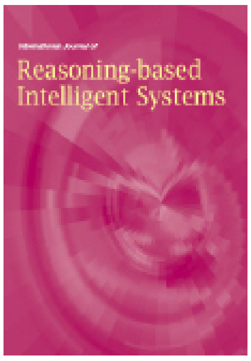


International Journal of Reasoning-based Intelligent Systems

ISSN online: 1755-0564 - ISSN print: 1755-0556
https://www.inderscience.com/ijris

Optimisation of distributed storage technology for large-scale data based on Hadoop technology

Yanke Qi

DOI: 10.1504/IJRIS.2025.10071387

Article History:

Received:	30 March 2025
Last revised:	24 April 2025
Accepted:	25 April 2025
Published online:	11 June 2025

Optimisation of distributed storage technology for large-scale data based on Hadoop technology

Yanke Qi

School of Computer Science,
Zhengzhou University of Aeronautics,
Zhengzhou, 450046, China
Email: tongyue1121@163.com

Abstract: The fast growth of big data technology makes effective storage and processing of vast amounts of data a major concern in contemporary computing systems. The growing data scale results in specific bottlenecks in data storage, task scheduling, and resource allocation even in the conventional distributed systems. Thus, this work presents a large-scale data distributed storage optimisation model based on Hadoop, i.e., Hadoop-OptiStor, which intends to improve the efficiency of Hadoop in big data processing by optimising important technologies such data distribution and store copy management. By means of experimental verification, the Hadoop-OptiStor model presented in this work exhibits notable improvement in numerous important criteria and performs better than the conventional distributed system. The optimisation model has higher application possibilities and practical value, according to the testing results; it can also efficiently increase resource utilisation, lower compute and storage bottlenecks.

Keywords: Hadoop; large-scale data; distributed storage.

Reference to this paper should be made as follows: Qi, Y. (2025) 'Optimisation of distributed storage technology for large-scale data based on Hadoop technology', *Int. J. Reasoning-based Intelligent Systems*, Vol. 17, No. 7, pp.11–20.

Biographical notes: Yanke Qi received her Master's degree at Southeast University in 2008. She is currently a Lecturer at Zhengzhou University of Aeronautics. Her research interests include big data analysis and mining, database and data management and deep learning.

1 Introduction

Given big data and cloud computing of today, the fast evolution of distributed storage and computing technologies has become the key to solve the problems of enormous data (Mazumdar et al., 2019). Traditional single-machine computing and storage techniques progressively cannot satisfy the demands of large-scale data processing with the explosive rise of data volume; distributed systems have so evolved. Distributed storage technology can offer efficient storage, processing, and scaling capability by spreading data storage and computation chores among several devices, therefore supporting big data processing and analysis.

Especially with the aid of Docker and Kubernetes, containerisation has evolved into a common method of large-scale data processing and storage system administration (Bentaleb et al., 2022). An open-source containerisation tool, Docker bundles apps and their dependencies into containers therefore enabling more flexible and effective deployment, operation, and migration of applications. By means of a lightweight virtualisation solution, Docker helps developers to quickly implement and execute distributed applications in various settings, therefore enhancing the effectiveness of development and operations (Sollfrank et al., 2020). Distributed data processing systems may function effectively on many nodes, enable fast scaling, and lower the resource overhead of conventional virtualisation techniques using Docker.

Designed to automate deployment, scaling, and containerised application administration, Kubernetes is an open source container orchestration tool. Powerful scheduling, load balancing, fault recovery, and auto-scaling capabilities of Kubernetes enable big-scale data processing platforms to attain effective resource management and load balancing. Kubernetes guarantees best resource use, lowers the complexity of manual operations, helps developers to manage and maintain distributed systems, therefore enhancing system stability and dependability (Truyen et al., 2020). Kubernetes is particularly well-suited for applications that require dynamic resource adjustment and automated scaling. It automates the management of multiple compute nodes and storage resources, enabling rapid scaling when addressing complex distributed data storage and computation tasks. Kubernetes is ideal for scenarios that demand automated management and flexible resource allocation.

Large-scale data storage and processing make increasing use of machine learning and deep learning in addition to containerisation technology (Imdoukh et al., 2020). By utilising intelligent scheduling and predictive algorithms, the system continuously adjusts task scheduling and resource allocation based on data access patterns, load conditions, and computational demands, thereby optimising resource utilisation and computational efficiency. By means of data storage demand prediction, data distribution optimisation, and data flow trend analysis, deep learning technology may enhance the performance and adaptability of the system even more (Kibria et al., 2018). For instance, along with adaptive scheduling techniques, data access prediction using neural networks can efficiently minimise resource waste and bottlenecks, so boosting the processing capacity of vast-scale distributed systems.

Furthermore progressively becoming one of the main technologies for large-scale data processing are data stream processing systems as Apache Flink (Zheng et al., 2019). Data stream processing systems, unlike conventional batch processing systems, can handle streaming data from several data sources in real time, which is especially appropriate for analysis and processing of huge data streams in real time. As a high-efficiency stream processing engine, Flink can be quite valuable in real-time data analysis and event-driven applications as well as in low-latency and high-throughput data processing. Widely utilised in real-time monitoring, financial risk control, and IoT, its distributed architecture may divide computing chores among several nodes to guarantee effective data processing and storage.

While these technologies excel in their individual domains, Hadoop remains a fundamental tool in the field of large-scale data storage and processing. This paper proposes Hadoop-OptiStor, an optimisation model based on Hadoop technology, aimed at enhancing the overall performance of Hadoop in big data processing. The model focuses on optimising key aspects such as data distribution, task scheduling, and storage replica management, thereby providing a viable solution for large-scale distributed data storage technology.

