Optimization of RDMA-Based HDFS Data Distribution Mechanism

Xiao Zhang, Binbin Liu, Junhao Zhao, Cong Dong

Optimization of RDMA-based HDFS Data Distribution Mechanism

Xiao Zhang(✉), Binbin Liu, Junhao Zhao, and Cong Dong

School of Computer Science, Northwestern Polytechnical University,
Xian 710129, Shaanxi, China
zhangxiao@nwpu.edu.cn

Abstract. Hadoop Distributed File System (short for HDFS) is a high availability file system designed to run on commodity hardware. It uses multiple replicas to ensure high reliability, and many data are transmitted between storage nodes. The performance of data transmission has a great impact on the latency of writing operations. Remote Direct Memory Access (short for RDMA) is a protocol with low latency and high through which is running on the Infiniband network. When HDFS runs on the Infiniband network, the default protocol IPoIB can not take advantage of the high-speed network. The latency of the writing process is similar to a TCP/IP network. In this paper, we present a new RDMA-based HDFS writing mechanism. It enables *DataNodes* to read data parallelly from the *Client* through RDMA. And by using RDMA primitive, the transmission latency is slower than the original TCP/IP protocol. The experiments show that our approach reduces the latency of the writing process by 10.11%-40.81% compared with the original HDFS.

Keywords: Distributed File System · RDMA ·
Data Distribution Mechanism · HDFS · Performance Optimization

1 Introduction

HDFS provides high-throughput data access, and programs running on HDFS usually have large data sets. Typical HDFS file sizes are in GBs or TBs volume level. An HDFS cluster can support hundreds of nodes and thousands or millions of files. Through the analysis of the existing HDFS, it is found that the communication protocol used is mainly TCP/IP. Due to the excessive processing delay of the TCP/IP protocol, there are multiple memory copies, and now the memory bandwidth performance is very large with high CPU bandwidth and network bandwidth. The difference causes HDFS to have a higher write latency. With the popularity of IB equipment and the rapid progress of research on RoCE (RDMA Over Converged Ethernet), iWARP (Internet Wide Area RDMA Protocol) and other Ethernet-based analog IB equipment, RDMA has gradually stepped out of the field of high-performance computing and has been applied to the optimization of distributed file systems. RDMA can be used to obtain higher data transmission performance, reducing the time for replicas to be transmitted

between nodes, thereby shortening the execution time of HDFS write operations and reducing write latency. In order to accelerate the data transmission speed between nodes and reduce the write latency of HDFS, this paper designs a new write data distribution mechanism for HDFS based on RDMA, so that each data node can read data in parallel. The experiment results show that the optimized HDFS write latency is reduced by 10.11%-40.81%. The main contributions of this paper are as follows:

(1) Analyze the advantages and disadvantages of the HDFS writing process and communication mechanism.
(2) Analyze the three implementation methods of IB network and the two primitives of RDMA, determine the most suitable primitive usage scheme for HDFS optimization, minimize the modification of the source code architecture, and greatly reduce the HDFS write delay.
(3) A data distribution mechanism based on RDMA technology is proposed, so that each data node can read data in parallel.
(4) Evaluate the optimizations achieved in this paper and reduce the overall time-consuming of HDFS write operations.

2 Related Work

RDMA was first used in the field of high-performance computing. With the reduction of hardware costs and the progress of Ethernet-based RDMA research (such as RoCE and iWARP), RDMA is gradually used in the field of distributed storage. The paper of Jiesheng Wu and others used RDMA to optimize the IO performance of PVFS [1]. They designed a RDMA-based transport layer to make PVFS data transmission transparent and stable in performance; designed a buffer management method for flow control , Dynamic and balanced buffer sharing; an efficient memory registration and deregistration scheme was proposed. Brent Callaghan proposed a storage protocol that supports RDMA on the WAN(Wide Area Network) [2], and use it to accelerate NFS performance [3]. The paper by Qingchao Cai et al. proposes an efficient distributed memory platform GAM, which provides directory-based cache coherence protocol through RDMA, so it can integrate the memory of each node, and ensure consistency through state transition, read through RDMA and take the data of the remote node [4]. Anuj Kalia's paper proposes FaSST [5], a high-performance, scalable, and distributed memory transaction processing system that uses RDMA message primitives and datagram transmission methods to design a new RPC (Remote Procedure Call), so it can maintain low overhead and simple system design. These studies show that RDMA can benefit traditional distributed and parallel file systems.

