applied sciences

MDPI

Article

Design and Optimization of a Distributed File System Based on RDMA

Qinlu He [1,*], Pengze Gao [1], Fan Zhang [1], Genqing Bian [1], Weiqi Zhang [1] and Zhen Li [2,*]

[1] School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an 710054, China; zhangweiqi@163.com (W.Z.)
[2] Shaan Xi Institute of Metrology Science, Xi'an 710043, China
* Correspondence: luluhe8848@hotmail.com (Q.H.); lizhen@163.com (Z.L.)

Abstract: With the rapid development of computer and network technology, to deal with explosively growing mass data correctly, distributed file systems have come into being, and they have been applied in many enterprises. However, in the packet storage process, the traditional TCP/IP transmission protocol has to pass through the operating system and application software layer. It takes up many server resources and memory bus bandwidth, which places a heavy burden on the server's CPU and memory. In this paper, we analyze the research status of the distributed file system based on analyzing and comparing the advantages of RDMA over traditional TCP Ethernet communication architecture. We propose an optimization scheme for the network transmission performance of the original FastDFS to replace the original TCP module with an RDMA network communication module, and we design a lightweight distributed file system with higher performance than the original FastDFS, which is suitable for small file storage. Using the zero-copy technology and kernel bypass technology of soft RDMA, the network communication mechanism in FastDFS is optimized to free the server's CPU from heavy data processing work. In this paper, the RDMA-based FastDFS is compared with the original FastDFS. The test results show that the network transmission performance of the optimized FastDFS is improved compared with the original FastDFS, which achieves the expected effect and can better meet the requirements of mass small file storage for today's Internet applications.

Keywords: DFS; RDMA; kernel-bypass; zero-copy

Citation: He, Q.; Gao, P.; Zhang, F.; Bian, G.; Zhang, W.; Li, Z. Design and Optimization of a Distributed File System Based on RDMA. *Appl. Sci.* **2023**, *13*, 8670. https://doi.org/10.3390/app13158670

Academic Editor: Dimitris Mourtzis

Received: 7 June 2023
Revised: 22 July 2023
Accepted: 25 July 2023
Published: 27 July 2023

1. Introduction

The birth of the distributed file system has been a milestone in file storage, effectively solving the problem of massive data management [1]. The distributed file system has many advantages such as cost savings, convenient management, good scalability, and strong reliability [2]. However, with continuous changes in the Internet environment, the functions and performance of the distributed file system in various application scenarios are also changing with each passing day.

With the development of modern information science and technology, the era of big data has already quietly begun. Statistics show that my country's Internet penetration rate has reached 57.7%. The utilization rates of social applications such as WeChat Moments and Qzone are as high as 86.9% and 64.7%, respectively; 74.1% of netizens use short video applications such as Douyin and Kuaishou; mobile online shopping users have increased by 10.2% compared to the number of users at the end of 2017 [3]. When netizens use social applications, share short videos, or shop online, the amount of data generated in small files is vast, and it continues to grow rapidly. The Internet produces files such as pictures, texts, and short videos, which generates massive amounts of small file data. To better store and manage this massive data, it is necessary to design and develop a high-performance distributed file system suitable for small files, which is an urgent problem to be solved in today's Internet environment [4].

In the transmission of large files, network transmission delay is the main factor of the entire file transmission delay. On the contrary, in the new set of small files, the processing overhead of the file sending and receiving ends dominates the file transmission delay [5]. Specifically, processing overhead refers to buffer management, message replication in different memory spaces, and system interruption after message transmission is completed. In today's communication scenarios of the Internet, the information of small files such as messages, pictures, and short videos accounts for the vast majority of file transmissions. Therefore, most of the communication delay in the current distributed file system comes from the processing overhead of the message sender and receiver.

The nodes of the existing distributed file system are mainly interconnected through Ethernet. In the traditional Ethernet communication method, multiple memory copies, interrupt handling, context switching, complex TCP/IP protocol processing, store-and-forward modes, and packet loss all lead to additional delays, resulting in large transmission delays and low system performance.

Therefore, in the face of high-performance computing, big data analysis and surge I/O high concurrency, low latency applications, the existing TCP/IP hardware, software architecture, and high CPU consumption communication technology can no longer meet the needs of emerging applications. Under high-speed network conditions, the high overhead of host processing related to network I/O not only limits the bandwidth between network nodes but also reduces the performance and flexibility of network communication. Specifically, in the traditional TCP/IP protocol, the reason is mainly that message transmission needs to be passed through the kernel. This communication method causes high data movement and data replication overhead. At the same time, compared with the continuous improvement of processor speed, the development of the traditional Ethernet field is slightly lagging. It has gradually become a shortcoming of distributed file system performance [6]. Furthermore, it is difficult to support new network protocols and sending and receiving interfaces, which also have poor flexibility [7].

For this problem, RDMA network transport is a suitable solution. It reduces computing resource consumption while guaranteeing extremely low transmission latency. However, most of the current optimization work of RDMA for distributed file systems only uses RDMA libraries or plug-ins as communication submodules of the system, and does not make full use of the excellent features of RDMA [8–27]. Accelio, the RDMA middleware used in CephFS, encapsulates the underlying RDMA network functions and only provides an abstract call interface [28,29]. In this way, the application cannot customize RDMA network transmission according to the characteristics of its own network transmission. The work of N.S. Islam et al. for HDFS was also carried out by supporting RDMA network modules; it has been difficult to further improve performance through the rich features of RDMA and full use of high-performance network resources has not been achieved. GlusterFS's strategy is to apply RDMA libraries instead of TCP/IP to complete network transmission, but there is no customization of transmission schemes according to different application scenarios, nor is it supported by a more effective memory model [30].

In summary, in these optimized designs, the network transport layer of the distributed file system is basically strictly isolated from the logical processing layer. In addition, they do not effectively integrate and adapt with the data processing mechanism of the file system according to the characteristics of RDMA transmission, and the software and hardware advantages of RDMA transmission cannot be fully utilized by replacing the data transmission function in the system. In the case of GlusterFS, software latency accounts for about 1–3% of total latency in disk media file systems, and close to 100% in memory file systems with stronger storage hardware [31]. Obviously, after the storage performance and network performance are improved, the isolation of the file system software layer and the network transport layer makes the distributed system too complex, making the software part of the system a bottleneck and limiting the performance of hardware devices.

In view of the objective requirements of distributed systems' RDMA network transmission in a high-performance hardware environment and the inability of current research

work to effectively coordinate the system software layer and network layer to give full play to the performance of RDMA, in this paper, we propose a lightweight distributed file system based on RDMA technology that focuses on the RDMA communication module of FastDFS and proceeds from the communication process. The system optimizes the communication process to maximize the advantages of RDMA communication technology, improves the performance of the original FastDFS distributed file system, and achieves the purpose of improving system bandwidth, reducing latency, and reducing server CPU utilization.

2. Background and Motivation

2.1. Comparison of RDMA Technology and TCP

Derived from previous studies [9–12,32–34], one of the primary reasons affecting the TCP transmission rate is that data are copied back and forth between the kernel and user space, significantly increasing CPU usage. Therefore, the critical problem in improving the performance is eliminating cumbersome data in the host CPU. Copy and move operations reduce CPU usage in the communication process, and RDMA uses these two methods to optimize communication performance. It is a technology that can directly access the memory of a remote host. There are two types of operations in RDMA: RDMA Read and RDMA Write. Whether RDMA Read or RDMA Write, the requester only needs to obtain the memory address of the accessed remote node in advance, register the target memory, obtain the memory key of the target memory, and then perform kernel bypass data access.

In addition to the studies mentioned above, there are other studies [13–18] that have also compared TCP with RDMA technology. It can also be seen from the research conclusions of these documents that the RDMA protocol significantly reduces the number of memory copies due to its zero-copy and kernel bypass characteristics. Compared with the TCP, it has obvious advantages and can achieve higher performance. For example, a 40 Gbps TCP stream consumes the CPU resources of mainstream servers. RDMA addresses the technical pain points of traditional TCP/IP communication: In the 40 Gbps scenario, the CPU usage is reduced from 100% to 5%, and the network latency is reduced from *ms* level to less than 10 *us* [19].

Analyzing the TCP architecture, when an application wants to use TCP/IP, it must use the programming interface socket provided by the protocol to realize network communication between applications or processes. Commonly used TCP/IP protocols for network communication programs use sockets for programming. Whether you write a client-side or server-side program, the system creates a socket handle for each TCP connection. This results in each transmission of traffic, through the OS and protocol stack management, so whether it is synchronous communication or asynchronous communication socket, there will be a phenomenon of high CPU consumption.

In the RDMA model, the core problem is how to realize the simplest, more efficient, and direct communication between applications. RDMA provides peer-to-peer communication based on message queues. As long as it is registered in advance, each application can directly obtain its messages from the remote end. Figure 1 shows the architecture of the RDMA technology and its comparison with the TCP/IP architecture.

The thin blue line in the figure represents the data flow in the TCP, and the thick red line represents the data flow in the RDMA communication. It is clear from the figure that RDMA saves a lot of data copying trouble compared to TCP, and the data are directly transmitted from the application buffer to the application buffer of another node through the host adapter HCA without passing through the kernel space of both parties.

RDMA uses core bypass and zero-copy technology to provide low-latency features while reducing CPU usage, breaking through the memory bandwidth bottleneck, and providing high bandwidth utilization. This benefits from how the RDMA message service is presented to the application and the underlying technology used to send and deliver these messages [31]. Compared with a TCP/IP network, RDMA technology also eliminates the retransmission delay caused by data packet loss. There is no need to confirm and resend the lost data packet as in the TCP/IP network, saving this part of the time overhead, so

that the overall performance is improved in this respect, and the transmission efficiency is also improved.

