Journal of Network and Innovative Computing
ISSN 2160-2174 Volume 14 (2026) pp. 1-15

Article

Computer Algorithm Based Modeling of Distributed Storage Performance Enhancement and Data Consistency

Yichi Zhang [1,*]

[1] Mathematics and Computer Science, University of California, San Diego, California, US, 92092
* Correspondence author: zhangyichi20070203@163.com

Abstract: This paper focuses on the performance of distributed storage and data, using computer algorithms to achieve the enhancement of distributed storage performance, and constructing a data consistency model to enhance data synchronization during distributed storage. The consistency hash algorithm is used to realize the mapping of data and storage host nodes, and the adjustment mechanism for load balancing is generated based on weights and load balancing metrics. A causal consistency model based on hot data governance (Horae) is constructed to record the changes in version vectors (SVV) due to data synchronization using a visible service, while hot data is sorted and backed up in Horae storage service to speed up the search and simplify the consistency verification process. Test results show that the method in this paper can effectively realize the performance enhancement of distributed storage, and the load balancing difference under dynamic storage clusters is still below 8%. In the data consistency test, the model of this paper also has obvious advantages in throughput, operation response time and remote update visible delay method compared with other models. The results of this paper provide scientific references for further optimization and improvement of distributed storage systems.

Keywords: computer algorithm; consistent hash algorithm; Horae model; data consistency; distributed storage

1. Introduction

With the rapid development of the Internet information age and the diversification of user demands, distributed cloud storage technology with the advantages of high performance and scalability has emerged [1-2]. In order to provide high-speed, reliable and low-cost distributed storage services, cloud service providers generally distribute server nodes in different geographical locations. User data is distributed and stored in these servers according to a certain strategy [3-5]. The state of synchronizing the data among the nodes in a distributed system so that it remains consistent is called data consistency [6]. It is categorized according to its strength into final consistency, causal consistency and strict consistency. Final consistency is the weakest strength, which refers to the final realization of the data in each node, which can maintain the data consistency of the distributed environment [7-8]. Strict consistency is a kind of strong consistency, which means that the update in each node is immediately visible to other nodes, which has very high requirements on the performance of the distributed system, and it is difficult to be realized in practical application scenarios [9-11]. Causal consistency belongs to the intermediate strength data consistency, as long as the update events in a node satisfy the causal order can be visible to other nodes [12-13]. Currently, data consistency in distributed environment has become one of the key issues in the application of distributed storage technology.

Distributed storage systems widely use read-write consistency algorithms to ensure data availability, and by using specific protocols, it effectively ensures the consistency of data read and written by users [14-15]. Compared to the classical Byzantine problem where data tampering exists, the read-write consistency problem assumes that all nodes will abide by the communication protocol to transmit user data [16-18]. Meanwhile, by deploying read-write consistency algorithms, distributed storage systems can avoid the problem of service interruption in case of failure of some nodes, thus effectively

improving the availability of stored data [19-20]. However, read-write consistency algorithms usually incur large network communication and storage overheads, and reduce the ability of user data access in distributed storage systems [21-23]. In the process of developing distributed storage systems, developers need to deploy read-write consistency algorithms that meet the requirements of storage application scenarios and adjust the algorithm execution mechanism [24-25]. However, selecting the applicable read-write consistency algorithms according to the application scenarios is still very challenging.

This paper provides a brief introduction to the distributed storage system architecture and analyzes the shortcomings of the Controlled Replication Under Scalable Hashing (CRUSH) algorithm in the traditional distributed storage to provide an innovative direction for this paper to study the performance enhancement methods for distributed storage. The consistent hash algorithm is utilized to map each data in each storage system to the hash ring one by one, while the FNV-1 (Flower-Noll-Vo-1) algorithm is used to compute the hash value of the node and the data, thus realizing the mapping of the data and the storage host nodes. Considering the performance differences of each host, virtual nodes are allocated based on weights to realize load adaptation and establish a dynamic allocation mechanism for virtual nodes. Taking data governance as a ballistic entry point, the causal consistency model based on hot data governance is proposed to simplify the causal sequence test by sorting, accelerate data stability, and optimize the model update visibility and read response latency. The proposed method in this paper is tested and compared with other methods or models respectively, and the enhancement of distributed storage performance and data consistency of this paper's method is demonstrated based on the comparison results.

2. Distributed storage system architecture

2.1. OSD storage

The Object Storage Device (OSD) is responsible for the actual data storage and processing.Each OSD node is an independent daemon that manages and maintains the object data in the distributed storage cluster.The OSD nodes are responsible for persistently storing the object data to the local hard disk. The local hard disk splits the data into fixed-size blocks and uses the Reliable, Self-Healing Distributed Object Storage (RADOS) protocol to store the blocks to different OSD nodes in the cluster.The OSD nodes can receive data access requests from clients or other nodes and provide data read and write services.The OSD nodes use distributed object storage protocols such as RADOS Block Device and RADOS Gateway, to communicate with clients and handle data transfer and response.

2.2. Monitor

In distributed storage system architecture [26], Monitor nodes are responsible for cluster monitoring and management.Monitor nodes regularly detect and monitor various components in the cluster, including OSD nodes, RADOS storage pools, Placement Group (PG) status, etc.Monitor nodes collect and maintain the status information of these components and store it into Monitor Map. The Monitor nodes in a distributed storage cluster are elected using the Paxos algorithm, which ensures that a sufficient number of Monitor nodes are always active in the cluster. When a Monitor node is down or offline, the other Monitor nodes will re-elect a new Monitor node.

2.3. MDS

The Metadata Service (MDS) is the component responsible for managing metadata, providing a file system interface that enables users to access and manage file data stored in the cluster as if they were using a traditional file system.The MDS is responsible for managing the file system's metadata, including information about directory structure, file attributes, permissions, and so on.The MDS maintains a global metadata table, which records metadata about files and directories, and is responsible for handling metadata read, write, and update operations. To enhance access performance, the MDS maintains a metadata cache that stores the most commonly used metadata information in memory, thereby reducing access to the storage cluster and speeding up metadata read and update operations.

2.4. Client

Client refers to an application or user that uses a distributed storage cluster and interacts with the cluster through specific interfaces (librados, libcephfs, etc.) to read, write, and manage data.Client can send read and write requests to the cluster through the interfaces, requesting data for a specific object or

file.Client can specify the object's Client can specify the object's identification number (ID) or file path and send a read request to the cluster. The cluster calculates the location of the object based on the Controlled Replication Under Scalable Hashing (CRUSH) algorithm and returns the data to the Client.