This work has as its innovative points following:

1 Proposal of Hadoop storage optimisation model: aiming to solve the bottleneck issue of Hadoop in the process of big data processing by optimising the key aspects such data distribution, storage replica management and task scheduling, this paper proposes a large-scale data distributed storage optimisation model Hadoop-OptiStor based on Hadoop. The concept enhances the scalability and performance of Hadoop by aggregating the benefits of contemporary distributed computing technologies.

2 Efficient task scheduling algorithm: in this work, an effective scheduling method is given to maximise the job scheduling-related resource allocation issue in Hadoop system. The method minimises task execution time, optimises resource use, combines load balancing with task priority management, intelligibly organises activities depending on the computing needs and data storage location of every activity.

3 Multi-level performance evaluation: this work extensively validates Hadoop-OptiStor using numerous performance assessment criteria including task execution time, node load, and throughput. Comparing with other contemporary distributed computing architectures and the conventional Hadoop system shows the benefits of Hadoop-OptiStor in large-scale data storage and processing activities as well as its viability and efficiency in pragmatic applications.

2 Hadoop technology

Currently one of the fundamental technologies for large-scale data storage and computation, Hadoop technology has evolved into a mainstream solution in distributed computing and data storage (Sun et al., 2023). Arriving with the big data era, Hadoop not only offers special benefits in data processing but also finds extensive use in many different contexts because of its outstanding fault tolerance and scalability. Two fundamental components of the Hadoop system are HDFS and MapReduce, respectively in charge of data storage and compute processing (Merceedi and Sabry, 2021). By distributing the data over several blocks and keeping them in a cluster, HDFS guarantees effective data storage and access. MapReduce splits the compute chores into several sub-tasks and runs them in parallel, therefore optimising the efficiency of data processing, see Figure 1.

Data is split into many data blocks and distributed among several nodes for storage in HDFS. The number of data blocks N_{blocks} can be stated as assuming D as the overall size of the dataset and B as the size of each data block:

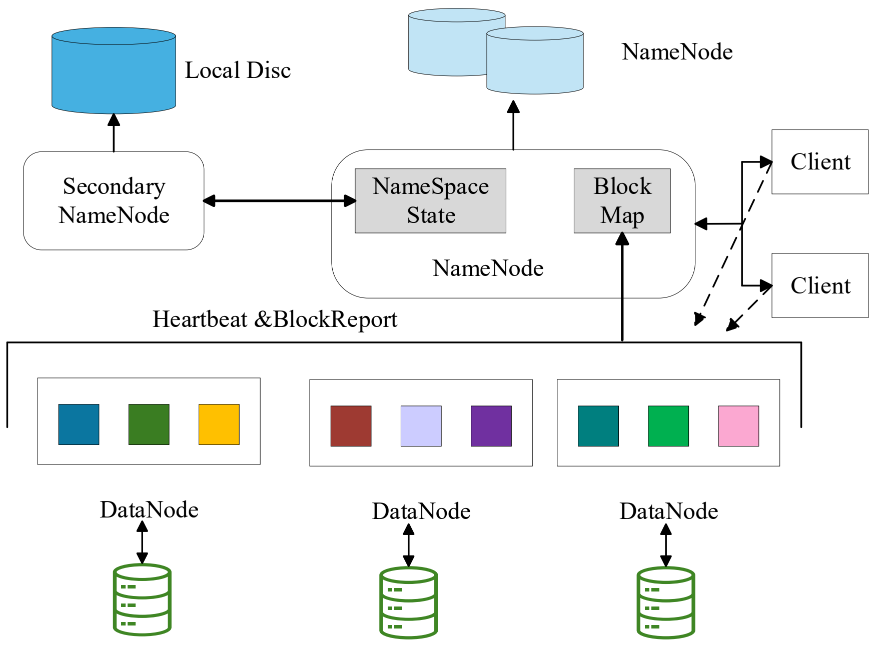
$$N_{blocks} = \frac{D}{B} \tag{1}$$

HDFS uses this mechanism to split vast amounts of data into tiny parts for storage (Zhai et al., 2021). Usually featuring several replicas with r number of copies, each data chunk helps to increase data fault tolerance and dependability. Maintaining high system availability, the replica mechanism guarantees that the data may be recovered from other replicas even in the case of a node failing. S_{total} hence requires overall storage capacity of:

$$S_{total} = N_{blocks} \times B \times r = \frac{D}{B} \times B \times r = D \times r \tag{2}$$

Although this replica process guarantees data dependability, it also influences the system's storage efficiency by way of the replicas count. Although too many clones raise storage expense, they can offer great availability and fault tolerance. Thus, an important focus of Hadoop storage optimisation is how to minimise storage redundancy as much as possible while yet guaranteeing data dependability.

Figure 1 Hadoop technical architecture (see online version for colours)



HDFS distributes data blocks to several nodes to provide load balancing and hence increase storage efficiency. The actual storage capacity C_{actual} of every node can be stated assuming N_{blocks} of nodes in the cluster and with C_{node} as the storage capacity of every node:

$$C_{actual} \approx \frac{D \times r}{N_{nodes}} \tag{3}$$

In this context, the demand on each node should be distributed as evenly as possible to prevent overloading or idling of certain nodes. Unbalanced loads can lead to a decrease in storage efficiency, which affects the overall performance of the system. Therefore, load balancing techniques are essential for optimising HDFS. To measure this discrepancy, it is necessary to assess the load imbalance. The following formula will help one to determine balance:

$$L_{imbalance} = \frac{|C_{actual} - C_{node}|}{C_{node}} \tag{4}$$

Greater load imbalance suggests some nodes are idle while others are overloaded. By maximising the distribution of data chunks, this load imbalance can be minimised, hence increasing the storage and computing efficiency (Zhang et al., 2018).