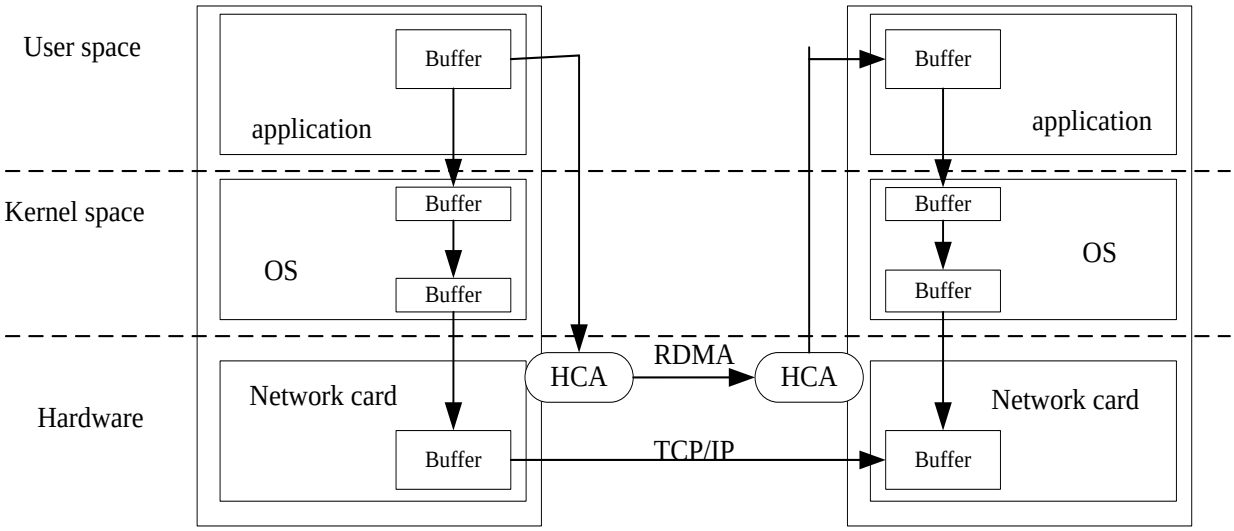

Figure 1. The difference between RDMA and TCP.

In contrast to traditional DMA's internal bus I/O, RDMA can remotely transfer data directly between application software at two endpoints over a network; Compared with traditional network transmission, RDMA can easily achieve ultra-low latency and ultra-high throughput transmission between remote nodes without the intervention of the operating system and protocol stack, and basically does not require CPU resource participation, and in a high-speed network environment, it is no longer necessary to consume too many system resources for the processing and repeated copying of network data. In short, these three characteristics of RDMA, i.e., CPU offloading, kernel bypass, and zero copy [32], make its advantages self-evident, and the transmission performance it can achieve far exceeds that of the traditional TCP, which, indeed, become a major tool for improving distributed file systems.

In summary, using RDMA communication instead of the original TCP can avoid the time delay caused by frequent data copying, thereby improving the performance of the distributed file system.

2.2. Asymmetry of RDMA Unilateral Communication

From the perspective of transport principle, RDMA unilateral operations are completely different from the requesting side of the initiator and the requesting side of the response. Here, we further divide the unilateral RDMA into two types, one is active operation, and the other is passive operation.

For a distributed file system with C/S architecture, if you want to achieve data transmission between the server and a client through unilateral RDMA communication, there are two designs and modes. One is adopted by the server active RDMA operation, which transmits data to the client by initiating a written request to the client, or forms a read operation to read data from the client; the other server only uses passive RDMA operation, that is, the client initiates a read request to read data from the server, or the client initiates a written request to write the data into the server memory.

However, the resources and latency consumed by initiating an RDMA operation proactively and the corresponding RDMA operation passively should be different for the server. The inference is drawn by analyzing the operating principle and process of unilateral RDMA. Due to the particularity of RDMA communication, some data processing-related functions of RDMA are directly integrated into the RNIC hardware without going through the system and application programs. Analyzed from the one-sided RDMA

transmission process, the work of the passive RDMA (i.e., receive/respond operation) is all handled by the hardware of the RNIC. In contrast, the other party's active RDMA operation (i.e., making/initiating a request) requires interaction between hardware and software to ensure that the operation is sent and completed. Therefore, under the same conditions, processing a passive RDMA operation is much less expensive than initiating an active RDMA operation.

It is clear, as shown in from Figure 2, from the experimental results that the server's active RDMA IOPS reached saturation quickly after reaching 4 threads, while passive RDMA IOPS was at a very high level, with 16 threads failing to saturate it. This validates the previous view that often, RDMA active operations are the bottleneck of RDMA performance, and passive RDMA operations are far from saturated when active RDMA processing operations reach saturation. This shows that under the same conditions, RDMA can handle many times more passive requests than active operations in a certain period of time, and the general use cases do not give full play to the passive processing power of RDMA. This is also the asymmetry of RDMA unilateral operation.

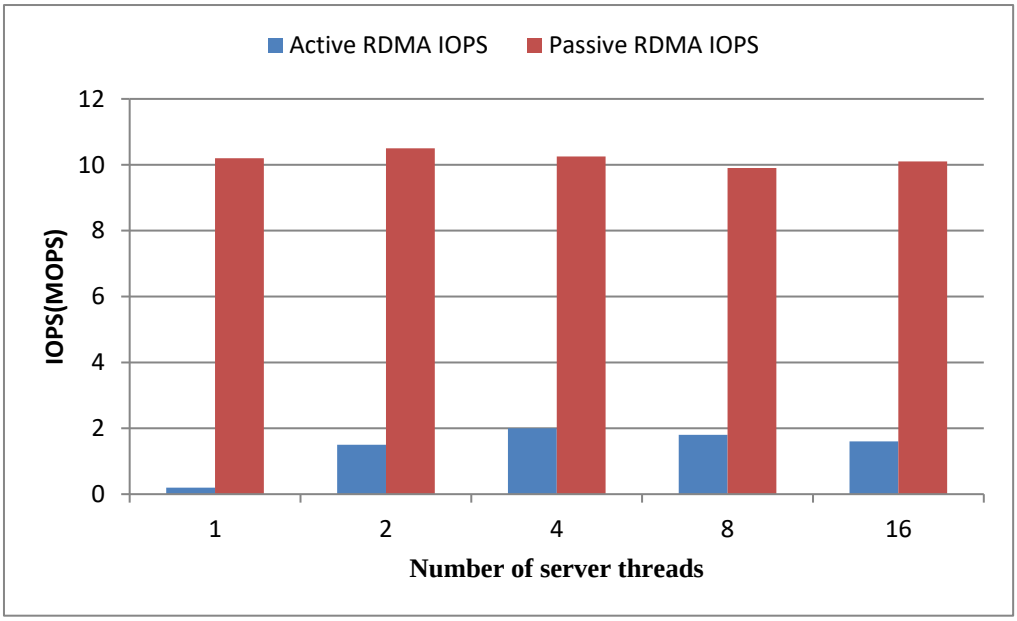

Figure 2. Server IOPS under different thread counts.

When the local RNIC sends an RDMA request to the remote RNIC, the local RNIC needs to complete the following operations in sequence: prepare the request content and communication status in the hardware, send the request, wait for the ACK confirmation message from the remote RNIC, and when the request is completed, generate complete event information [33]. These operations require the local INIC to maintain many states in the hardware, which significantly limits the ability of the local RNIC to perform more RDMA operations.

However, for passive RDMA Read, the RNIC in the server machine has almost no processing burden and only needs to respond to requests sent from the remote RNIC. In this case, RNIC does not need to maintain many states or perform additional operations on the received request. The capabilities of RNIC can be fully utilized, resulting in higher IOPS.

3. Design of a Lightweight Distributed File System Based on RDMA Communication Mode

Based on the original FastDFS, the original TCP module is replaced by the RDMA network communication module. The features of RDMA technology are zero copy and kernel bypass, and a lightweight distributed file system suitable for small file storage is designed with higher performance than the original FastDFS.

3.1. Design of a Lightweight Distributed File System Based on RDMA Communication Mode

Based on the analysis in Section 2, RDMA communication technology has many advantages over the traditional TCP/IP communication. If RDMA technology is used in a distributed file system to replace the original TCP/IP, theoretically, the distributed file system performance can significantly improve.

Based on the discussion in the previous sections, we chose FastDFS as the basis for optimization. In this section, we mainly present a design based on the original FastDFS, allowing RDMA communication technology to be integrated into the system to improve the performance of the original FastDFS.

(1) Architecture of the Lightweight Distributed File System

The overall architecture of the system is shown in Figure 3.

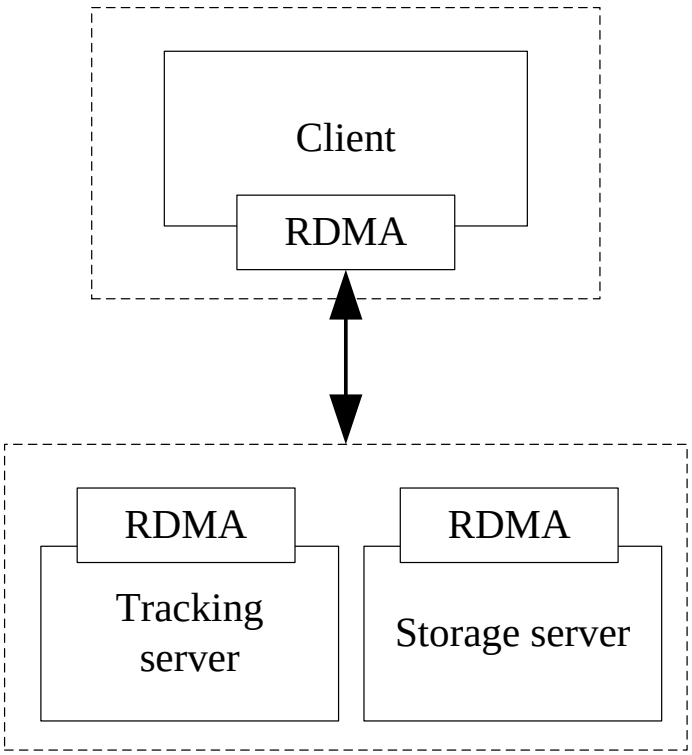

Figure 3. Overall system architecture.

