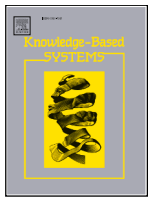
Knowledge-Based Systems 337 (2026) 115399

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Dynamic chunking-driven intelligent transmission mechanism for distributed systems

Enliang Lv [a], Xingwei Wang [a,*], Bo Yi [a,1], Hao Lu [a], Min Huang [b], Yue Kou [a], Keqin Li [c,2]

[a] School of Computer Science and Technology, Northeastern University, Shenyang, 110819, China
[b] School of Information Science and Engineering, Northeastern University, Shenyang, 110819, China
[c] Department of Computer Science, State University of New York, New York, 12561, USA

ARTICLE INFO

Keywords:
Distributed systems
Data transmission optimization
Data chunking
Deep reinforcement learning

ABSTRACT

As network complexity and scale continue to grow, ensuring efficient data transmission among nodes and maintaining system stability have become critical challenges for network enterprises. However, traditional distributed data transmission mechanisms often fail to meet performance requirements under complex and dynamic network conditions, resulting in inefficient data transmission, increased latency, and potential node failures. To address the above issues, we propose an intelligent transmission mechanism driven by dynamic chunking, designed to ensure efficient and stable data transmission in distributed systems. Network conditions are monitored in real time, and the data chunking problem is formulated as a Markov Decision Process (MDP). To derive optimal chunking decisions, a deep reinforcement learning framework is designed to autonomously solve the MDP and adaptively learn chunking policies in response to network performance dynamics. Furthermore, to ensure system stability, we employ a peer-to-peer (P2P) mechanism for node discovery, integrate Distributed Hash Tables for efficient chunk location, and leverage P2P protocols to coordinate the exchange and transmission of data chunks among nodes. Extensive simulation results validate that the proposed mechanism achieves substantial improvements over traditional methods in terms of transmission duration and throughput, while exhibiting robust adaptability to dynamic and complex network conditions.

1. Introduction

Driven by powerful economies of scale and technical innovations like cloud computing, web systems have become increasingly centralized, with more consumers relying on cloud service providers for data storage, sharing, and computing services [1–3]. Consequently, centralized network architectures have emerged as a promising solution to offer greater controllability and higher convenience, such as Amazon S3 [4], Google Drive [5], and Dropbox [6]. However, this centralized architecture introduces several risks, such as single points of failure, data silos, and load imbalance, all of which can significantly degrade system performance and ultimately diminish the user experience. For instance, a report from Amazon's e-commerce platform has revealed that the financial losses caused by service disruptions exceed $66,000 per minute [7,8].

To address these issues, distributed architecture has become a research hotspot, as it distributes control permissions across multiple nodes, thereby reducing reliance on a central server. As a result, it has become the foundational infrastructure for various technological domains, including cloud computing [9], the Internet of Things (IoT) [10], and Blockchain [11]. Although distributed systems have continuously evolved and improved over time, data transmission performance remains a major bottleneck to their future development. This bottleneck not only affects the performance and stability of distributed systems but also hinders their further advancement.

Consequently, to optimize data transmission efficiency, data chunking has been extensively employed in leading distributed transmission systems such as InterPlanetary File System (IPFS) [12], BitTorrent [13], HDFS [14], Ceph [15], and modern cloud storage platforms [16]. Data chunking partitions large objects into smaller chunks distributed across

* Corresponding author.
E-mail addresses: enlianglv@163.com (E. Lv), wangxw@mail.neu.edu.cn (X. Wang).
[1] Member, IEEE
[2] Fellow, IEEE

https://doi.org/10.1016/j.knosys.2026.115399
Received 27 October 2025; Received in revised form 19 January 2026; Accepted 21 January 2026
Available online 2 February 2026

different nodes, enabling independent storage and transmission to improve throughput, reduce latency, and enhance system fault tolerance. However, this gives rise to a new challenge, as most of these systems utilizes a pre-defined and fixed chunk size, limiting their ability to adapt to dynamic network conditions and heterogeneous data characteristics. Moreover, variations in chunk size can significantly impact the transmission efficiency for the same file. Particularly, excessively small chunks may introduce high overhead due to increased metadata and connection management, while overly large chunks may result in poor adaptability to network fluctuations. These issues can severely compromise the stability and efficiency of the system. Although implementing dynamic adjustment strategies offers a potential solution, relying on pre-defined heuristic rules is often insufficient. These rigid methods fail to capture the complex, non-linear correlations between high-dimensional network dynamics and optimal chunking strategies. Furthermore, traditional supervised learning methods are ill-suited for this context. Models trained on static historical datasets are fundamentally incapable of capturing the stochastic volatility and real-time variability inherent in complex distributed networks, leading to strategy mismatches when environmental conditions shift. In contrast, Deep Reinforcement Learning (DRL) [17] overcomes these limitations by formulating chunking as a sequential decision-making process. Instead of relying on static knowledge, DRL enables the system to autonomously learn and update its policy through continuous interaction and exploration within the environment. This capability allows the mechanism to adaptively optimize transmission efficiency in response to dynamic and unseen network conditions, offering a level of robustness that static or supervised approaches cannot match.

Therefore, to overcome the adaptability limitations of fixed-size chunking in complex and dynamic network environments, we propose a dynamic chunking mechanism and design a novel intelligent and efficient distributed transmission method based on dynamic chunking. Crucially, this mechanism enables the system to adaptively adjust chunk sizes based on real-time network conditions, thereby optimizing transmission performance. This capability is a key characteristic that distinguishes it from traditional distributed transmission systems. The chunking mechanism utilizes a consensus algorithm to elect a master node responsible for continuously monitoring network conditions and data characteristics throughout the distributed system. Given the unpredictable nature of network dynamics and data heterogeneity in distributed systems, where traditional heuristic methods often fall short, we leverage DRL to learn optimal chunk size decisions. Through the training of the DRL model, the mechanism is capable of predicting the optimal chunk size in real time, enabling dynamic adjustments tailored to both network conditions and data characteristics, thereby enhancing the efficiency and stability of the transmission process. Additionally, a distributed hash table (DHT), a decentralized storage system that maps keys to specific nodes for scalable retrieval [18] is employed to enable efficient lookup of data chunks. Furthermore, a peer-to-peer (P2P) protocol [19] is employed to support node discovery, connection establishment, and data transmission. The Bitswap [20] protocol is also integrated to manage chunk requests and exchanges between nodes, enabling parallel data transmission across the system. To evaluate the effectiveness of the designed mechanism, we conduct extensive experiments on data transmission. The experimental results demonstrate that the proposed mechanism effectively adapts to dynamic network conditions and heterogeneous data characteristics, offering a robust, efficient, and scalable solution for data transmission in distributed systems.

The contributions are summarized as follows:

- We present a dynamic chunking-driven transmission mechanism for distributed systems. The mechanism jointly considers real-time network states and data characteristics when adjusting chunk sizes. These indicators are detected and aggregated by a master node, which is elected through a consensus-based mechanism to enable global coordination of chunking decisions across all nodes in the system.
- Compared to traditional heuristic methods that are often static and lack adaptability, we formulate the chunking decision problem as a MDP and propose a DRL-based dynamic chunking mechanism to learn the optimal chunking policy. To enhance the flexibility of the mechanism, supplementary components such as a delayed experience feedback scheme and a model migration mechanism are designed to ensure continuous operation even in the presence of node failures.
- We propose an efficient distributed data transmission mechanism. The mechanism leverages a P2P network for decentralized node discovery and data transmission, while employing a DHT to locate and identify data chunks. By enabling the parallel transfer of data chunks from multiple nodes, this approach significantly enhances the overall performance of distributed data transmission.
- To assess the performance of the proposed mechanism, we developed a distributed data transmission testbed and conducted comprehensive experiments under realistic distributed transmission scenarios. The experimental results show that our mechanism outperforms the others in multiple aspects, achieving a 22% improvement in transmission performance and a 28% increase in throughput, with a 31% optimization rate compared to traditional fixed-size chunking approaches.

The remainder of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 details our proposed system architecture. Section 4 presents the dynamic chunking and intelligent decision-making mechanisms. Section 5 describes the efficient transmission method based on dynamic chunking. Section 6 evaluates the performance of our proposed method through experimental results. Finally, Section 7 concludes the paper.

2. Related works

In this section, we provide an overview of two main areas closely related to our work: distributed data transmission techniques and data chunking, and further briefly discuss our design methodology in the context of existing approaches.

2.1. Distributed data transmission technology

Gnutella [12] was among the earliest distributed networks designed to support decentralized file sharing across all file types. Building on this foundation, a range of distributed networks subsequently emerged, many of which targeted specific application domains. Systems such as CAN [21], Chord [22], and Pastry [19] introduced structured overlay network protocols to support efficient routing and topology maintenance. Freenet [23] focused on enabling anonymous storage and retrieval in distributed environments. A notable advancement came with BitTorrent [13], which introduced an incentive-based mechanism that significantly improved transmission efficiency and resource utilization. BitTorrent remains widely used today. Parallel efforts have also explored distributed file systems, including Google FS and Hadoop, which have been foundational in big data infrastructures [24]. In recent years, IPFS [25] has gained increasing traction, particularly as a decentralized storage layer for blockchain-based applications [26,27]. Some researchers even envision IPFS as a potential successor to the Hypertext Transfer Protocol (HTTP). Other emerging distributed file storage systems include Arweave [28], Hypercore [29], Swarm [12], etc., which are also actively developing decentralized storage and transmission solutions that aim to address the limitations of traditional centralized approaches. Moreover, recent studies on adaptive networks [30] and multi-objective optimization [31] have validated the importance of adaptive strategies in handling dynamic topologies, offering valuable references for optimizing transmission efficiency in complex distributed systems.

