Journal of Combinatorial Mathematics
and Combinatorial Computing
www.combinatorialpress.com/jcmcc

Optimization Design of Massive Data Storage System Based on Distributed Computing Model

Xiang Li[1,✉]

[1] *Image and Text Information Center, Jiangsu Province Nantong Industry & Trade Technician College, Nantong, Jiangsu, 226010, China*

ABSTRACT

With the arrival of the big data era, the demand for massive data storage is growing, and distributed storage systems have become a key technology to solve this problem. The traditional HDFS system has a large storage overhead, this paper in order to improve the storage efficiency of massive data, the introduction of corrective deletion code (RS code) technology, to ensure the reliability of the data at the same time significantly reduce the cost of storage. In order to improve the storage efficiency of massive data, this paper introduces the corrective censoring code (RS code) technology, which ensures the data reliability and significantly reduces the storage cost. In addition, to address the problems of low coding efficiency and high repair overhead in the practical application of RS code, this paper further introduces the local repair code (LRC) technology, which reduces the data repair overhead, and compares and analyzes the application effect of optimization model (RS-LRC-HDFS). The experimental results show that after RS-LRC optimization, the time overhead of the HDFS storage system in the write process and read process is significantly improved by 81.12% and 93.01%, respectively, compared with the pre-optimization period, and the repair time of massive file data is reduced by 87.25%. It can be seen that it provides an efficient and reliable solution for massive data storage.

Keywords: HDFS system, corrective censoring code, local repair code, RS-LRC-HDFS, data storage

1. Introduction

With the continuous development of information technology, the technology of storing and analyzing massive data has also been widely concerned and applied. Massive data usually refers to a collection of data that is extremely large in quantity and very complex in type and structure, such as the Internet, social media, Internet of Things, remote sensing, genomics and other fields [1-4]. How to store and analyze massive data efficiently, accurately, and securely has become one of the important issues in business, science, and government [5-6].

✉ Corresponding author.
E-mail address: Lixiang19832010@126.com (X. Li).
Received 20 December 2024; Revised 05 February 2025; Accepted 25 March 2025; Published Online 16 April 2025.
DOI: 10.61091/jcmcc127b-148

Massive data processing usually requires features such as high concurrency, high reliability, high performance and high scalability. Aspects such as the size of the data, the speed of writing and reading, the frequency of data access, and the stability requirements of the system need to be considered in the design phase [7-10]. Issues such as data security, availability and consistency also need to be considered. Optimization is a very important aspect in the process of mass data processing and storage system design [11-13]. Optimization can start from many aspects, such as optimizing algorithms and data structures, optimizing query and storage strategies, optimizing system architectures and hardware configurations, etc. Distributed computing models can play an important role [14-16]. Distributed storage is a way of sharing data storage among many nodes on a computer network, and its core idea is to decentralize the storage of massive data on multiple nodes, so as to improve the reliability, availability and performance of data [17-20]. Distributed storage has many advantages and can effectively scale storage capacity and computational performance.Distributed file systems such as HDFS (Hadoop Distributed File System) and GlusterFS (Gluster File System), have become the mainstream choice in the industry [21-23].

In this paper, we first carry out the overall design of the HDFS system platform and analyze its architectural features and storage mechanisms. For the application of RS code in data storage, it explores the problem that the disk I/O and network bandwidth required for repair are too high in the actual process. Through the introduction of local repair code (LRC) technology, the traditional RS code is optimized, and then the optimized RS-LRC technology is used to reduce the computational overhead of writing and reading in the process of massive data storage and improve the efficiency of node repair. Finally, the performance of the optimized system is verified through actual cases, which provides a reference basis for the practical application of distributed massive data storage system.

2. RS-LRC-HDFS mass data storage system

2.1. Overall Design of Massive Data Storage System Platform

2.1.1. General framework of the system platform. The system platform is generally divided into four layers: user layer, business logic layer, data storage layer and data resource layer. The overall framework model of the platform is shown in Figure 1. User layer: it is the client browser. Users send requests to the system on the client side, and the system feeds back the processing results to the client. Business Logic Layer: It is mainly used for the communication between the application program and the storage system. It provides functions such as system platform settings, security management, catalog management, resource management, etc. Data storage layer: mainly composed by using HDFS [24] and Map Reduce in the system cluster, responsible for the management of data and assignment of tasks, providing distributed computing and storage for the system. Data Resource Layer: is the basis of the hardware resources of the whole platform, composed of multiple computers, used to store and manage massive teaching resource data.

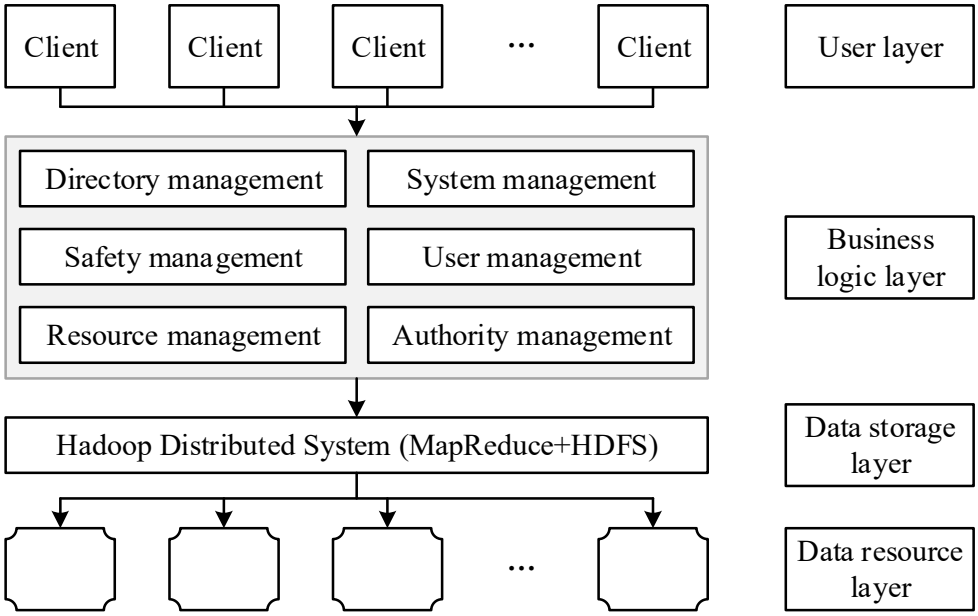


Fig. 1. Design of the overall framework of the platform

2.1.2. Functional modules of the system platform. This system provides a management platform to solve the problem of massive data storage. Its functional modules mainly include four modules: user management module, administrator module, system platform management module and teaching resource management.

User management module: including user's login, file uploading, downloading and deleting functions.

Administrator module: including account rights management, user management, log management and general user functions.

Platform Management Module: includes functions of server management, fault management, early warning alarm and fault logging.