The overall system architecture is designed based on FastDFS, as shown in Figure 3. Similar to the structure of traditional FastDFS, the system is mainly divided into a Tracking Server tracker, Storage Server storage, RDMA communication module, and Client module. The Tracking Server is the core module of the entire system, managing all the Storage Servers. When the system is started, the Storage Server will periodically send heartbeat data containing its status information to the Tracking Server, stored by the Tracking Server. The Tracking Server is responsible for processing various requests from the Client, queries the appropriate Storage Server according to the request, and sends the relevant information (IP and port number) of the depositor server back to the Client, which plays a load-balancing role. The Storage Server provides file-related services, and the Client communicates directly with the Storage Server to complete access to the file. The RDMA communication module is embedded in the Storage Server, Tracking Server, and Client, and is responsible for communicating data in RDMA mode. The system network communication topology diagram is shown in Figure 4.

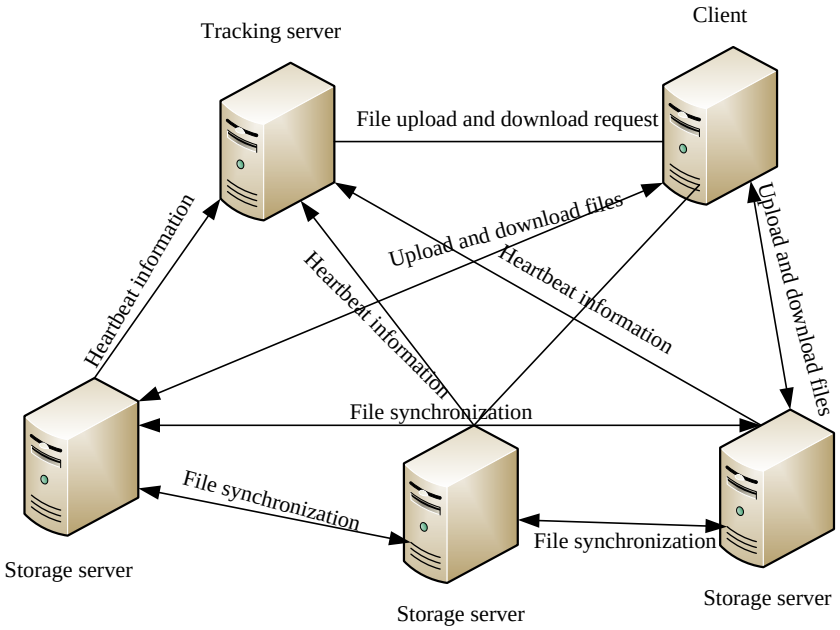

Figure 4. Network communication topology diagram.

(2) Functions of Lightweight Distributed File System

The server of the system is mainly divided into three modules: Tracking Server, Storage Server, and RDMA communication module. The Tracking Server, as the load balancing and scheduling role of the system, is mainly responsible for processing the requests sent by the Client, making decisions about the most appropriate storage for the Client, and conveying communication interface information. Another function of the Tracking Server is to manage the storage group. When a specific storage disk space alarms, it closes the storage write permission; when a particular storage is abnormal or fails, it sends alarm information to the entire system. The Storage Server module provides primary file services and reports real-time status information such as its synchronization status, CPU usage, remaining storage space, and memory usage to the Tracking Server. The RDMA communication module shields the original TCP/IP and through RDMA verb interface programming, completes the communication between the Client, Storage Server, and Tracking Server.

1. File upload

File upload is one of the main functions of the system. First, the Client initiates a file upload request, and the Tracking Server receives the file upload request and selects a Storage Server with a lower load to return. The Client, in turn, tries to communicate with storage in the provided Storage Server list through RDMA and writes the file to be uploaded directly into the memory registered in storage in advance to complete the file upload. Subsequently, when the corresponding CQE in the Storage Server's completion queue is consumed, the Storage Server is notified that the upload has completed. After the Storage Server receives the message, it will synchronize the file with each Storage Server in the same group through the synchronization thread and directly write its memory area through RDMA.

2. File download

The file download process is similar to the file upload process. The Client sends the file FID to the Tracking Server, and the Tracking Server parses the FID to determine whether the file exists. When the file does not exist, it directly returns the file's non-existence information to the client. If the file exists, it will correspond to the Storage Server. The IP and port information is returned. The Client connects with the corresponding Storage Server and downloads files through RDMA.

3. File synchronization

File syncing comes into play after a file upload or update ends. When the Client uploads a file to a Storage Server, Storage sends the file to Storage Servers in the same group through the synchronization thread, using incremental synchronization. The Storage Server storing the source file obtains the related information of other Storage Servers in the group through shared memory, such as IP addresses and ports. In addition, file synchronization is also required when there are new servers in the Storage group or when a new Storage group is added to the system. At this time, the Tracker is responsible for receiving the status information of all Storage (including new members of Storage), updating the locally stored system grouping information and the status information of each server, and pushing the new information to the storage group and keeping it in shared memory for synchronization files between Storage Servers. In the same group, there will be storage that takes the role of active synchronization. According to the list information provided by the Tracker, it will communicate with each storage that needs to be synchronized, register the memory, and gradually synchronize the files to its memory.

4. File deletion

File deletion is also a function in the system. When deleting a file, the Client will first send a file deletion request to the tracker, and the tracker will locate the directory where the file is located according to the FID contained in the parsed file deletion request. If the file is not in the manual, the tracker will send a file directly to the Client code name that does not exist. If the file is found in the manual, the Tracker will obtain the IP and port of the corresponding Storage Server and send a file deletion request directly to the corresponding Storage Server.

(3) Module design of a lightweight distributed file system

The system is divided into the following main modules: RDMA network communication, Tracking Server, and Storage Server. Among them, the RDMA network communication module is the core of communication, mainly by shielding the original TCP communication mode and changing the communication mode to RoCE to improve the performance of the entire system. The storage entity of the system is the Storage Server, and the Tracking Server is responsible for scheduling and load balancing the whole system. The implementation and design details of each module are described in detail in the following submodules:

1. Tracker module design

The Tracker retains most of the functions of the original FastDFS, such as load balancing, processing, and server reporting status. The Tracker stores the grouping information of storage in the system through a linked list. The SQL Server functions are encapsulated in the Tracker, and basic operations such as inserting, deleting, and querying the SQL Server are realized. At the same time, the interface of the RDMA network communication module is reserved to facilitate the connection of the RDMA communication module. The Tracker stores log information in log files, such as request records, message queues, and storage query information.

2. Storage module design

Storage adopts a dual-process mode, and the process that interacts with the Tracker is called the S–T process. There are two main types of data interaction, one type is to report its heartbeat status information to the Tracker, and the other type is to obtain the listening port and IP address of other storage required from the Tracker, or the current grouping status of the system. Another thread is mainly responsible for data interaction with the Client, called the S–C process. When the Client initiates a request, it needs to use this thread to respond to the Client's request and upload or download file data transmission. In addition, the S–C process will communicate with other storage in the same group, especially when files need to be synchronized.

3. RDMA communication module

The RDMA communication module is the core communication and underlying message management module. By changing the file, the RDMA module can cover the original TCP and UDP communication methods and can assume the communication function of

the entire system. During the initialization process, the physical memory is preregistered through the *ibv_reg_mr()* function, that is, the memory management is handed over to the RDMA module. The process of this module when transmitting data is as follows: First, the Client module initiates a data request, which requires the bottom layer to start an RDMA Tracking server. At the same time, the Tracker will let the remote receiving node start an RDMA Client. A queue pair and a completion queue are established through the *ibv_create_cq()* and *ibv_create_pq()* functions, complete initialization, and a connection is established. The relationship needs to be installed before the client communicates with the storage. After the connection is established, the Client and storage can transfer files. Once the data transfer is complete, you can notify the end of the data transfer and consume the corresponding elements in the completion queue.

3.2. FastDFS Communication Process Optimization Scheme Based on RDMA

In the discussion of the asymmetry of unilateral RDMA communication, it can be found that the server's active RDMA IOPS has become the server's bottleneck, its passive RDMA operation is far from saturated, and the resource utilization is not high. Therefore, if you change its communication flow and change the active RDMA operation on the server side to a passive RDMA operation (that is, the Client initiates the RDMA request), you can further improve the performance of the distributed file system proposed in Section 3.1. In this section, we detail the main process of optimized file upload and file download, and propose and solve problems that may arise from other modification processes.

(1) Main process logic

The following describes the optimized FastDFS communication process, in which the critical active RDMA operation steps on the server side are changed to passive.

1. Optimized upload process

As shown in Figure 5, the optimized file upload process is as follows (the circle marked by the red line is the modified process).

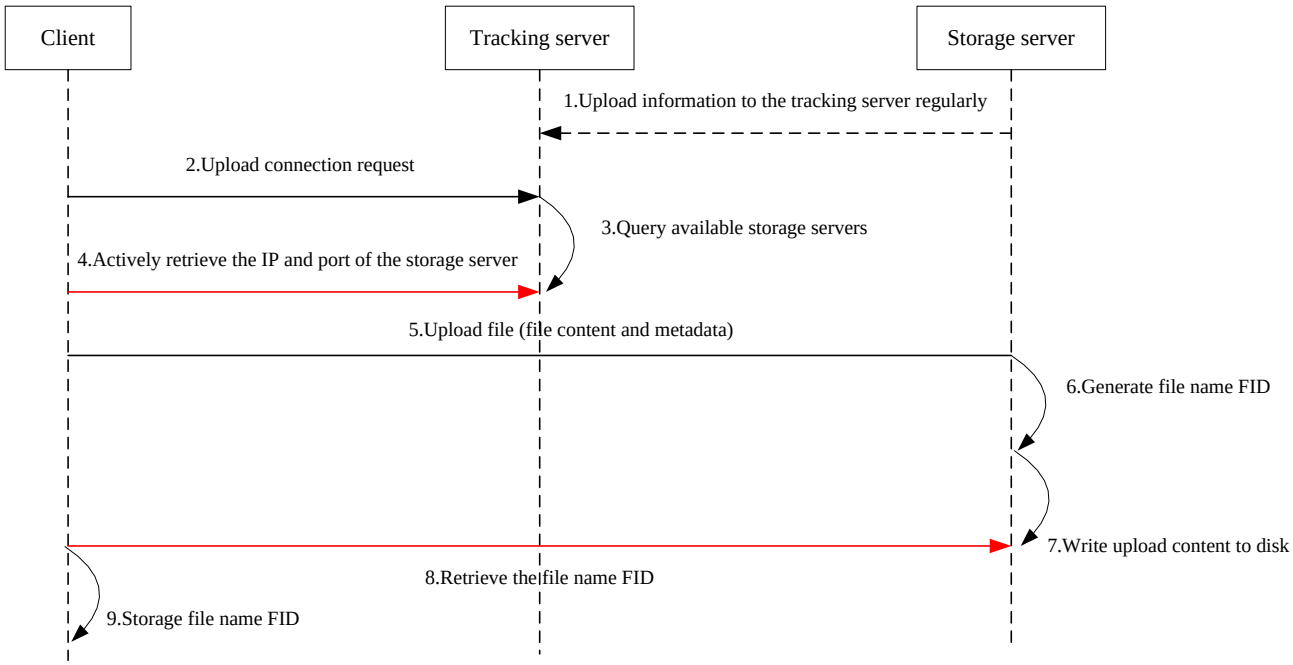

Figure 5. The optimized file upload process.