With the development of Hadoop, the performance of HDFS has received a lot more attention. They applied RDMA to HDFS to improve its performance. Sayantan Sur's paper studies the impact of high-performance interconnection networks on HDFS [6]. When the underlying storage medium is HDD (Hard Disk Drive), the network equipment is replaced by Gigabit Ethernet card to IB

network card (IPoIB), the performance can be improved by 11%-100%; when the underlying storage medium is replaced by SSD (Solid State Drive), the high-speed Internet can bring 48% -219% performance improvement. At present, the network becomes the IO bottleneck because of the emergence of NVMe SSD and even NVM (Non-volatile Memory) devices, and the application of RDMA can bring more obvious performance improvement effects. Since Hadoop has released many versions, and many HDFS-based optimizations are tightly coupled with the RDMA design, it cannot be easily applied to the production environment; so Adithya Bhat and others have designed a RDMA-based HDFS plug-in that can flexibly interact with Hadoop2.5, 2.6 version combination [7].

HDFS communication is mainly based on RPC, and using RDMA to modify the RPC interface to optimize communication bottlenecks is also a solution. For example, Li Liang designed and developed a set of RDMA-based network communication architecture in his paper, and implemented and provided an RPC over RDMA communication interface [8]; Yang Heng optimized RDMA-based RPC in his paper. Registered memory blocks are used repeatedly to reduce memory registration time and improve performance [9]. They eventually increased the RPC communication rate by 30%, but it was not applied to HDFS, but only provided us with an idea. After all, RPC still has serialization overhead, and if it does not cooperate with the distribution mechanism of modifying write data, it will still waste RDMA communication performance, and the reduction in HDFS write latency is not obvious. As stated in Dushyanth Narayanan's paper, although RDMA reading is not as flexible as RPC, it can bring benefits in terms of latency and throughput, and we should use reasonable design to give full play to its advantages [10].

Nusrat Sharmin Islam et al. performed a series of optimizations on HDFS based on RDMA. Firstly, they designed a set of java buffer management mechanism, and modified the data transmission protocol of HDFS using RDMA-based UCR (Unified Communication Runtime) library [11]. Next, they further optimized the RPC of HDFS to make RDMA compatible with Java's IO interface and reduce its memory copy [12]. Finally, due to the improvement of IO performance, the HDFS software stack has become a new bottleneck, and they have proposed SOR-HDFS [13], using the SEDA(Staged Event-Driven Architecture) [14] to improve HDFS write performance. SOR-HDFS divides the HDFS write process into four stages, each of which has an input queue and a thread pool associated with it. At any stage, the threads in the thread pool will use the data in the input queue, process it and provide it to the input queue in the next stage. This allows the different stages of data transmission and IO to overlap to the greatest extent, thereby accelerating the overall speed of the write process.

3 Background and Motivation

In this section, firstly, we outline the existing HDFS write data process and determine its performance bottlenecks. Then, we introduced the different implementations of RDMA, determine our usage plan and clarify the reasons.

3.1 HDFS Introduction

HDFS is an important part of the Hadoop ecosystem. As a distributed file system, it is built on a cluster of multiple servers, and each server can be responsible for one or more roles, which are *NameNode*, *DataNode*, and *Client*. As shown in the figure, the various roles are interconnected through the network, using RPC for control flow, and using Socket for data flow. As shown in Figure 1, where *NameNode* acts as a cluster metadata server, it stores cluster metadata information, contacts the *DataNode* through a heartbeat mechanism and monitors the cluster status. The *Client* must also pass the *NameNode* to register and obtain the required file information. As a data center of the cluster, *DataNode* provides storage services, sends heartbeat information to *NameNode* through RPC, reports the status of data blocks, and transmits data with other nodes through the pipeline flow constructed by Socket. The Client provides an interactive interface between the cluster external application and HDFS. The upper layer application can interact with the *NameNode* to obtain cluster information and file metadata information through the *Client*, and interact with the *DataNode* to read or write files through the *Client*.