3. Computer algorithms for performance enhancement

As mentioned before in the distributed storage system, the distributed storage cluster will calculate the object location and return the data to the Client according to the CRUSH algorithm.CRUSH is a pseudo-random data distribution algorithm based on hash, which maps the data objects to the storage device through a controlled process to ensure the reliability of the data, and it is the guarantee of the distributed qualities and flexibility of the storage system. However, the CRUSH algorithm also has some limitations: in the implementation of distributed storage, the input parameters of the storage node selection function crush_do_rule include CRUSH map, replica placement rule number, weight vector, etc., in which the weight of the storage node recorded by the CRUSH map is by default proportional to the capacity of the storage node, the probability of the OSD to be selected and the value of weight are positively correlates with the weight value, and the weight value is the only constraint for CRUSH to map PGs to OSDs when other conditions (cluster topology, replica placement rules, etc.) remain unchanged. This indicates that storage capacity is the only factor for CRUSH to select OSD under deterministic conditions, which theoretically can only satisfy the demand of data distribution balance, and in practice, there are defects in data balance, ignoring the impact of the underlying network state as well as the heterogeneous and load conditions of storage nodes (including CPU resources, memory resources, OSD disk type, and OSD load, etc.) on the performance of clusters. Due to the variable read/write loads and the limited uniformity of data distribution, OSDs with poor network conditions, lower configurations, or higher loads are also indiscriminately selected to respond to client read/write operations, which leads to cluster performance degradation. Therefore, it is necessary to consider network state information as well as storage node heterogeneity and load conditions to enhance the performance of distributed storage.

3.1. Information mapping based on consistent hash algorithm

To address the shortcomings of CRUSH algorithm in distributed storage system, this paper considers the performance enhancement of distributed storage by using consistent hash algorithm. The key to realize distributed data storage using consistent hash algorithm [27] is the need to realize the mapping of data and storage host nodes. In this paper, the IP address of the storage host is used as the feature value of the storage host, while the name of the distributed storage system is used as the Key value, and each data is mapped to the hash ring one by one using the consistent hash algorithm. The FNV-1 (Flower-Noll-Vo-1) algorithm is used to compute the hash value of the nodes and the data, the FNV-1 algorithm is able to compute a large amount of data while maintaining a small conflict rate, because of its highly decentralized nature, it is suitable for computing some very similar strings.

3.2. Load balancing adjustment mechanism

Mapping host nodes and bracket nodes on a hash ring, monitoring data and host nodes are ideal if they are uniformly distributed on the hash ring, as shown in Fig. 1(a). The data node searches for the nearest host node in a clockwise direction, and key 1 falls on host node 2, i.e., data 1 is stored on host 2, and when there is a large amount of data, and so on for the allocation of storage tasks. When there are fewer storage hosts, as the storage nodes are allocated to the hash ring with randomness after the hash algorithm calculation, it may lead to uneven distribution of storage nodes and hash offset phenomenon, as shown in Fig. 1(b). In this paper, virtual nodes are introduced, as shown in Fig. 1(c), and the virtual nodes added are indicated by the “#” sign.

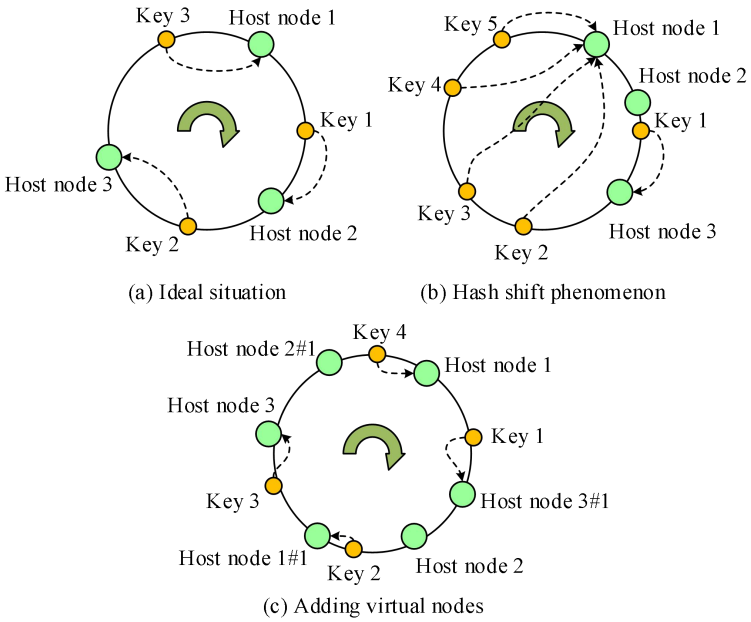

Figure 1. Task allocation based on the consistent hash algorithm

3.2.1. Assigning virtual nodes based on weights

According to the definition of host weights, the ratio of host weights is the ratio of the inverse of the host's initialized performance index P. The ratio of each host's weights is calculated using equation (1):

$$\begin{cases} P_i = 100 \times \left(a_c P_c + a_m P_{mi} \right) And\ a_c + a_m = 1 \\ W_1 : W_2 : \cdots : W_i \approx 1/P_1 : 1/P_2 : \cdots : 1/P_i \end{cases} \tag{1}$$

where: P_i is the initialized performance evaluation index of each host, P_{ci} and P_{mi} are the initialized CPU occupancy and memory occupancy of the i th host, respectively. a_c , a_m are the degree of influence of the initialized CPU occupancy and memory occupancy of the hosts, respectively; W_i is the weight of each host, taken as a positive integer. The number of virtual nodes is set according to the weights:

$$\begin{cases} N = N_1 + N_2 + \cdots + N_i \\ N_1 : N_2 : \cdots : N_i = W_1 : W_2 : \cdots : W_i \end{cases} \tag{2}$$

where: the total number of host nodes is N and N_i is the number of nodes of the i rd host, the number of virtual nodes added to each host is $N_i - 1$.

3.2.2. Dynamic allocation mechanisms

In the process of system operation, when the load balancing evaluation index S of the system exceeds the set threshold, the set system triggers the dynamic allocation mechanism to sort all the host load condition evaluation indexes L_i , subtract 1 from the host virtual node with the largest value of L_i , add 1 to the host virtual node with the smallest value of L_i , and then re-execute the load balancing evaluation of the system until S is within the set threshold.