2. Optimized download process

Similar to the upload, Figure 6 illustrates the optimized download process.

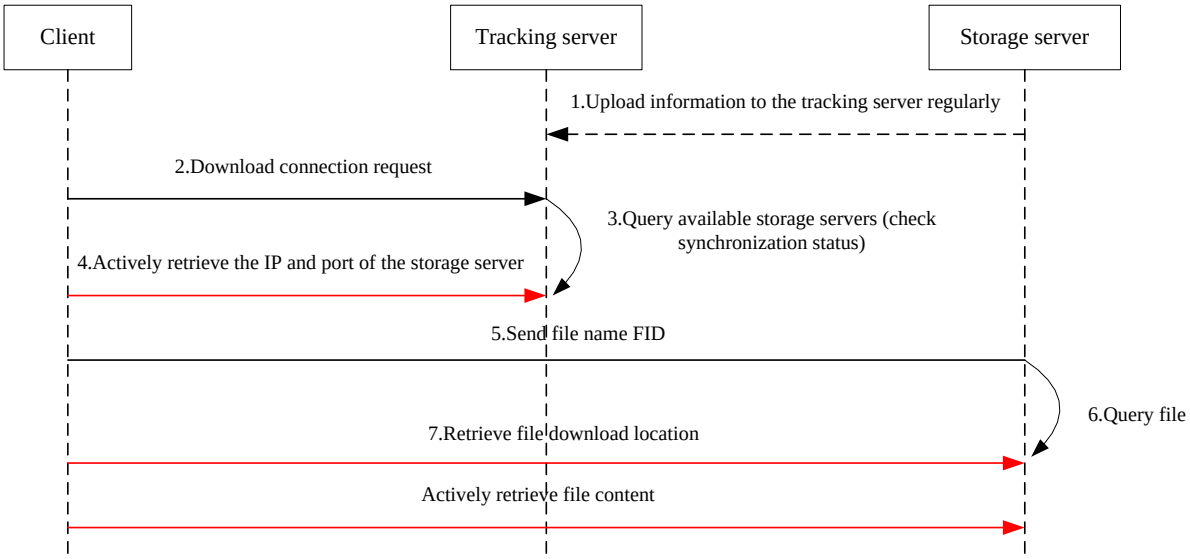

Figure 6. Optimized download process.

(2) Determination of relevant parameters

In the design described above, a critical problem needs to be solved. Since the client needs to initiate an active RDMA operation with the server to obtain the result after starting the upload request or download request, the server does not actively respond. At this time, the client will face two problems:

(1) Determining when to obtain results from the server;
(2) Determining the size of the result obtained from the server at a time.

For the first problem, the easiest way is to let the Client repeatedly retry.

RDMA Read obtains the result from the response buffer of the server. Similarly, on the server side, the server side constantly checks whether there are new requests in the request buffer. However, this method has a clear disadvantage, that is, it wastes the Client's CPU cycles, especially when the number of retries is too high, making its CPU too high, and also wastes the server's passive processing of IOPS. Therefore, one should try to set a threshold T, and when the retry time is greater than T, it is considered to be a timeout, and the connection is disconnected. The setting of the threshold T must be reasonable. It must not be too small, otherwise, the link may be disconnected before the tracker is ready for data, and T must not be too large, otherwise, it will consume too much Client CPU and reduce throughput.

To achieve a better result, the value of the threshold T must be determined. T has the following relationship with throughput TH:

$$TH = argmax\, f(T,F,P,S)$$

In the formula, T represents the time of the attempt to read the RDMA from the Client before disconnecting, F stands for the extract size that the Client uses to read remote results from the server, P represents the processing time of the request on the server, and S represents the size of the result after the server has processed it.

The solution to the first problem is to conduct multiple experiments. First, record the average processing time of the server, then refer to the processing time, and temporarily set T to a rough guarantee that the Client can correctly retrieve the value of the result $Tmax$. Then, conduct multiple experiments using the idea of dichotomy. Set the value of T one by one, and look for the lowest possible time from the experimental results that can fully guarantee that the results are retrieved correctly. However, regardless of the number of T settings, the client may fail at a certain fetch of results, which requires an error-handling mechanism. The system sets that, when the correct result is not obtained, a retry

is performed first, and when the number of retries is greater than the maximum number of retries, the connection is disconnected, and the error message is output to the screen.

For the second problem, it can be seen from the content of the previous section that the main results need to be retrieved by the Client, and through a one-by-one analysis, corresponding solutions can be given for these results:

(3) Download the location information into storage;
(4) Download the content of the file.

For the first three results, the content size that the Client needs to retrieve is fixed. The structure of the storage information, FID, and download path defined in the code can easily calculate the number of bytes the content occupies. The Client only needs to read the content of the corresponding size at the corresponding stage to obtain the correct result. For the fourth result, although the file name the Client needs to download is not a fixed value, the FID contains the file size information. The Client needs to pre-parse the value of the file size through FID before downloading, pass the obtained value as a parameter to the RDMA Read function, and download the file completely and without redundancy.

By solving the above two problems, the optimization scheme further breaks through the transmission process, enabling the server to circumvent the high-consumption active RDMA operation, fully using its passive RDMA operation capability, and processing as much as possible at the same time. There may be more requests to improve the transmission efficiency of the entire system.

4. Experimental Analysis

4.1. Test Environment

(1) Test environment configuration

To evaluate the performance of the RDMA-based FastDFS presented in this paper, seven virtual machines were built under the VMware virtual machine platform. The physical hardware and software configuration of the virtual machine is shown in Table 1, indicating the processor, memory, hard disk size and other information of the server, and the operating system uses Windows 7 64-bit.

Table 1. Hardware environment configuration.

Category	Configuration
CPU	Intel® Core™ i7-8300HQ CPU @2.30 GHZ
RAM	32.0 GB
Disk	500 GB
Translator	GCC 4.47
OS	Windows 10 64-bit

There are seven nodes in the cluster, two nodes as Tracking Servers, four nodes as Storage Servers (of which Storage 1 and Storage 2 form Group 1, Storage 3 and Storage 4 form Group 2), and one node as the Client. Deploy the CentOS 7 environment for each of the seven nodes. The detailed configuration of the IP, role, and virtual machine for each server in the system is shown in Table 2.

Table 2. Test environment configuration.

ID	IP Address	Configuration	Role
A	192.168.156.61	CentOS7/2 GB RAM/20 G Disk/CPU/100 M	Tracker 1
B	192.168.156.62		Tracker 1
C	192.168.156.63		Group 1 Storage 1
D	192.168.156.64		Group 1 Storage 2
E	192.168.156.65		Group 2 Storage 3
F	192.168.156.66		Group 2 Storage 4
G	192.168.156.67		Client

The system architecture diagram is shown in Figure 7.

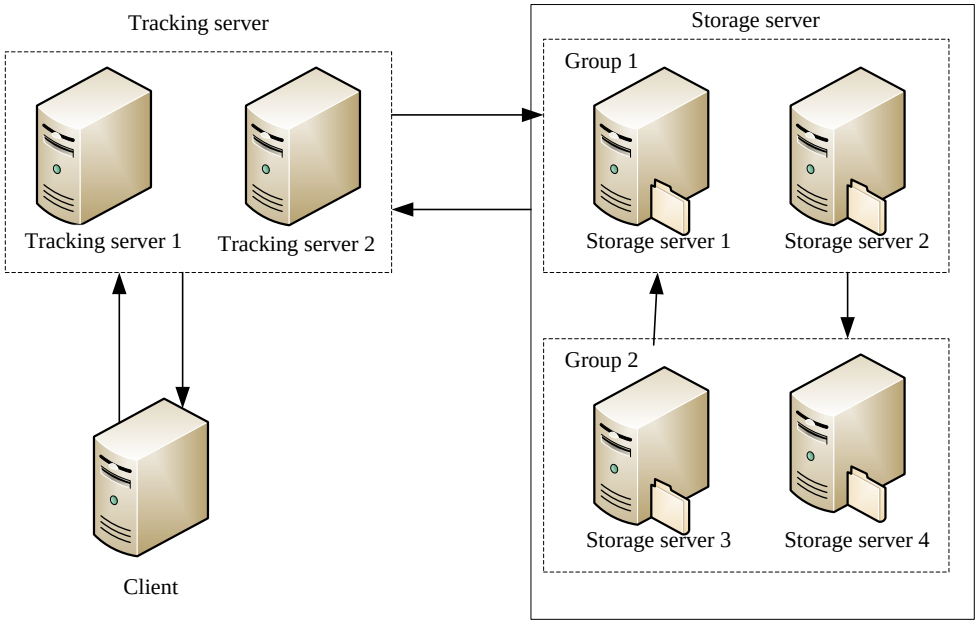

Figure 7. System test architecture diagram.

(2) Optimized FastDFS environment deployment

After installing the operating system CentOS 7 for each virtual machine, one can start to install FastDFS. Download the installation package from the official website address for decompression and installation. The FastDFS version used in this experiment is v5.05.