Teaching Resources Module: Includes the classification and storage of teaching resources such as course lesson plans, course teaching videos, test banks, practical training guides, professional construction and high-quality course construction.

2.1.3. Network topology of system platforms. The network topology of the massive teaching resources data platform mainly includes the user operation area at the front end and the data sharing service area at the back end. Among them, the data sharing service area includes all kinds of servers of Hadoop cluster network [25]. These servers can be distributed in different areas. With the increase of data volume and changes in application requirements, the platform can be flexibly expanded, and these operations are transparent to users.

2.1.4. Disadvantages of HDFS storage. The file size of HDFS is usually GB to TB level, therefore, HDFS is adapted to support the storage of large files during the design, and when storing data smaller than the HDFS chunk size, HDFS also stores it according to the chunk size, which increases the storage space and also affects the access efficiency. Therefore, this paper optimizes the traditional HDFS storage model as follows.

2.2. RS-LRC based HDFS storage optimization

2.2.1. Problems with RS codes. Even though the cost of disk is getting lower and lower, the data volume is growing faster. It is still a challenge to store the huge amount of data efficiently while

ensuring the reliability of the data.RS class corrective deletion code [26] is widely adopted by many distributed storage systems because of its high reliability and high storage efficiency. However, it is highly restricted because the disk I/O and network bandwidth required for repair are too high.

$RS(k,n-k)$ code needs to read k blocks even to repair a single block, while the three-copy system needs to read only one block to repair, we take RS(12,4) code as an example for optimization.

RS-type deletion codes have the MDS property and the optimal minimum distance d , the only problem is that the locality r of RS-type deletion codes is equal to k. Generally speaking, codes with small locality have smaller disk blocks to be read during repair, so the IO consumed by reads and writes and network bandwidth consumed by transfers are smaller.

For an encoding of (n,k) with localizability, the minimum distance must satisfy the following constraint:

$$d \le n-k-\left\lceil \frac{k}{r} \right\rceil +2 \tag{1}$$

For $RS(n,k,r)$ code, $r=k, d=n-k+1$. While the minimum distance does reach the maximum, the locality r is too large, resulting in a repair that is too inefficient.

2.2.2. Adding Local Checksum Block Optimization. In this paper, in the previous subsection, we have identified the problems and the root causes of the problems, for which we propose an improved RS code. The repair efficiency and system performance of RS codes is improved by adding extra check blocks to the RS codes and improving the localization of RS codes. We call the improved RS code RS-LRC code [27], LRC is the abbreviation of local repairable code.RS-LRC code adds local checksum blocks ($m=n-k$, is the number of global checksum blocks) on top of $RS(k,n-k)$ code, dividing $k+m$ blocks into l groups of $(k+m)/l$ data blocks each, which $(k+m)/l$ do the dissimilarity operation together to generate a local checksum P_i block.

The $RS-LRC(k,n-k,l)$ code represents data blocks k blocks, $n-k$ checksum blocks, l local checksum blocks, and locality $r=(k+m)/l$ .The parameters of the RS code we have chosen are $k=12, n-k=4, n=16$, and since we have added four local checksum blocks, the improved scheme we call RS-RLC(12, 4, 4) code.

Since we have to add extra checksum blocks on top of the RS code, it will inevitably lead to a further increase in the extra storage space. For the above reasons, we select RS(12, 4) code as the basis for improvement, which can gain more performance improvement at the cost of smaller additional storage space.The structure of RS-LRC code (k=12,m=4,l=4) is shown in Fig. 2.In RS(12, 4) code, the whole strip has 16 blocks, including 12 data blocks and 4 global checksum blocks.The 16 blocks can be equally divided into 4 groups of 4 blocks each.

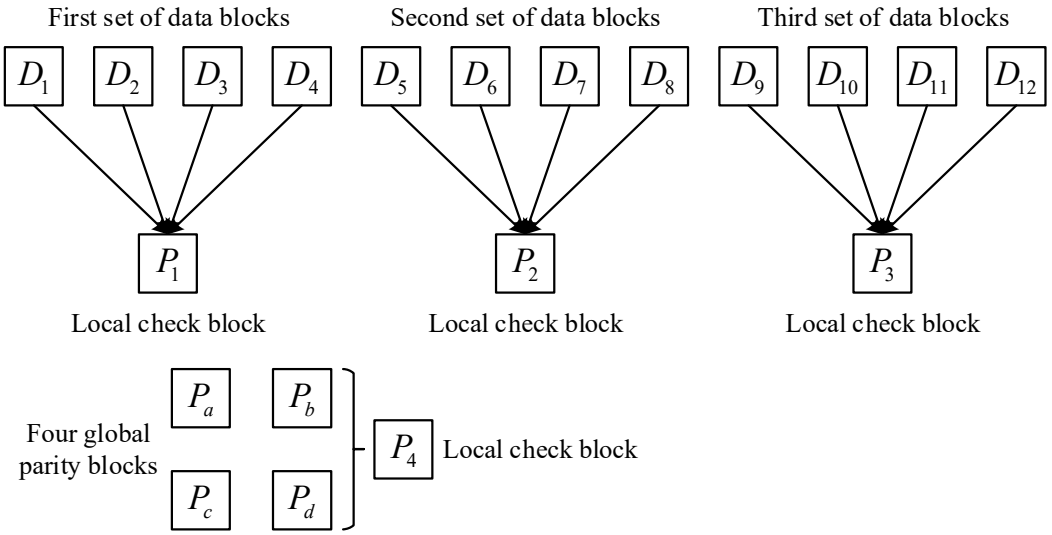


Fig. 2. RS-LRC code structure, k=12, m=4, l=4

Assume that the data blocks are respectively $D_1, D_2, \ldots, D_{12}$. First the global checksum block P_a, P_b, P_c, P_d is generated by generating matrix of RS(12, 4). then $D_1, D_2, ..., D_{12}$, and P_a, P_b, P_c, P_d are divided into four groups, the first group is (D_1, D_2, D_3, D_4), the second group is (D_5, D_6, D_7, D_8), the third group is $(D_9, D_{10}, D_{11}, D_{12})$ and the fourth group is (P_a, P_b, P_c, P_d).