The implementation and performance optimization of these distributed systems heavily rely on key technologies such as DHT, P2P networking, data chunking, and content addressing. DHT [18] serves as the

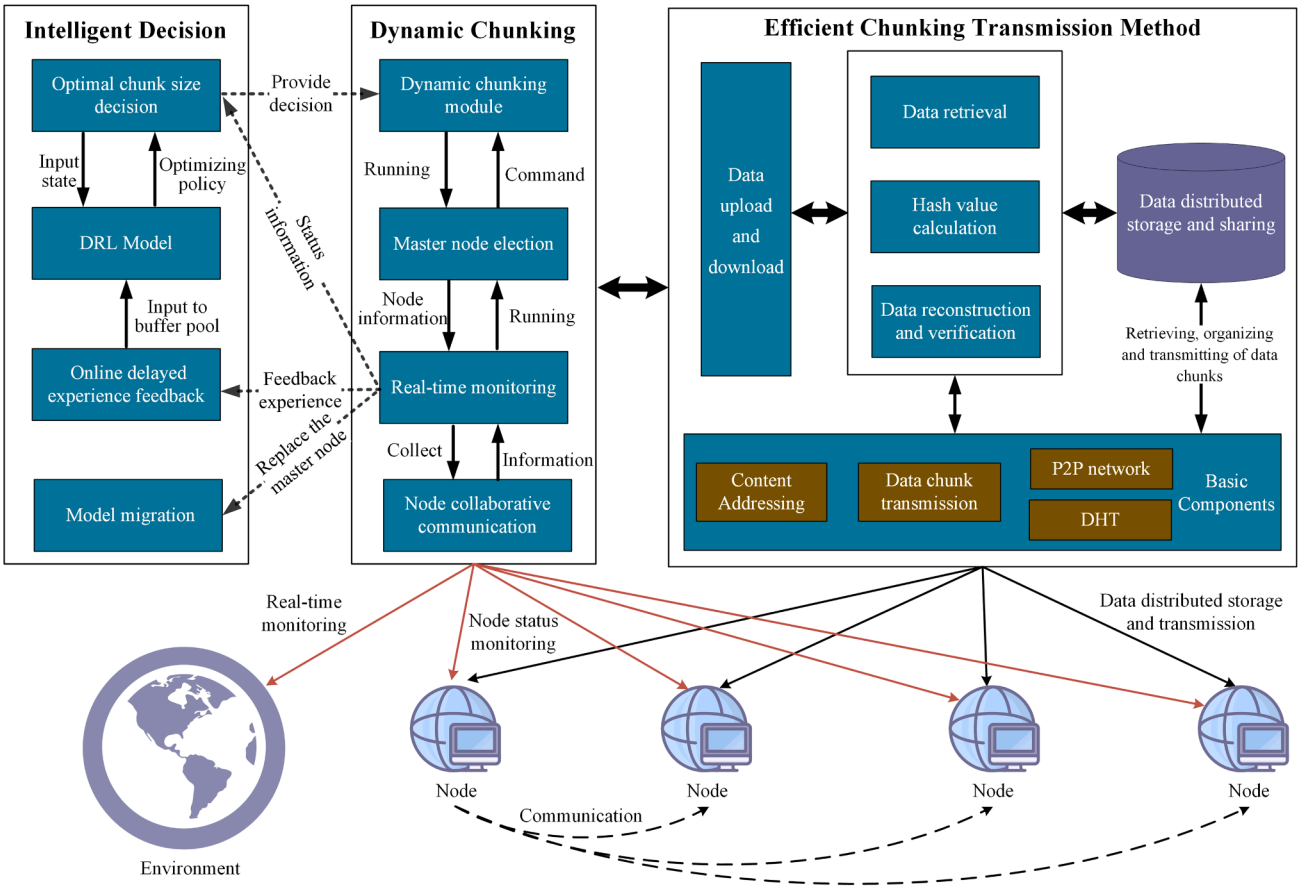


Fig. 1. The overall architecture of dynamic chunking transmission mechanism.

foundation for decentralized storage, enabling efficient data retrieval and storage by distributing data and indexes across various nodes in the network. This technology has been widely adopted in systems like IPFS, BitTorrent, Freenet, and Arweave. P2P networks [32] reduce reliance on centralized servers through direct node interaction and are extensively applied in IPFS, BitTorrent, Storj, and SAFE. Content addressing technology [23] locates data via unique data identifiers, ensuring data integrity and consistency, and is broadly applied in systems such as IPFS and Swarm. The Bitswap protocol [20], as one of IPFS's core protocols, optimizes bandwidth utilization and transmission efficiency by managing data chunk exchanges between nodes.

2.2. Data chunking

Data chunking [33] is a crucial processing technique widely used in many distributed transmission systems. At its core, it involves dividing large data objects into several smaller data chunks and distributing them across different nodes [21]. During the data chunking process, each data chunk is typically assigned a unique content hash value, enabling content-based addressing and retrieval without relying on physical locations. Compared to traditional centralized transmission methods, data chunking supports parallel downloading from multiple nodes [34], allowing better utilization of network bandwidth and faster overall transmission. Furthermore, the method increases fault tolerance, as data transfer can continue from other available nodes even if some fail, and only the missing chunks need to be retransmitted, reducing overhead.

Data chunking has been adopted to improve transmission efficiency and fault tolerance by several systems such as IPFS, BitTorrent, and Storj. In IPFS, data chunking is employed to split uploaded files into fixed-size chunks, each uniquely identified by hash value and stored within DHT [12]. This method not only enables efficient data retrieval and storage, but also ensures data integrity and consistency through content addressing. Within the BitTorrent protocol, data chunking enables users to download different pieces of a file in parallel from multiple peers, thereby speeding up the transmission process [13]. Furthermore, data chunking has been used to optimize data management in cloud storage systems. By splitting data into smaller chunks, the data chunking technique enables cloud services [35] to manage storage resources more flexibly, increase utilization rates, and accelerate access through parallel operations.

However, most existing distributed systems still rely on fixed-size chunking strategies, which present notable limitations in dynamic network environments. Fixed chunking lacks the flexibility to adapt to fluctuating network conditions and diverse data characteristics, often resulting in decreased transmission efficiency and underutilized bandwidth. Especially in environments with frequent changes of node status or highly heterogeneous data resources, fixed-size strategies fail to provide the required adaptability, becoming a major bottleneck for improving transmission performance in distributed systems. These limitations drive further research into more intelligent chunking strategies to better adapt to complex and volatile distributed environments.

3. System design

In this section, we present the overall architecture design of our proposed distributed data transmission system. To enhance the adaptability and efficiency of distributed data transmission, the system is designed with a modular architecture comprising three key components: the intelligent decision module, the dynamic chunking module, and the efficient chunking transmission method. The overall system architecture is illustrated in Fig. 1. Their functionalities are described below:

- **Dynamic Chunking Module**: This module coordinates and executes dynamic chunking strategies. The master node, elected via a election algorithm, continuously monitors the network environment, data characteristics, and node status. Based on this information, it analyzes and dynamically adjusts data chunk sizes. This module is also responsible for scheduling DRL model training and the dynamic chunking process. In the event of the current master node failure, it ensures continuous system operation through re-election.
- **Intelligent Decision Module**: This module is responsible for intelligent chunking decisions. It uses a DRL model to interact with the network environment in real time, continuously learning the optimal chunking policy. It dynamically adjusts data chunk sizes based on current data characteristics, node states, and network conditions. Additionally, this module includes a model migration mechanism, ensuring that the DRL model automatically migrates to a new master node if the current one fails, thereby guaranteeing the continuity of model training.
- **Efficient Chunking Transmission Method**: This method handles the indexing, addressing, exchange, and transmission of data chunks within the distributed network. It employs content addressing to uniquely identify each data chunk, utilizing DHT for data chunk retrieve and localization. The transmission method also utilizes P2P network and the BitSwap protocol to facilitate data chunk exchange and transmission between nodes.

Algorithm 1 The overall process of the dynamic chunking transmission mechanism.

Input: Real-time environment monitoring information
Output: Execution of dynamic chunking and data transmission

Initialize the system and elect a master node.
The Intelligent Decision Module runs the optimal chunking DRL model.
The Dynamic Chunking Module monitors bandwidth, latency, data size, type, and number of data providers for all nodes.
for each data item **do**
    Construct environment state.
    The Intelligent Decision Module predicts the optimal chunk size using the DRL model.
    if re-chunking is needed **then**
        The Dynamic Chunking Module notifies provider nodes to re-chunk the data.
        The Efficient Chunking Transmission Method generates CID for each chunk.
        Store chunks locally and update DHT.
    end if
end for
for each data transmission request **do**
    The Efficient Chunking Transmission Method queries DHT to locate data chunks.
    Download chunks in parallel via P2P network.
    Cache chunks locally and update DHT.
end for
The Intelligent Decision Module collects transmission feedback and stores in experience buffer.
Optimize the DRL model and the optimal chunking policy.
if master node fails **then**
    Re-elect a new master node.
    The Intelligent Decision Module migrates the DRL model to the new master.
end if
return Completion information of dynamic chunking and data transmission.

The overall workflow of the system is as follows: As illustrated in Algorithm 1, the master node is elected through a consensus algorithm. The master node interacts with slave nodes in the distributed network via the dynamic chunking module. It continuously monitors key metrics, including network bandwidth, transmission latency, data size, data type, and the number of providers. Based on these observations, the intelligent decision module analyzes the network conditions. It then leverages the DRL model to predict the optimal chunk size for different data and determines whether re-chunking is necessary. If chunking adjustments are required, the master node issues instructions to the relevant slave nodes to re-split the data. The newly sliced data chunks are then processed by the efficient chunking transmission method, which generates unique content identifier (CID), a cryptographic hash serving as the immutable fingerprint for addressing the content, and stores them in the DHT. When a node requests specific data, the chunking transmission method queries the DHT to locate the corresponding data chunks and initiates parallel transmission from multiple data provider nodes using P2P protocols. Upon completion, the received data chunks are cached in the requester's local storage, and the DHT index is updated accordingly. After data transmission concludes, the intelligent decision module collects feedback information such as transmission duration, data characteristics, network conditions and node status. The information is fed back to the DRL model, and stored in experience replay buffer to support the model's continuous learning and optimization. In the event of the current master node failure, the consensus algorithm detects the issue and promptly initiates a re-election process to select a new master. Simultaneously, the model migration mechanism automatically transfers the DRL model to the new master. This ensures that the agent can continue incremental learning based on previous experience without restarting from scratch.

