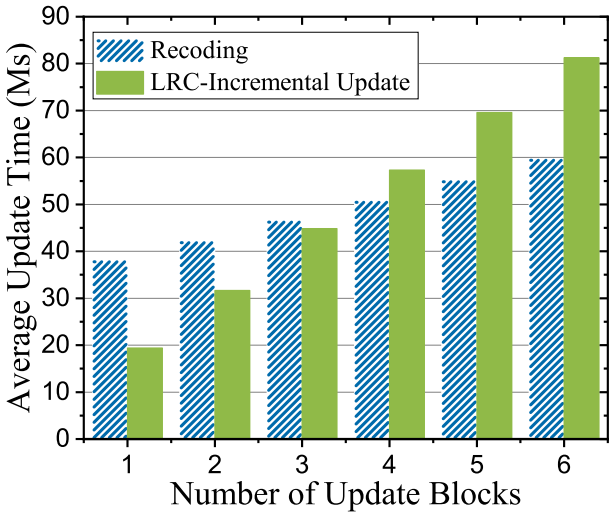

Figure 8. Cold tablets update performances based on locally repairable code (LRC) (6,2,2).

5.5. Parallel Recovery Performance

In this experiment, five tables (7 GB each table) were employed for the data recovery test, and the average amount of data stored on each CS node was 3.5 GB. Recovery time refers to the total time to recover the lost data of the failure CS node after down without affecting the CBase performances. Hence, we designed a timer to kill one or two CS processes at intervals to simulate the node failure scenario. Figure 9 shows the total recovery time of the three strategies while the number of failed nodes was 1 or 2. The three strategies were three-replicas, coarse-grained CBase-EC which minimum granularity is a tablet, and fine-grained CBase-EC which minimum granularity is a block. It can be clearly seen from Figure 9 that the data recovery time of the CBase based on CBase-EC was

significantly higher than that of the original Cbase, not only because the recovery strategy needed to distinguish the hot and cold tablets, but also because the CPU was required to participate in the calculation during the repair process. Nevertheless, the recovery rate of CBase-EC with a fine-grained coding scheme was better than that of CBase-EC with coarse-grained coding, because the parallel pipeline technology could effectively decrease the idle waiting time of disk I/O, CPU, and network transmission.

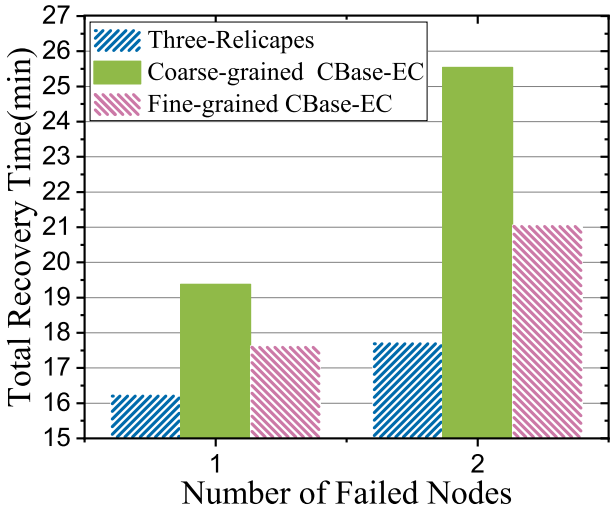

Figure 9. Parallel repair performance.

6. Conclusions and Future Work

The CBase-EC strategy is a new redundant mechanism for the CBase, which is a distributed relational database system. CBase-EC can dynamically recognize and convert hot-cold tablets and maintain loading balance by using ECs to trade off transactions processing throughput-storage efficiency. To evaluate this new strategy, we ran simulations on the CBase cluster obtained by 10 PC servers. We conclude that the CBase-EC delivers the optimal throughput-storage efficiency tradeoff and outperforms the three-replicas strategy in the CBase database system.