The host load condition evaluation metrics are determined by several factors, including the host's CPU occupancy, memory occupancy, and task volume. The single host load condition evaluation index L_i is:

$$L_i = 100 \times \left(b_c L_d + b_m L_{mii} + b_t L_u \right) And\ b_c + b_m + b_i = 1 \tag{3}$$

where: A smaller value of L_i means a smaller load. L_{ci} , L_{mi} , and L_{1i} are the CPU occupancy, memory occupancy, and task volume percentage of the i th host, respectively. b_c , b_m , and b_i are the degree of influence of CPU occupancy, memory occupancy, and task volume percentage,

respectively.

Using the standard deviation-based load balancing evaluation method, the load balancing of the system is evaluated based on the host load condition evaluation, and the load balancing evaluation indexes of the distributed storage system are calculated using Eqs. (4) and (5) S :

$$\overline{L} = \frac{\sum_{i=1}^{N} L_i}{N} \tag{4}$$

$$S = \frac{1}{L}\sqrt{\frac{\sum_{i=1}^{N}\left(L_i - \overline{L}\right)^2}{N}} \tag{5}$$

where: L_i is the load condition evaluation index for each host. L is the average of the load condition evaluation metrics for each host. N is the total number of hosts. S is the load balancing evaluation metrics of the storage system.

4. Data consistency model

4.1. Data consistency requirements

A complete distributed storage system is formed by many nodes in different locations connected together through a network, a huge amount of data is distributed throughout the system in different nodes. All computers connected to the distributed storage system can access the data in any of the nodes. Some of them are read-only data, some are writable data, and a very important issue when reading and writing data is to ensure that the data read each time is credible. Here not only refers to the authenticity of the data, this paper is more concerned about whether the data is dirty data, i.e., data that has been updated, but is not reflected in the access.

In distributed storage systems the amount of data is huge and the number of copies of each data object is large, so maintaining the consistency of the copies in each node becomes a critical issue at this point.

4.2. Data consistency classification

Data consistency can be categorized into strong and weak consistency. Strong consistency means that the modification of a writable data by one node is immediately updated to the whole system. Weak consistency, also known as final consistency, means that a node's modification of a writable data is not immediately synchronized to each replica, but is passed to one replica and then to other replicas, which ultimately makes the data consistent across replicas.

Strong consistency ensures that all concurrent accesses to any data will not produce data inconsistency errors. However, it has high requirements on the system, which limits the availability of the system, and this method is basically unachievable when the number of nodes is large. Therefore, strong consistency is an ideal data consistency scheme, and in reality, it will be a trade-off between the degree of dependence on data consistency in practical applications and the performance of the system.

Compared with strong consistency, weak consistency can not guarantee that the data at each moment is the latest data, but its algorithms and system requirements are relatively low, relatively easy to realize. For most practical applications, do not need too strong real-time, as long as you can ensure that the final data can achieve consistency, within a certain range has been sufficient to meet the application requirements, therefore, very often choose the weak consistency program.

4.3. Causal consistency model based on hot data governance

4.3.1. Model Architecture Design and Notation Definition

(1) Model Architecture Design

The causal consistency model based on hot data governance (Horae) is based on the traditional causal consistency model [28], which reduces the read response latency and update visibility latency based on the increase of throughput through the Horae service. Fig. 2 shows the Horae model infrastructure, where an undirected graph is used between data centers to indicate that the data center storage model is a fully geo-replicated strategy.

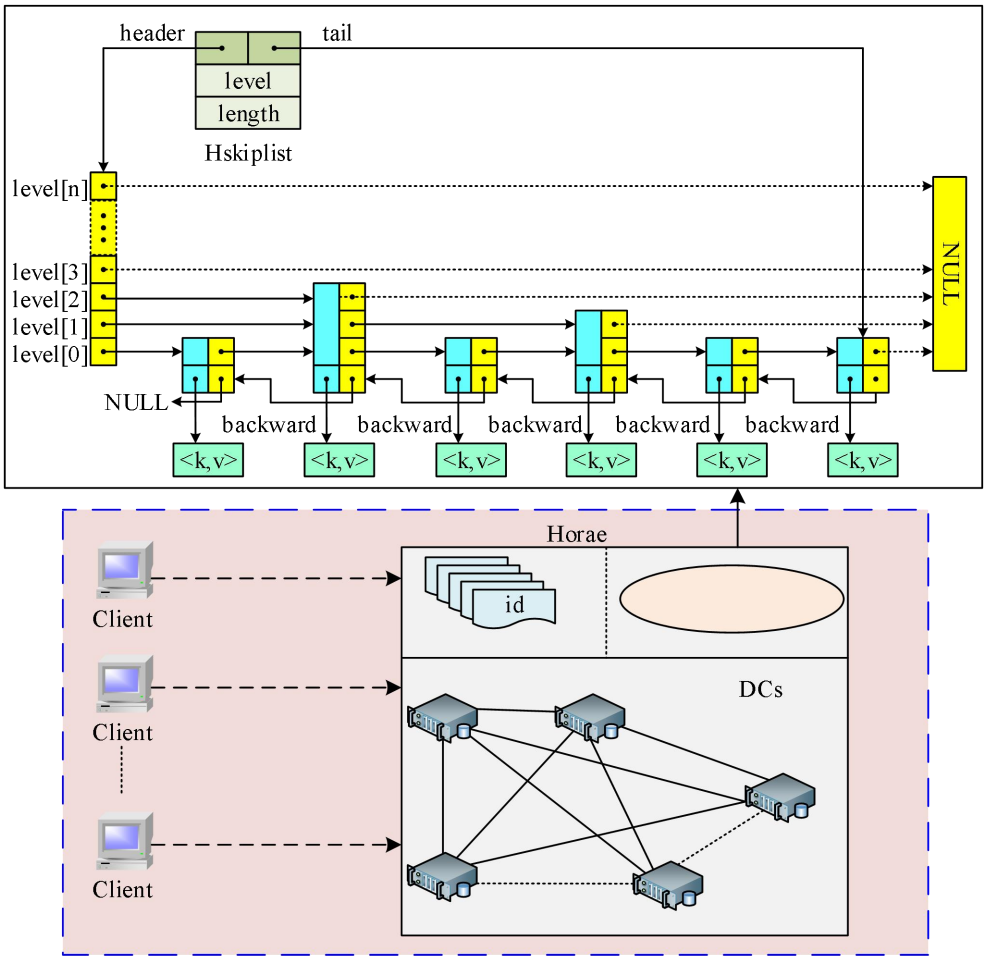

Figure 2. Horae model architecture

(2) Definition of notation

The fundamentals and design of the Horae model are presented in this section.The Horae model usage approach can be used to improve any existing solution based on a global stabilization process to enhance causal consistency across geographic locations. The notation used in the model in this paper is specified as N denotes a partition, M denotes a replica of each partition, p_n^m denotes the n th partition of the m replica, DS_c denotes the dependency vector of the client c, SRV_n^m is the stabilized remote vector of the p_n^m, SRV_c is the stabilized remote vector of the client c, $Clock_n^m$ is the physical clock of the p_n^m, and SVV_n^m is the version vector of the p_n^m. sr for the source replica, dt for the timestamp, Ds for the dependency set, HDS for the Horae service storage dependency set, and PVL_n^m for the p_n^m partition visible label.