Complexity Analysis. The computational complexity of the proposed mechanism comprises the DRL decision cost (T_{dec}) and the data execution cost (T_{exec}). In the decision phase, the DRL agent employs a fully connected Deep Neural Network (DNN) with $K = 2$ hidden layers, an input state dimension $I = 8$, a hidden dimension $H = 128$, and an output action dimension $O = 8$. The computational complexity is dominated by layer-wise matrix multiplications, formulated as $O(I \cdot H + (K-1) \cdot H^2 + H \cdot O)$. Since these parameters are constants independent of the data scale, the decision complexity T_{dec} is asymptotically $O(1)$. Conversely, the execution complexity scales linearly with the file size, yielding $T_{exec} = O(L)$. Consequently, the overall complexity is bounded by $O(L)$, demonstrating that the intelligent chunking mechanism incurs negligible computational overhead relative to the data transmission task itself.

4. Design of dynamic chunking and intelligent decision mechanism

In this section, we will introduce the design of the dynamic chunking and intelligent decision-making mechanisms, including the master election algorithm, DRL-based chunking decision-making, and the process of dynamic data chunking. In addition, we will present the delayed experience feedback mechanism and the model migration mechanism as effective supplements to the core mechanisms, enhancing the system's adaptability and continuity of training.

4.1. Master node election algorithm

The election algorithm is responsible for selecting a master node to manage and coordinate all nodes in the distributed transmission network. It ensures system continuity and stable chunking decision training by handling node failures and dynamic network changes. The overall election process is illustrated in Fig. 2.

In terms of role assignment, the election algorithm classifies nodes in the transmission system into three roles: leader, candidate, and follower. Beyond maintaining log consistency as in traditional protocols [36,37], the leader undertakes several key responsibilities. These include monitoring network status, scheduling chunking decisions, and

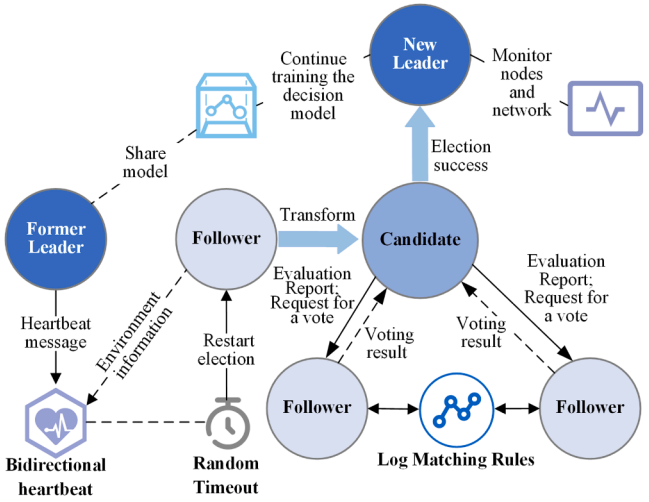


Fig. 2. The process of master node election.

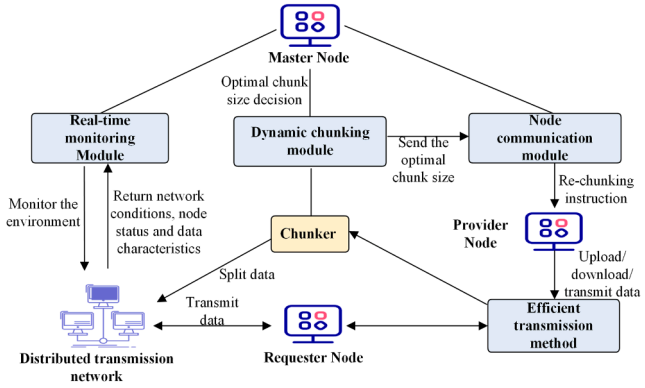


Fig. 3. The process of dynamic chunking.

training the DRL model. When campaigning, candidates must submit a comprehensive evaluation report including the node's computing capacity, network bandwidth, and storage performance. Followers use an improved bidirectional heartbeat mechanism to both receive instructions from the leader, and report local network status and data possession. In addition, the leader periodically shares the DRL model parameters and state information across the distributed transmission network.

During the leadership phase, the leader manages all followers and regularly sends heartbeat log to maintain its leadership. Each follower has an election timeout mechanism, which sets a timer that resets upon receiving a heartbeat log. If a follower does not receive a heartbeat log within the timeout period, it switches to a candidate, increases its term number, and initiates a new election process. Candidates send vote requests to other nodes, which determine whether to vote based on log consistency and comprehensive evaluation information. If a candidate receives votes from the majority of nodes, it becomes the new leader and enters the leadership phase. If it does not receive enough votes, it restarts the election process.

The election algorithm also introduces various mechanisms to handle node failures and abnormal behavior during the election process. To avoid the vote-splitting problem caused by multiple candidates requesting votes at the same time, the algorithm sets a randomized election timeout to reduce the probability of multiple nodes becoming candidates simultaneously. In addition, the algorithm ensures that only candidates with the latest logs can receive votes through log matching rules. During the election process, if a node fails to respond in time due to network problems, other nodes will wait for a certain period before voting to avoid invalid elections caused by temporary failures.

Correctness and Convergence Analysis. To theoretically validate the reliability of the election algorithm, we formally analyze its correctness and convergence time. Regarding Correctness, assume that a distributed system with N nodes elects two leaders in term t with vote sets V_1 and V_2, which implies $|V_1| + |V_2| > N$. It follows that $V_1 \cap V_2 \neq \emptyset$, meaning at least one node voted more than once in the same term, which contradicts the algorithmic constraint of "at most one vote per term"; thus, the assumption is invalid. Furthermore, the randomized timeout mechanism resolves split-vote deadlocks. Let $P_{conf} < 1$ be the probability of a conflict in a single round. As the election rounds $k \to \infty$, the cumulative success probability $P_{suc} = 1 - P_{conf}^{k}$ approaches 1, ensuring that a valid leader is eventually elected, thereby proving the correctness. Regarding Convergence, the system recovery time T_{rec} depends on the failure detection latency T_{det} and the election duration. Assuming the election timeout follows a uniform distribution $T_{timeout} \sim U[T_{\min}, T_{\max}]$, the network Round-Trip Time (RTT), defined as the bidirectional delay is τ, and the number of election rounds R follows a geometric distribution, the expected failure detection time is $\mathbb{E}[T_{det}] = (T_{\min} + T_{\max})/2$. Consequently, the total convergence time is formulated as $\mathbb{E}[T_{rec}] = \mathbb{E}[T_{det}] + \mathbb{E}[R] \cdot (T_{\min} + \tau)$. Since the randomized timeout minimizes the conflict probability (implying $\mathbb{E}[R] \approx 1$), the convergence time is bounded by $O(T_{\max})$, demonstrating that the system completes failure recovery with constant time complexity.

4.2. Dynamic chunking mechanism based on DRL

To improve adaptability to dynamic network environments and optimize performance of data transmission, we introduce dynamic chunking into distributed data transmission. The core of this approach lies in making real-time and intelligent decisions on the optimal data chunk size. However, the highly dynamic nature of network conditions and data characteristics in distributed systems make the dynamic chunking decision problem challenging to model. Traditional heuristic or rule-based methods typically rely on preset rules or empirical thresholds. Due to the lack of clear patterns in complex environmental changes, these methods frequently lead to outdated or mismatched strategies, thereby preventing optimal transmission performance.

To address this issue, we integrate DRL into the optimal chunking decision process. The mechanism consists of two core components: dynamic chunking and DRL model. The dynamic chunking module continuously monitors environment conditions and executes chunk adjustments, with the DRL module making optimal chunk size decisions. These two modules cooperate closely to enable adaptive chunking.

4.2.1. Dynamic chunking

In the mechanism, the dynamic chunking module plays a key role in bridging environment perception and chunking execution. As demonstrated in Fig. 3, it operates in close coordination with the DRL model by receiving chunking decisions based on real-time network and data conditions, and then performing the corresponding chunk size adjustments. The system, through the master node, continuously monitors key metrics across all nodes in the distributed network, including network bandwidth, latency, data size, data type, and the number of providers for different data. The collected information serves as the environment state input. Based on this input, the DRL model outputs the optimal chunking decision using a policy function trained on historical transmission experience. After data transmission is completed, the system collects transmission performance indicators as experience feedback to continuously optimize the chunking policy, enabling the DRL model to self-learn and adapt over time.

Leveraging the optimal chunking decision provided by the DRL model, the dynamic chunking module initiates and coordinates the corresponding chunk adjustment operations across participating nodes. The module first evaluates whether the existing chunk size of different data need to be adjusted based on the decision, and if necessary, the master

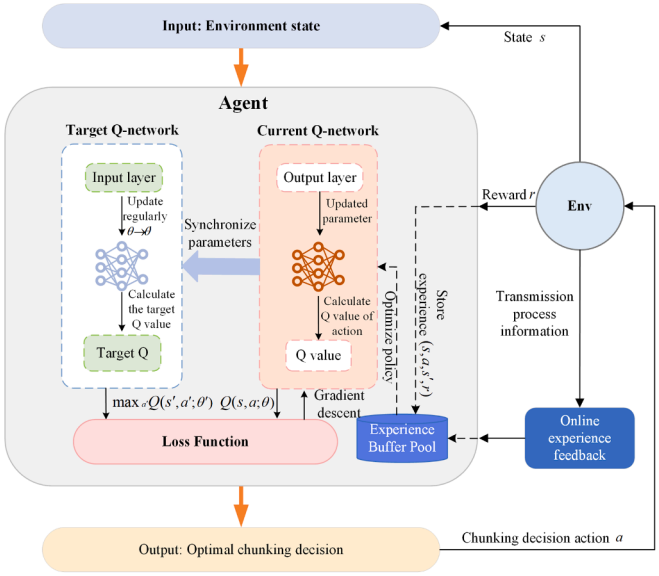


Fig. 4. DRL design.

node notifies the relevant nodes to perform re-chunking. During the re-chunking process, the master node instructs all nodes owning the data to split it into multiple smaller data chunks according to the optimal chunk size. Each chunk is assigned a unique CID through hash computation and stored in the node's local repository. The sliced data chunks are organized into a Directed Acyclic Graph (DAG), a topological structure that eliminates circular dependencies to ensure data integrity, using the Merkle DAG [38] structure, with the root node's CID serving as the unique identifier of the entire data. At the same time, the system records the mapping between the CIDs and the owner nodes in the DHT. This allows a requesting node to quickly locate multiple owner nodes via the DHT, download different data chunks in parallel, and finally reassemble them into the complete file locally.

