Optimization of the Master/Slave Model in Supercomputing Global Storage Systems

Xiaobin He[1][a], Wei Xiao[1][b], Qi Chen[2][c], Xin Liu[1][d] and Zuoning Chen[3][e]

[1]*National Research Center of Parallel Computer Engineering & Technology, Xueyuan Road, Haidian District, Beijing, China*

[2]*School of Computer Science and Technology, Tsinghua University, Beijing, China*

[3]*Chinese Academy of Engineering, Beijing, China*

Keywords: Supercomputing, High Concurrency IO, Global Storage, Master/Slave Model, Bandwidth.

Abstract: The global storage system is an important infrastructure for supercomputing systems, providing a globally consistent view of data for large-scale compute nodes. The Master/Slave model is used extensively in the software implementation of global storage to enhance the server-side capability of handling IO requests. In recent years, as network performance and IO concurrency of compute nodes have increased, the problem of high performance loss of the Master/Slave model in high concurrency scenarios has come to the fore. Taking LWFS, the data forwarding software of the Sunway supercomputers, as an example, this paper proposes two performance optimization methods: optimizing the request receiving capability of the Master based on a multi-threaded parallel processing approach, and reducing the lock overhead of multiple Slave synchronization based on a no-wait request queue mechanism. Tests in the Sunway E-class prototype validation system show that the peak bandwidth of 1M block sequential read/write and 4K block random read/write are increased by 16% and 90% respectively after the optimization methods are overlaid, and the peak bandwidth of 4K block random read/write is increased by 80% and 48% respectively, which shows that the optimized Master/Slave model has a significant increase in read/write bandwidth under high concurrency scenarios and can better adapt to the application IO requirements under high concurrency scenarios.

1 INTRODUCTION

Global storage system is one of the most important supporting system of supercomputing system (hereinafter referred to as supercomputing), which provides globally consistent shared data view for large-scale supercomputing nodes, and ensures data reading and writing for supercomputing applications. In recent years, with the continuous expansion of supercomputing scale, the performance of data transmission network has increased significantly, and supercomputing applications have put forward higher requirements for the performance of global storage system. However, compared with the rapid growth of computing scale and the network performance, the performance and scale of the storage system is relatively stable. How to reduce the IO competition caused by multi process of applications in high concurrency scenarios based on limited storage node resources and improve the overall efficiency of the system has become a problem that must be faced by the designer of modern global storage systems(Yildiz et al., 2016).

The design of modern supercomputing global storage system is complex, which is often divided into metadata storage services, data storage services, etc. It provides global storage space and data view through distributed data access protocols. However, no matter how complex the global storage system design is, it can be abstracted into a simple C/S architecture. C represents clients, often running on computing nodes, user service nodes, etc. And S represents server, which connects with clients through high-speed storage networks to support efficient data

[a] https://orcid.org/0000-0001-6785-1561
[b] https://orcid.org/0000-0002-3047-5520
[c] https://orcid.org/0000-0003-2256-1143
[d] https://orcid.org/0000-0002-7870-6535
[e] https://orcid.org/0000-0003-2256-1143

He, X., Xiao, W., Chen, Q., Liu, X. and Chen, Z.
Optimization of the Master/Slave Model in Supercomputing Global Storage Systems.
DOI: 10.5220/0011949700003612
In *Proceedings of the 3rd International Symposium on Automation, Information and Computing (ISAIC 2022)*, pages 416-421
ISBN: 978-989-758-622-4; ISSN: 2975-9463

access services for clients. In supercomputing, the number of clients of the global storage system is often much larger than that of the server. In order to ensure the efficiency, the server uses the Master/Slave model to improve the ability of processing parallel data access requests. The Master/Slave model is a commonly used network message packet processing model(Leung and Zhao, 2008). Generally, the master thread is responsible for receiving data access requests through the network and distributing them to a fixed number of slave threads for execution. This mechanism is simple to implement and stable in operation, and is widely used in the field of network data packet processing.

In the construction of global storage system, the Master/Slave model is also widely used due to the wide range of highly concurrent data transmission. However, in recent years, with the increasing scale of supercomputing and the improvement of supercomputing network performance, the performance overhead of the traditional Master/Slave model is increasing, which has been unable to meet the needs of global storage. There are two main reasons. First, the master's single thread implementation has been unable to meet the efficient request reception needs of high-speed networks, causing the bottleneck of data access request reception. Secondly, the cost of multi slaves lock synchronization operation in high concurrency scenarios is increasingly significant, and the fixed slave cannot flexibly solve the load imbalance problem of multi threads , resulting in the reduced efficiency of large-scale synchronous IO in supercomputing high concurrency scenarios.