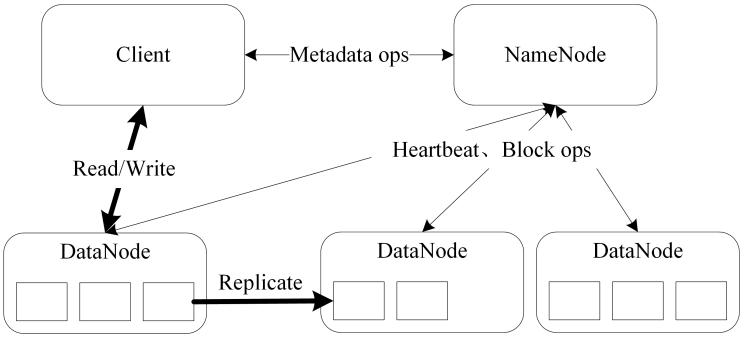


Fig. 1. HDFS Architecture

HDFS saves redundant copies of data on multiple *DataNode* nodes. The default number of copies is 3, which are stored on this node, the same rack node, and adjacent rack nodes. This design not only performs data backup, but also improves security. It also makes it possible to access neighboring nodes as much as possible when reading data to improve access efficiency; but at the same time it also increases the overall data transmission amount when data is written, this phenomenon will be more serious under the poor network performance and pipe line data transmission.

3.2 The Writing Process of Original HDFS

In HDFS, the completion of a file write operation requires the cooperation of the entire cluster. The process of writing a file is shown in Figure 2. First, the Client

needs to do some preparation work. For example, contact the *NameNode* to apply for the data block information, initialize the *DataStreamer* thread; create a file lease to avoid conflicting file write operations; *NameNode* will also update the metadata information after receiving the request from the client, allocate the data block and return block information; the *client* can establish a Pipeline connection with the *DataNode* based on the returned information. Then, the *client* can perform formal data transmission operations. For example, cyclically writes data to the data queue; at the same time, the *DataStreamer* thread will also take out the data in the data queue in parallel, calculate the checksum, and encapsulate it into a packet to write into the pipeline stream; *DataNode* reads the packet from the pipeline stream and check the checksum, write it to disk, and pass the packet to the downstream node. Finally, after receiving all ACK messages, the Client performs the finishing work. For example, close the pipeline flow and contact the *NameNode* to update the file status.

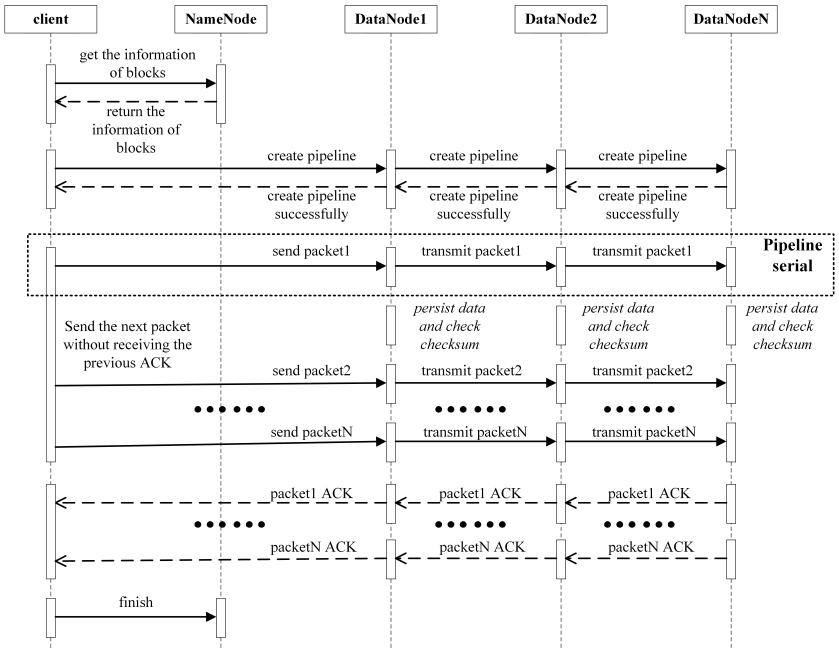


Fig. 2. The flowchart of HDFS writing process

It can be seen from the previous introduction that although the *Client* does not need to wait for the ACK information of the previous packet when passing the packet, each *DataNode* needs to wait for the packet data from the upstream node. This serial pipeline flow slows down the entire write process. On the *DataNode* side, each node can check the checksum and persist the data only after receiving the entire packet from the upstream node. We divide the HDFS writing process

into four parts: communicating with *NameNode* (registering file information and obtaining data block information), establishing PipeLine, transmitting data, and completing files; and the process of transmitting data can be divided into four at each *DataNode* Stage: Receiving the packet, checking the checksum, persist the data, and transmitting the packet. As shown in Figure 3, receiving and transmitting packets account for 66% of the data transmission stage, and then account for 43% of the entire write process, is the bottleneck of the entire writing process. Therefore, improving the data transmission environment and optimizing the data distribution mechanism can greatly reduce HDFS write latency.