4.2.2. DRL design

To implement the decision process, the dynamic chunking problem is formulated as a Markov Decision Process [39]. The MDP can be represented as a tuple $< S, A, P, R >$, where:

State, S: As shown in Fig. 4, the state space represents the set of environmental features perceived by the model, including network conditions, node status, and data characteristics. In distributed environments, numerous factors affect data transmission efficiency, so the state space design must sufficiently reflect the environmental state information. Among these, the network bandwidth of each node directly impacts data transmission speed. When the bandwidth is high, a larger data chunk size can be used to reduce the time cost of data retrieval and reassembly, thereby improving overall transmission efficiency. However, in distributed environments, nodes may frequently join or leave the network, and their total number is typically large and variable. Directly including the bandwidth of all nodes in the state space may result in an excessively large state dimension, thereby increasing model complexity and computational cost. Therefore, we use the average bandwidth of all nodes to represent the overall network bandwidth. Suppose there are N nodes in the network and the bandwidth of the node i is b_i, then the average bandwidth B_{avg} is represented as follows:

$$B_{avg} = \frac{1}{N} \sum_{i=1}^{N} b_i \tag{1}$$

The latency between nodes refers to the time it takes for a data packet to travel from a source node to a destination node, also known as one-way delay, which reflects the network responsiveness between nodes. In this paper, we use RTT to calculate the one-way delay between nodes. Thus, the average delay ($\bar{d}_i$) is defined as the average one-way delay between that node and all other nodes, as follows:

$$\bar{d}_i = \frac{1}{N-1} \sum_{\substack{j=1 \\ j \neq i}}^{N} d_{ij} \tag{2}$$

where N is the number of nodes in the distributed network; $d_{ij} = RTT_{ij}/2$ is the one-way delay between node i and node j.

Furthermore, the overall average network delay D_{avg} can be calculated as follows:

$$D_{avg} = \frac{1}{N} \sum_{i=1}^{N} \bar{d}_i \tag{3}$$

In addition to network bandwidth and latency, the inherent characteristics of the data-such as its size and type-also have a significant impact on the performance of dynamic chunking-based transmission in distributed environments. The data size is denoted as $size_f$ and the data type (e.g., video, compressed files, audio, and text) is encoded using a 4-dimensional one-hot vector denoted as $type_f$. Moreover, the current chunk size of the data (denoted as $chunksize_{now}$) and the number of providers in the network (denoted as num_p) also influence the dynamic chunking decision. When there are more providers, the chunk size can be appropriately reduced to enable more effective parallel transmission. Thus, the state s is a vector represented as follows:

$$S = \left(B_{avg}, D_{avg}, size_f, type_f, num_p, \text{chunksize}_{now}\right) \tag{4}$$

Action, A: In the optimal chunking decision, the action refers to the adjustment of data chunk size made by the agent based on the current distributed environment and data state. To reduce the computational complexity of the model and ensure the effectiveness of the action space, we employ a discrete design, defining the action space as a discrete set. The range of data chunk sizes is set to [128, 1024] KB, with a fixed step size of 128 KB to construct the discrete action sequence. Specifically, the action is an 8-dimensional one-hot vector, represented as:

$$action = [a_1, a_2, \ldots, a_8] \tag{5}$$

where a_i has a value of $128 * i$, representing the chunk size that should be adopted for the data under the current state.

Reward, R: The reward function is the core element guiding the optimization direction of the DRL model, used to evaluate the reward obtained after the agent takes an action. In the actual data transmission process within the distributed environment, different states s and chunking actions a will result in different data transmission durations, denoted as $T(s, a)$. In the optimal chunking decision, the objective of the policy is to select the most appropriate chunk size to minimize the transmission time. Therefore, the reward function can initially be defined as the negative value of the transmission time, where a shorter transmission time yields a higher reward. However, the transmission time is significantly influenced by the data size, which may lead to large variations in reward values and affect the stability of model training. To keep the reward values within a reasonable range, we use the maximum transmission time $T_{\max}$ and the minimum transmission time $T_{\min}$ collected from the initialized experience buffer as reference values for normalization. To ensure the normalization remains robust to distribution shifts, the values of $T_{\max}$ and $T_{\min}$ are periodically updated by retrieving the maximum and minimum transmission times currently stored in the experience replay buffer. This ensures that the normalization range adaptively evolves based on the actual transmission dynamics observed throughout the training process. The reward function r is defined as:

$$r(s, a) = -\frac{T(s, a) - T_{\min}}{T_{\max} - T_{\min}} \tag{6}$$

The overall objective is to maximize the cumulative reward, which can be defined as follows:

$$R = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \gamma^t r_t], \tag{7}$$

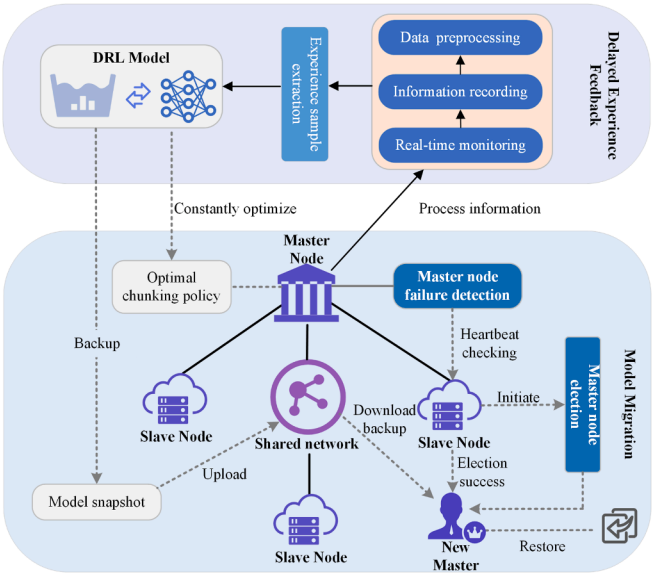


Fig. 5. Supplementary mechanisms.

where τ represents a sequence of states, actions, and rewards from the dynamic chunking decision process, and π_θ is the chunking policy parameterized by θ. T denotes the total number of steps in the trajectory, where each step corresponds to a data chunking operation. γ is the discount factor, used to balance immediate and long-term cumulative rewards in the model.

Transition, P: The state transition function $P(s_{t+1}|s_t, a_t)$ defines the probability of transitioning to a new state s_{t+1} after taking action a_t in the current state s_t. In the distributed environment, each data is monitored in real time with respect to its size, type, current chunk size, the overall average network bandwidth and latency, as well as the number of data providers. When it is necessary to adjust the chunk size, the most suitable chunk size under the current state is selected for re-chunking. After re-chunking, the data-related parameter $chunksize_{now}$ is updated accordingly. Mathematically, we formulate this transition dynamics as a coupled probability density function:

$$P(s_{t+1}|s_t, a_t) = \delta(c_{t+1} - a_t) \cdot \mathcal{T}(\Omega_{t+1}|s_t, a_t), \tag{8}$$

where $\delta(\cdot)$ is the Dirac delta function representing the deterministic control update, ensuring that the chunk size dimension c_{t+1} ($chunksize_{now}$) in the next state is strictly set to the action a_t. The term $\mathcal{T}$ denotes the stochastic transition kernel of the environment, governing how the network dimensions Ω_{t+1} (e.g., bandwidth B_{avg} and delay D_{avg}) evolve based on the aggregated monitoring data collected during the subsequent transmission phase. Thus, the transition implicitly captures the interaction between the chunking decision and the dynamic network environment.

4.3. Supplementary mechanisms for agent deployment and training

To further support the effective deployment and continuous training of the DRL agent in distributed environments, we design two supplementary mechanisms. The supplementary mechanisms are illustrated in Fig. 5. These mechanisms enhance the reliability and adaptability of the system by addressing practical challenges related to delayed feedback and master node failures.

4.3.1. Online delayed experience feedback

In the chunking decision mechanism, the agent is deployed on the master node of the distributed transmission system, and is responsible for online execution and training. However, after the agent makes a chunking decision during actual operation, it cannot immediately obtain feedback on the transmission result. If data transmission need to be performed directly to obtain instant feedback, it will result in frequent occupation of network bandwidth, thereby affecting normal data transmission among nodes. To address this, we design a staged online delayed experience feedback mechanism. After completing data transmission, each node records transmission process information and periodically sends it to the master node. The master node aggregates, analyzes, and preprocesses the collected information to extract valid experience samples. Once a predefined threshold is reached, these samples are added to the experience replay buffer of the chunking decision model. The approach not only reduces bandwidth consumption caused by frequent communication but also effectively leverages inter-node transmission information to continuously optimize the chunking decision policy.

4.3.2. Model migration

In the distributed environment, the master node is responsible for core tasks such as chunking decision and model training. Once the master node fails, the training process of the DRL model may be interrupted, affecting the continuity of dynamic chunking decision and the stability of performance. To address this, we design a model migration mechanism that allows the DRL model to transfer from the old master node to the new one, continuing its incremental learning based on the previous state. During normal system operation, the master node periodically saves and uploads the DRL model parameters and states to the system's shared network storage to ensure that the latest model data is not lost in case of master node failure. When a failure of master node is detected, the election algorithm promptly initiates a new election process to select a new master node to take over the system. After the new master node is selected, it first retrieves the latest DRL model parameters and states from the most recent shared network backup. Meanwhile, the DRL model on the new master node also synchronizes the existing experience replay buffer to continue incremental learning based on the latest data. The model migration mechanism ensures the model can keep optimizing its optimal chunking policy in complex and changing network environments without interrupting the learning process due to master node changes, thus guaranteeing system stability.

5. Efficient transmission method based on dynamic chunking

In this section, we present the efficient transmission method based on dynamic chunking, specifically covering data chunking, distributed storage, transmission, and reassembly verification.