Then each group computes a separate check block, called P_1, P_2, P_3, P_4. In computing global check block $P_i, i \in \{a, b, c, d\}$, the computation is based on a finite domain; while in computing local check blocks P_i, $i \in [4]$, it is a hetero-or operation, which is simple and less computationally expensive. The formula is as follows:

The remaining 3 checksum blocks can be obtained by the same calculation. With the extra checksum blocks, the network bandwidth and disk I/O consumption required to repair a single block can be greatly reduced. For example, for D_1, repairing D_1 requires only $P_1, D_2, D_3 +, D_4$. By using the formula: i.e:

$$D_1 = P_1 + D_2 + D_3 + D_4 \tag{2}$$

$$P_1 = D_1 + D_2 + D_3 + D_4 \tag{3}$$

2.2.3. Re-optimization. The optimized RS-LRC code composition is shown in Fig. 3.

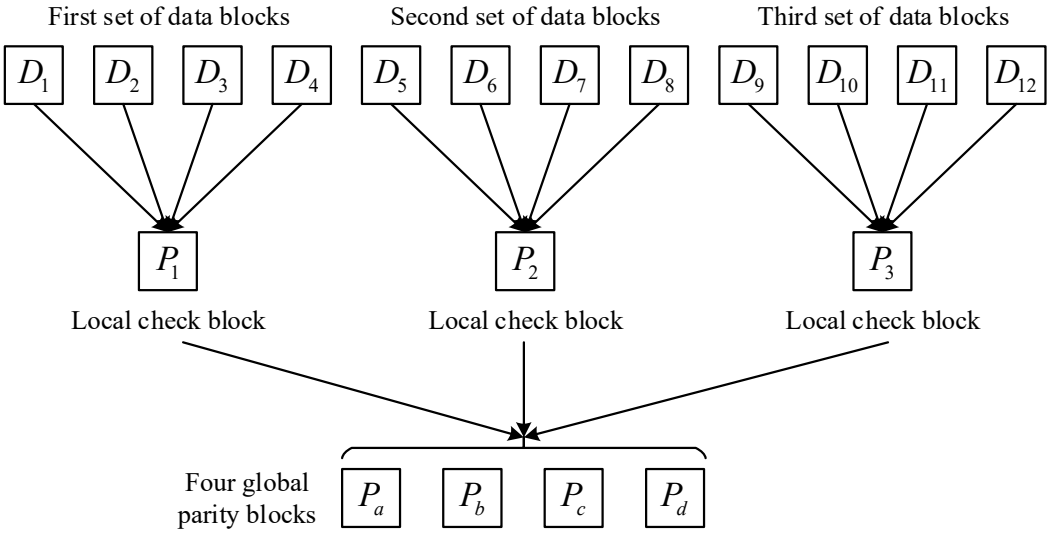


Fig. 3. Re-optimized RS-LRC

The fourth local check block P_4 can be replaced by the other three local check blocks P_1, P_2, P_3 due to properties specific to the van der Merwe matrix:

$$P_4 = P_1 + P_2 + P_3 \tag{4}$$

This eliminates the need for us to save P_4, thus saving one block of space per strip. When the global checksum block of the fourth group is lost, say P_a, it can be recovered by the following equation:

$$P_a = P_1 + P_2 + P_3 + P_b + P_c + P_d \tag{5}$$

In practical systems, it is very rare to directly use Vandermonth matrices for RS code coding and decoding, and most system implementations use generating polynomials to realize RS codes. Generating polynomials and matrix RS codes, although in different forms, belong to the same two different ways of defining RS codes, and the generating polynomials can be converted into matrix form by certain constructions. For RS(12, 4) code, the generating polynomial is $g(x)$:

$$\begin{aligned} g(x) &= (x-\alpha^0)(x-\alpha)(x-\alpha^2)(x-\alpha^3) \\ &= g_4x^4 + g_3x^3 + g_2x^2 + g_1x + g_0 \end{aligned} \tag{6}$$

where g_i is the coefficient of the generating polynomial. The addition, subtraction, multiplication and division operations described above are all operations in a finite field. The above generating polynomial $g(x)$ can be represented as an array $g = [g_0, g_1, g_2, g_3, g_4]$, where each value in the array corresponds to a coefficient of the generating polynomial. From this array, a $k \times n$ -dimensional generating matrix G can be constructed: the first 5 elements of the first row of G are g_0, g_1, g_2, g_3, g_4, and the next 11 elements are 0; the second row is obtained by cyclically shifting the first row to the right by one bit, i.e., the first element is 0, and the second to sixth elements are g_0, g_1, g_2, g_3, g_4, and the next 8 elements are 0; the next 10 columns are obtained in the same way, and the last 5 elements of the final column are g_0, g_1, g_2, g_3, g_4.

The parity check matrix H and the generating matrix G are orthogonal, i.e. $GH^T = 0_{k\times n}$. for each row h_i in H there is $Gh_i^T = 0_k \times 1$. And the first row in H is a n-dimensional vector of all 1's, which is known from $h_i, j = \alpha^{(i-1)}(j-1)$. So $G1^T = 0_{k\times 1}$. and $G1^T = 0_{k\times 1} \Leftrightarrow \sum_{i=1}^{16} g_i = 0_{k\times 1}$. g_i denotes row i of the generating matrix, $i \in [16]$. Which shows that $g_i (i \in [16])$ is a function of all other 15 columns. For g_{16} there is:

$$\sum_{i=0} g_{16} = 0 \tag{7}$$

This means that any of the columns can be obtained by adding the other 15 columns with the following formula:

$$g_{16} = \sum_{i=1}^{15} g_i = \sum_{i=1}^{4} g_i + \sum_{i=5}^{8} g_i + \sum_{9}^{12} g_i + g_{13} + g_{14} + g_{15} \tag{8}$$

$\sum_{i=1}^{4} g_i$ is the first local checksum block, $\sum_{i=5}^{8} g_i$ is the second local checksum block and $\sum_{i=9}^{12} g_i$ is the third local checksum block. So for RS(12, 4, 3), the data block and the checksum block conform to the following equation:

$$\sum_{i=1}^{12} D_i + P_a + P_b + P_c + P_d = 0 \tag{9}$$

$$P_1 + P_2 + P_3 + P_a + P_b + P_c + P_d = 0 \tag{10}$$

D, P represents the data block and checksum block respectively. According to the above equation,

if any one of the four global checksum blocks is lost, it can be recovered by the other three global checksum blocks and the three local checksums by performing the different-or operation together. In this way, the global checksum block can be recovered by six blocks even if no local checksum block is generated for the global checksum block.

The original form of G is not systematic, where k data blocks are encoded and replaced by new ones, which will result in reading a single data block, all of which need to be read into at least k blocks to be reconstructed, which will greatly degrade the system performance. So first an elementary transformation is done on the Vandermont matrix [28] to convert the submatrix of $k \times k$ into a unitary matrix. The final generated matrix is transformed to the following form: $G = [I_k \mid A]$. With the transformed form, the relationship $GH^T = 0_{k\times(n-k)}$ is more intuitive. $H = [-A^T \mid I_{n-k}]$, A^T is $(n-k)\times k$, so H is $(n-k)\times n$. i.e:

$$\begin{aligned} GH^T &= \left[I_k \left| A \right|\right]\left[-A^T \left| I_{n-k} \right|^T\right] \\ &= \left[I_k \mid A\right]\left[\frac{-A}{I_{n-k}}\right] \\ &= -A + A \\ &= 0_{k\times(n-k)} \end{aligned} \tag{11}$$

Finally, this paper designs a distributed massive data storage system (RS-LRC-HDFS) based on the optimization of class corrective censoring code and locally repairable code (RS-LRC for short) and verifies its effectiveness in storing massive data.