This paper proposes two optimization methods to address the problems of the Master/Slave model used for GFS. Firstly, the multi-threaded master technique is used to solve the performance bottleneck of a single master receiving data access requests, and secondly the wait-free queue mechanism is used to reduce the synchronous locking overhead of multiple slave threads. This paper takes evaluations on the Sunway E-class Prototype Verification System(SEPVS)(Jiangang Gao, 2021). The evaluation results show that the proposed method can significantly improve the performance of the Master/Slave model of global storage system under high concurrency scenarios.

2 BACKGROUND

At present, in order to support the global data access requirements of supercomputing's large-scale computing nodes, the global storage system for supercomputing is generally built using the hierarchical storage architecture(Chen et al., 2020) composed of the global file system(GFS) layer and the forwarding layer. The GFS layer is responsible for building a globally shared high-capacity and high-performance storage space, which is composed of metadata storage node cluster, data storage node cluster, and storage service node cluster. The metadata storage node cluster and data storage node cluster provide distributed metadata and data storage services for the storage service node cluster respectively. High speed network is used to connected the above nodes, providing metadata and data service communication based on efficient Remote Direct Memory Access(RDMA) protocol(Chen and Lu, 2019). The data forwarding layer consists of a storage service node cluster and a computing node. The storage service node provides a data forwarding service for the computing node to access data, and the nodes are allocated according to the grouping of computing nodes(Ji et al., 2019). Generally, one computing node group corresponds to several storage service nodes. Due to the huge scale of supercomputing system, the number of computing nodes is often much larger than the storage service nodes, which leads to the storage service nodes need to bear huge load pressure(Bez et al., 2021). Taking SEPVS as an example, it contains 1024 computing nodes, and only 18 storage service nodes provide IO services for these computing nodes. For supercomputing applications that pursue the ultimate performance of data access, this ratio poses a great challenge to the efficient processing of data access requests in high concurrency scenarios at the software level. The storage system of SEPVS is built with the Lustre file system as GFS(Rao et al., 2018), and Light Weight File System (LWFS) as data forwarding file system(Chen et al., 2020). The LWFS resides on top of the GFS client and provides data access request forwarding services for multiple computing nodes as a shared service. Both Lustre and LWFS use the Master/Slave model to support high concurrent access to large scale compute nodes.

The workflow of Master/Slave in GFS is shown in Figure 1. Generally master is a single thread, receives network packets by multiplexed IO mechanism, and stores the data access requests parsed from the packets into the request queue. The current mainstream multiplexed IO mechanism mainly includes epoll, poll, select(Duan et al., 2004), and epoll is widely used because of its performance advantages(Gammo et al., 2004). The request queue is generally a FIFO queue, which is accessed synchronously by the master and slave through locks. Slave is generally a group of threads running independently, which fetches requests from the request queue and then executes the requested operations in parallel. After the operation is executed, the result is sent to the client via the network transmission component. In a traditional HPC system, the CPU computation rate is much higher than the network rate, so the master can complete the task of receiving network messages in a single thread. Due to the high latency of storage operations, multiple slave processes are running in parallel to ensure that the maximum performance of the underlying storage resources is efficiently utilized. In the era of E-class

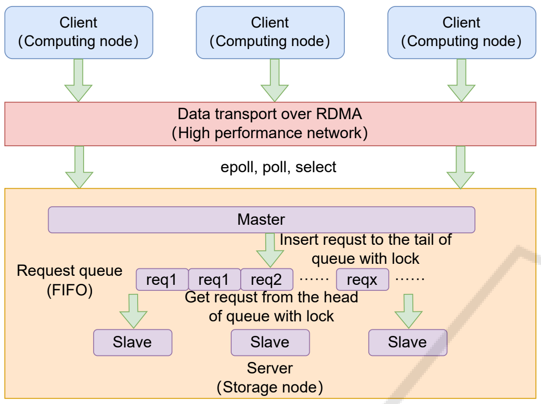

Figure 1: Workflow of Master/Slave Models in Global Storage.

supercomputing, network performance has improved rapidly, and the single port rate of high-speed network has exceeded 200Gbps(Yamazaki et al., 2017). As a result, the single-process master is often not capable of receiving concurrent requests from multiple clients. At the same time, the scale of supercomputing in the E-class era has not only led to an increase in data access concurrency, but also to a large variation in data requests from high-performance computing applications. These factors together make it difficult to adapt the traditional Master/Slave model to the future demands of shared storage in the E-class era. Taking the LWFS deployed in SEPVS as an example, the bandwidth loss of 1MB block read/write bandwidth of LWFS compared with the direct access to GFS reaches 44% and 29% respectively when 24 compute nodes are concurrently accessed, which is mainly due to the single-threaded processing bottleneck of the master. When the concurrency rises further to 32 compute nodes, the loss of LWFS read/write bandwidth rises further, mainly due to the greater overhead of the slave shared request queue's lock at high concurrency. Similar phenomenon is observed for the random access performance of 4KB block size. The above tests show that the performance overhead is high in high-concurrency scenarios, resulting in a huge waste of storage resources, which cannot meet the needs of future supercomputing high-speed network environments and high-concurrency storage access scenarios in the E-class era.