4.3.2. Design methodology

In the Horae model, the basic services are divided into visible and storage services.The Horae model uses visible services to record SVV changes due to data synchronization, seeking to improve data real-time performance while meeting causal consistency requirements. In addition, hot data is sorted and backed up in the Horae storage service to speed up search and simplify the consistency verification process.

(1) Horae Model Visibility Service

The Horae Visibility Service pages Partitioned Visibility Labels (PVLs) using the data center ID as an index to record changes in data visibility.The Horae model stores SVV vectors in real-time via PVL labels to visually represent the local application state of remote data. As a result, the Horae model can read data in causal order without waiting for periodic computations of the global stabilization process. This reading pattern enables the Horae model to maximize the update visibility of thermal data under

the requirement of causal consistency.

(2) Horae model storage services

The underlying Horae storage service is implemented as a Hskiplist, which consists of a dictionary and a jump table. The jump table stores $< dt , k , SRV >$, representing the corresponding timestamps dt, keys k and stable remote vectors SRV. The jump table is used to assist the dictionary in sorting by maintaining multiple pointers to other nodes in each node for fast access to nodes. The dictionary then holds the mapping and identification information from k to v. The implementation looks up the corresponding key values with O(1) complexity. Thus Hskiplist implements node lookups with average O(logN) and worst O(N) complexity, and can also batch process nodes through sequential operations.

(3) Data Activity Policy

The Horae model periodically prunes the Horae storage service using the data activity policy to ensure that the data is real-time.The Horae storage service records only an approximation of the frequency of accesses (co) of the data item in relation to the data activity and uses the data activity policy to maintain the data activity on the basis of the activity updates.The Horae model maintains the data activity in the background using equations (6) and (7):

$$idle = t_{now} - dt \tag{6}$$

$$co = co - \frac{idle}{d_f} \tag{7}$$

In Eq. t_{now} is the physical clock at the current moment, $idle$ represents the data item access idle time, i_f is the growth factor, d_f is the discount factor, while co represents the data activity. Equation (6) aims to obtain the idle waiting time of the data item, and then the data activity is discounted by equation (7), and the data activity incremental strategy is the suffix increment.

In the process of comparison calculation of data activity using approximation calculation method, through the random number between 0-1 r and the activity factor p to compare, when $r < p$, increase the data activity, p factor calculation formula as shown in Equation (8):

$$p = \frac{1}{(co - i_c) * i_f + 1} \tag{8}$$

The formula indicates that the higher the current data activity, the lower the probability that the raw access frequency will increase. This approach reduces the case of inactivity after a short period of high frequency accesses, i.e., the access frequency decreases after a short period of frequent accesses.

In addition, to prevent new key values from being pruned, Equation (9) is used to assign the default data activity value of i_c to the frequency of newly inserted key values:

$$co = i_c \tag{9}$$

4.3.3. Horae causal consistency agreement

(1) Data synchronization mechanism

In Horae model, data center write operation triggers real-time incremental synchronization. Data center i sends a REPLICATE message to remote data center m, which receives the REPLICATE message and inserts it into the local version chain according to the causal order. After the insertion is complete, the partition key value timestamp updates the version vector $SVV_n^m[i]$. Unlike the traditional causal consistency model, the Horae model synchronizes the SVVs through PVLs to record the synchronization status of the data centers.

However, in a real-world environment, a special case exists: since no update operation is triggered, p_n^m does not send a REPLICATE message to its corresponding replica. As a result, the m th term of the corresponding version vector and the m th term of the stable remote vector SRV cannot be monotonically incremented. This situation can expose the model to slow replicas. To avoid this, it is stipulated that if a partition is not updated within Δt time, its current hybrid logical timestamp is sent to its replica as a broadcast.

(2) Global stabilization policy

To save metadata storage overhead, the Horae model uses a propagation tree to exchange information for nodes (partitions) within the same data center. Every Δt unit of time, the partitions

within the data center share their version vectors SVV. When $k \neq m$, $svv_n^m[k]$ is the latest timestamp received by the server p_n^m from the p_n^k update. Otherwise it indicates the current physical timestamp of the corresponding partition. p_n^m computes the minimum value of the known version vector as SRV_n^m. Once the partition computation is complete, it is shared with the other peer partitions to compute the datacenter SRV. The SRV is a vector timestamp that contains visible clock entries for all datacenters in the system. If in data center i. $SRV[j]=t$ is equivalent to an update in data center j with a timestamp less than t already visible locally. Figure 3 shows the global stabilization schematic.