5.1. Data distributed storage

The overall architecture of the efficient transmission method based on dynamic chunking is illustrated in Fig. 6. Distributed storage is the foundation of the efficient transmission method in distributed systems, responsible for tasks such as chunking, identifying, and storing shared data uploaded by nodes. When a node needs to store data in the shared network, the uploaded data is first split into multiple data chunks. The size of each data chunk is determined by the DRL-based decision mechanism, which predicts the optimal size based on the real-time network environment. Each resulting data chunk is individually hashed using the SHA-256 [40] algorithm to generate a unique content identifier. The approach not only enables content-based addressing of data throughout the distributed environment but also ensures data integrity and consistency verification. Any tampering or loss of data chunks will change their hash values, meaning that acquiring data through its hash index directly points to the correct data entity. The sliced data chunks are organized using the Merkle DAG data structure, with the root node's hash serving as the unique identifier for the entire data. All hash indexes and storage addresses are recorded in the DHT. The DHT establishes a mapping between data hash indexes and their provider nodes, enabling fast data retrieval and location.

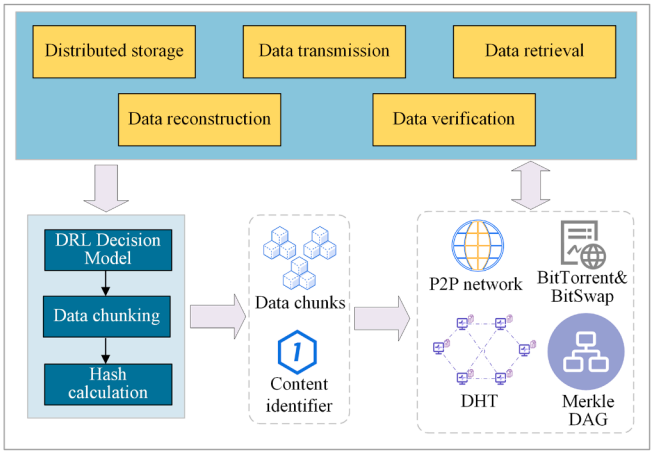


Fig. 6. Efficient transmission method.

When a node needs to obtain data, the requester can directly query the DHT for data providers and transfer data in parallel from multiple nodes via the P2P network. It achieves decentralized data transmission, ensuring security and reliability during transfer. After obtaining data chunks, nodes cache them locally and subsequently serve as new providers, facilitating data exchange and transmission to other requesters. Distributed storage ensures high data integrity and availability through redundant storage across various node caches. Even if some nodes fail, as long as at least one provider is operational, the data can still be provided.

5.2. Data retrieval

When a node needs to acquire specific data, it uses the data's hash value along with the DHT and content-addressing technology to locate the providers of the data. After obtaining the locations of providers, the node can download different data chunks in parallel from multiple providers via the P2P network. During the data retrieval process, each node maintains a routing table that splits the entire distributed network into multiple k-buckets. Each k-bucket stores information about nodes within a specific distance range from the current node, with fixed capacity and ordered by their most recent contact time. Additionally, nodes utilize the Kademlia routing algorithm for node location and data retrieval. The algorithm calculates the logical distance between the IDs of two nodes using XOR operations, where a smaller XOR distance implies closer proximity between nodes, enabling faster discovery of the target node and optimizing the data retrieval process.

Ultimately, the system obtains a list of nodes storing the target data via the distributed algorithm and selects a subset of them to request the data. The specific workflow of the data retrieval algorithm is as follows. First, the current node computes the XOR distance to the target hash value and selects the closest nodes from its routing table as initial candidate nodes. Then, it sends query requests to these candidates. Each candidate node checks whether it stores the data entity corresponding to the target hash value. If not found, the candidate returns a list of nodes from its routing table that are closer to the target hash value. This iterative process continues until the node corresponding to the target hash value is found, or when the nodes returned by the iteration are no longer closer to the target hash value. If a data chunk has multiple providers, the requesting node can also select the top-priority nodes for retrieval based on current network conditions and transmission performance.

5.3. Data transmission

During data transmission, nodes communicate directly with data owner nodes via a P2P network. Each data chunk is transmitted independently, and different chunks can be retrieved in parallel from multiple providers. Nodes can also dynamically adjust the transmission priority of data chunks and nodes based on data needs, network conditions, and transmission history. Additionally, the BitSwap protocol manages the data exchange process by recording transmission history between nodes, allowing the system to prioritize certain nodes for data retrieval and thereby optimize parallel transmission.

During the data transfer process, when a node needs to obtain specific data, it first queries its local cache and neighboring nodes. If the data is not found, the DHT-based iterative retrieval process is initiated. Based on the list of provider nodes returned by the DHT, the requesting node selects several target nodes and sends a data request, which includes the hash of the target data and the number of chunks needed. The request also specifies the list of desired data chunks. Upon receiving the request, the provider nodes check their local repositories and respond with the availability of the requested chunks, which are then transmitted via the P2P network. Throughout the process, the node continuously adjusts the priority of chunks to ensure higher-priority data chunk is transmitted first. The BitSwap protocol enables concurrent transfers of data chunks from multiple nodes, improving overall transmission efficiency. A sliding window mechanism manages out-of-order chunk arrivals, and the system organizes received chunks using hash-based structures to ensure data integrity and consistency. If any chunks are missing or corrupted, the node will re-request them from available providers and increase their transmission priority, thereby enhancing the reliability of the transmission process.

5.4. Data reconstruction and verification

After receiving all the data chunks, the requesting node reconstructs the original data based on the hash index and chunking information. During the reassembly process, all chunks are organized in their original order using a Merkle DAG structure, where the hash identifiers of child nodes are recursively used to generate those of parent nodes, ultimately forming the root hash identifier. The root identifier serves as the CID of the complete data, which encapsulates information such as the identifier encoding format, data encoding type, and content hash value. During the integrity verification phase, the system traverses the Merkle DAG starting from the root hash, recursively verifying the hash identifiers of child nodes. If a mismatch is found between the computed and expected hash values of any chunk, the system determines that the chunk has been tampered with or lost. In such cases, the requesting node will initiate a retransmission request to the provider node for that chunk. For large files, the mechanism also supports on-demand verification. If only a portion of the file is needed, the verification mechanism can check just the path from the root node to the target data chunk, without traversing the entire data structure. Through collaboration between data reconstruction and verification, the system ensures that data chunks remain untampered and intact throughout the transmission process.

6. Evaluation

In this section, we evaluate the transmission performance of the proposed mechanism on the constructed distributed platform. We also compare it with other baselines in terms of transmission duration, throughput, and optimization rate to better validate the effectiveness and adaptability of the mechanism.

6.1. Experimental settings

For the evaluation, we constructed a distributed emulation platform utilizing five independent virtual machine nodes hosted on a high-performance cloud server. The server is equipped with an Intel Xeon Gold 6238R CPU (@ 2.20 GHz), 64 GB of RAM, and a 512 GB SSD, running the Ubuntu 18.04 LTS operating system. To emulate the dynamic

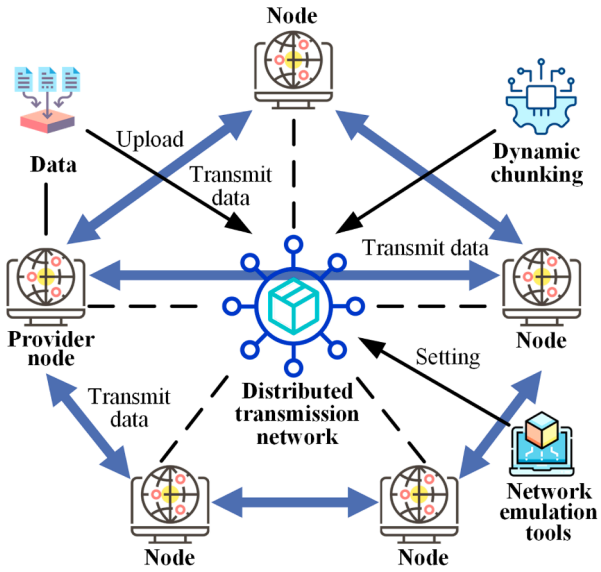


Fig. 7. Experimental emulation platform.

Table 1
Hyperparameter settings.

Description	Symbol	Value
Hidden Layer Dimensions	n_{hidden}	128, 128
State Dimension	d_s	8
Action Dimension	d_a	8
Learning Rate	α	10^{-4}
Discount Factor	γ	0.9
Exploration Rate	ϵ	0.9
Target Update Frequency	N_{target}	200
Replay Buffer Capacity	C	1000
Batch Size	B	32
Min Sampling Threshold	$N_{\min}$	200
Gradient Clipping	G_{clip}	1.0

and heterogeneous network conditions of real-world distributed environments, we employed network emulation tools such as TC to precisely configure the network bandwidth, latency, and packet loss rate for each node. The architecture of this experimental platform is illustrated in Fig. 7. In the implementation of the proposed mechanism, the hyperparameter settings for the DRL model are detailed in Table 1. To ensure a fair and rigorous comparison, all baseline methods were tuned and evaluated under the exact same environmental conditions and datasets as the proposed mechanism. Additionally, we selected the following three baselines to demonstrate the effectiveness of the proposed mechanism, as follows:

(1) **Fixed**: In previous works [13,33], the fixed chunking approach typically splits data using a chunk size of 256 KB.
(2) **Random**: The random chunking approach selects a chunk size randomly for splitting the data.
(3) **Fitted formula**: The fitted-formula-based chunking approach [41] builds a transmission time prediction model through mathematical modeling and polynomial fitting, then calculates the chunk size that minimizes the transmission time for splitting the data.

6.2. Performance comparison

In this part, we simulate various network conditions, file properties, and node states to comprehensively evaluate the performance of our proposed mechainism. In the experiments, the network bandwidth is set to ten different values ranging from 16 Mbps to 160 Mbps with a step of 16 Mbps. The number of data provider nodes varies from 1 to 4, and the inter-node delay is configured to 0 ms, 50 ms, and 100 ms to simulate no delay, normal delay, and high-delay scenarios, respectively. In terms of data characteristics, various file types (e.g., txt, zip, mp4, and mp3) are involved, with file sizes ranging from 100 MB to 1000 MB.