3. Results and analysis of RS-LRC-HDFS mass data storage

3.1. Analysis of data storage optimization effect of RS-LRC

3.1.1. LRC vs. RS. In order to analyze the decoding efficiency of the improved LRC over RS censored codes, (12,2,2) LRC and (12,4) RS are taken as an example, since they have the same storage cost of 1.33x. 40 test experiments are carried out respectively, and the comparative results of (12,2,2) LRC and (12,4) RS are shown in Fig. 4. From the figure, it can be seen that under the same storage cost, the time for LRC to decode any 2 faults is only about half of the time used for RS corrective censoring decoding, which achieves the purpose of this paper to improve RS corrective censoring. Of course, consider that (12,4)RS tolerates any 4-block faults, while (12,2,2)LRC has only roughly 85% decoding success rate for any 4-faults. However, both are more fault tolerant than three copies. In summary, compared to RS censoring codes, LRC trades the disadvantage of lower fault tolerance for a substantial increase in decoding performance, and maintains the same or even lower storage cost as RS censoring codes.

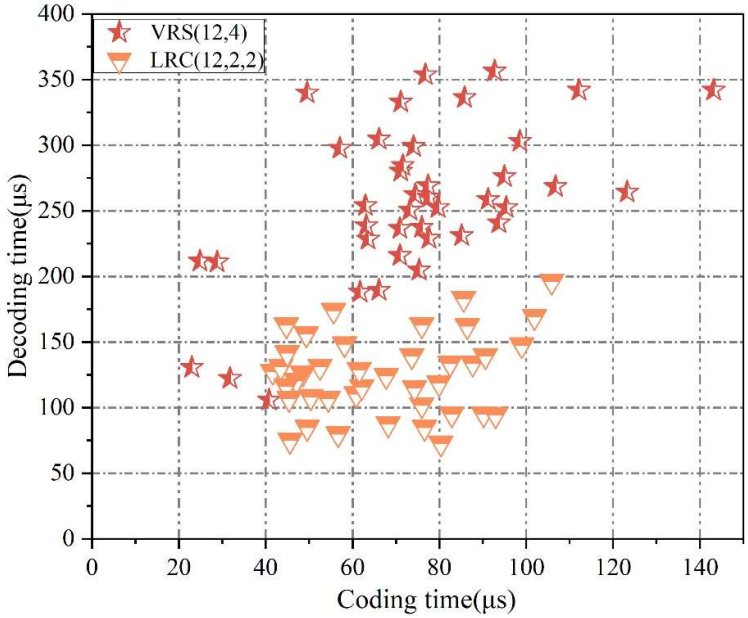


Fig. 4. Comparison results of LRC and RS

3.1.2. Parameter changes. This test is done by changing the parameters of the corrective censoring code and comparing the (6,2,2) LRC and (12,2,2) LRC, only the number of data blocks is increased and other parameters remain unchanged, the results of the experiments comparing the parameter changes of the LRC (16,3,3) and the LRC (6,2,2) are shown in Fig. 5. From the figure, it can be seen that the encoding time and decoding time of (12,2,2) LRC are higher than that of (6,2,2) LRC.The average values calculated from the respective test data are 46.54 microseconds for (6,2,2) LRC and 10.42 microseconds for decoding, whereas (12,2,2) LRC has an average of 62.58 microseconds and decoding average of 15.6 microseconds. Here it can be seen that changing the parameter variations to double the number of data blocks does not significantly improve the encoding and decoding times, and the encoding time increases more than the decoding time.

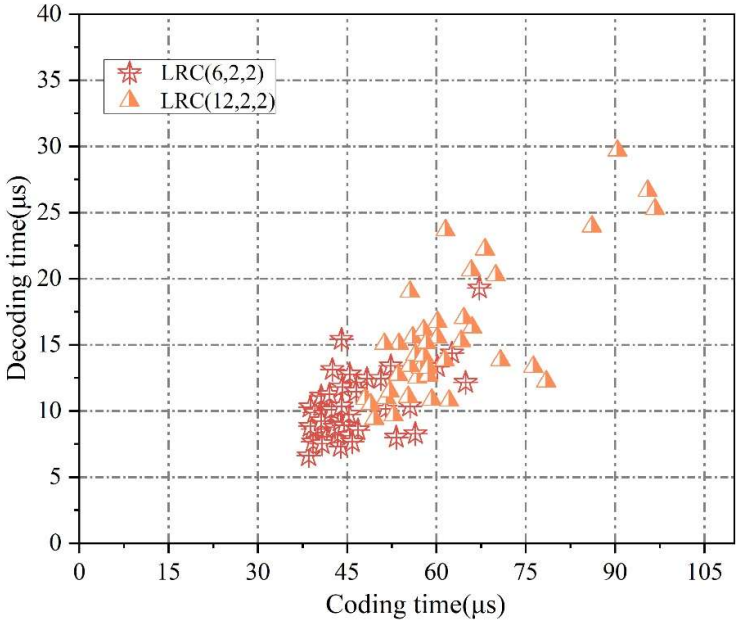
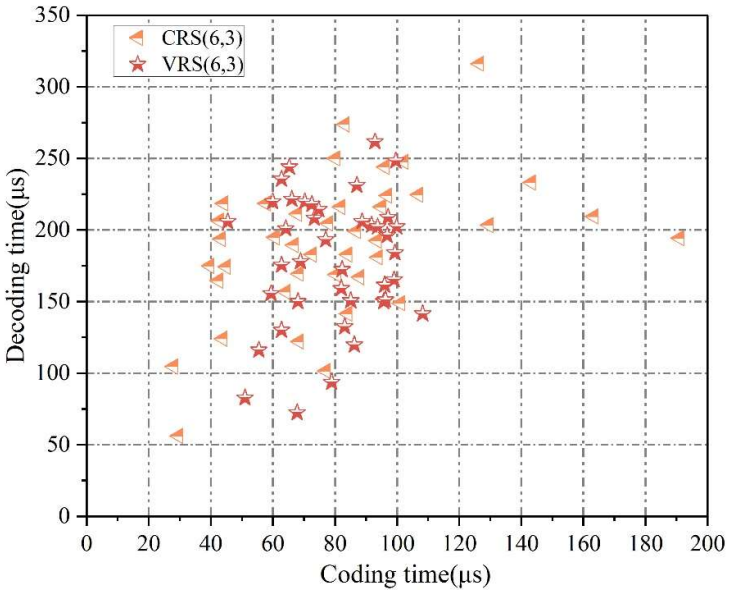