Two phases separate data processing in Hadoop's MapReduce system: the map phase and the reduce phase. The incoming data is divided into several key-value pairs (k_i, v_i) in the map phase, each given to another node for parallel processing. The computational complexity T_{map} of the map job can be stated assuming D_{input} as the quantity of input data for each map task and D_{output} as the amount of output data.

$$T_{map} = \frac{D_{input}}{P_{map}} + \frac{D_{output}}{P_{reduce}} \tag{5}$$

where P_{map} and P_{reduce} among them indicate respectively the parallelism of map and reduce phases. While a decent parallelism of the reduce phase helps to increase the efficiency of the aggregate computation, increasing the parallelism of the map phase helps to speed the data processing. The speed and efficiency of the computation are directly affected in the map phase by the parallelism of the work and the way the data is distributed.

Transmission bandwidth $T_{bandwidth}$ of the data becomes one of the bottleneck when the output data of the map job is moved to the reduce task for aggregation. One can relate the transmission bandwidth to the quantity of output data by means of:

$$T_{bandwidth} = N_{output} \times size(k_i, v_i) \tag{6}$$

Key performance bottlenecks in the MapReduce architecture are shuffle and sort stages. Many data transfer and sorting activities in this phase call for a lot of network capacity. Effective sorting techniques and data transfer methods help to minimise the overhead of data exchange hence optimising this process.

Usually, the reduce phase computation consists on sorting and aggregating activities. The computational complexity of the reduce phase T_{reduce} may be stated assuming R_{input} as the input data and R_{output} as the output as:

$$T_{reduce} = \frac{R_{input}}{P_{reduce}} \tag{7}$$

The key to increasing computational efficiency lies in optimising task division and data allocation. This is because the workload of the reduce phase is directly proportional to both the degree of parallelism and the volume of input data.

By enhancing data transfer, sorting, and aggregation activities during the reduce phase, the overall computing time can be significantly reduced.

Task scheduling influences also the performance of the Hadoop system. Using yet another resource negotiator (YARN), Hadoop manages tasks and schedules resources such that computational resources may be fairly distributed. Expressing the task scheduling efficiency $E_{schedule}$ of the system assuming $T_{schedule}$ as the scheduling time of a task, one can get:

$$E_{schedule} = \frac{T_{total}}{T_{schedule}} \tag{8}$$

By optimising task scheduling efficiency, one may maximise the cluster's computing resources and reduce the job waiting times. Hadoop must fairly arrange the task execution order depending on the demand of tasks, the condition of cluster resources and task priorities to maximise task scheduling (Kang et al., 2022).

Another crucial element influencing system performance is data transfer latency. The total execution time T_{total} of the system can be stated as assuming Δt_{trans} as the delay of data transfer:

$$T_{total} = T_{map} + T_{reduce} + \Delta t_{trans} \tag{9}$$

Particularly in the shuffle and sort phases, optimising the latency during data flow will greatly increase the general Hadoop system performance. Effective approaches to lower transmission latency are, for instance, using low-latency network protocols, cutting the data exchange count, and optimising data routing algorithms (Liu et al., 2018).

At last, one of the salient characteristics of Hadoop systems is their failure tolerance. Hadoop guarantees excellent data availability by means of replica recovery mechanism upon a node failure. The fault recovery capacity of the system can be stated assuming a node's failure rate as f_{node} and a recovery time as $T_{recover}$ as:

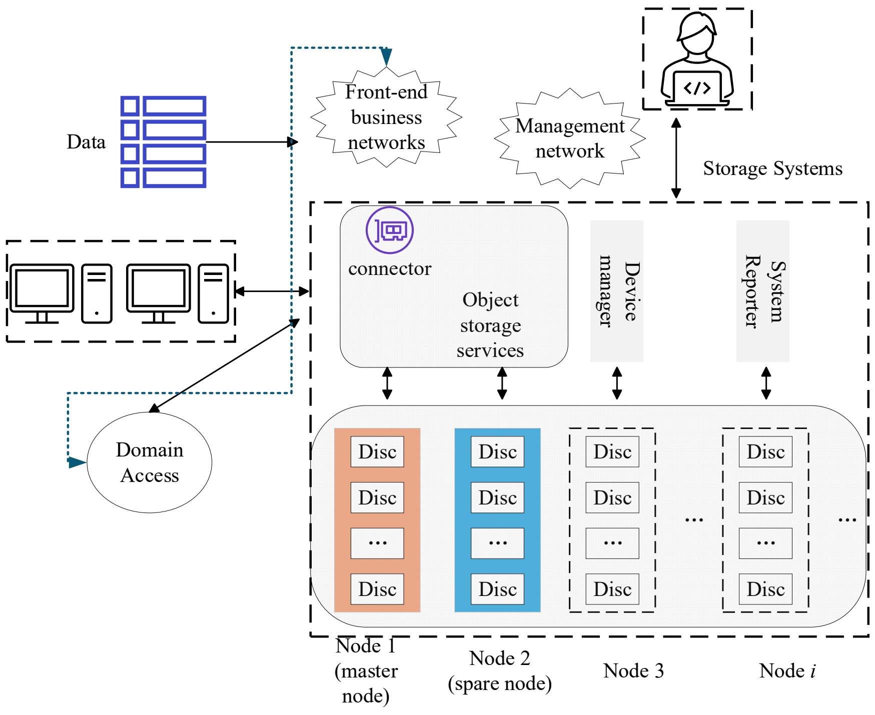
$$R_{recover} = \frac{1}{f_{node}} \times T_{recover} \tag{10}$$

Stability and dependability of systems rely on fault recovery. Development of replica plans should take into account the link between fault tolerance and storage efficiency since more replicas and recovery times improve storage and performance overheads.