6.2.1. Transmission duration

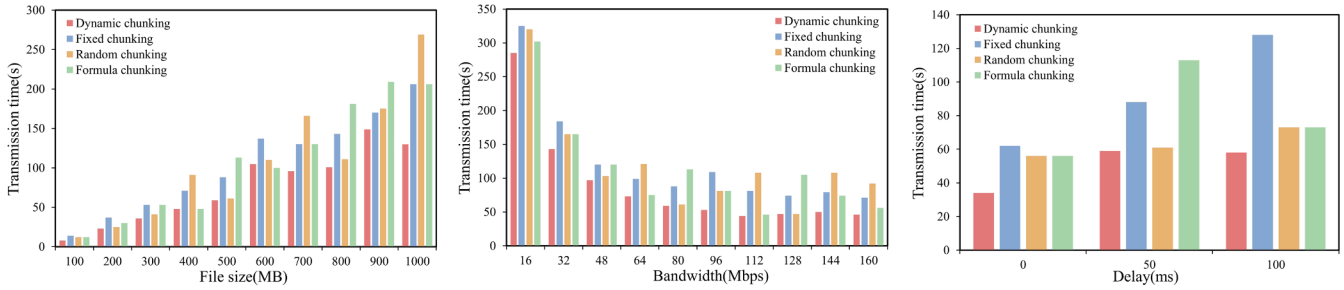
Fig. 8 illustrates the transmission duration under different conditions for the four chunking-based transmission strategies when the number of data provider nodes is fixed. Firstly, the Fig. 8(a) shows the variation in transmission duration with respect to data size when the node bandwidth is 80 Mbps and the network delay is 50 ms. It can be observed that the proposed method achieves lower transmission duration than the other three strategies across most data sizes. Its advantage is particularly noticeable for larger files, where the transmission duration is significantly lower than the other baselines. Although the random chunking and fitted formula chunking may yield lower transmission duration in certain cases, their performance is highly unstable. This may be due to the selection of excessively small chunk sizes, resulting in a larger number of chunks and thus increased retrieval time. Then, the Fig. 8(b) shows that under a fixed delay of 50 ms, the proposed mechanism achieves lower transmission time than the baselines across different bandwidth for data of the same data size. Notably, the traditional fixed chunking strategy performs significantly worse in many cases, especially at higher bandwidths. This indicates that fixed chunking designs tend to waste a large portion of time on chunk retrieval and other operations, leading to poor bandwidth utilization and increased transmission duration. Finally, the Fig. 8(c) demonstrates that for 500 MB of data size under different delay settings, the transmission duration of the dynamic chunking remains significantly lower than that of the baselines. Moreover, compared to random chunking and formula-based chunking, the dynamic strategy maintains more stable performance under higher delay, indicating better adaptability.

6.2.2. Overall throughput

Fig. 9 presents a comparison of overall throughput for the four chunking strategies under different conditions with the number of data provider nodes fixed. Firstly, the Fig. 9(a) shows how throughput varies with data size when each node has a bandwidth of 80 Mbps and a network delay of 50 ms. It can be observed that the proposed mechanism consistently achieves higher overall throughput than the baselines in most cases. The advantage is particularly pronounced for larger data sizes, indicating that the dynamic chunking mechanism effectively optimizes the chunking strategy to reduce transmission wait time and retrieval delays, thereby improving overall throughput. Then, the Fig. 9(b) shows the throughput of each method when transferring 1000 MB file under a fixed delay of 50 ms and varying bandwidth levels. The results indicate that the dynamic chunking mechanism demonstrates clear advantages in throughput across most bandwidth settings, especially in higher-bandwidth scenarios. While the throughput performance of all methods is relatively close under lower bandwidths, the dynamic chunking method still maintains a noticeable edge. In contrast, under high bandwidth conditions, the throughput of the other baselines does not significantly improve, suggesting that their chunking strategies fail to fully utilize the available bandwidth. Finally, the Fig. 9(c) compares the throughput of the four strategies for transmitting a 1000 MB file under varying delay conditions while keeping bandwidth fixed. The results show that the dynamic chunking strategy maintains more stable throughput performance as network delay increases, outperforming the other baselines in terms of stability and adaptability.

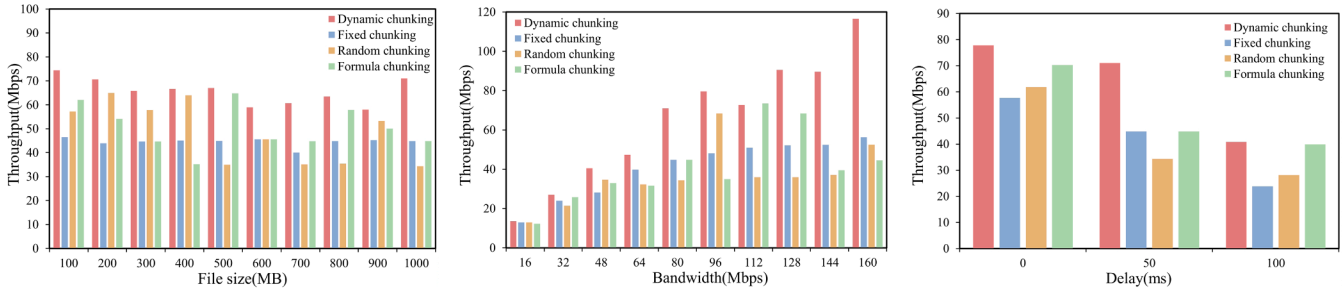
6.2.3. Transmission optimization rate

Fig. 10(a) to (c) illustrate the transmission optimization rate of the proposed mechanism compared to the traditional fixed chunking



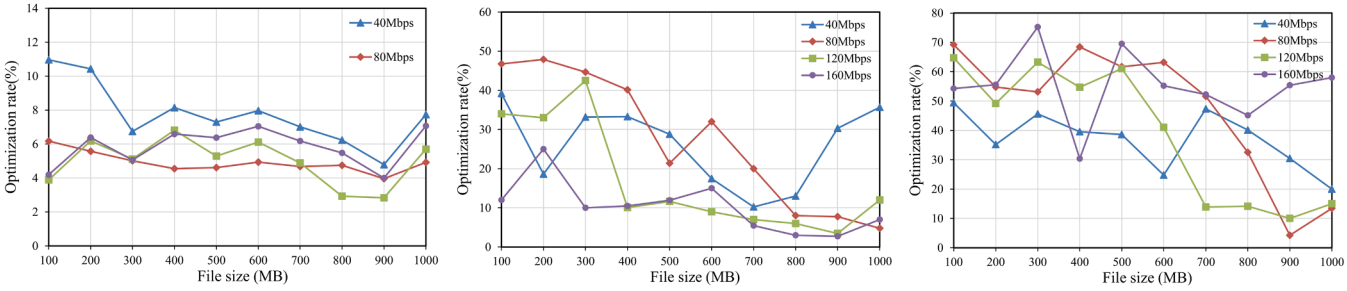
(a) 80Mbps bandwidth and 50ms latency. (b) 500MB size and 50ms latency. (c) 500MB size and 80Mbps bandwidth.

Fig. 8. Transmission duration under different network conditions.



(a) 80Mbps bandwidth and 50ms latency. (b) 1000MB size and 50ms latency. (c) 1000MB size and 80Mbps bandwidth.

Fig. 9. Overall throughput under different network conditions.



(a) 0ms latency. (b) 50ms latency. (c) 100ms latency.

Fig. 10. Transmission optimization rate under different network conditions.

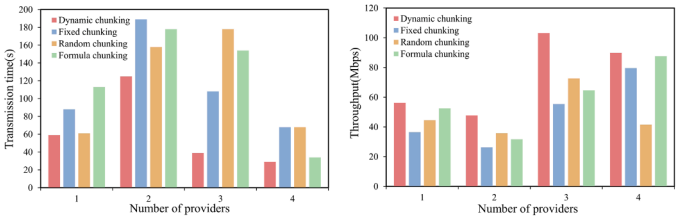
Table 2
Comparison of transmission duration and throughput.

Method	Transmission Duration (s)		Overall Throughput (Mbps)	
	Mean ± SD	Variance	Mean ± SD	Variance
Fixed Chunking	87.40 ± 12.50	156.25	45.77 ± 6.80	46.24
Random Chunking	101.50 ± 28.50	812.25	39.41 ± 16.50	272.25
Formula Chunking	72.50 ± 9.20	84.64	55.17 ± 8.40	70.56
Dynamic Chunking	**59.20 ± 6.50**	**42.25**	**67.57 ± 7.50**	**56.25**

approach. Overall, the proposed mechanism demonstrates superior optimization rates under various conditions, including different bandwidths, delays, and file characteristics. In particular, under high-latency environments, our proposed mechanism effectively adapts to network fluctuations and significantly improves transmission performance and efficiency. Firstly, the Fig. 10(a) shows the optimization effect under a no-delay (0 ms) environment, where the optimization rate generally ranges between 3% and 11%, with a more noticeable improvement for small file transfers. Then, the Fig. 10(b) presents the results under a typical delay (50 ms), where the optimization rate can reach up to 45%, with the most significant improvements observed for files smaller than 500 MB. Under low-bandwidth conditions (40 Mbps), the proposed mechanism achieves higher optimization rates for large file transfers. This indicates that our dynamic chunking mechanism adapts better to low-bandwidth and higher-latency conditions compared to traditional fixed chunking. Finally, the Fig. 10(c) demonstrates that the proposed mechanism achieves the highest optimization under high-delay environments (100 ms), with peak optimization rates reaching 72% and an average of around 40%. For larger file transfers, the optimization rate slightly decreases, hovering around 20%. The results confirm the mechanism's strong adaptability to high-latency conditions. By dynamically adjusting chunk sizes according to current network conditions, the dynamic chunking strategy effectively optimizes distributed transmission performance.

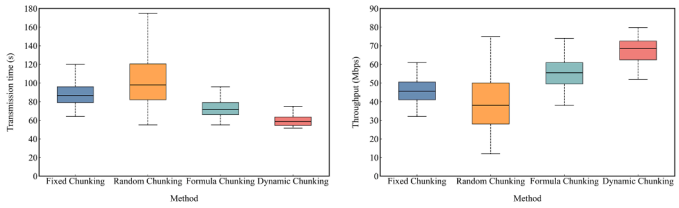
6.2.4. *Scalability*

To evaluate the scalability of the system, we analyze the effectiveness of the four chunking methods as the number of provider nodes



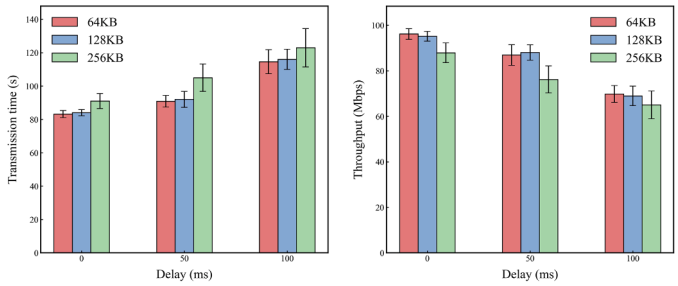
(a) Transmission duration. (b) Overall throughput.