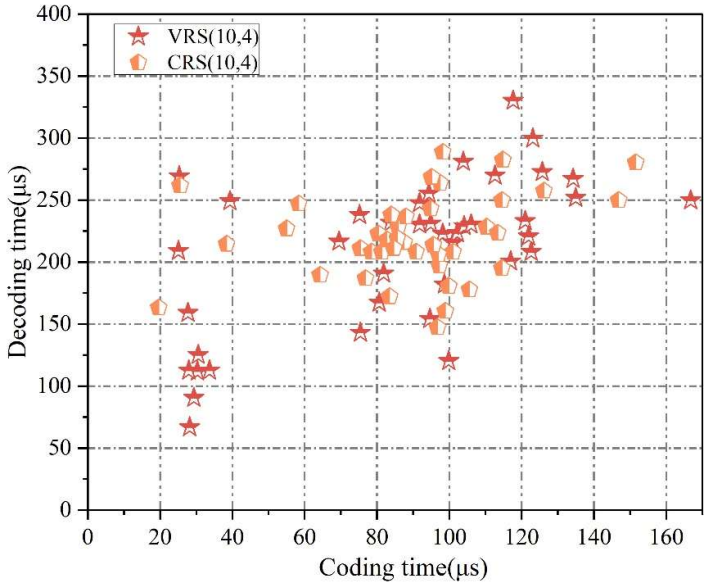
Fig. 5. LRC (12,2,2) and LRC (6,2,2) parameters change the results

3.1.3. Generating Matrix Comparisons. The main comparison is between Vandermonth RS corrective censoring codes and Cauchy RS corrective censoring codes, which also reflects the comparison of efficiency between Vandermonth LRC and Cauchy LRC.The results of the comparison of VRS and CRS generation matrices are shown in Fig. 6, where (a) stands for VRS (6,3), CRS (6,3), and (b) stands for VRS (10,4), CRS (10,4).

From Fig. (a), it can be seen that the Vandermonth RS censored code is more concentrated than the Cauchy RS censored code in each group of test results, while the coding span of the Cauchy RS censored code is much larger, which shows that the stability of the Cauchy RS censored code is poorer. From Fig. (b), it is found that the coding and decoding spans of both Kersey RS censored code and Vandermonth RS censored code are larger after changing the parameter combinations, which indicates that the performance of Vandermonth RS censored code and Kersey RS censored code are similar.



(a) VRS(6,3), CRS(6,3)



(b)VRS(10,4), CRS(10,4)

Fig. 6. VRS and CRS generation matrix contrast results

3.2. Validation of RS-LRC-HDFS Data Storage Effectiveness

3.2.1. Evaluation indicators. In the paper, we propose to introduce local repair coding based on systematic RS-LRC codes into HDFS systems, and we focus on the following evaluation metrics in this section of experiments.

1) Storage overhead: given the same amount of data volume while achieving the same fault tolerance, we pursue the system with higher efficiency, i.e., the system that requires less space is superior.

2) Repair bandwidth: the total amount of data that needs to be fetched from a normal node in order to repair a failed data. The normal node sends coded data to the new node with a bandwidth equal to the size of the transmitted coded block. The new node will perform decoding operation on the received coded block to recover the lost data.

3) Disk I/O: For the disk I/O operation, the main concern in the paper is the disk I/O during the

restoration process, because the encoded data block received by the new node can only be decoded on this node to restore the original data, so the biggest indicator to reduce the disk I/O is that the number of normal nodes can be reduced.

In this section, the main purpose is to further verify the time of reading and writing different files in the RS-LRC optimized HDFS data storage file system based on the HDFS storage system (HDFS for short), the original RS's HDFS storage system (RS-HDFS for short), and the method in this paper (RS-LRC-HDFS), respectively. The vertical coordinates in the following results are all time, and the units are all milliseconds.

3.2.2. Time overhead of the data writing process. The time overhead of the data writing process is shown in Figure 7. The results show that when the size of the file increases from 100MB to 1000MB, the time overhead of writing data for HDFS, RS-HDFS and RS-LRC-HDFS systems are at 7.12×104, 7.43×104 and 1.44×104 milliseconds, respectively. The time overhead of writing data using RS-LRC optimized distributed data storage coding technique is reduced by 81.12% compared to the traditional HDFS system when the file size reaches 1000MB. This shows that the RS-LRC based HDFS data storage optimization proposed in this paper is extremely effective.

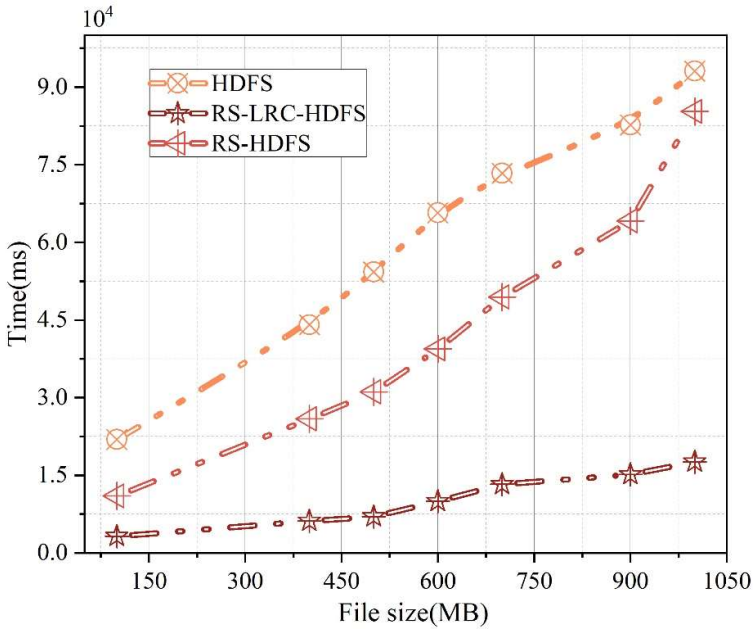


Fig. 7. The time overhead of the data writing process

3.2.3. Time overhead of the data reading process. The time overhead of the data reading process is shown in Fig. 8. The results show that the HDFS system has the largest time overhead for the data reading process under the three algorithms, followed by RS-HDFS, while the RS-LRC-HDFS system has the smallest. When the size of the file is 100MB, the three systems consume 1.05×104, 0.35×104 and 0.0735×104 milliseconds to read the data, respectively; and when the size of the file reaches 1,000MB, the time consumed by its data reading is 4.37×104, 2.94×104 and 0.775×104 milliseconds, respectively. The RS-LRC optimized distributed data storage based coding technique proposed in this paper reduces the time to read the file by 93.01% and 82.27% compared to the traditional and RS-HDFS systems for file sizes of 100MB and 1000MB, respectively. This may be related to the fact that the HDFS system takes longer time because it needs to decode the coded blocks during the massive data reading process, whereas the localized repair coding of the RS-LRC-HDFS system can directly perform the reading operation on the original data, so it takes less time.