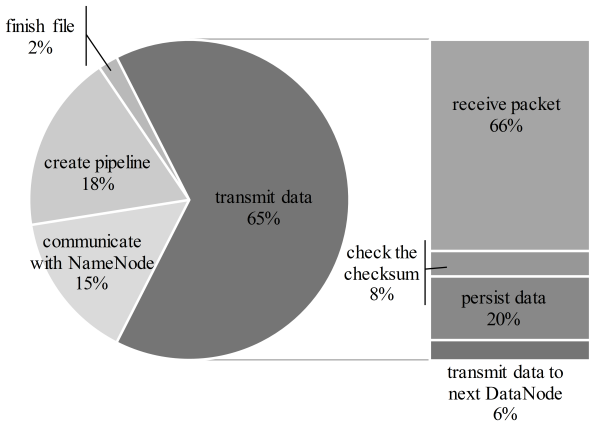


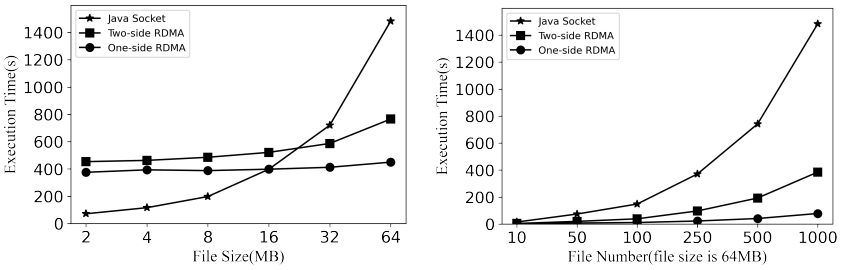
Fig. 3. The proportion of time cost in each phase of the write process of HDFS

3.3 RDMA Introduction

RDMA technology contains three major features: low CPU usage, kernel bypass, and zero copy. RDMA reduces the server-side data processing delay in network transmission. It transmits data directly from the memory of one computer to another without the intervention of both operating systems. There are three different implementations of RDMA, namely InfiniBand, RoCE and iWARP. Among them, InfiniBand is a kind of network specially designed for RDMA, which guarantees reliable transmission from the hardware level, while RoCE and iWARP are RDMA technologies based on Ethernet and support the corresponding verbs interface. InfiniBand network supports RDMA new generation network protocol, with good performance, but the price of network card and switch is also very high. With the development of technology, the emergence of RoCE and iWARP which are based on Ethernet, reduced the cost of hardware and promoted the

research on RDMA. But in order to obtain the best optimization effect, this paper uses InfiniBand to build the corresponding RDMA network.

RDMA accesses remote memory using two types of primitives [15]: 1) message primitives, *send* and *recv*, similar to socket programming, before sending a message, the receiver needs to call the *recv* primitive in advance to specify the received message The stored memory address, such primitives are also called two-side RDMA; 2) memory primitives, *read* and *write*, these primitives can directly read or update remote memory without intervention from the remote CPU, They are called one-side RDMA. Obviously, the advantage of memory primitives is that they can obtain higher performance, and save CPU resources when there are computationally intensive tasks at the remote end; the disadvantage is that because there is no CPU participation, it is easy to cause consistency problems. On the contrary, the advantages of message primitives are that information can be transmitted in time, and there is feedback for reading and writing; the disadvantage is that the overhead is large and the delay is high. As shown in Figures 4(a) and 4(b), due to the need to create queue pairs and maintain memory buffers in advance when establishing RDMA connections, it takes some time, and Java Socket does not need to maintain the above information; therefore, in small-scale data transmission, the performance of RDMA is not as good as Java Socket; but with the increase of data scale, after all, the preliminary preparation of RDMA only needs once, so the performance is greatly overtaken. In view of the above characteristics of RDMA, in the design of this paper, we will choose to use message or memory primitives according to whether the communication process needs feedback, and only choose to use RDMA when the network connection can be established or be used many times, so as to give full play to RDMA performance, reduce HDFS write latency.