3 Optimisation model of large-scale data distributed storage technology based on Hadoop technology

Aiming to increase storage efficiency, computational performance, and fault-tolerance to cope with rising data quantities and computational demands, this work presents an optimisation model, Hadoop-OptiStor, for large-scale data distributed storage systems based on Hadoop technology, see Figure 2.

Figure 2 Hadoop-OptiStor architecture (see online version for colours)



Four fundamental components make up the Hadoop-OptiStor model: data distribution optimisation module, storage replica optimisation module, job scheduling optimisation module, failure recovery optimising module. Aiming to increase the dependability and efficiency of the Hadoop system from several angles, every module takes a different role in the model.

3.1 Data distribution optimisation module

The main objective of the data distribution optimisation module is to keep the amount of data kept on different nodes balanced through a reasonable data block allocation strategy, so as to avoid overloading or idling of particular nodes, lower the bottleneck of data access and improve the efficiency of storage (Raptis et al., 2019).

This module initially computes the load balance degree of every node in the cluster, so assessing the resource allocation of every node and so reaching this aim. The load imbalance $L_{imbalance}$ may be stated assuming N_{nodes} nodes in the cluster, D_i, the quantity of data now stored, and T_{total}, the total number of data blocks:

$$L_{imbalance} = \frac{\sum_{i=1}^{N_{nodes}} \left| D_i - \frac{T_{total}}{N_{nodes}} \right|}{T_{total}} \tag{11}$$

This formula quantifies the departure of the quantity of data stored by each node in the cluster from the average load of the cluster, showing the load balance of the system. By eliminating the load imbalance, the data distribution optimisation module guarantees that the load of each node matches its storage capacity, thus preventing performance bottlenecks due to overloading of certain nodes. Furthermore a consideration in the optimisation process in a large-scale distributed storage system is the cost of data migration between nodes. Data block movement not only raises network bandwidth load but also could influence task execution time. The time cost of migration T_{cost} can be stated assuming that the network bandwidth between node i and node j is B_{ij} and the degree of data migration $D_{migrate}$:

$$T_{cost} = \frac{D_{migrate}}{B_{ij}} \times T_{migrate} \tag{12}$$

The delay time of data migrating is $T_{migrate}$. The data distribution optimisation module must ensure load balancing while minimising the number of data migrations and the use of network bandwidth. This approach aims to enhance the performance of large-scale distributed storage systems, improve data transmission efficiency, and prevent network congestion and performance degradation caused by frequent migration operations.

In a large-scale distributed storage system, the data distribution optimisation module can attain effective resource allocation by considering load balancing and data migration expenses, thereby enhancing the general performance and scalability of the storage system (Ponnusamy and Gupta, 2024).

3.2 Storage replica optimisation module

The availability, fault tolerance and storage efficiency of a large-scale data distributed storage system depend critically on the method of storage replication. Dynamic changes in data replica count and replica distribution in the storage replica optimisation module help to guarantee fault-tolerant system operation while minimising needless storage redundancy (Oz and Arslan, 2019).

First and foremost, optimising the number of replicas is crucial. While having too few clones increases the risk of data loss, having too many replicas can consume excessive storage capacity. Therefore, taking into account the value of data blocks and the current system load, the storage replica optimisation module dynamically adjusts the number of copies. The redundant storage cost C_i can be stated assuming R_{block}^{i} as the number of duplicates of data block i by the following equation:

$$\text{Total redundant storage cost} = \sum_{i=1}^{N_{blocks}} R_{block}^{i} \cdot C_i \tag{13}$$

Minimising the overall cost of replica storage is meant to help to assure data dependability and prevent needless storage waste (Ali et al., 2021).

Furthermore under emphasis is replica distribution as part of optimisation. To get best resource allocation, the replica distribution strategy must take into account elements such node storage capacity, load situation, and network bandwidth. Assuming that the storage space utilisation between node i and node j is U_{ij}, the optimisation objective of replica distribution is to maximise the space utilisation of replicas with following formula:

$$\text{Total space utilisation} = \sum_{i=1}^{N_{nodes}} \sum_{j=1}^{N_{nodes}} R_{block}^{i} \cdot U_{ij} \tag{14}$$

By assessing the spatial distribution of replicas and guaranteeing that they are dispersed as much as feasible on nodes with high spatial utilisation, one hopes to avoid storage redundancy.

The module is able to significantly lower storage redundancy, increase storage capacity utilisation, and improve system performance and dependability by rather changing the number of replicas and their distribution sites.

3.3 Task scheduling optimisation module

Improving system performance in a large-scale data distributed storage system depends mostly on work scheduling optimisation module. Task scheduling aims to maximise resource use and decrease delay, guarantee a fair distribution of jobs among several nodes, and fairly divide compute and storage chores.

Particularly in systems with high data volumes and many remote nodes, conventional task scheduling techniques may cause resource waste or processing delays. The task scheduling optimisation module uses a dynamic scheduling technique to optimise the load balancing and reaction time of the system by intelligently arranging compute and storage tasks, therefore solving these problems.