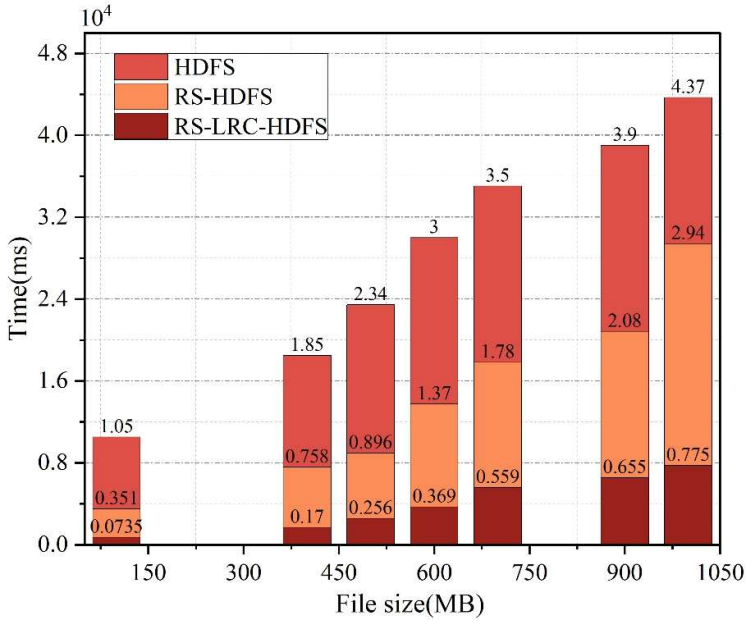


Fig. 8. The time overhead of the data reading process

3.2.4. Time overhead of the node repair process. The time overhead of the node repair process is shown in Figure 9. As can be seen from the figure, when the file size increases from 5MB to 1000MB, the time overhead of the node repair process for the three systems HDFS, RS-HDFS and RS-LRC-HDFS increases by 91.09%, 93.16% and 85.36%, respectively. When the file size is 1000MB, the time overhead of the node repair process is 2.51×104, 11.9×104 and 19.69×104 milliseconds for HDFS, RS-HDFS and RS-LRC-HDFS systems, respectively. In comparison, the optimization model proposed in this paper has a larger difference in the time overhead of the node repair process for massive data than the traditional HDFS, and the time overhead of the RS-LRC-HDFS system is reduced by 17.18 × 104 milliseconds compared to the HDFS system. And the time gap increases as the failed data block increases. Since inexpensive machines are mostly used in HDFS system, the possibility of failure is more, so it is more important to repair the data quickly with less bandwidth consumption.

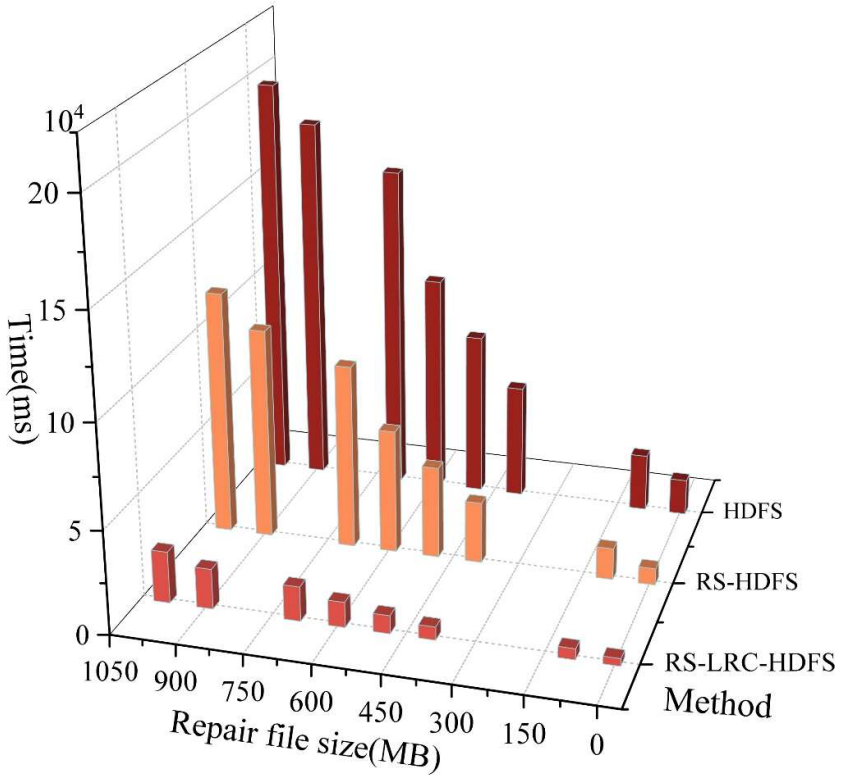


Fig. 9. The time overhead of the node repair process

3.3. Case Study of Massive Data Storage Based on RS-LRC-HDFS

3.3.1. Comparison of data storage characteristics:

1) Select the datasets smaller than HDFS chunk size (128M) from the test dataset.

2) Select five test data sets from the re-selected test set with different number sets of 100, 500, 1000, 5000 and 10000 respectively.

3) Record the storage time under different test data sets.

The result of data storage efficiency comparison between HDFS and RS-LRC-HDFS system is shown in Figure 10. From the results it can be seen that in the case of the same amount of data RS-LRC-HDFS is more efficient in storing files of different data levels, and in the case of different orders of magnitude of files, the use of RS-LRC-HDFS storage system is significantly more effective than the HDFS storage system. At 10,000 files, the RS-LRC-HDFS system is significantly more effective at storing 83.27% more files than the HDFS system. Overall, RS-LRC-HDFS is more suitable for massive data file storage than HDFS.

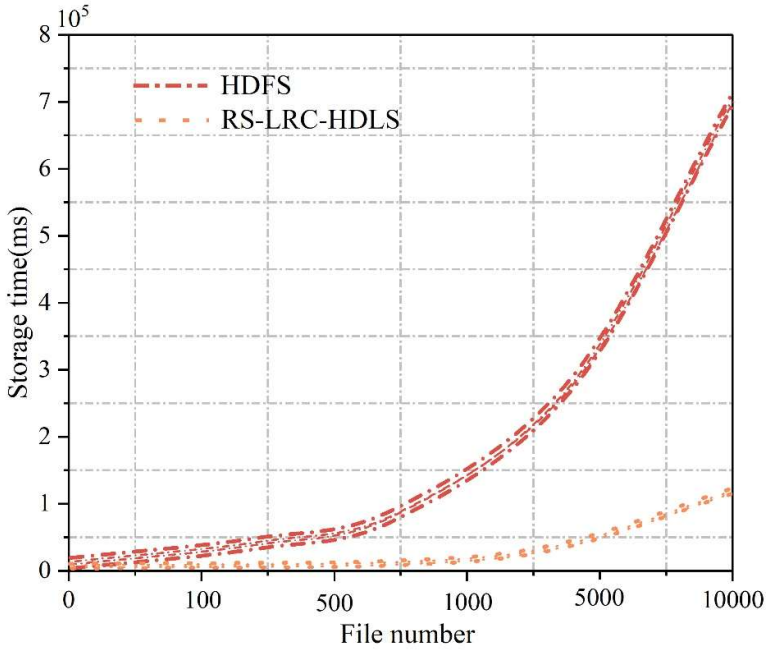


Fig. 10. The storage efficiency of HDFS and RS-LRC-HDFS systems

3.3.2. Data storage efficiency comparison results. Data storage query efficiency is an important indicator of massive data storage system, in the test data randomly selected data sets of different data volume sizes, by verifying the native distributed storage scheme in the same hardware configuration environment and the storage scheme used in this paper for comparison test.