(a) small file (b) more files

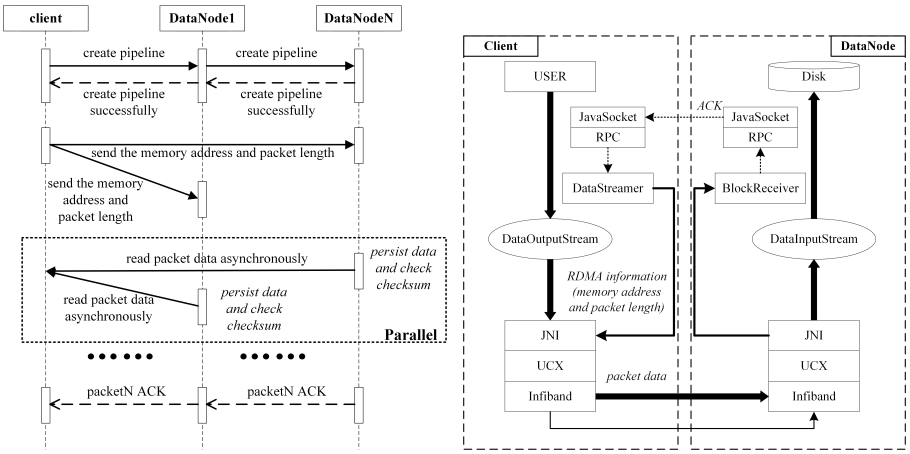
Fig. 4. The time cost of the different communication ways

4 The Design and Implementation of JUCX HDFS

According to the previous analysis, data distribution is the bottleneck of the entire writing process. The read performance of one-side RDMA is much higher

than other network transmission methods, and reading data from the *Client* to the *DataNode* does not require feedback, so here is the most It is suitable to use one-side RDMA. However, one-side RDMA need to determine the corresponding memory address, data transmission also needs to know the packet size, and these two information are small, the *DataNode* and *Client* need to interact in time, so use the message primitives of RDMA. Finally, the operation of Pipeline and ACK, because it is only a simple RPC transmission control flow, and the amount of data is small, almost no impact on performance, so retain the original Java Socket communication method.

The overall write process after optimization is shown in Figure 5(a). Since the data distribution part was mainly modified, the process of contacting the *NameNode* in the early stage and completing the file in the later stage has not changed, so it is not reflected in the figure. In the packet data transmission process, first, the client sends the memory address and packet length of the packet data to each *DataNode* through the message primitive, that is, the *send* function. Then, the *DataNode* can read the packet data directly through the memory primitive, namely the *get* function, according to the received information. Although the *send* function still serially sends RDMA information to each *DataNode*, due to the small amount of data, it can be considered that each *DataNode* starts to read data from the *Client* at almost the same time, that is, to persist data in parallel.



(a) The write process (b) The data stream and control flow

Fig. 5. RDMA-based HDFS Write Distribution Mechanism

In this paper, RDMA uses JNI(Java Native Interface) to call the underlying UCX(Unified Communication X) library. The detailed data flow after optimization is shown in Figure 5(b). First, the client-side *DataStreamer* thread constructs a packet, temporarily stores the data in memory, and sends related in-

formation to the *DataNode* through the RDMA *send* function. Then, the *BlockReceive* thread of the *DataNode* can receive information through the *recv* function. After determining the remote address and read length, the packet data in the client memory can be directly read through RDMA and persisted to the disk. Finally, the ACK message is returned to the client via the original RPC. The experiment uses HDFS cluster built by Hadoop 2.9.0 to analyze the performance of HDFS under different storage media and different networks. This paper directly uses the put file system interface provided by HDFS to test the execution time of write operations. The put command can upload files locally to the HDFS cluster, that is, the complete HDFS write process is executed. Use the put command to upload 1-5GB files to the HDFS cluster and record the execution time of the command to analyze the reduction effect of this optimization on HDFS write latency.

DataNode can also implement a parallel data distribution mechanism using TCP/IP, but TCP/IP will perform multiple memory copies, so the transmission performance is far lower of RDMA. And the first consideration in this paper is to apply RDMA technology to HDFS more efficiently, so this paper does not implement the data distribution mechanism under TCP/IP.