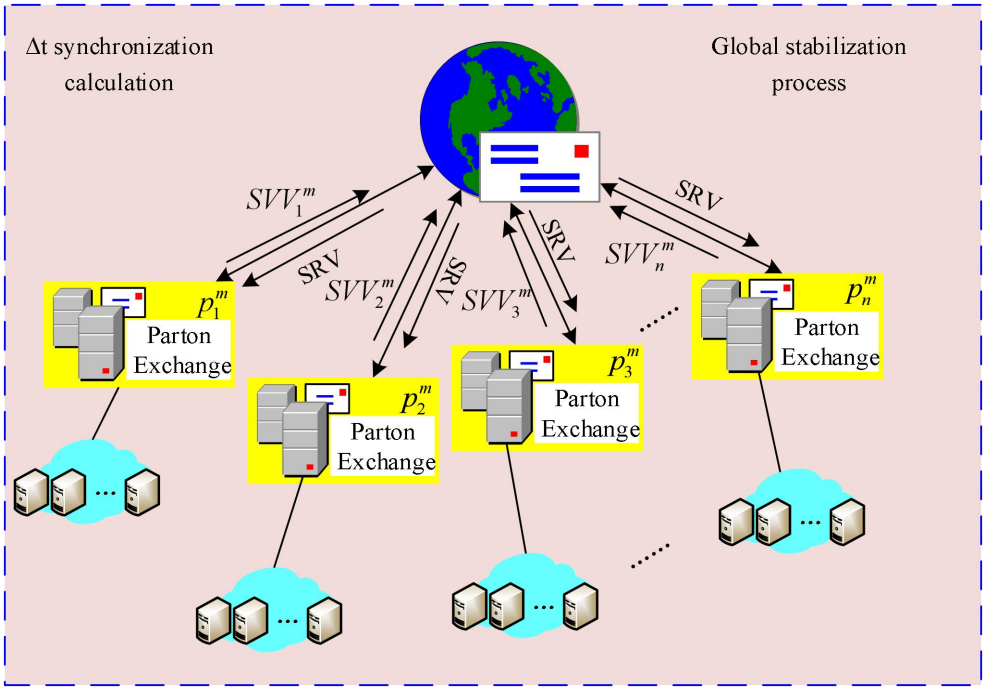

Figure 3. Global stability process

5. Analysis of methodological tests

5.1. Distributed Storage Performance Enhancement Testing

5.1.1. Experimental platform and deployment

The experiment was conducted using a high performance computer on VMware Workstation software. The parameters of the system platform are respectively: CPU: Intel core i7 8750H, memory: 16GB DDR4, hard disk: 2*512GB, the core server is Windows 10 professional, the virtual machine system is VMware Workstation pro, the server operating system is Linux CentOS 7. By default, the distributed storage system experimental platform has been built.

5.1.2. Selection of experimental data

In this paper, the research for the postal industry courier data storage, courier company staff every scanning a courier face sheet, the system will record a piece of data. Experimental data selected from a province postal industry big data information statistics system within the data provided by a courier company. This data is part of the information of the scanned express face sheet, a total of 11 fields such as A ~ K, such as the scanning time, express bill number.

5.1.3 Load balancing tests

Using the experimental data in the previous section, the courier single number as a unique identifier, mapped to the hash ring, the real server to the IP address is also mapped to the ring, the use of IntelliJ IDEA development software to write Java code, the consistency of the hash algorithm through the API to connect to the storage interface of the distributed storage system, overriding its self-contained block storage strategy. Under the same size of data, a number of data storage experiments to take the

statistical average, you can get the load difference of different storage nodes, the results are shown in Figure 4. As can be seen from the figure, this paper's algorithm (Ours) and ordinary virtual nodes of the consistency of hash storage data strategy (Method2) is not much different, different cluster size in the division of data, the load difference in the node in 5.0% to 7.5%, compared to the HDFS storage data allocation strategy (Method1) has a certain degree of improvement. It can be speculated that the test results will also stabilize in this interval as the amount of test experiment data increases with the number of cluster storage nodes.

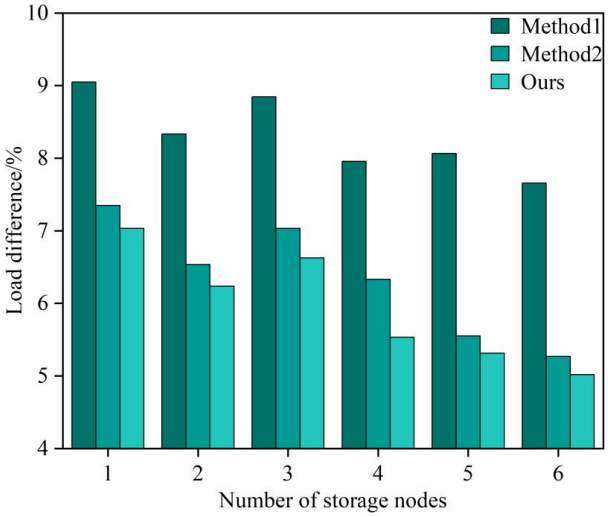

Figure 4. Isostatic load balancing

In the configuration phase of the virtual machine, the disk space of the storage nodes is set to be different from each other to differentiate the load factor and set the heterogeneous condition of the cluster. Repeat the previous experimental step, distribute the data to different slave nodes, conduct multiple data storage experiments to take statistical averages and get the load difference of different slave nodes, as shown in Figure 5. It can be seen that HDFS and virtual nodes of the consistency of the hash two data storage strategy in the face of heterogeneous clusters, the load difference fluctuates greatly, while this paper's algorithm according to the set load balancing evaluation index S , the calculation of the node in the distribution of data according to the weight of the proportional division, with the increase in the amount of data and the number of nodes, it can be speculated that the load difference of the entire cluster stabilized at around 5%, cluster load difference decreases significantly.

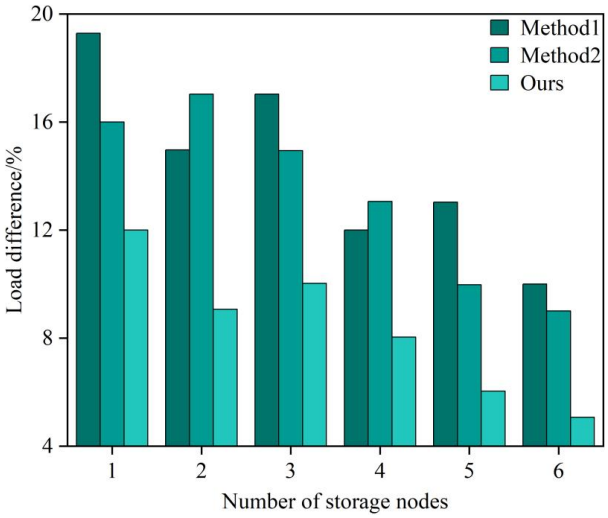

Figure 5. Heterogeneous cluster load equalization

In order to verify the dynamic data balancing of the cluster, on the basis of Figure 5, 5000 read and write operations are performed on the data of the storage nodes to experiment with the load shifting strategy between data nodes, and the results are shown in Figure 6, where the arrows in the figure

indicate the gap between the load difference values of different methods. It can be seen that compared to the HDFS storage strategy, the data distribution of the consistent hash node using virtual nodes is relatively more uniform, and the load difference is stable at about 20~25%. The algorithm in this paper introduces the load balancing evaluation index with threshold S, realizes the dynamic distribution of the cluster, and takes the average in many tests, and the load difference value reaches less than 8% (about 7.88%), and the practicability has been significantly improved to meet the design requirements. At the same time, the system load in the experiment changes gently, the number of nodes changes without large fluctuations in the overall load, indicating that the operation of adding and deleting nodes in the storage system will not cause excessive impact on the system.