Fig. 11. Performance when the number of provider nodes changes.



(a) Transmission duration. (b) Throughput.

Fig. 12. Box plots of transmission duration and throughput.



(a) Transmission duration. (b) Throughput.

Fig. 13. Performance comparison of different action granularities.

increases. Fig. 11(a) presents the transmission duration of a 500 MB file under different numbers of providers, with a bandwidth of 80 Mbps and a latency of 50 ms. The results indicate that the proposed mechanism consistently achieves lower transmission duration as the number of providers increases, demonstrating excellent scalability. In contrast, the fixed and formula-based chunking methods show only marginal or unstable improvements, with transmission duration fluctuating and overall performance less stable. Fig. 11(b) compares the throughput of the four chunking transmission methods for a 1000 MB file at a bandwidth of 160 Mbps and a latency of 50 ms. The results reveal that our proposed mechanism maintains superior throughput in most cases, especially when there are three provider nodes, significantly outperforming the other baselines. Meanwhile, the fixed chunking method exhibits large fluctuations in throughput, indicating poor stability. Overall, the proposed mechanism proves more effective in adapting to environments with multiple provider nodes. It can more efficiently utilize available node resources, validating its superiority when node resources vary.

6.2.5. Stability analysis

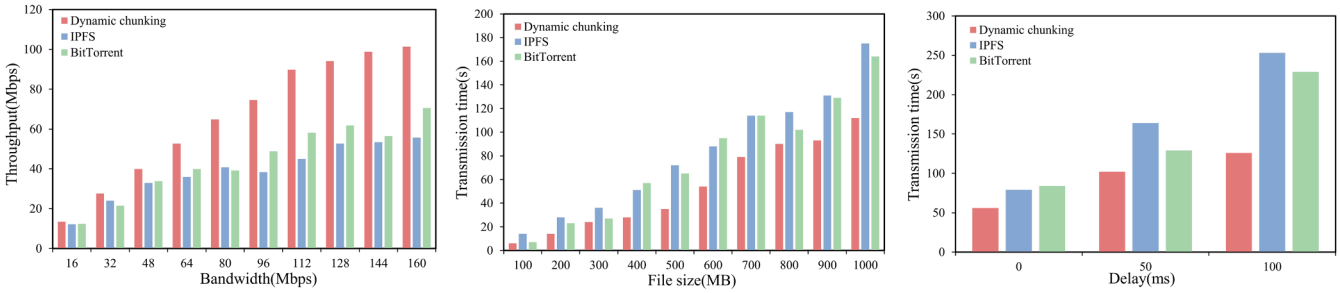
Beyond efficiency, transmission stability is a critical performance metric in distributed systems. To quantitatively assess the stability of our proposed mechanism, we conducted a series of independent trials under representative network conditions (80 Mbps bandwidth, 50 ms latency, and 500 MB data size), followed by a comprehensive statistical analysis. Fig. 12 presents box plots illustrating the distributions of transmission duration and throughput. As observed, the dynamic chunking method exhibits a significantly more compact interquartile range (IQR) compared to the baselines, indicating superior performance stability. Furthermore, Table 2 provides a detailed breakdown of the mean, standard deviation, and variance for both transmission duration and throughput. The data demonstrates that the proposed mechanism consistently outperforms the baselines across all metrics, confirming that the dynamic chunking approach maintains high transmission efficiency while achieving robust stability.

6.2.6. Impact of action space granularity

In the design of the DRL model, the step size within the action space dictates the granularity of chunk adjustments. To systematically validate the rationality of selecting 128 KB as the fixed step size, we conducted a series of parameter sensitivity analysis experiments. Under identical network conditions (100 Mbps bandwidth and 1000 MB data size), we selected three representative granularities within the chunk size range of [64, 1024] KB for comparative evaluation: fine-grained (64 KB), baseline (128 KB), and coarse-grained (256 KB). The fine-grained setting constructs a larger action space, offering higher control precision but incurring increased training costs. Conversely, the coarse-grained setting results in an action space with only four discrete actions, reducing computational overhead but yielding a relatively coarse control policy. Fig. 13 illustrates the performance in terms of transmission duration and throughput across various granularities under different latency conditions. It can be observed that transmission performance significantly degrades under the coarse-grained setting. In contrast, the experimental results for the fine-grained setting exhibit a distinct saturation effect, showing only marginal improvements compared to the baseline. However, the action space size doubles, which significantly increases the agent's computational complexity and convergence time. Therefore, the 128 KB step size strikes an optimal trade-off between transmission performance and training overhead, validating the rationality of our design choice.

6.3. Comparison with mainstream distributed protocols

While algorithmic comparisons have validated the theoretical advantages of the dynamic chunking strategy, it is equally critical to evaluate its performance gains relative to other transmission protocols within a distributed network environment. To this end, we selected two mainstream chunk-based distributed transmission protocols (IPFS [25] and BitTorrent [13]) as baselines. We conducted comparative evaluations against the proposed mechanism on our constructed distributed platform to verify the mechanism's effectiveness in practical applications. IPFS employs a rigid fixed-size chunking design, where data is presegmented into independent blocks of a predetermined size before entering the transmission layer. Similarly, BitTorrent, a dominant protocol for large-scale P2P data sharing, enforces a fixed chunk size determined during torrent generation, which remains immutable throughout the network. Fig. 14 illustrates the transmission performance of the three methods under varying conditions of bandwidth, file size, and latency. Fig. 14(a) reveals that as network bandwidth increases, the throughput growth of both IPFS and BitTorrent plateaus, exhibiting distinct saturation effects and failing to fully utilize available bandwidth resources. In contrast, the dynamic chunking transmission mechanism effectively enhances bandwidth utilization through real-time adjustments, yielding throughput performance significantly superior to both baselines. Fig. 14(b) shows that the proposed mechanism effectively adapts to varying data scales. By optimizing chunk sizes to boost efficiency, it achieves lower transmission durations than other methods across diverse file sizes. Fig. 14(c) further elucidates the impact of network latency. While traditional distributed protocols suffer severe performance degradation in high-latency scenarios, our proposed mechanism effectively mitigates the adverse effects of high latency through intelligent



(a) 500MB size and 50ms latency. (b) 160Mbps bandwidth and 50ms latency. (c) 1000MB size and 160Mbps bandwidth.

Fig. 14. Performance comparison of different transmission protocols.

decision-making. Overall, traditional distributed transmission protocols exhibit distinct limitations when confronting network fluctuations. Conversely, the proposed mechanism demonstrates superior adaptability to complex and dynamic distributed network environments through dynamic decision-making. It outperforms the baselines across all evaluated metrics, verifying both the effectiveness and robustness of the mechanism.

7. Conclusion

In this paper, we propose an intelligent and efficient transmission mechanism based on dynamic chunking for distributed environments. The proposed mechanism adopts a master-slave architecture to perceive real-time environmental states, dynamically adjusts data chunk sizes to improve transmission efficiency. Concurrently, it leverages DHT and P2P technologies to enable parallel transmission. In addition, to optimize chunking decisions, we innovatively introduce a DRL approach, modeling the chunking process as Markov Decision Process, where an agent interacts with the environment to learn optimal dynamic chunking policy. The dynamic chunking module aggregates network-wide node information to environment states in real time and applies DRL decisions by coordinating chunk size adjustments across nodes. Furthermore, we design a master node election algorithm, delayed experience feedback mechanism, and model migration mechanism to ensure system robustness and continuous optimization of chunking decisions. Extensive simulation experiments demonstrate that the proposed method offers significant advantages over conventional ones in key metrics such as transmission duration and throughput, and maintains high adaptability in dynamic and complex environments. Currently, the proposed mechanism primarily operates based on global environmental observations. However, given the complexity of real-world distributed systems, the heterogeneity of individual nodes and the stochastic nature of feedback signals are non-negligible factors. Therefore, future work will consider exploring fine-grained decision frameworks to better adapt to diverse node characteristics. Additionally, we will consider investigating robust learning techniques, such as PSSCL [42], to mitigate the potential impact of measurement noise and further enhance system stability.

CRediT authorship contribution statement

Enliang Lv: Writing – review & editing, Writing – original draft, Software, Project administration, Formal analysis, Data curation, Conceptualization; **Xingwei Wang:** Writing – review & editing, Resources; **Bo Yi:** Writing – review & editing; **Hao Lu:** Writing – review & editing; **Min Huang:** Writing – review & editing; **Yue Kou:** Writing – review & editing; **Keqin Li:** Writing – review & editing.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 62432003, U25A20431.