Several researchers have proposed Regeneration codes [15,16], Zigzag codes [17], Butterfly codes [18] and Hitchhiker codes [19] to balance storage overhead and repair bandwidth. The research on the recovery rate has mainly been considered from two aspects. From the perspective of system performance improvement, it can increase the network bandwidth transmission [20,21] or increase the parallelism of data recovery [22–24]. From the perspective of algorithm optimization, there were some repair solutions that use heuristic algorithms to find the optimal repair path. Xiang et al. proposed their solution of joint latency and storage cost minimization via the computation of a sequence of convex approximations with provable convergence [25]. In addition, some new data block placement strategies have also been proposed. Venkatesan et al. [26,27] show that, for a replication factor of two, all possible placement schemes have mean times to data loss (MTTDLs) within a factor of two for practical values of the failure rate, storage capacity, and rebuild bandwidth of a storage node. The theoretical results are confirmed by means of event-driven simulation. They also show that the declustered placement scheme, contrary to intuition, offers a reliability for replication factors greater than two that does not decrease as the number of nodes in the system increases. Since all ECs rely on the arithmetic calculation over finite fields, the encoding performance of erasure codes can be accelerated by improving the utilization efficiency of the CPU.

In the future, we will deeply investigate the distributed database systems reliability storage problem and find more effective solution to reduce the storage overhead. In addition, we will continue to study low bandwidth recovery overhead strategies, data

block storage placement strategies and efficient coding schemes for different workloads based on this paper.

Author Contributions: Conceptualization, X.G.; methodology, C.X.; software and validation, C.X. and Y.X.; formal analysis, C.X.; investigation, Q.Z.; resources and data curation, L.Z.; writing—original draft preparation, C.X.; rewriting and editing, C.X. and L.Z.; supervision, Q.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Key Research and Development Project grant number 2019YFB2102600 and National Natural Science Foundation grant number 61572194, 61672233.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barber, R.; Garcia-Arellano, C.; Grosman, R.; Müller, R.; Raman, V.; Sidle, R.; Spilchen, M.; Storm, A.; Tian, Y.; Tözün, P.; et al. Evolving Databases for New-Gen Big Data Applications. In Proceedings of the 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, 8–11 January 2017.
2. Huang, D.; Liu,Q.; Cui, Q.; Fang, Z.; Ma, X.; Xu, F.; Shen, L.; Tang, L.; Zhou, Y.; Huang, M.; et al. TbidB: A Raft-based HTAP database. *Proc. VLDB Endow.* **2020**, *13*, 3072–3084. [CrossRef]
3. Nachiappan, R.; Javadi, B.; Calheiros, R.N.; Matawie, K.M. Cloud storage reliability for big data applications: A state of the art survey. *J. Netw. Comput. Appl.* **2017**, *97*, 35–47. [CrossRef]
4. Borovica-Gajić, R.; Appuswamy, R.; Ailamaki, A. Cheap data analytics using cold storage devices. *Proc. VLDB Endow.* **2016**, *9*, 1029–1040. [CrossRef]
5. Balakrishnan, S.; Black, R.; Donnelly, A.; England, P.; Glass, A.; Harper, D.; Legtchenko, S.; Ogus, A.; Peterson, E.; Rowstron, A. Pelican: A building block for exascale cold data storage. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, CO, USA, 6–8 October 2014; pp. 351–365.
6. IDC. *Technology Assessment: Cold Storage Is Hot Again Finding the Frost Point*; International Data Corporation: Framingham, MA, USA, 2013; May 2013 IDC #241005.
7. Plank, J.S. T1: Erasure codes for storage applications. In Proceedings of the 4th USENIX Conference on File and Storage Technologies, San Francisco, CA, USA, 13–16 December 2005; pp. 1–74.
8. Rashmi, K.V.; Shah, N.B.; Gu, D.; Kuang, H.; Borthakur, D.; Ramchandran, K. 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, 27–28 June 2013.
9. O'Neil, P.; Cheng, E.; Gawlick, D.; O'Neil, E. The log-structured merge-tree (LSM-tree). *Acta Inform.* **1996**, *33*, 351–385. [CrossRef]
10. Papailiopoulos, D.S.; Dimakis, A.G. Locally repairable codes. *IEEE Trans. Inf. Theory* **2014**, *60*, 5843–5855. [CrossRef]
11. Reed, I.S.; Solomon, G. Polynomial codes over certain finite fields. *J. Soc. Ind. Appl. Math.* **1960**, *8*, 300–304. [CrossRef]
12. Li, H.; Zhang, Y.; Zhang, Z.; Liu, S.; Li, D.; Liu, X.; Peng, Y. PARIX: Speculative Partial Writes in Erasure-Coded Systems. In Proceedings of the 17th USENIX Annual Technical Conference (USENIXATC), Santa Clara, CA, USA, 12–14 July 2017; pp. 581–587.
13. Bloemer, J.; Kalfane, M.; Karp, R.; Karpinski, M.; Luby, M.; Zuckerman, D. *An XOR-Based Erasure-Resilient Coding Scheme*; Technical Report No. TR-95-048; Berkeley International Computer Science Institut: Berkeley, CA, USA, August 1995.
14. Zhou, T.; Tian, C. Fast erasure coding for data storage: A comprehensive study of the acceleration techniques. *ACM Trans. Storage (TOS)* **2020**,*16*, 1–24. [CrossRef]
15. Dimakis, A.G.; Godfrey, P.B.; Wu, Y.; Wainwright, M.J.; Ramchandran, K. Network coding for distributed storage systems. *IEEE Trans. Inf. Theory* **2010** , *56*, 4539–4551. [CrossRef]
16. Jiekak, S.; Kermarrec, A.M.; Le Scouarnec, N.; Straub, G.; Van Kempen, A. Regenerating codes: A system perspective. *ACM SIGOPS Oper. Syst. Rev.* **2013**, *47*, 23–32. [CrossRef]
17. Tamo, I.; Wang, Z.; Bruck, J. Zigzag codes: MDS array codes with optimal rebuilding. *IEEE Trans. Inf. Theory* **2012**, *59*, 1597–1616. [CrossRef]
18. Pamies-Juarez, L.; Blagojević, F.; Mateescu, R.; Gyuot, C.; Gad, E.E.; Bandic, Z. Opening the Chrysalis: On the Real Repair Performance of MSR Codes. In Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16), Santa Clara, CA, USA, 22–25 February 2016; pp. 81–94.
19. Rashmi, K.V.; Shah, N.B.; Gu, D.; Kuang, H.; Borthakur, D.; Ramchandran, K. A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. In Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM'14), Chicago, IL, USA, 17–22 August 2014; pp. 331–342.
20. Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. In Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM'08), Seattle, WA, USA, 17–22 August 2008; pp. 63–74.