4.2. Comparative Test and Result Analysis

For the sake of the narrative, the original FastDFS system is referred to as Scheme 1 [35], Section 3.1 design Scheme the RDMA communication-based FastDFS is as defined as Scheme 2, and Section 3.2 design Scheme the RDMA FastDFS is referred to as Scheme 3. First, verify the functional correctness of Schemes 2 and 3, and then simulate the concurrency of multiple clients through multi-threading of the Client, upload, download, and delete experiments, analyze the experimental results, and evaluate and analyze the performance of the above three schemes from the three aspects of CPU occupancy, throughput, and transmission delay.

(1) CPU usage test

In the actual use case, when a large number of users use the distributed file system, the server needs to respond to multiple different requests in a very short time and return the processing results to the user, which requires the server to withstand a certain pressure when working. The specific method of testing is to have the Client establish a connection with the Tracking Server and continuously make a certain number of requests to the Tracking Server, observing the Tracking Server's CPU usage in this concurrent pressure environment. Select Tracker 2 as the experimental observation object and use the system's top tool to observe CPU usage. After the Client is started, the Client continuously sends requests to the server with multiple threads. This experiment uses the test script that comes with FastDFS./test_upload.sh for upload testing. Figure 8 shows the resulting interface after testing using *./test_upload.sh.*

First, modify the value of PROCESS_COUNT in *test_upload.c*. This value represents the maximum number of connections the Client generates, and the default is 10. Since the test needs to fully observe the processing power of the server when the number of connections is large, it is changed to 500 and recompiled. The ./gen_files command can generate 5 k, 50 k, 200 k, 1 M, 10 M, and 100 M test files each. Use ./gen_files to generate several test files of different sizes, with a total size of 5 G. Use the command vim test_upload.sh to

modify the number of connections simulated by the Client, change them to 10, 50, 100, 200, 300, 400, and 500, respectively, and execute the script ./test_upload.sh one by one for upload testing. Perform the above experiments several times, and obtain the average test value as the final experimental result. After the test is completed, the CPU usage rate of the Tracker 2 server is counted, as shown in Figure 9.

```
[root@itcast02 upload]# ../combine_result 10
total_count=44500, success_count=44500, success ratio: 100.00% time_used=302s, avg time used: 6ms, QPS=147.35

file_type total_count success_count time_used(s) avg(ms) QPS success_ratio
5K 34162 34162 213 6 160.38 100.00
50K 6449 6449 42 6 153.55 100.00
200K 3522 3522 28 8 125.79 100.00
1M 367 367 14 40 26.21 100.00
10M 0 0 0 0 0.00 0.00
100M 0 0 0 0 0.00 0.00

IO speed = 5210 KB
ip_addr  total_count success_count time_used(s) avg(ms) QPS success_ratio
192.168.156.65 11129 11129 68 6 163.66 100.00
192.168.156.66 11129 11129 90 8 123.66 100.00
192.168.156.64 11115 11115 68 6 163.46 100.00
192.168.156.63 11127 11127 72 6 154.54 100.00
```

Figure 8. The system includes a testing tool, i.e., *./test_upload.sh*.

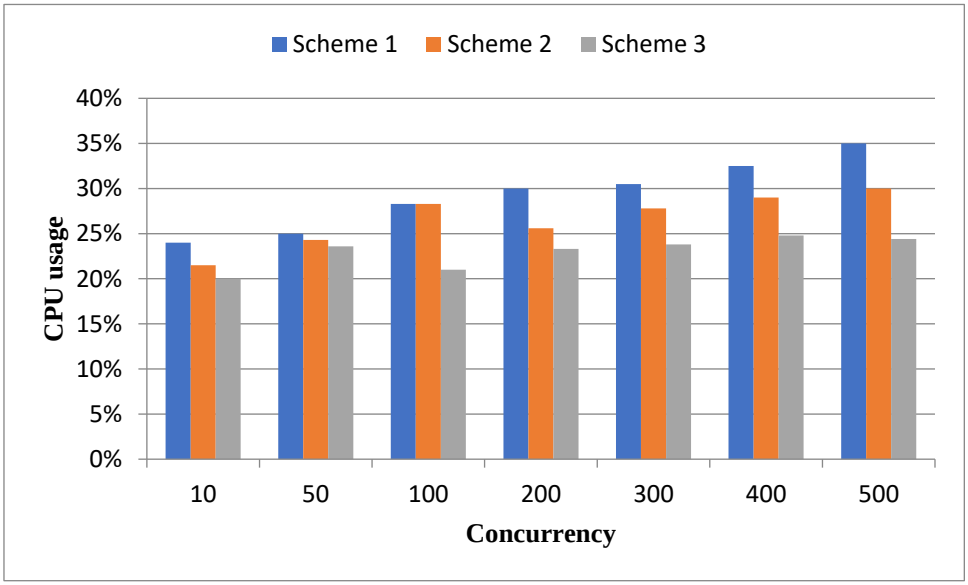

Figure 9. Server CPU utilization test results.

It can be seen from the experimental results that as the amount of concurrency increases, the overall CPU usage of the system shows an upward trend. When the amount of concurrency is small, there is no significant difference in CPU usage across the three scenarios because the CPU usage does not increase significantly. However, as the amount of concurrency increases, the advantages of Schemes 2 and 3 gradually become apparent, where Scheme 3 has a more obvious advantage over Scheme 2, because in Scheme 2, the Tracking Server still needs to respond to the Client, and in Scheme 3, the Tracker only puts the results into the specified memory and is fetched by the Client through the RDMA Read operation.

(2) I/O rate and the number of queries per second test

While testing the CPU usage, the I/O rate and QPS of queries per second were also tested. QPS reflects the number of requests that the system can handle per second.

The I/O access speed test results are shown in Figure 10.

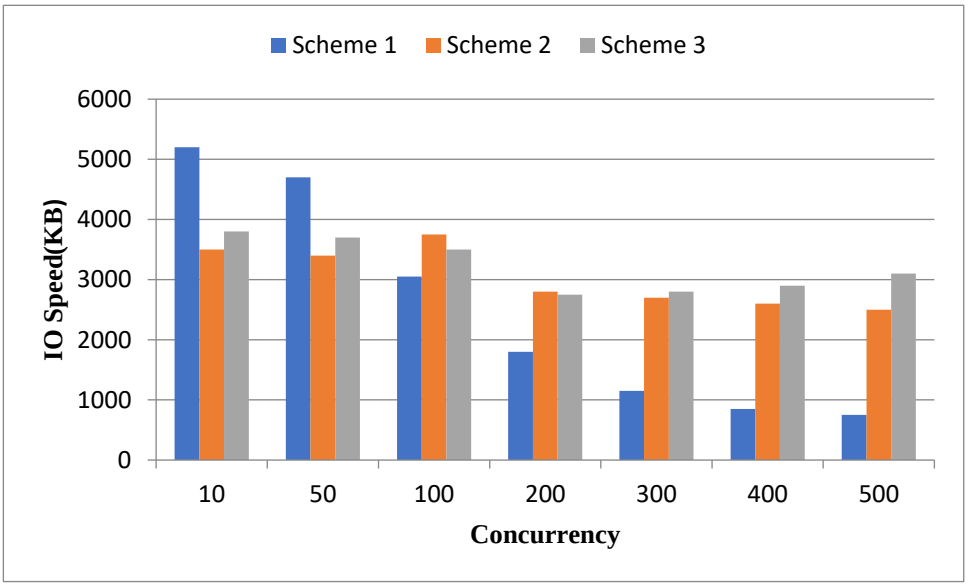

Figure 10. I/O rate test results.

The test results of QPS queries per second are shown in Figure 11.

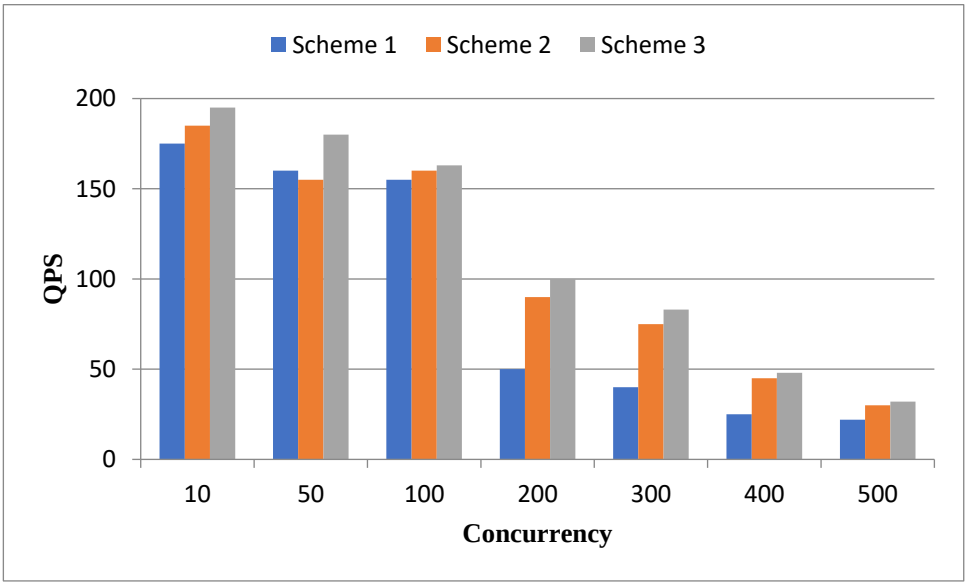

Figure 11. Query QPS test results every second.

From the test results, it can be concluded that when the amount of concurrency increases, the I/O rate and QPS of the disks have a significant downward trend. However, Schemes 2 and 3 are significantly improved compared to Scheme 1. RDMA communication has dramatically enhanced the I/O rate and QPS, particularly when the concurrency exceeds 200. Schemes 2 and 3 have the same I/O rate when the concurrency is less than 200, but when the concurrency is larger, the processing power of the server is fully utilized due to the process optimization of Scheme 3, and the gap between Schemes 2 and 3 is widened. At the same time, in terms of QPS, Scheme 3 also has certain advantages over Scheme 2.