1) Randomly select data sets of 10GB, 100GB, 500GB, 800GB, and 1TB from the test data.

2) Store different test datasets in HDFS, RS-HDFS and RS-LRC-HDFS storage schemes, respectively.

3) Perform 15 storage operations for each test set respectively and take the average of 15 storage operations as the final consumption time.

The results of the storage efficiency comparison of the different models are shown in Fig. 11. It can be seen that when the number of files is small, the number of large and small files is relatively balanced, and both storage models need to send requests to the main storage node, although the number of requests of this research scheme is much less than the original storage technology scheme, the storage time of the two is similar. With the gradual increase in the number of files, the advantages of the storage management model are gradually reflected. When the data size is increasing, the storage efficiency of this research scheme (RS-LRC-HDFS) gradually increases and remains stable, and when the size of the file increases from 10GB to 1000GB, the improvement of the storage efficiency of this paper's model increases by 10.45%-57.7% compared with the original distributed storage method (HDFS).

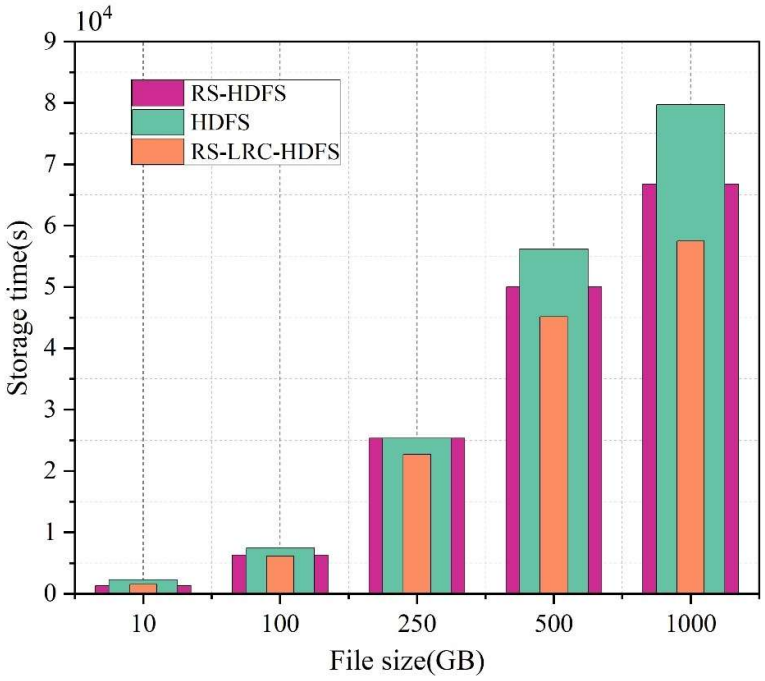


Fig. 11. Comparison of storage efficiency of different models

3.3.3. Data query efficiency comparison results. In order to verify the difference in data query speed between the storage management model of this study and the original distributed storage scheme, five groups of test data are selected from the test data, respectively, stored and 50 files are randomly selected in each group of test data for query operation, and the average of 15 times is taken as the final query time. The results of the query efficiency comparison of different models are shown in Figure 12. It can be seen that the query time is saved by reducing the number of requests to the main storage node when retrieving remote sensing data files and querying the files directly from the index, so the query speed of the storage management model in this study is better than the query speed of the original distributed storage scheme. The query efficiency is consistently much higher than the original distributed query scheme and increases by 55.3%-81.5% and 62.95%-74.54% for this paper's model over the traditional HDFS model and RS-HDFS model, respectively, as the data size increases from 10GB to 1000GB. When the data size of the file is increased to 1000GB, the query time required by the three models, HDFS, RS-HDFS and RS-LRC-HDFS, is 71.43s, 54.25s and 13.94s, respectively. It can be seen that the data querying efficiency is highest in this paper's proposed data querying, and the model has the best application.

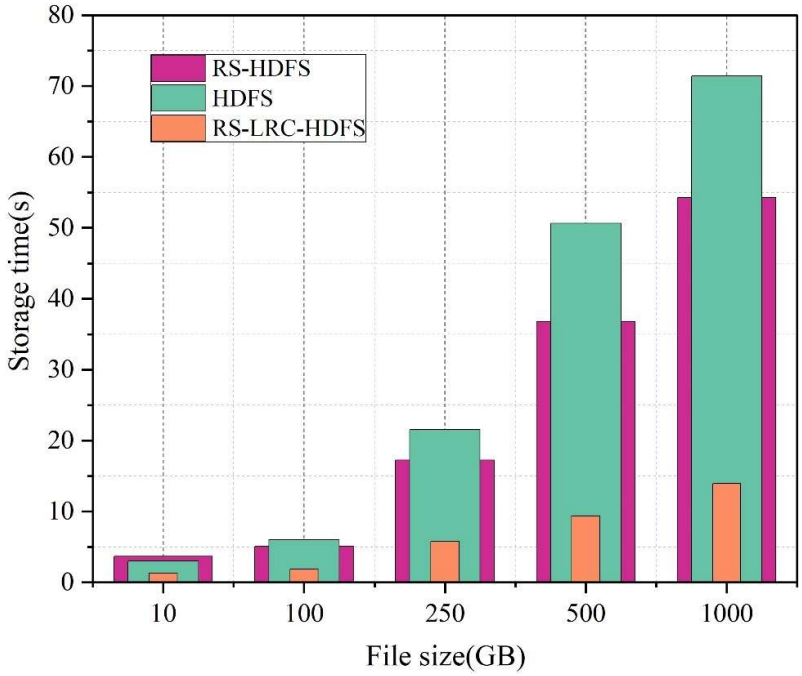


Fig. 12. Comparison results of query efficiency of different models

4. Conclusion

In this paper, for the massive data storage requirements, the RS-LRC technology is used to optimize the design of the storage system of the distributed computing model (HDFS), and the optimized storage model (RS-LRC-HDFS) is obtained, and the application effect of the model before and after the optimization is analyzed. The experimental results show that:

1) LRC is a good solution to the problem that RS censored codes consume a large amount of system I/O resources in the decoding process, and the change of LRC parameter will produce a big performance difference, and only increase the number of coded data blocks, LRC has no effect on the decoding performance, and the effect on coding performance is very small compared with that of RS coding, which means that LRC can provide more parameter combinations to choose from without decreasing the performance than RS censored codes. This shows that LRC can provide more choices of parameter combinations than RS censored codes without decreasing the performance. The performance of the Vandermonth matrix is similar to that of the Cauchy matrix in terms of coding and decoding performance.