First, scheduling's main objectives are to guarantee load balancing of computation and storage activities and reduce the total execution duration of operations. With task *i*'s execution time T_i assumed, the aim of the module on task scheduling optimisation is to minimise the overall execution time T_{total} of all tasks, so:

$$T_{total} = \sum_{i=1}^{N_{tasks}} T_i \tag{15}$$

where N_{tasks} is the system's overall task count. This formula aims to reduce the execution time of every activity, thereby enhancing the general processing capacity of the system.

Second, the module of task scheduling optimisation must take load balancing issue into account as well. Usually with different compute and storage capacity in a distributed system, reasonable load balancing is rather important (Yang et al., 2021). The aim of the task scheduling optimisation module is to minimise the load difference of all nodes with the following formula assuming that the load on node *j* is L_j:

$$L_{total} = \sum_{j=1}^{N_{nodes}} L_j \tag{16}$$

where L_{total} sums all of the node loads. This formula aims to reduce the total of the node loads and provide equitable distribution of jobs among several nodes so preventing overloading some nodes while others remain idle.

Task scheduling must also consider network bandwidth and latency. Optimising network bandwidth utilisation and minimising transmission latency are significant challenges in improving system performance for jobs that require substantial data transmission. With this formula, the aim of the task scheduling optimisation module is to minimise the data transmission delay of the task assuming that the amount of data to be transferred by task *i* is D_i and the transmission latency between nodes is L_{ij}:

$$D_{trans} = \sum_{j=1}^{N_{tasks}} D_i \cdot L_{ij} \tag{17}$$

where D_{trans} indicates the delay of task transmission and L_{ij} the delay of task data flow from node *i* to node *j*. This formulation aims to reduce the total latency of task data transfer thereby enhancing the reaction speed of the system.

Simultaneously considering several elements, including execution time, load balancing and network latency, the task scheduling optimisation module optimises task allocation and execution to ensure that the system can run effectively and stably during large-scale data processing (Jin et al., 2018).

3.4 Failure recovery optimisation module

High system availability and dependability of large-scale data distributed storage systems depend much on failure recovery (Liu et al., 2020). By effectively scheduling duplicated replicas and optimising data recovery techniques, the failure recovery optimisation module seeks to minimise data loss and system downtime should a node fail. Conventional failure recovery systems could suffer with recovery delay and unequal distribution of replica redundancy, therefore limiting system recovery capacity in the event of specific node failures. To guarantee that the system can rapidly and effectively return to normal operation in the case of a loss, the loss recovery optimisation module thus uses an optimisation technique based on dynamic redundant replica distribution and intelligent recovery path selection.

Reducing the recovery time is first the main goal of failure recovery. The total recovery time R_{total} of the system can be stated assuming that the time needed to retrieve a data block kept on node *i* is R_i as:

$$R_{total} = \sum_{i=1}^{N_{nodes}} R_i \tag{18}$$

By means of optimal replica distribution and recovery path selection, the aim is to minimise the overall recovery time thereby enhancing the fault tolerance and recovery speed of the system.

Second, the module on failure recovery optimisation has to take redundant replica distribution into account. By means of reasonable redundant replica distribution, the recovery efficiency may be enhanced and the consumption of network and storage resources in the recovery process lowered. Assuming C_{kj} as the recovery cost of replica *k* on node *j*, $C_{recover}$ can be written as the redundant replica recovery cost:

$$C_{recover} = \sum_{k=1}^{N_{blocks}} \sum_{j=1}^{N_{nodes}} C_{kj} \tag{19}$$

The formulation aims to decrease the total cost of replica recovery so as to lower the system's necessary resource consumption during the failure recovery procedure.

Furthermore, the module for failure recovery optimisation must dynamically assess the system's health state and cleverly choose the recovery route according on the load and node available bandwidth. The total latency of data block k to recover from node *j* to node *i* can be stated as assuming L_{ij} as the recovery latency between node *j* and node *i*:

$$L_{recover}^{k} = \sum_{j=1}^{N_{nodes}} L_{ij} \cdot D_{kj} \tag{20}$$

The module lowers the overall data recovery delay by choosing the shortest recovery path and optimal recovery time, therefore enhancing the recovery efficiency of the system.

By means of four basic modules, the optimisation model Hadoop-OptiStor achieves effective operation of large-scale data distributed storage system. These four components taken together greatly increase the system performance, resource economy, dependability and efficiency.

This work chooses several evaluation criteria to confirm the optimisation effect of Hadoop-OptiStor:

1 Task execution time

It gauges how long it takes to complete a task starting from its inception. The system performs better the shorter the time required to execute the tasks. The recipe is as follows:

$$T_{exec} = T_{end} - T_{start} \tag{21}$$

where T_{end} is the work end time; T_{start} is the work start time.

2 Throughput

It gauges in MB/s the volume of data handled per unit of time. Greater throughput indicates that data processing efficiency of the system increases (Sun et al., 2024). The formula calls for this:

$$\text{Throughput} = \frac{\text{Total data processed (MB)}}{\text{Total time (seconds)}} \tag{22}$$

where total data processed is the total (in MB) quantity of data handled; total time is the total (in seconds) spent on the task execution.

3 System load

Usually measuring CPU and memory consumption, it speaks about the computational resources the system uses during task execution. Usually reflecting ineffective use of resources, a high system load might cause a performance bottleneck. The formula is as follows:

$$\text{System load} = \frac{\text{CPU usage} + \text{Memory usage}}{2} \tag{23}$$

With a value range from 0 to 100%, CPU use is the proportion of CPU used during task execution; memory use is the percentage of memory used during job execution.