Figure 12 shows the comparison of the transmission performance of the traditional mode in the RDMA network in the single-client connection environment. It can be seen that the bandwidth utilization rate of the proposed model is higher than that of the traditional method in different data I/O scenarios, and the bandwidth utilization rate is increased by about 27% in the case of 4 KB data I/O. At the same time, the transmission performance

gap between the two is large when the amount of data is small, and gradually narrows with an increase in the amount of data transferred, and the performance is close after the amount of 256 KB data. This is because the proposed model simplifies the interaction process, optimizes the transmission mechanism, and greatly reduces the time overhead of the software side outside the actual network transmission. I/O requests with small data volumes spend less time on network transmission, and software time consumption will greatly affect their transmission performance. However, I/O requests with large data volumes take a long time to transmit over the network, and this part of the time is not sensitive. It is for this reason that as the amount of data transferred increases, the bandwidth utilization of both increases.

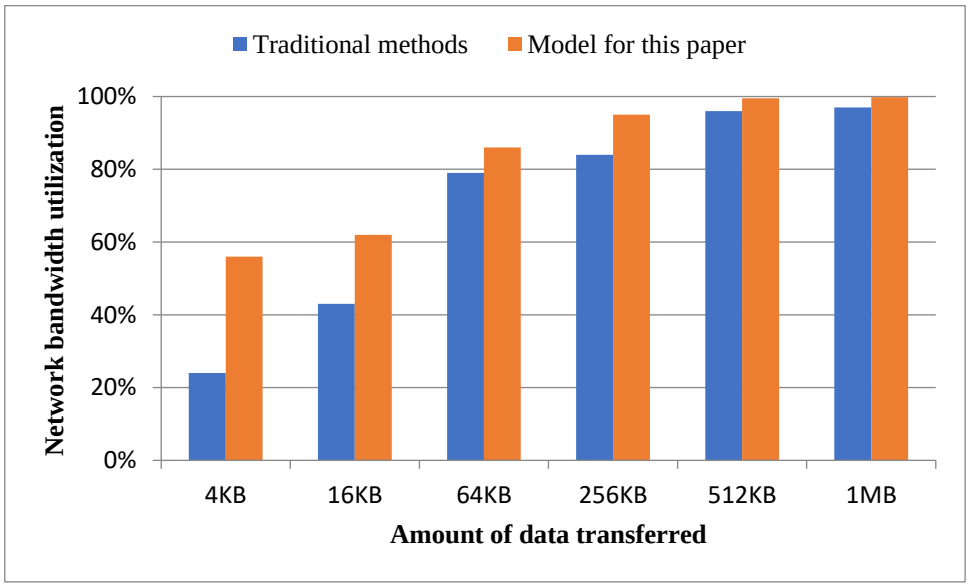

Figure 12. Comparison of system data transmission performance under single-client connection.

Figure 13 shows the transmission performance of the traditional mode in the RDMA network compared with the model in this article in the multi-client connection environment. Compared with the previous experiment, this experiment mainly evaluates the improvement of the network transmission performance of the proposed model in the concurrent environment. Since the transmission performance in the large data I/O scenario mainly depends on the actual transmission capacity of the network hardware, we still mainly focus on the performance comparison in the small data I/O scenario. Because concurrent processing complements the data processing capabilities on the software side, the system can make full use of the network transmission bandwidth in the multi-client connection environment. Among them, the bandwidth utilization of the traditional method in 4 KB data transmission reaches about 46%, and the model in this paper increases it to about 80%. The proposed model shows greater advantages in the concurrent environment, because the efficient one-sided interaction mode of the Client consumes less resources on the server side, so that the server side can support more Client data access when the concurrency reaches the upper limit.

In addition to the utilization of network bandwidth, latency during transmission is also an important aspect of performance evaluation. Figure 14 shows the test results of the transmission delay between the traditional method and the model in this article in a single-client connection environment. From the experimental results, it can be seen that the proposed model is better than the traditional method in terms of transmission delay. The reduction in the delay of the proposed model is mainly due to the optimization of the network interaction model and the reduction in memory replication by using the RDMA feature. Analysis of the data in the figure shows that the delay difference between the two transmission methods is maintained in the range of 50–100 *us*, and does not change

significantly with a change in data volume. This shows that the reduction in the delay of each request in this model is about such an order. Similar to the previous experiment, we can conclude that the increase in transmission latency is more pronounced for small data volume I/O requests.

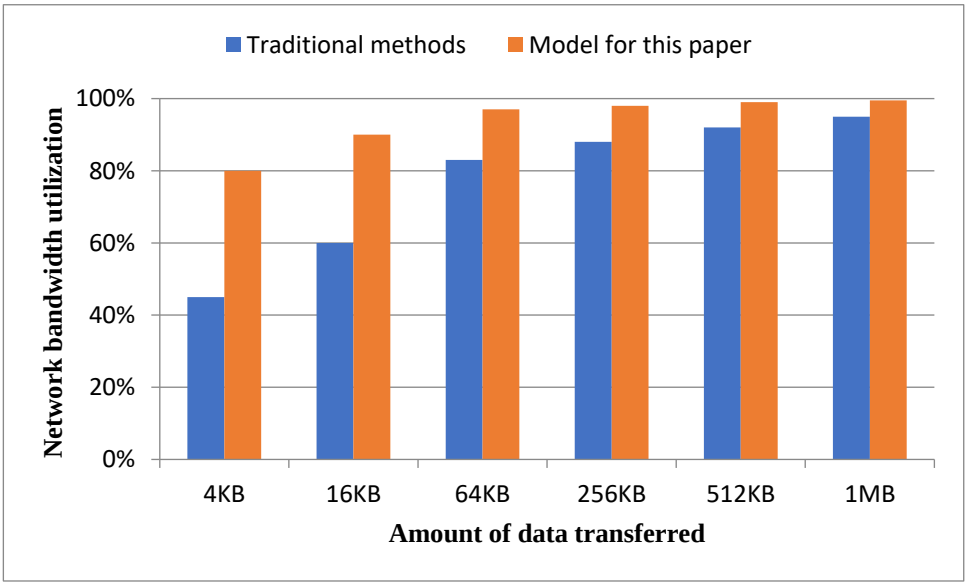

Figure 13. Comparison of system data transmission performance under multi-client connection.

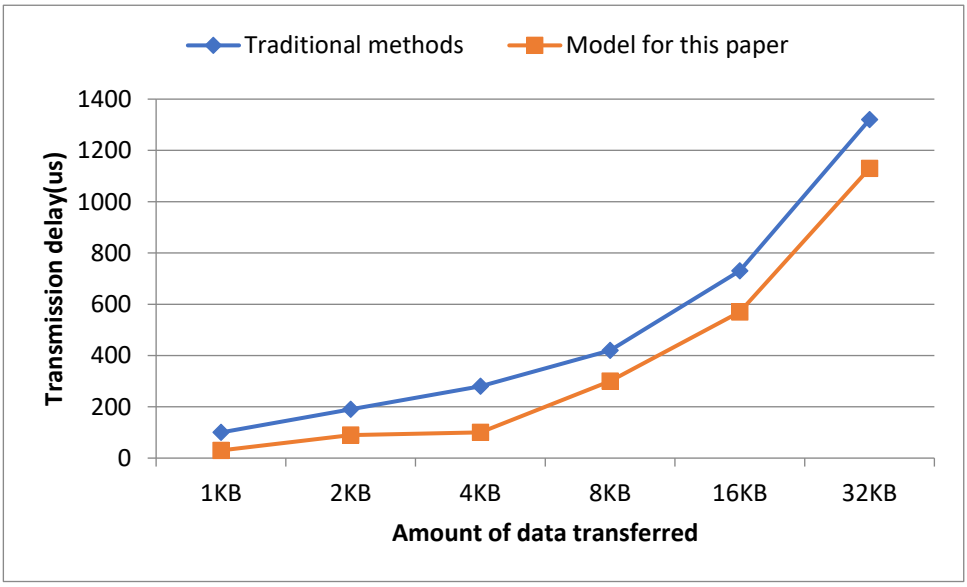

Figure 14. Comparison of system data transmission delay.

Figure 15 shows the comparison of the bandwidth achieved by network transmission when using the model in this paper and retaining a user-mode data replication under large data volume. As can be seen from the figure, although the bandwidth peak fluctuates, the network transmission bandwidth of this model with fewer data copies is relatively higher. The reason for this is that data replication limits the performance of network transmission, and the design of this model to reduce data replication increases the bandwidth limit of network transmission. In addition, the stronger the performance of the network hardware, the greater the negative impact of the time overhead of data replication, and correspondingly, the better the optimization of data replication in this model.

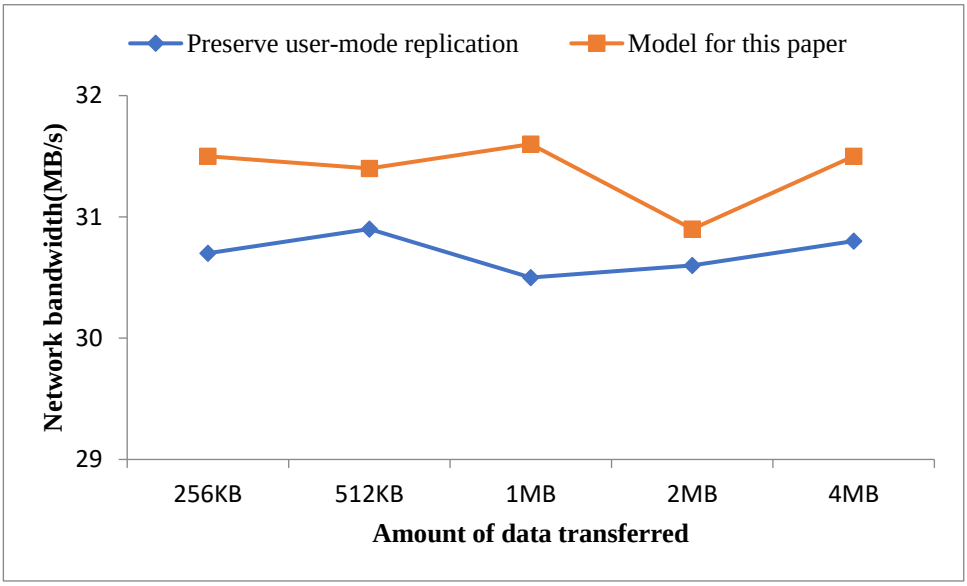

Figure 15. Comparison of network bandwidth for different times of data replication.