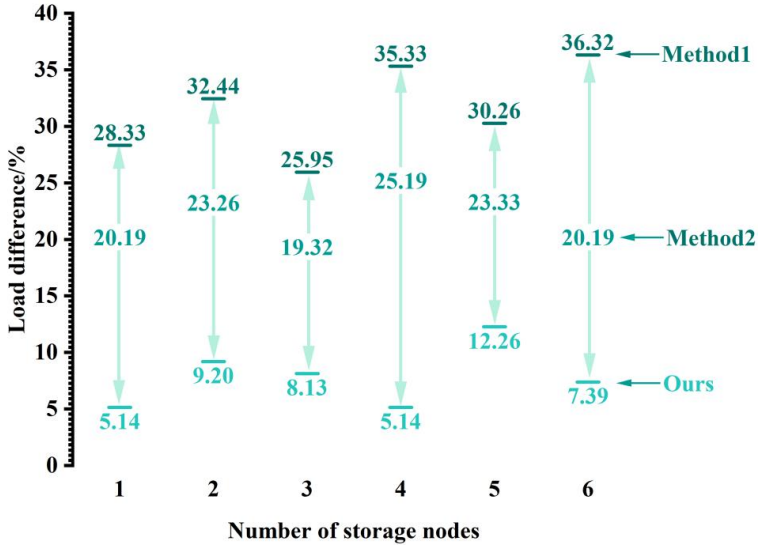

Figure 6. The cluster load equilibrium contrast after the read-write task

5.2. Data consistency testing

5.2.1. Experimental setup

To test the performance of the protocols in the Horae model, experiments are conducted using the DKVF platform to lay out and manage distributed replica nodes in the same environment as in the previous section. The experiments are also compared with the latest models Wren and POCC that use scalar dependencies.The DKVF platform uses the Java programming language and the Berkeley db database. The database supports full ACID transaction semantics and also provides a simple database programming interface in distributed storage systems.

5.2.2. Throughput analysis

Throughput is the total amount of key values updated by the model per unit of time, and is an important measure of model performance. The experiments in this subsection take the traditional experimental approach of increasing the number of data centers and partitions respectively to evaluate the throughput performance of the model.

First, the impact of the number of data centers on the model throughput is analyzed by controlling the partition variable and setting the number of partitions in each data center to 6. The client sends read and write requests to the local data center, and the data centers are periodically synchronized with each other. Figure 7 shows the variation of throughput corresponding to the number of data centers from 2 to 12. With the increase of the number of data centers, the throughput of the Wren model changes significantly, while the throughput of the POCC model and the Horae model changes less. The reason for this is that the Wren model is based on partial geographic replication strategy, and the increase in the number of data centers leads to an increase in the data synchronization overhead, which reduces the throughput of the model.The POCC model and the Horae model are based on a full geographic replication strategy, and only some of the data centers in the model are synchronized with each other, and the throughput of the model does not change much. When the number of data centers is 8, the Horae model improves the throughput performance by 32.12% over the Wren model and 15.09% over the POCC model. The POCC model and Horae model use the full geo-replication strategy to improve the throughput of the model compared to the Wren model. The global stabilization strategy of Horae

model avoids operational delays and achieves higher throughput compared to POCC model.

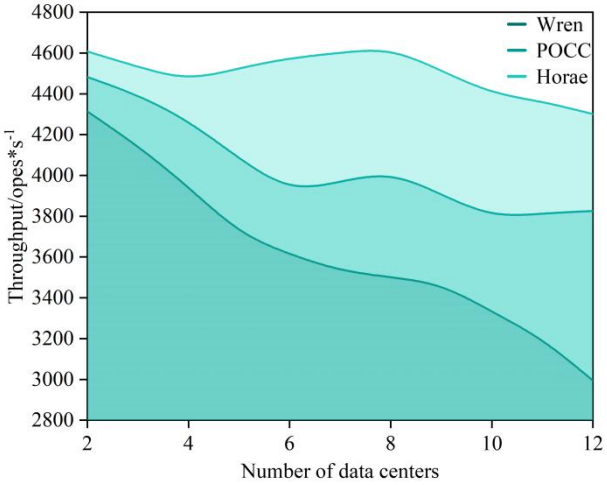

Figure 7. Throughput of the number of different data centers

Second, the impact of the number of partitions on the model throughput is analyzed by controlling the number of data centers variable, setting the number of data centers to 4, and changing the number of partitions in the data centers. Fig. 8 shows the variation of throughput corresponding to the number of partitions from 2 to 12, with the increase of the number of partitions, the throughput of the model shows an upward trend. Compared with Wren and POCC models, Horae model's its partition vector stores only part of the data center metadata with adjacencies, and the metadata is propagated through synchronous messages, which reduces the metadata overhead and achieves the throughput improvement.

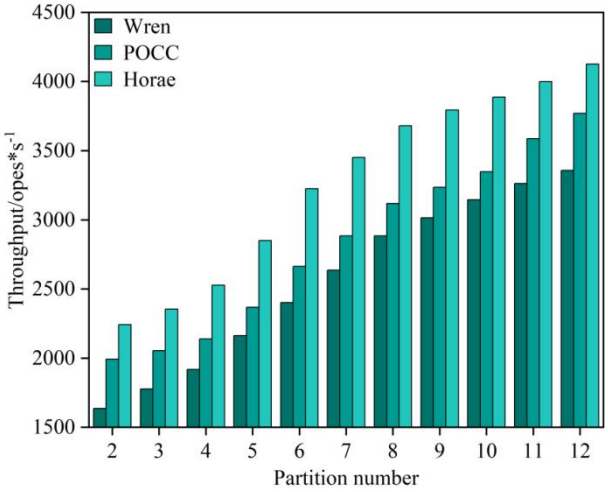

Figure 8. Throughput of the number of different partitions

5.2.3. Operational response time analysis

According to the traditional experimental way of testing the operation response time, this section of the experiment is divided into two parts to measure the operation response time in the model. In the first part, no clock deviation is set between partitions, and the change of the operation response time in the model is observed by changing the number of partitions in the data center. In the second part, two virtual servers are deployed on a physical device, and 2,000 operation requests are sent in a cyclic manner to artificially control the value of clock deviation between the servers and accurately analyze the effect of clock deviation between the servers on the operation response time.

Figure 9 shows the variation of operation response time with the number of partitions in the model, and Figure 10 shows the variation of operation response time with the value of clock deviation between partitions in the model. As the number of partitions and the value of clock deviation between partitions increase, the operation response time of the model also increases.Compared with Wren and Horae

models, the use of physical clocks in the global stabilization strategy of the POCC model is highly affected by the clock deviation between partitions, which produces a high operation latency.The use of hybrid logical clocks in the global stabilization strategy of the Wren model can provide operation consistent with the cause and effect of the clock deviation between partitions when there is a clock deviation between partitions. When there is clock deviation between partitions, it can provide timestamps for operations that conform to causal dependencies, which effectively avoids operational delays. Compared with the Wren model, the Horae model speeds up part of the global stabilization process by simplifying the causal dependency validation process of the hot data metadata and reduces the operation response time by 26.60%.