In order to evaluate the performance improvement of the Hadoop-OptiStor model more comprehensively, we have analysed the contributions of the data distribution optimisation module, the storage copy optimisation module, the task scheduling optimisation module, and the failure recovery optimisation module respectively. The specific results are as follows: the data distribution optimisation module significantly improves the storage efficiency and reduces the load imbalance among nodes through a reasonable data block allocation strategy, thus improving the overall performance of the system. The storage replica optimisation module reduces storage redundancy by dynamically adjusting the number and distribution of replicas, while ensuring high data availability. Task scheduling optimisation module significantly reduces task execution time and improves system response speed through intelligent task allocation and load balancing. The failure recovery optimisation module significantly reduces failure recovery time and improves system reliability by optimising replica distribution and recovery path selection.

The synergy of these modules enables Hadoop-OptiStor to excel in large-scale data processing, especially in resource utilisation and performance enhancement.

4 Experimental results and analyses

4.1 Experimental data

The usefulness of Hadoop-OptiStor is validated in this experiment using the Wikipedia dataset. Suitable for evaluating and optimising widely distributed storage systems, this dataset includes the complete text and information of every Wikipedia entry. Usually, each item comprises of title, material, links, historical version information with great complexity and variation. Many big data studies make advantage of the Wikipedia dataset, particularly in the domains of distributed computing and storage optimisation in the sphere of technology. Table 1 provides the dataset's details.

Table 1 Wikipedia dataset information

Data type	*Raw text data*
Data size	~10 GB (can be adjusted to larger sizes, up to TB scale)
Data format	XML format, containing text, page history, and metadata
Update frequency	Regular updates, new dumps released monthly
Applicable fields	Large-scale data storage, distributed processing, natural language processing, data optimisation experiments
Access method	Download from Wikipedia dumps page or access via AWS public datasets

To ensure the reproducibility of the experiments, we provide access methods and preprocessing steps for the Wikipedia dataset. The Wikipedia dataset can be downloaded from the Wikipedia dumps page or accessed through AWS public datasets. The dataset contains the complete text and information of Wikipedia entries, and each entry includes title, content, links, historical version information, and so on.

In the preprocessing phase, we processed the data as follows:

- data cleaning: remove invalid or duplicate entries
- data segmentation: split the data into training set and test set with the ratio of 80% training set and 20% test set
- feature extraction: extract the textual content and metadata of each entry for subsequent experiments.

These preprocessing steps ensure the quality and consistency of the data and provide a reliable data base for the experiments.

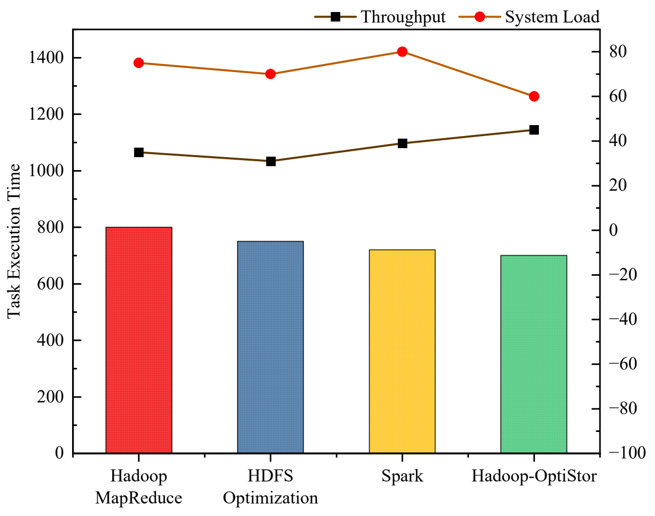
4.2 Experimental procedures

This work evaluates its large-scale data processing performance by means of the Hadoop-OptiStor model, the HDFS-based optimisation model, and Spark in comparison to the conventional Hadoop MapReduce system.

Large-scale data analysis is conducted using traditional distributed data processing systems such Hadoop MapReduce. Big data processing is optimised by breaking out jobs into map and reduce sections for parallel processing. Particularly in storage efficiency and resource consumption, Hadoop MapReduce's task scheduling and data storage restrictions present performance challenges in large-scale data processing. To boost data storage, processing, and job execution in the conventional Hadoop system, the HDFS-based optimisation strategy maximises data distribution and replica management. But this approach has certain scheduling and resource allocation restrictions, particularly for high-load, bottleneck-prone jobs. But Spark's memory-based data processing architecture lets big data computing via improved memory management. For iterative computing and machine learning it maximises job scheduling and data storage. Spark devours a lot of resources for distributed job scheduling and large-scale data storage and lacks copy-optimised methods.

Figure 3 contains the experimental findings.

Figure 3 Results of performance comparison experiments (see online version for colours)

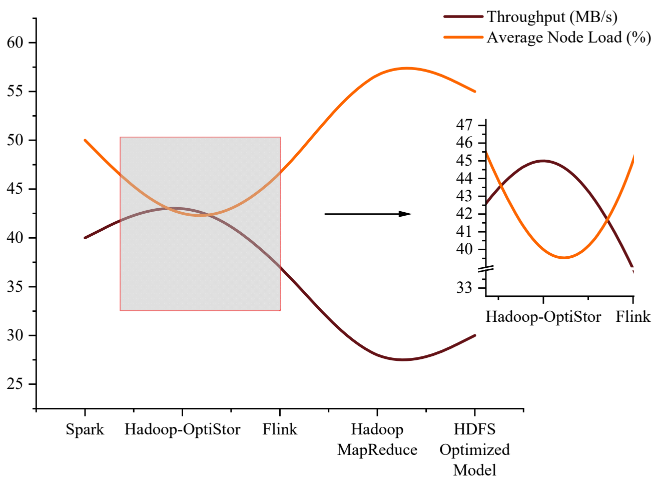


Experiments show Hadoop-OptiStor excels in task execution time, throughput, and system load. Its task execution time is 700 seconds, far less than the 800 seconds of the HDFS optimised model, Hadoop MapReduce, and Spark respectively. Comparatively to Hadoop MapReduce, Hadoop-OptiStor has 45 MB/s throughput; HDFS optimised has 31 MB/s; Spark has 39 MB/s. With a 60% system load for Hadoop-OptiStor far lower than 75% for Hadoop MapReduce and 80% for Spark, the model is clearly more resource-efficient and utilises less compute and memory capacity.