21. Roy, A.; Zeng, H.; Bagga, J.; Porter, G.; Snoeren, A.C. In side the Social Network's (Datacenter) Network. In Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15), London, UK, 17–21 August 2015; pp. 123–137.
22. Ongaro, D.; Rumble, S.M.; Stutsman, R.; Ousterhout, J.; Rosenblum, M. Fast Crash Recovery in RAMCloud. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11), Cascais, Portugal, 23–26 October 2011; pp. 29–41.
23. Mitra, S.; Panta, R.; Ra, M.R.; Bagchi, S. 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, 18–21 April 2016; pp. 30:1–30:16.
24. Li, R.; Li, X.; Lee, P.; Huang, Q. Repair Pipelining for Erasure-Coded Storage. In Proceedings of the USENIX Annual Technical Conference (USENIX ATC '17), Santa Clara, CA, USA, 12–14 July 2017; pp. 567–579.
25. Xiang, Y.; Lan, T.; Aggarwal, V.; Chen, Y.F.R. Joint latency and cost optimization for erasure coded data center storage. *ACM Sigmetrics Perform. Eval. Rev.* **2014**, *42*, 3–14. [CrossRef]
26. Venkatesan,V.; Iliadis, I.; Hu, X.Y.; Haas, R.; Fragouli, C. Effect of Replica Placement on the Reliability of LargeScale Data Storage Systems. In Proceedings of the 18th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'10), Miami Beach, FL, USA, 17–19 August 2010; pp. 79–88.
27. Venkatesan, V.; Iliadis, I.; Fragouli, C.; Urbanke, R. Reliability of Clustered vs. Declustered Replica Placement in Data Storage Systems. In Proceedings of the 19th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'11), Singapore, 25–27 July 2011; pp. 307–317.