(3) Client single-sided I/O effect test

In addition to reducing data replication, another important aspect of this model for file data transfer is the Client single-sided I/O network transfer design. In this section, the read and write performances of bilateral I/O and the Client one-sided I/O modes are compared and tested in different data scale environments, and the bandwidth utilization in different environments is recorded. Figure 16 shows the performance test results of the read operation of the two network transmission models, and the performance test results of the write operation are shown in Figure 17.

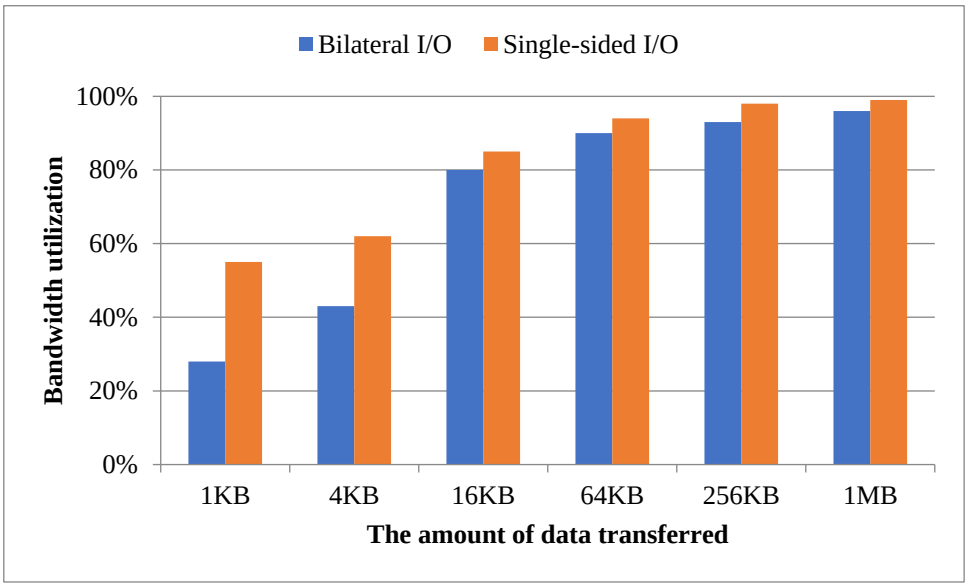

Figure 16. Comparison of client read operation performance of different network transport models.

As can be seen from the experimental results, whether reading data or writing data, the Client's single-sided I/O has improved the performance of network transmission. The Client's single-sided I/O performance improvements for read and write operations have some differences. Taking the 1 KB data volume as an example, the Client's single-sided I/O performance improvement on read operations is about 97%, while the performance improvement on write operations is about 203%. The reason for this is that in bilateral

I/O, after the Client completes the data write operation, the server side needs to check the integrity of the relevant data, but does not use it after reading the data. Therefore, for the Client's one-sided I/O mode, the optimization of the write data operation is relatively large.

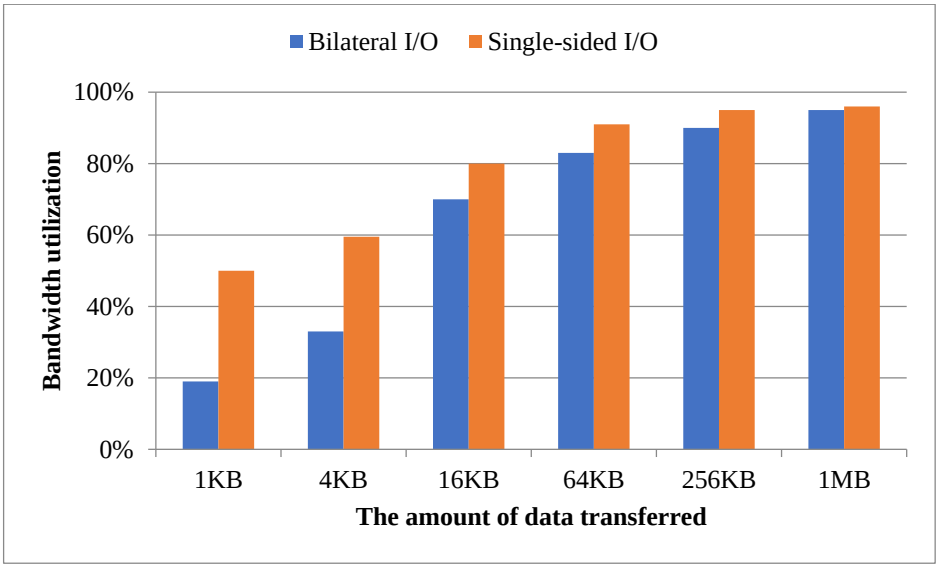

Figure 17. Comparison of client write operation performance of different network transport models.

(4) Transmission delay test

Latency is also an important indicator to measure the performance of a distributed file system. To test the transmission capacity of the three scheme systems, use ./gen_files to generate random files with a total size of 10 G, upload the experiments one by one, and observe the experimental results, as shown in Figure 18.

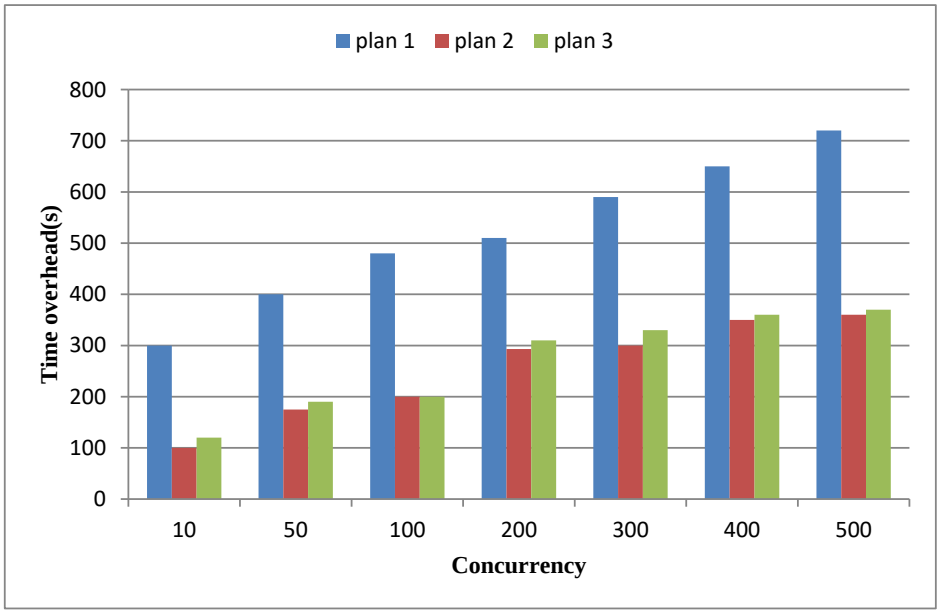

Figure 18. Time cost of the file transfer.

From Figure 18, it can be concluded that RDMA communication has a significant advantage in transmission delay compared to TCP. The transmission overhead of 1 is significantly reduced.

The experimental results are shown in Figure 19, and the transmission performance of the proposed model is better, on the whole, than that of the traditional method. When the

two parts are separated, when using traditional methods to interact, network transmission time accounts for about 22%, and software processing time accounts for about 77%. It can be seen that, in the traditional method, the time-consuming software processing is the bottleneck of the system, which limits the efficiency of network transmission and the overall data processing capacity of the system.

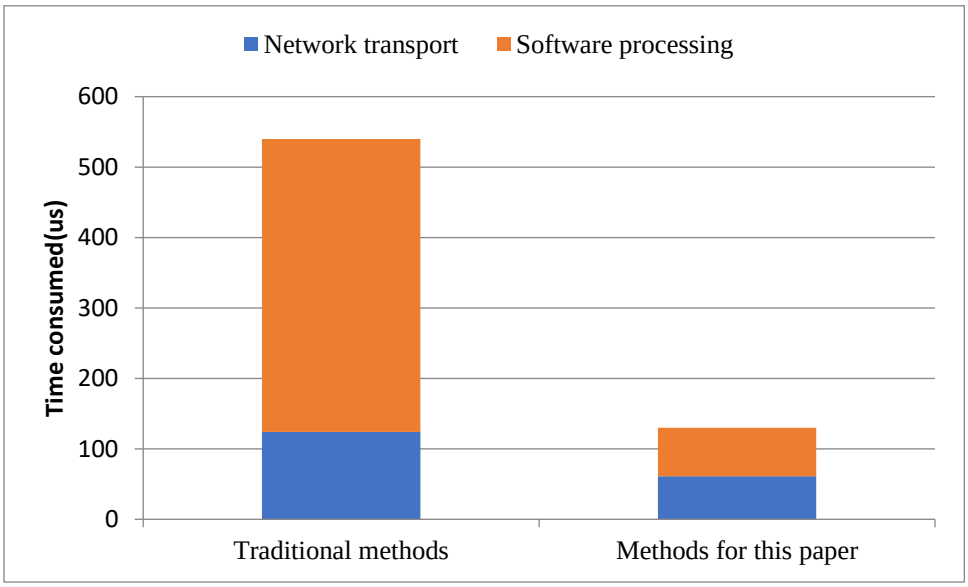

Figure 19. Proportion of time spent by each process of network interaction.

On the contrary, when using the model in this article for interaction, the network transmission time accounts for about 47%, and the software processing time accounts for about 53%. By reducing data replication and simplifying the more efficient use of RDMA, the model in this paper greatly reduces the proportion of time-consuming time on the software side, enhances the coordination between various levels of the system, and improves the network transmission efficiency and overall data processing capability of the system.