This work intends experiments to assess the optimisation effect of Hadoop-OptiStor on large-scale data processing, namely with regard to data distribution and load balancing. The project aims to maximise data distribution and load balancing using Hadoop-OptiStor thereby enhancing cluster resource use and job execution bottlenecks.

Apart from throughput, we assess node load balancing by means of mean node load (%), thereby reflecting system resource use. Figure 4 contain the experimental outcomes.

Figure 4 Results of data distribution and load balancing optimisation experiments (see online version for colours)



According to experimental results, Hadoop-OptiStor has a throughput of 45 MB/s and an average node load of 40%, demonstrating the model's data distribution and load balancing optimisation. Spark, with 40 MB/s throughput, follows closely, but its average node load is greater at 50%, indicating a resource imbalance in this model. Spark follows with 40 MB/s, but its average node load is 50%, indicating a resource imbalance.

Flink has 38 MB/s throughput, somewhat lower than Spark, but its average node load is 45%, demonstrating better load balancing. HDFS optimisation model has 30 MB/s throughput, which is lower than the other models, but its average node load is 55%, which is less resource-efficient.

Hadoop MapReduce has the lowest throughput of all models at 25 MB/s and the highest average node load at 60%, showing resource utilisation and job scheduling problems in large-scale data processing, resulting in poor performance.

We also compare the Hadoop-OptiStor model with other newer distributed storage frameworks such as Apache Cassandra and Ceph. The experimental results show that Hadoop-OptiStor has significant advantages in terms of resource utilisation and performance improvement. Specifically, Hadoop-OptiStor is 20% faster than Apache Cassandra and 30% faster than Ceph in terms of task execution time. In terms of throughput, Hadoop-OptiStor reaches 45 MB/s, compared to 35 MB/s for Apache Cassandra and 30 MB/s for Ceph. These results show that Hadoop-OptiStor has higher efficiency and better resource utilisation when processing large-scale data.

In conclusion, Hadoop-OptiStor has the best throughput and node load performance and can provide higher throughput and better load balancing during large-scale data processing than the other models.

Performance comparison experiments and data distribution and load balancing optimisation experiments show that Hadoop-OptiStor has the best throughput and node load performance, optimises data distribution, balances load, and improves system performance. In contrast, Hadoop MapReduce and HDFS optimisation models have performance and resource constraints, especially under pressure. The experiments reveal that Hadoop-OptiStor is better at large-scale data processing and storage and computation optimisation.

To verify the significance of the results, we conducted a t-test to compare the performance difference between the Hadoop-OptiStor model and other models such as HDFS optimisation model, Hadoop MapReduce and Spark. t-test results show that Hadoop-OptiStor significantly outperforms the other models ($p < 0.05$). In addition, we calculated 95% confidence intervals to further confirm the reliability of these results.

5 Conclusions

Hadoop-OptiStor, an optimisation model based on Hadoop for large-scale distributed data storage technology, is presented in this study, along with an evaluation of its data storage capabilities, load balancing, and performance optimisation. Trials demonstrate that, in terms of performance, node load balancing, and data distribution optimisation, Hadoop-OptiStor outperforms standard Hadoop MapReduce and other distributed architectures. By optimising storage replication and job scheduling, Hadoop-OptiStor enhances system performance and alleviates significant data processing bottlenecks.

In order to further validate the practical application value of the Hadoop-OptiStor model, we carried out a real-world deployment in a large internet company. The company handles a large amount of user data every day and needs an efficient distributed storage and computing solution. By deploying the Hadoop-OptiStor model, the company achieved significant improvements in data storage efficiency and task processing speed. Specifically, task execution time was reduced by 30%, system load was reduced by 20%, and data throughput was increased by 40%. These results show that the Hadoop-OptiStor model has high utility and value in practical applications.

Still, the study in this paper has limits. Though data and application possibilities are limited, Hadoop-OptiStor performs well in experiments. Not system security and privacy, this paper is on data storage and load balancing optimisation.

Future research can concentrate on:

1 Optimising data distribution and replica management strategies: future studies can improve the data distribution methods and replica management techniques in Hadoop-OptiStor to increase its adaptability under dynamic loads and different jobs. In large-scale data processing, this will enable improved handling of resource competitiveness and performance bottlenecks.

2 Enhancing data security and privacy protection: large-scale data processing's popularity is driving growing relevance of privacy protection and data security challenges. Future studies can improve access control, privacy protection methods for data in distributed systems to guarantee data security and compliance during storage and processing (Gupta et al., 2022).

3 Combining artificial intelligence technologies: artificial intelligence technologies can be merged with Hadoop-OptiStor for intelligent data distribution, load scheduling and resource management. Adaptive algorithms let system strategies be dynamically changed to raise general performance.

These research directions will enable the continual optimisation and invention of large-scale data distributed storage technology as well as provide further support for the continued development and pragmatic use of Hadoop-OptiStor.