2) In addition to paying some storage overhead as well as computational overhead, the proposed method in this paper of applying the local repair encoding of the system LRC code to the HDFS system reduces the disk I/O, as well as reduces the repair bandwidth. The optimized system improves the data write speed by 81.12%, read speed by 93.01%, and repair time by 87.25%.

3) It is concluded through simulation test that RS-LRC-HDFS is more suitable for storing massive data files than HDFS. And in the case of 1000GB data query, the storage efficiency of HDFS optimized with RS-LRC technology is 10.45%-57.7% and 55.3%-81.5% higher than that of pre-optimization storage efficiency and query efficiency, respectively.

References

[1] Maftei, A. A., Lavric, A., Petrariu, A. I., & Popa, V. (2023). Massive data storage solution for IoT devices using

blockchain technologies. Sensors, 23(3), 1570.

[2] Li, Z., Kong, X., Zhang, J., Shao, L., Zhang, D., Liu, J., ... & Qiu, C. W. (2022). Cryptography Metasurface for One-Time-Pad Encryption and Massive Data Storage. Laser & Photonics Reviews, 16(8), 2200113.

[3] Fayoumi, A., Orachorn, C., & Shi, X. (2024, July). From Big Data to Massive Data: Towards a Massive Data Storage Solution for the Internet of Senses. In 2024 IEEE 7th International Conference on Big Data and Artificial Intelligence (BDAI) (pp. 284-289). IEEE.

[4] Lv, T., He, C., Zhang, J., & Song, Z. (2021, June). Massive AIS data storage and query based on Hadoop platform. In Journal of Physics: Conference Series (Vol. 1948, No. 1, p. 012016). IOP Publishing.

[5] Alghushairy, O., & Ma, X. (2022). Data storage. In Encyclopedia of Big Data (pp. 338-341). Cham: Springer International Publishing.

[6] Sangat, P., Indrawan-Santiago, M., & Taniar, D. (2018). Sensor data management in the cloud: Data storage, data ingestion, and data retrieval. Concurrency and Computation: Practice and Experience, 30(1), e4354.

[7] Li, R., Song, T., Mei, B., Li, H., Cheng, X., & Sun, L. (2018). Blockchain for large-scale internet of things data storage and protection. IEEE Transactions on Services Computing, 12(5), 762-771.

[8] Harb, H., Makhoul, A., & Abou Jaoude, C. (2018). A real-time massive data processing technique for densely distributed sensor networks. IEEE Access, 6, 56551-56561.

[9] Qiu, J., Wu, Q., Ding, G., Xu, Y., & Feng, S. (2016). A survey of machine learning for big data processing. EURASIP Journal on Advances in Signal Processing, 2016, 1-16.

[10] Shen, H., Zhang, M., Wang, H., Guo, F., & Susilo, W. (2021). Efficient and privacy-preserving massive data processing for smart grids. IEEE Access, 9, 70616-70627.

[11] Chandy, A. (2019). Smart resource usage prediction using cloud computing for massive data processing systems. J Inf Technol, 1(02), 108-118.

[12] Leskovec, J., Rajaraman, A., & Ullman, J. D. (2020). Mining of massive data sets. Cambridge university press.

[13] Peng, J., Tang, M., Li, M., & Zha, Z. (2017). A load balancing method for massive data processing under cloud computing environment. Intelligent Automation & Soft Computing, 23(4), 547-553.

[14] Hsu, K. (2022). Big data analysis and optimization and platform components. Journal of King Saud University-Science, 34(4), 101945.

[15] Fu, W., Liu, S., & Srivastava, G. (2019). Optimization of big data scheduling in social networks. Entropy, 21(9), 902.

[16] Abdalla, H. B., Ahmed, A. M., & Al Sibahee, M. A. (2020). Optimization driven MapReduce framework for indexing and retrieval of big data. KSII Transactions on Internet and Information Systems (TIIS), 14(5), 1886-1908.

[17] Balaji, S. B., Krishnan, M. N., Vajha, M., Ramkumar, V., Sasidharan, B., & Kumar, P. V. (2018). Erasure coding for distributed storage: An overview. Science China Information Sciences, 61, 1-45.

[18] Dau, H., & Milenkovic, O. (2018). MaxMinSum Steiner systems for access balancing in distributed storage. SIAM Journal on Discrete Mathematics, 32(3), 1644-1671.

[19] Raman, R. K., & Varshney, L. R. (2018, June). Dynamic distributed storage for blockchains. In 2018 IEEE International Symposium on Information Theory (ISIT) (pp. 2619-2623). IEEE.

[20] István, Z., Sidler, D., & Alonso, G. (2017). Caribou: Intelligent distributed storage. Proceedings of the VLDB Endowment, 10(11), 1202-1213.

[21] Sohn, J. Y., Choi, B., Yoon, S. W., & Moon, J. (2018). Capacity of clustered distributed storage. IEEE

Transactions on Information Theory, 65(1), 81-107.

[22] Shahid, F., Ashraf, H., Ghani, A., Ghayyur, S. A. K., Shamshirband, S., & Salwana, E. (2020). PSDS–proficient security over distributed storage: a method for data transmission in cloud. IEEE Access, 8, 118285-118298.

[23] Elyasi, M., & Mohajer, S. (2020). Cascade codes for distributed storage systems. IEEE Transactions on Information Theory, 66(12), 7490-7527.

[24] JunnanLiu,ShengyiJin,DongWang & HanLi. (2024). An archive - based method for efficiently handling small file problems in HDFS. Concurrency and Computation: Practice and Experience(24),e8260-e8260.

[25] Monia Hamdi,Heni Bouhamed,Fady Badreddine & Reem Ibrahim Alkanhel. (2024). Deep recurrent neural networks distributed on a Hadoop/Spark cluster for fall detection. INTERNATIONAL JOURNAL OF COMPUTERSCOMMUNICATIONS & CONTROL(3),

[26] Ying Song,Jialin Liu,Yingai Tian & Bo Wang. (2025). An Optimized Method for Batch Recovery Based on Erasure Coding in Heterogeneous Network. Sensors(2),346-346.

[27] Li Ruihu,Yang Sen,Rao Yi & Fu Qiang. (2021). On binary locally repairable codes with distance four. Finite Fields and Their Applications

[28] Acharya Kuldip,Ghoshal Dibyendu & Bhattacharyya Bidyut K.. (2020). Segmentation of images through curve fitting analysis by modified Vandermonde matrix and modified Gram-Schmidt method. IET Image Processing(17),4588-4598.