5. Conclusions

With the rapid development of the Internet, the number of small files generated in an age of fragmented information has increased exponentially, and the demand for massive storage of small files has gradually increased. By comparing and analyzing the advantages and disadvantages of RDMA technology and TCP, combined with the requirements of applications for distributed file systems in the current Internet environment, in this paper, we design a lightweight distributed file system based on FastDFS by using RDMA technology and change the file upload and download mechanism of FastDFS from the transmission process by further analyzing the asymmetric characteristics of RDMA unilateral operation, so that the performance of server-side RDMA virtual NICs can be fully exerted. The server resource utilization is improved, the file transfer process is optimized, and finally, the performance of the optimized FastDFS is verified by experiments. The experimental analysis results show that the optimization scheme proposed in this paper significantly improves various performance indicators compared with the original FastDFS. In terms of server-side CPU utilization and QPS, the optimized scheme with improved transmission process further reduces the CPU utilization rate compared with the scheme optimized for communication mode and increases the number of QPS on the server.

Author Contributions: Q.H. and F.Z. wrote the main manuscript text; Z.L. prepared the figures. All authors reviewed the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: The authors would like to thank anonymous referees for their invaluable suggestions and comments. This work is supported by the National Natural Science Foundation of China (61872284); Industrial Field of General Projects of Science and Technology Department of Shaanxi Province (2023-YBGY-203, 2023-YBGY-021); Industrialization Project of Shaanxi Provincial Department of Education (21JC017); and "Thirteenth Five-Year" National Key R&D Program Project (Project Number 2019YFD1100901), Natural Science Foundation of Shannxi Province, China (2021JLM-16).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank anonymous referees for their valuable suggestions and comments.

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

References

1. Kumar, K.J. Implementing Network File System Protocol for Highly Available Clustered Applications on Network Attached Storage. In Proceedings of the 2013 5th International Conference and Computational Intelligence and Communication Networks, Mathura, India, 27–29 September 2013; pp. 496–499.
2. Subramoni, H.; Lai, P.; Luo, M.; Panda, D.K. RDMA over Ethernet—A preliminary study. In Proceedings of the 2009 IEEE International Conference on Cluster Computing & Workshops, New Orleans, LA, USA, 31 August–4 September 2010.
3. Liu, J.; Li, B.; Song, M. THE optimization of HDFS based on small files. In Proceedings of the 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), Beijing, China, 26–28 October 2011.
4. Qi, X.; Hu, H.; Guo, J.; Huang, C.; Zhou, X.; Xu, N.; Zhou, A. High-availability in-memory key-value store using RDMA and Optane DCPMM. *Front. Comput. Sci.* **2023**, *17*, 3. [CrossRef]
5. Wasi-ur-Rahman, M.; Islam, N.S.; Lu, X.; Jose, J.; Subramoni, H.; Wang, H.; Panda, D.K.D. High-Performance RDMA-based Design of Hadoop MapReduce over InfiniBand. In Proceedings of the IEEE International Symposium on Parallel & Distributed Processing Workshops & Phd Forum, Cambridge, MA, USA, 20–24 May 2013.
6. Kim, H.; Yeom, H. Improving Small File I/O Performance for Massive Digital Archives. In Proceedings of the 2017 IEEE 13th International Conference on e-Science (e-Science), Auckland, New Zealand, 24–27 October 2017; pp. 256–265.
7. Dragojević, A.; Narayanan, D.; Castro, M.; Hodson, O. FaRM: Fast remote memory. In Proceedings of the Usenix Symposium on Networked Systems Design & Implementation, Seattle, WA, USA, 2–4 April 2014.
8. Kalia, A.; Kaminsky, M.; Andersen, D.G. Using RDMA efficiently for key-value services. In Proceedings of the 2014 ACM Conference on SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 295–306.
9. He, Q.; Bian, G.; Zhang, W.; Li, Z. RTFTL: Design and implementation of real-time FTL algorithm for flash memory. *J. Supercomput.* **2022**, *78*, 18959–18993. [CrossRef]
10. Chen, W.; Yu, S.; Wang, Z. Fast In-Memory Key-Value Cache System with RDMA. *J. Circuits Syst. Comput.* **2018**, *28*, 1950074. [CrossRef]
11. He, Q.; Bian, G.; Zhang, W.; Wu, F.; Li, Z. TCFTL: Improved Real-Time Flash Memory Two Cache Flash Translation Layer Algorithm. *J. Nanoelectron. Optoelectron.* **2021**, *16*, 403–413. [CrossRef]
12. Campbell, R. *Managing AFS: The Andrew File System*; Prentice-Hall: Saddle River, NJ, USA, 1998.
13. He, Q.; Zhang, F.; Bian, G.; Zhang, W.; Li, Z.; Chen, C. Dynamic decision-making strategy of replica number based on data hot. *J. Supercomput.* **2023**, *79*, 9584–9603. [CrossRef]
14. Vasile, M.; Martoiu, S.; Boukhadida, N.; Stoicea, G.; Micu, P.; Dumitru, A.; Iordache, C. Integration of FPGA RDMA into the ATLAS Readout with FELIX in High Luminosity LHC. *J. Instrum.* **2022**, *18*, C01025. [CrossRef]
15. Wang, Y.; Meng, X.; Zhang, L.; Tan, J. C-Hint: An Effective and Reliable Cache Management for RDMA-Accelerated Key-Value Stores. In Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 3–5 November 2014.
16. Mittal, R.; Shpiner, A.; Panda, A.; Zahavi, E.; Krishnamurthy, A.; Ratnasamy, S.; Shenker, S. Revisiting Network Support for RDMA. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 20–25 August 2018.
17. Dalessandro, D.; Devulapalli, A.; Wyckoff, P. iWarp protocol kernel space software implementation. In Proceedings of the International Conference on Parallel & Distributed Processing, Rhodes, Greece, 25–29 April 2006.
18. Fan, K.F. System and Method for RDMA QP State Split between RNIC and Host Software. U.S. Patent 8,161,126, 14 April 2012.
19. Michael, S.; Zhen, L.; Henschel, R.; Simms, S.; Barton, E.; Link, M. A study of lustre networking over a 100 gigabit wide area network with 50 milliseconds of latency. In Proceedings of the Fifth International Workshop on Data-Intensive Distributed Computing Date, Delft, The Netherlands, 19 June 2012; pp. 43–52.
20. Miao, M.; Ren, F.; Luo, X.; Xie, J.; Meng, Q.; Cheng, W. SoftRDMA: Rekindling High Performance Software RDMA over Commodity Ethernet. In Proceedings of the Asia-Pacific Workshop on Networking, Hong Kong, China, 3–4 August 2017.

21. Balaji, P.; Narravula, S.; Vaidyanathan, K.; Krishnamoorthy, S.; Wu, J.; Panda, D.K. Sockets Direct Protocol over InfiniBand in clusters: Is it beneficial? In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, USA, 10–12 March 2004; pp. 28–35.
22. Sur, S.; Koop, J.M.; Chai, L.; Panda, K.D. Performance analysis and evaluation of Mellanox ConnectX Infiniband architecture with multi-core platforms. In Proceedings of the 15th Annual IEEE Symposium on High Performance Interconnects (HOTI'07), Stanford, CA, USA, 22–24 August 2007; pp. 125–134.
23. Balaji, P.; Shivam, P.; Wyckoff, P.; Panda, K.D. High Performance User Level Sockets over Gigabit Ethernet. In Proceedings of the IEEE International Conference on Cluster Computing (Cluster'02), Chicago, IL, USA, 26 September 2002; pp. 179–186.
24. Grant, R.E.; Rashti, M.J.; Afsahi, A. An Analysis of QoS Provisioning for Sockets Direct Protocol vs. IPoIB over Modern InfiniBand Networks. In Proceedings of the International Conference on Parallel Processing-Workshops (ICPP-W'08), Portland, OR, USA, 8–12 September 2008; pp. 79–86.
25. Yu, W.; Rao, N.S.; Wyckoff, P.; Vetter, J.S. Performance of RDMA-capable storage protocols on wide-area network. In Proceedings of the Petascale Data Storage Workshop, Austin, TX, USA, 17 November 2008; pp. 1–5.
26. Yu, W.; Tian, Y.; Vetter, J.S. Efficient zero-copy noncontiguous I/O for globus on infiniband. In Proceedings of the 2010 39th International Conference on Parallel Processing Workshops, San Diego, CA, USA, 13–16 September 2010; pp. 362–368.
27. Huang, J.; Ouyang, X.; Jose, J.; Wasi-ur-Rahman, M.; Wang, H.; Luo, M.; Subramoni, H.; Murthy, C.; Panda, D.K. High-Performance Design of HBase with RDMA over InfiniBand. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, Shanghai, China, 21–25 May 2012; pp. 774–785.
28. Ceph over Accelio. 2014. Available online: https://www.cohortfs.com/ceph-over-accelio (accessed on 6 June 2023).
29. Islam, N.S.; Rahman, M.W.; Jose, J.; Rajachandrasekar, R.; Wang, H.; Subramoni, H.; Murthy, C.; Panda, D.K. High performance RDMA-based design of HDFS over InfiniBand. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 10–16 November 2012; pp. 1–12.
30. GlusterFS Docs. Available online: https://docs.gluster.org/en/latest/ (accessed on 6 June 2023).
31. Clark, D.D.; Jacobson, V.; Romkey, J.; Salwen, H. An Analysis of TCP processing overhead. *IEEE Commun. Mag.* **1989**, *27*, 23–29. [CrossRef]
32. Taranov, K.; Rothenberger, B.; De Sensi, D.; Perrig, A.; Hoefler, T. NeVerMore: Exploiting RDMA Mistakes in NVMe-oF Storage Applications. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022. [CrossRef]
33. Shi, W.; Wang, Y.; Corriveau, J.P.; Niu, B.; Croft, W.L.; Peng, M. Smart Shuffling in MapReduce: A Solution to Balance Network Traffic and Workloads. In Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, 7–10 September 2015; pp. 35–44.
34. Wu, Y.; Ma, T.; Su, M.; Zhang, M.; Chen, K.; Guo, Z. RF-RPC: Remote Fetching RPC Paradigm for RDMA-Enabled Network. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1657–1671. [CrossRef]
35. Liu, X.; Yu, Q.; Liao, J. Fastdfs: A high performance distributed file system. *ICIC Express Lett. Part B Appl. Int. J. Res. Surv.* **2014**, *5*, 1741–1746.