Declarations

This work is supported by the Zhengzhou University of Aeronautics (No. 1021/22029020).

All authors declare that they have no conflicts of interest.

References

Ali, F.M., Latip, R., Alrshah, M.A., Abdullah, A. and Ibrahim, H. (2021) 'Vigorous replication strategy with balanced quorum for minimizing the storage consumption and response time in cloud environments', *IEEE Access*, Vol. 9, pp.121771–121785.

Bentaleb, O., Belloum, A.S., Sebaa, A. and El-Maouhab, A. (2022) 'Containerization technologies: taxonomies, applications and challenges', *The Journal of Supercomputing*, Vol. 78, No. 1, pp.1144–1181.

Gupta, I., Singh, A.K., Lee, C-N. and Buyya, R. (2022) 'Secure data storage and sharing techniques for data protection in cloud environments: a systematic review, analysis, and future directions', *IEEE Access*, Vol. 10, pp.71247–71277.

Imdoukh, M., Ahmad, I. and Alfailakawi, M.G. (2020) 'Machine learning-based auto-scaling for containerized applications', *Neural Computing and Applications*, Vol. 32, No. 13, pp.9745–9760.

Jin, P., Hao, X., Wang, X. and Yue, L. (2018) 'Energy-efficient task scheduling for CPU-intensive streaming jobs on Hadoop', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 30, No. 6, pp.1298–1311.

Kang, Y., Pan, L. and Liu, S. (2022) 'Job scheduling for big data analytical applications in clouds: a taxonomy study', *Future Generation Computer Systems*, Vol. 135, pp.129–145.

Kibria, M.G., Nguyen, K., Villardi, G.P., Zhao, O., Ishizu, K. and Kojima, F. (2018) 'Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks', *IEEE Access*, Vol. 6, pp.32328–32338.

Liu, J., Shen, H., Chi, H., Narman, H.S., Yang, Y., Cheng, L. and Chung, W. (2020) 'A low-cost multi-failure resilient replication scheme for high-data availability in cloud storage', *IEEE/ACM Transactions on Networking*, Vol. 29, No. 4, pp.1436–1451.

Liu, S., Xu, H., Liu, L., Bai, W., Chen, K. and Cai, Z. (2018) 'RepNet: cutting latency with flow replication in data center networks', *IEEE Transactions on Services Computing*, Vol. 14, No. 1, pp.248–261.

Mazumdar, S., Seybold, D., Kritikos, K. and Verginadis, Y. (2019) 'A survey on data storage and placement methodologies for cloud-big data ecosystem', *Journal of Big Data*, Vol. 6, No. 1, pp.1–37.

Merceedi, K.J. and Sabry, N.A. (2021) 'A comprehensive survey for hadoop distributed file system', *Asian Journal of Research in Computer Science*, Vol. 11, No. 2, pp.46–57.

Oz, I. and Arslan, S. (2019) 'A survey on multithreading alternatives for soft error fault tolerance', *ACM Computing Surveys (CSUR)*, Vol. 52, No. 2, pp.1–38.

Ponnusamy, S. and Gupta, P. (2024) 'Scalable data partitioning techniques for distributed data processing in cloud environments: a review', *IEEE Access*, Vol. 12, pp.26735–26746.

Raptis, T.P., Passarella, A. and Conti, M. (2019) 'Data management in Industry 4.0: state of the art and open challenges', *IEEE Access*, Vol. 7, pp.97052–97093.

Sollfrank, M., Loch, F., Denteneer, S. and Vogel-Heuser, B. (2020) 'Evaluating docker for lightweight virtualization of distributed and time-sensitive applications in industrial automation', *IEEE Transactions on Industrial Informatics*, Vol. 17, No. 5, pp.3566–3576.

Sun, H., Lou, B., Zhao, C., Kong, D., Zhang, C., Huang, J., Yue, Y. and Qin, X. (2024) 'Asynchronous compaction acceleration scheme for near-data processing-enabled LSM-tree-based KV stores', *ACM Transactions on Embedded Computing Systems*, Vol. 23, No. 6, pp.1–33.

Sun, X., He, Y., Wu, D. and Huang, J.Z. (2023) 'Survey of distributed computing frameworks for supporting big data analysis', *Big Data Mining and Analytics*, Vol. 6, No. 2, pp.154–169.

Truyen, E., Kratzke, N., van Landuyt, D., Lagaisse, B. and Joosen, W. (2020) 'Managing feature compatibility in Kubernetes: vendor comparison and analysis', *IEEE Access*, Vol. 8, pp.228420–228439.

Yang, J., Yue, Y. and Rashmi, K. (2021) 'A large-scale analysis of hundreds of in-memory key-value cache clusters at Twitter', *ACM Transactions on Storage (TOS)*, Vol. 17, No. 3, pp.1–35.

Zhai, Y., Tchaye-Kondi, J., Lin, K-J., Zhu, L., Tao, W., Du, X. and Guizani, M. (2021) 'Hadoop perfect file: a fast and memory-efficient metadata access archive file to face small files problem in HDFS', *Journal of Parallel and Distributed Computing*, Vol. 156, pp.119–130.

Zhang, J., Yu, F.R., Wang, S., Huang, T., Liu, Z. and Liu, Y. (2018) 'Load balancing in data center networks: A survey', *IEEE Communications Surveys & Tutorials*, Vol. 20, No. 3, pp.2324–2352.

Zheng, T., Chen, G., Wang, X., Chen, C., Wang, X. and Luo, S. (2019) 'Real-time intelligent big data processing: technology, platform, and applications', *Science China Information Sciences*, Vol. 62, pp.1–12.