References

[1] T.V. Doan, R. van Rijswijk-Deij, O. Hohlfeld, V. Bajpai, An empirical view on consolidation of the web, ACM Trans. Internet Technol. (TOIT) 22 (3) (2022) 1–30.
[2] C.B. de Leusse, C. Gahnberg, The global internet report: consolidation in the internet economy, Internet Soc. (2019).
[3] J. Shi, R. Rao, Y. Song, X. Wang, B. Yi, Q. He, C. Zeng, M. Huang, S.K. Das, Service recommendation in JointCloud environments: an efficient regret theory-based QoS-aware approach, Comput. Netw. 254 (2024) 110716.
[4] M.R. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel, Amazon S3 for science grids: a viable solution?, in: Proceedings of the 2008 International Workshop on Data-aware Distributed Computing, 2008, pp. 55–64.
[5] S.J. Chen, Z. Qin, Z. Wilson, B. Calaci, M. Rose, R. Evans, S. Abraham, D. Metzler, S. Tata, M. Colagrosso, Improving recommendation quality in google drive, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 2900–2908.
[6] L. Caviglione, M. Podolski, W. Mazurczyk, M. Ianigro, Covert channels in personal cloud storage services: the case of dropbox, IEEE Trans. Ind. Inf. 13 (4) (2016) 1921–1931.
[7] D. Coldewey, Cloudflare DNS goes down, taking a large piece of the internet with it, Retrieved May 26 (2020) 2023.
[8] M. Rosemain, R. Satter, Millions of websites offline after fire at French cloud services firm, Reuters (2021).
[9] J. Shi, B. Yi, X. Wang, M. Huang, Y. Song, Q. He, C. Zeng, K. Li, JointCloud resource market competition: a game-theoretic approach, IEEE/ACM Trans. Netw. 32 (6) (2024) 5112–5127.
[10] P. Sun, S. Shen, Y. Wan, Z. Wu, Z. Fang, X.-z. Gao, A survey of iot privacy security: architecture, technology, challenges, and trends, IEEE Internet Things J. 11 (21) (2024) 34567–34591.
[11] Y. Liu, J. He, X. Li, J. Chen, X. Liu, S. Peng, H. Cao, Y. Wang, An overview of blockchain smart contract execution mechanism, J. Ind. Inform. Integr. 41 (2024) 100674.
[12] E. Daniel, F. Tschorsch, IPFS and friends: a qualitative comparison of next generation peer-to-peer data networks, IEEE Commun. Surv. Tutorials 24 (1) (2022) 31–52.
[13] A. Mazri, M. Mehdi, Unraveling decentralized data storage: a comparative analysis of IPFS and BitTorrent networks, in: 2024 2nd International Conference on Electrical Engineering and Automatic Control (ICEEAC), IEEE, 2024, pp. 1–6.
[14] N. Jagadish Kumar, D. Dhinakaran, A. Naresh Kumar, A.V. Kalpana, Swarm intelligence with a chaotic leader and a salp algorithm: HDFS optimization for reduced latency and enhanced availability, Concurrency Comput. Pract. Exper. 36 (17) (2024) e8127.
[15] F.K. Parast, S.A. Damghani, B. Kelly, Y. Wang, K.B. Kent, Efficient security interface for high-performance Ceph storage systems, Future Gener. Comput. Syst. 164 (2025) 107571.
[16] C. Manthiramoorthy, K.M.S. Khan, et al., Comparing several encrypted cloud storage platforms, Int. J. Math. Stat. Comput. Sci. 2 (2024) 44–62.
[17] P. Qin, T. Zhao, Knowledge guided deep deterministic policy gradient, Knowl. Based Syst. 311 (2025) 113087.
[18] J.C. Priya, G. Nanthakumar, T. Choudhury, K. Karthika, 6G-DeFLI: enhanced quality-of-services using distributed hash table and blockchain-enabled federated learning approach in 6G IoT networks, Wireless Netw. 31 (1) (2025) 361–375.

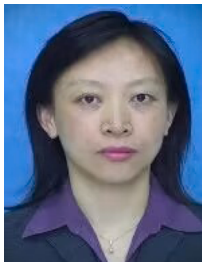
[19] V.C. Manduva, Review of P2P computing system cooperative scheduling mechanisms, Int. J. Modern Comput. 7 (1) (2024) 154–168.
[20] C. Karapapas, G. Xylomenos, G.C. Polyzos, Enhancing IPFS bitswap, in: International Conference on Information and Communication Technology for Intelligent Systems, Springer, 2024, pp. 189–199.
[21] J. Shen, Y. Li, Y. Zhou, X. Wang, Understanding I/O performance of IPFS storage: a client's perspective, in: Proceedings of the International Symposium on Quality of Service, 2019, pp. 1–10.
[22] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, ACM SIGCOMM Comput. Commun. Rev. 31 (4) (2001) 149–160.
[23] Y. Wei, D. Trautwein, Y. Psaras, I. Castro, W. Scott, A. Raman, G. Tyson, The eternal tussle: exploring the role of centralization in IPFS, in: 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), 2024, pp. 441–454.
[24] S.M. Almufti, S.R.M. Zeebaree, Leveraging distributed systems for fault-tolerant cloud computing: a review of strategies and frameworks, Acad. J. Nawroz Univ. 13 (2) (2024) 9–29.
[25] D. Trautwein, Y. Wei, Y. Psaras, M. Schubotz, I. Castro, B. Gipp, G. Tyson, IPFS in the fast lane: accelerating record storage with optimistic provide, in: IEEE INFOCOM 2024-IEEE Conference on Computer Communications, IEEE, 2024, pp. 1920–1929.
[26] R.S. Patil, G.S. Biradar, S. Terdal, Blockchain-integrated optimized cryptographic framework for securing cloud data, Knowl. Based Syst. (2025) 113830.
[27] S. Khatal, J. Rane, D. Patel, P. Patel, Y. Busnel, Fileshare: a blockchain and IPFS framework for secure file sharing and data provenance, in: Advances in Machine Learning and Computational Intelligence: Proceedings of ICMLCI 2019, Springer, 2021, pp. 825–833.
[28] S. Williams, V. Diordiiev, L. Berman, I. Uemlianin, Arweave: a protocol for economically sustainable information permanence, Arweave Yellow Paper (2019).
[29] P. Cao, Y. Chen, K. Liu, J. Zhao, S. Liu, W. Chong, HyperCore: hyperbolic and co-graph representation for automatic ICD coding, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 3105–3114.
[30] F. Wang, S. Zhang, E.-K. Hong, T.Q.S. Quek, Constellation as a service: tailored connectivity management in direct-satellite-to-device networks, (2025), arXiv:2507.00902
[31] J. Li, G. Sun, L. Duan, Q. Wu, Multi-objective optimization for UAV swarm-assisted IoT with virtual antenna arrays, IEEE Trans. Mob. Comput. 23 (5) (2023) 4890–4907.
[32] A.R. Naik, B.N. Keshavamurthy, Next level peer-to-peer overlay networks under high churns: a survey, Peer-to-Peer Netw. Appl. 13 (3) (2020) 905–931.
[33] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, Y. Psaras, Design and evaluation of IPFS: a storage layer for the decentralized web, in: Proceedings of the ACM SIGCOMM 2022 Conference, 2022, pp. 739–752.
[34] M.I. Khalid, I. Ehsan, A.K. Al-Ani, J. Iqbal, S. Hussain, S.S. Ullah, et al., A comprehensive survey on blockchain-based decentralized storage networks, IEEE Access 11 (2023) 10995–11015.
[35] C. Yang, J. Chen, A scalable data chunk similarity based compression approach for efficient big sensing data processing on cloud, IEEE Trans. Knowl. Data Eng. 29 (6) (2016) 1144–1157.
[36] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: 2014 USENIX Annual Technical Conference (USENIX ATC 14), 2014, pp. 305–319.
[37] H. Howard, R. Mortier, Paxos vs Raft: have we reached consensus on distributed consensus?, in: Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data, 2020, pp. 1–9.
[38] C. Roy, A. Mukherjee, N. Chaki, Merkle DAG-based distributed data model for content-addressed trust-less verifiable data, in: 2022 7th International Conference on Computer Science and Engineering (UBMK), IEEE, 2022, pp. 462–467.
[39] H. Kurniawati, Partially observable markov decision processes and robotics, Annu. Rev. Control, Rob. Auton. Syst. 5 (1) (2022) 253–277.
[40] R. Fotohi, F.S. Aliee, Securing communication between things using blockchain technology based on authentication and SHA-256 to improving scalability in large-scale IoT, Comput. Netw. 197 (2021) 108331.
[41] E. Lv, X. Wang, B. Yi, L. Qiu, J. Shi, J. Guo, K. Zhang, P. Li, Towards efficient collaborative data transmission in JointCloud: a dynamic chunking mechanism, in: EAI International Conference on Collaborative Computing: Networking, Applications and Worksharing, Springer, 2024, pp. 298–308.
[42] Q. Zhang, Y. Zhu, F.R. Cordeiro, Q. Chen, PSSCL: a progressive sample selection framework with contrastive loss designed for noisy labels, Pattern Recognit. 161 (2025) 111284.

Enliang Lv received the BS and MS degrees in Computer Science and Technology from Northeastern University, China, in 2023 and 2025 respectively. He is currently working toward the PhD degree in Computer Science and Technology with the College of Computer Science and Engineering, Northeastern University, China. His main research interests include jointcloud computing and distributed transmission.

Xingwei Wang received the BS, MS, and PhD degrees in computer science from Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively. He is currently a Professor with the College of Computer Science and Engineering, Northeastern University. His research interests include cloud computing and future Internet. He has published more than 100 journal articles, books, and refereed conference papers, and has received several best paper awards.

Bo Yi (Member, IEEE) is currently an Associate Professor of Computer Science and Engineering with Northeastern University, China. He has authored and co-authored more than 20 journal and conference articles on IEEE Transactions on Cloud Computing, IEEE Communications Letters, etc. He is also a reviewer for IEEE Communications Surveys & Tutorials, Computer Networks, Journal of Network and Computer Applications, etc. His research interests include service computing, virtualization, and cloud computing in SDN, NFV, and DetNet.

Hao Lu received the BS degree in Computer Science and Technology from Henan Polytechnic University, China, in 2022. He is currently working toward the MS degree in Computer Science and Engineering at Northeastern University, China. His current research interests include cloud computing, mechanism design, multi-objective optimization, and jointcloud.



Min Huang received the BS degree in automatic instrumentation, the MS degree in systems engineering, and the PhD degree in control theory from Northeastern University, Shenyang, China, in 1990, 1993, and 1999, respectively. She is currently a Professor with the College of Information Science and Engineering, Northeastern University. Her research interests include modeling and optimization for logistics and supply chain systems. She has published more than 100 journal articles, books, and refereed conference papers.

Yue Kou received the PhD degree in computer software and theory from Northeastern University, Shenyang, China, in 2009. She is currently an Associate Professor with the School of Computer Science and Engineering, Northeastern University. Her research interests include social network analysis and mining, recommender systems, and web data management.

Keqin Li (Fellow, IEEE) is a SUNY Distinguished Professor of Computer Science at the State University of New York. He is also a National Distinguished Professor with Hunan University, China. He has chaired many international conferences and is currently an Associate Editor of ACM Computing Surveys and CCF Transactions on High Performance Computing. He is an AAAS Fellow, an IEEE Fellow, and an AAIA Fellow, and a Member of Academia Europaea.