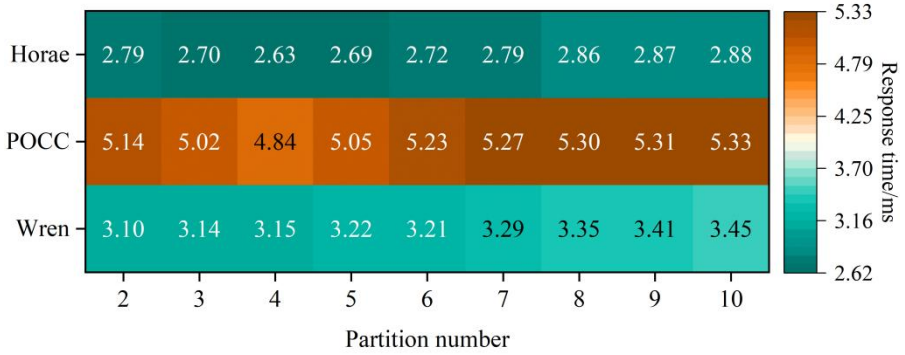

Figure 9. Operation response time of different partitions

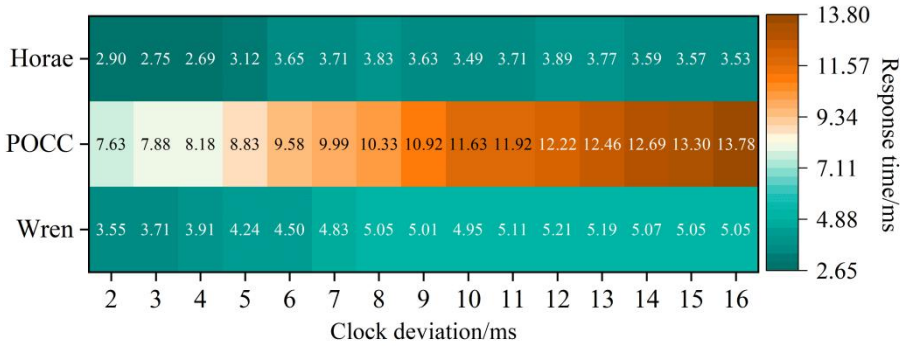

Figure 10. Operating response time of different clock deviations

5.2.4. Remote Update Visible Delay Analysis

Remote update visibility latency refers to the time interval between data written in the local data center and synchronized to the remote data center, and also plays an important role in measuring the model performance, even if a delay of a few milliseconds may cause anomalies by reading incorrect outdated data. The remote update visibility delay is determined by the minimum clock value received by the server, firstly the effect of clock deviation on the remote update visibility delay is analyzed by controlling the clock deviation value between the servers artificially. Secondly no clock deviation is set between the servers and the number of data centers in the model is increased to observe the change in the remote update visibility delay.

Figure 11 shows the variation of remote update visibility delay with respect to the value of clock deviation and Figure 12 shows the variation of remote update visibility delay with respect to the number of data centers. The results show that remote update visibility delay is affected by clock deviation between servers, which increases linearly with the value of clock deviation, and the number of heartbeat values received by its nodes increases linearly with the number of data centers, and the remote update visibility delay also increases. Compared with Wren model, POCC model and Horae model are based on a full geo-replication strategy where remote updates are synchronized among all data centers, which reduces the remote update visibility latency.The Wren model uses a generic stable vector UST, where a data center corresponds to a timestamp in the vector, and the remote updates are made visible by stipulating the completion of synchronization of all remote data centers, and the remote update visibility latency of the model depends on the synchronization of the local data center to the remote data center. The POCC model uses a global stable timestamp GST, and the remote update visibility delay of the model depends on the time when the local data center (i.e., the data center where

the data is written) is synchronized to the data center with the largest latency, and an increase in the number of data centers produces smaller global stable timestamps, thus causing the remote update visibility delay to be lower than that in the POCC model, and the remote update visibility delay to be higher than that in the POCC model. The Horae model uses a remote stabilization vector SRV_n^m, which includes only the timestamps of the local data center d_{t_1} and the timestamps of the remote data centers d_{t_2}, and the model's remote update visibility latency depends on how long it takes for the local data center i to synchronize with the data centers in the storage cluster. As the number of data centers increases, the remote update visibility delay of the Horae model is significantly lower than that of the POCC model.

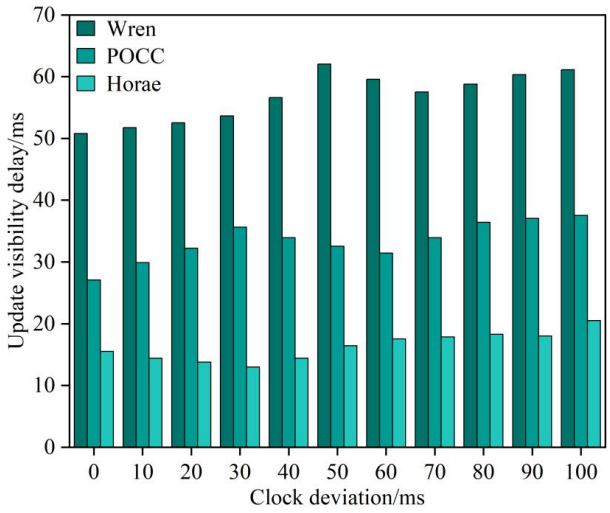

Figure 11. Remote update visible delay of different clock deviations

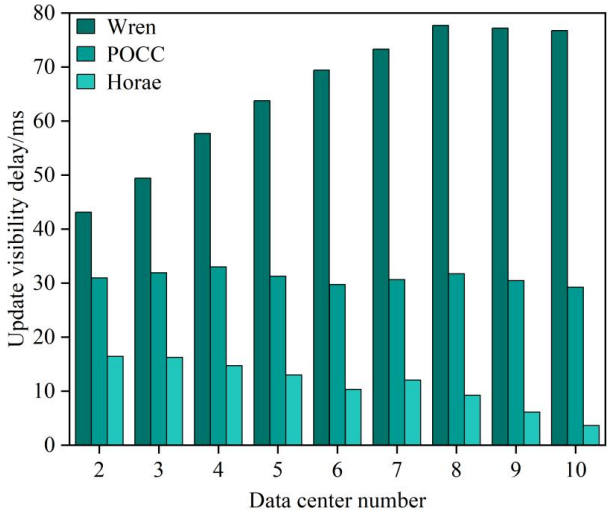

Figure 12. Remote update visibility latency of different data center numbers

6. Conclusion

This paper proposes a method to enhance the performance of distributed storage and data consistency based on computer algorithms, and tests and analyzes the method proposed in this paper on high-performance computers. The results show that the performance enhancement algorithm in this paper has obvious advantages over the allocation strategy of HDFS storage data and the strategy of consistent hash storage data of common virtual nodes, and achieves the lowest load balancing difference in both isomorphic and heteromorphic data, and the load balancing difference of this paper's algorithm is significantly smaller than that of the other two strategies in the dynamic clustered data, which shows the advanced distributed storage performance. Meanwhile, the throughput, operation response time and remote update visible latency of this paper's Horae model based on hot data

governance are smaller than those of the Wren model and POCC model. It shows that the model in this paper can fully realize the data consistency in distributed storage and further enhance the distributed storage performance.