5 Evaluation

5.1 Experimental Setup

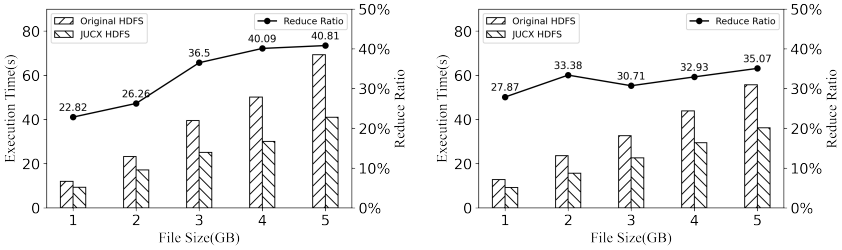
The experiment used 6 machines with the same hardware configuration to build the Hadoop 2.9.0 cluster. The software and hardware configuration of the machine is shown in Table 1. Among them, one machine is used as the *NameNode* node, one is used as the Client node, and the remaining four are used as *DataNode* nodes. Commands are executed under the *Client* node to avoid the influence of *NameNode* and *DataNode*.

Table 1. The configuration of platform

Name	Describe
Model	Sugon S650
CPU	AMD Opteron(TM) 6212, 2.6 GHz, 16 cores
Memory	40GB
HDD	1TB SEAGATE 7200 rpm
SSD	250GB SamSung MZ-76E2508
1GigE NetWork	Intel 82574L Gigabit Network Connection
IB NetWork	Mellanox Technologies MT26428
Operating System	Ubuntu 16.04 Linux, kernel 4.4.0-119-generic

5.2 Single Replica

First, in order to test the optimization effect of RDMA, regardless of the effect of the copy distribution mechanism, we set the cluster to single replica mode. Different back-end storage media will also affect the write performance, so we use HDD and SSD as *DataNode* storage disk for experiments. The experiment results are shown in Figure 6(a) and Figure 6(b). The optimized HDFS in this paper reduces the write latency of 22.87%-40.81%. Consistent with the previous analysis, due to the time-consuming preparations such as establishing RDMA connections, the performance improvement is not obvious when the data volume is small, and it can be greatly improved when the file size is 5GB. When the storage medium is replaced by HDD to SSD, the data persistence time is reduced, and the two stages of data transmission and data persistence have pipeline design, so the overall write performance of HDFS is improved, and the performance optimization of data transmission is covered. It is reflected in the test results that the overall optimization rate under SSD is lower than under HDD.



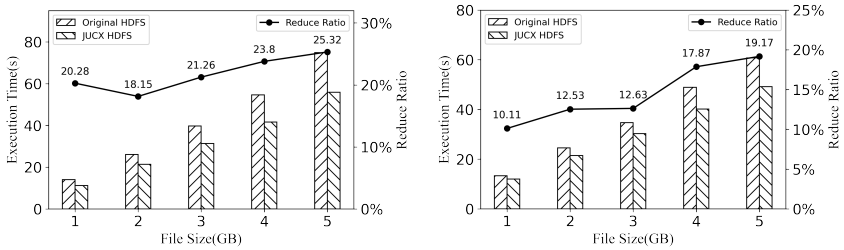
(a) HDD (b) SSD

Fig. 6. HDFS write performance in single-replica

5.3 Multiple Replicas

Next, in order to verify our write distribution mechanism, we adjusted the number of cluster replicas to perform the same test. Observing the experimental results, we can see that due to the increase in the number of replicas, the workload of data transmission increases. Whether it is the original HDFS or our optimized HDFS, the execution time of the write operation has increased. According to Figures 7(a) and 7(b), HDFS optimized in this paper reduces the write latency of 10.11%-25.32%. However, due to the pipeline mechanism in HDFS's own design, during the packet transmission, *DataNode* is also performing data persistence, so the performance improvement of the data transmission process will be covered by the data persistence process, so the performance improvement in the case of multiple replicas is not as high as in the case of single replica. In the future, we will design new solutions and optimize the data persistence process, but this is not the work of this paper. Through the new write distribution mechanism, we

have retained the complete RDMA optimization to improve the write performance, so that the new HDFS we designed has a large increase in write latency under two replica configurations and two underlying storage configurations.