References

1. Odun-Ayo, I., Ajayi, O., Akanle, B., & Ahuja, R. (2017, December). An overview of data storage in cloud computing. In 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS) (pp. 29-34). IEEE.
2. 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.
3. István, Z., Sidler, D., & Alonso, G. (2017). Caribou: Intelligent distributed storage. Proceedings of the VLDB Endowment, 10(11), 1202-1213.
4. Wang, T., Byna, S., Dong, B., & Tang, H. (2018, September). UniviStor: Integrated hierarchical and distributed storage for HPC. In 2018 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 134-144). IEEE.
5. Yang, H., Shin, W., & Lee, J. (2018). Private information retrieval for secure distributed storage systems. IEEE Transactions on Information Forensics and Security, 13(12), 2953-2964.
6. Krechowicz, A., Deniziak, S., & Łukawski, G. (2021). Highly scalable distributed architecture for NoSQL datastore supporting strong consistency. IEEE Access, 9, 69027-69043.
7. Chen, L., Dai, W., Qiu, M., Liu, M., & Xiong, Z. (2017, November). Flexible Consistency for Distributed Storage Systems. In 2017 IEEE International Conference on Smart Cloud (SmartCloud) (pp. 232-237). IEEE.
8. Sivaramakrishnan, K. C., Kaki, G., & Jagannathan, S. (2015). Declarative programming over eventually consistent data stores. ACM SIGPLAN Notices, 50(6), 413-424.
9. Roohitavaf, M., Demirbas, M., & Kulkarni, S. (2017, September). Causalspartan: Causal consistency for distributed data stores using hybrid logical clocks. In 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS) (pp. 184-193). IEEE.
10. Tang, Y., Sun, H., Wang, X., & Liu, X. (2017). Achieving convergent causal consistency and high availability for cloud storage. Future Generation Computer Systems, 74, 20-31.
11. Shudo, K., & Yaguchi, T. (2017). Causal Consistency for Data Stores and Applications as They are. Journal of Information Processing, 25, 775-782.
12. Lu, H., Liu, H., Ye, C., Liao, X., Mao, F., Zhang, Y., & Jin, H. (2023, May). Software-Defined, Fast and Strongly-Consistent Data Replication for RDMA-Based PM Datastores. In 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 90-101). IEEE.
13. Gavrielatos, V., Katsarakis, A., Joshi, A., Oswald, N., Grot, B., & Nagarajan, V. (2018, April). Scale-out ccNUMA: Exploiting skew with strongly consistent caching. In Proceedings of the Thirteenth EuroSys Conference (pp. 1-15).
14. 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.
15. Wang, M., & Zhang, Q. (2020). Optimized data storage algorithm of IoT based on cloud computing in distributed system. Computer Communications, 157, 124-131.
16. Li, Y., Gai, K., Qiu, L., Qiu, M., & Zhao, H. (2017). Intelligent cryptography approach for secure distributed big data storage in cloud computing. Information Sciences, 387, 103-115.
17. Khelaifa, A., Benharzallah, S., Kahloul, L., Euler, R., Laouid, A., & Bounceur, A. (2019). A comparative analysis of adaptive consistency approaches in cloud storage. Journal of Parallel and Distributed Computing, 129, 36-49.
18. Talluri, L. S. R. K., Thirumalaisamy, R., Kota, R., Sadi, R. P. R., KC, U., Naha, R. K., & Mahanti, A. (2021). Providing consistent state to distributed storage system. Computers, 10(2), 23.
19. Aikins, M. V. (2023). Distributed storage systems and how they handle data consistency and reliability. Faculty of Natural and Applied Sciences Journal of Scientific Innovations, 5(1), 83-89.
20. Nguyen, H. H. C., Nguyen, T. T., & Trinh, T. H. (2019). Consistency Maintenance in Distributed Cloud Storage Systems. Azerbaijan Journal of High Performance Computing, 2(2), 158-169.
21. Mansouri, Y., Toosi, A. N., & Buyya, R. (2017). Data storage management in cloud environments: Taxonomy, survey, and future directions. ACM Computing Surveys (CSUR), 50(6), 1-51.
22. Kim, B. H., & Yoon, Y. (2021). Cloud Storage Service Architecture Providing the Eventually Consistent Totally Ordered Commit History of Distributed Key-Value Stores for Data Consistency Verification. Electronics, 10(21), 2702.
23. Xu, Z., Gao, Y., & Davidson, A. (2023). Keep Your Distributed Data Warehouse Consistent at a Minimal Cost. Proceedings of the ACM on Management of Data, 1(2), 1-25.
24. Rashmi, K. V., Shah, N. B., & Ramchandran, K. (2017). A piggybacking design framework for read-and download-efficient distributed storage codes. IEEE Transactions on Information Theory, 63(9), 5802-5820.
25. Luo, T., Aggarwal, V., & Peleato, B. (2019). Coded caching with distributed storage. IEEE Transactions on Information Theory, 65(12), 7742-7755.
26. Sanaz Taheri Boshrooyeh,Alptekin Küpçü & Öznur Özkasap. (2025). Integrita: A BFT distributed storage system. Future Generation C[]omputer Systems107629-107629.

27. Qiu Ningjia,Hu Xiaojuan,Wang Peng & Yang Huamin. (2016). Research on Optimization Strategy to Data Clustered Storage of Consistent Hashing Algorithm. TELKOMNIKA (Telecommunication Computing Electronics and Control)(3),824-824.
28. Junfeng Tian & Yanan Pang. (2020). Adjoin: A causal consistency model based on the adjacency list in a distributed system. Concurrency and Computation Practice and Experience(22).