(a) HDD (b) SSD

Fig. 7. HDFS write performance in multiple-replica

6 Conclusion

This paper proposes a RDMA-based data distribution mechanism that improves the HDFS write process and optimizes HDFS write performance. By analyzing the existing HDFS writing process, it can be found that the data transmission process has a large overhead and a high time-consuming account. This is caused by the underlying TCP/IP protocol and the serial pipe stream data transmission method. Therefore, we used the high throughput and zero copy features of RDMA to modify the HDFS write process, so that the *DataNode* can read data from the *Client* in parallel, reducing the data transmission time for writing files and improving the overall write performance. Experiments show that, compared to the existing HDFS, the optimized HDFS using the method proposed in this paper reduces the write latency by 10.11%-40.81%.

Acknowledgements. This work is supported by the National Key Research and Development Project of China (2018YFB1004400), Beijing Municipal Natural Science Foundation-Haidian original innovation joint fund(L192027). Xiao Zhang is the corresponding author.

References

1. Wu, J., Wyckoff, P., Panda, D.: Pvfs over infiniband: Design and performance evaluation. In: 2003 International Conference on Parallel Processing, 2003. Proceedings., pp. 125–132. IEEE (2003)
2. Yu, W., Rao, N.S.V., Vetter, J.S.: Experimental analysis of infiniband transport services on wan. In: 2008 International Conference on Networking, Architecture, and Storage, pp. 233–240 (2008)

3. Callaghan, B., Lingutla-Raj, T., Chiu, A., Staubach, P., Asad, O.: Nfs over rdma. In: Proceedings of the ACM SIGCOMM Workshop on Network-I/O Convergence: Experience, Lessons, Implications, NICELI '03, p. 196–208. Association for Computing Machinery, New York, NY, USA (2003). DOI 10.1145/944747.944753. URL https://doi.org/10.1145/944747.944753
4. Cai, Q., Guo, W., Zhang, H., Agrawal, D., Chen, G., Ooi, B.C., Tan, K.L., Teo, Y.M., Wang, S.: Efficient distributed memory management with rdma and caching. Proc. VLDB Endow. **11**(11), 1604–1617 (2018). DOI 10.14778/3236187.3236209. URL https://doi.org/10.14778/3236187.3236209
5. Kalia, A., Kaminsky, M., Andersen, D.G.: Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram rpcs. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 185–201. USENIX Association, Savannah, GA (2016). URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kalia
6. Sur, S., Wang, H., Huang, J., Ouyang, X., Panda, D.K.: Can high-performance interconnects benefit hadoop distributed file system. In: Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO, p. 10. Citeseer (2010)
7. Bhat, A., Islam, N.S., Lu, X., Wasi-ur Rahman, M., Shankar, D., Panda, D.K.D.: A plugin-based approach to exploit rdma benefits for apache and enterprise hdfs. In: BPOE, pp. 119–132. Springer (2015)
8. Liang, L.: Research and implementation of communication protocol based on rdma across user and kernel space. Master's thesis, Huazhong University of Science and Technology (2016)
9. Heng, Y.: Research and implementation of optimization of rpc over rdma in user-space. Master's thesis, Huazhong University of Science and Technology (2016)
10. Dragojevic, A., Narayanan, D., Castro, M.: Rdma reads: To use or not to use? IEEE Data Eng. Bull. **40**(1), 3–14 (2017)
11. 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: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–12. IEEE (2012)
12. Lu, X., Islam, N.S., Wasi-Ur-Rahman, M., Jose, J., Subramoni, H., Wang, H., Panda, D.K.: High-performance design of hadoop rpc with rdma over infiniband. In: 2013 42nd International Conference on Parallel Processing, pp. 641–650. IEEE (2013)
13. Islam, N.S., Wasi-ur Rahman, M., Lu, X., Panda, D.K.: High performance design for hdfs with byte-addressability of nvm and rdma. In: Proceedings of the 2016 International Conference on Supercomputing, pp. 1–14 (2016)
14. Welsh, M., Culler, D., Brewer, E.: Seda: an architecture for well-conditioned, scalable internet services. ACM SIGOPS Operating Systems Review **35**(5), 230–243 (2001)
15. Kalia, A., Kaminsky, M., Andersen, D.G.: Design guidelines for high performance rdma systems. In: 2016 USENIX Annual Technical Conference (USENIX ATC 16), pp. 437–450 (2016)