3 METHODOLOGY DESIGN

In order to solve the problem of excessive bandwidth loss in the supercomputing global storage of the Master/Slave mechanism in the E-level era, this paper designs two optimization methods. One is the multi thread optimization of the master, which can realize the parallelization of the core process of network message receiving, and improve the concurrent receiving ability of the global storage for client data access requests. The second is the wait-free queue optimization of slave. This technology uses the wait-free queue to construct the request queue for multiple slaves, reduce the lock overhead of request queue management, and reduce the performance loss in high concurrency.

3.1 Master Optimization

Traditional single threaded master calls epoll_wait to wait for the IO event from network. If multiple threads are used to call epoll_wait to get IO event of a network interface, when a thread gets a event from the network, the corresponding IO event handler does not finish processing the event. At this time, the monitored file descriptor generates a new IO event, which may be acquired by another thread, causing another thread to compete for access to the file descriptor. To ensure data consistency, the file descriptor needs to use a lock protection mechanism, which will greatly reduce the processing efficiency.

To this end, we uses epoll's EPOLLONESHOT mechanism to solve the problem of multi thread competitive access, as shown in Figure 2. For a file descriptor registered with EPOLLONESHOT, the operating system can trigger at most one EPOLLIN, EPOLLOUT, or EPOLLERR event registered on it, and only once. In this way, when a thread process an event on a file descriptor, other threads have no chance to operate on the file descriptor. This mechanism ensures that a file descriptor is processed by only one thread at any time, ensuring data consistency, it avoids multi thread competitive access. At the same time, once the file descriptor registered with EPOLLONESHOT is processed by a thread, the thread immediately resets the EPOLLONESHOT event of the file descriptor to ensure that subsequent events of the file descriptor can be retrieved and processed by the thread. The EPOLLONESHOT implementation mechanism reduces the burden of consistency management on multiple masters, while ensuring data consistency.

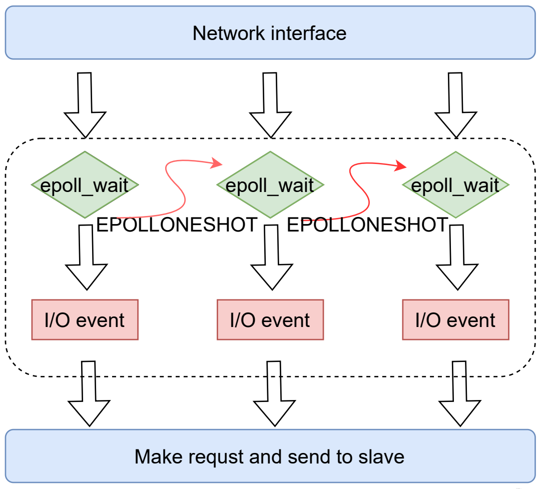

Figure 2: Workflow of multi thread master.

3.2 Slave Optimization

In the implementation of multi threaded parallel master, the receiving rate of network requests is accelerated, and the cost of lock of multiple slave threads will become more prominent. Therefore, this paper uses the wait-free queue provided by the userspace rcu library to solve this problem(Arbel and Attiya, 2014). The wait-free queue supports the request insert operation without lock protect, while the operation of obtaining requests from the queue requires a mutex so that multiple slave threads can obtain requests from the queue at the same time. However, this mutex only requires the collaboration between slaves, which can minimize the possibility of competition, as shown in Figure 3.

In addition, to further reduce competition, the number of slave threads can be dynamically adjusted according to the policy management service to adapt to different application workloads. To realize the dynamic scaling of slave threads, slave threads are composed of a management thread and multiple worker threads. All management operations of the request processing queue, such as creating more worker threads, waking up sleep worker threads, are completed by a separate management thread, which enables it to manage all threads scaling related information and thread lists without locking. In the specific implementation, the request processing queue management task is based on the signal to avoid multiple system calls per request, thus improving the system service efficiency.

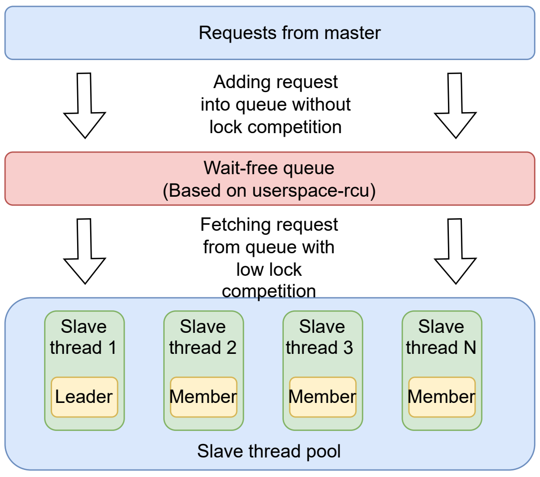
Requests from master
Adding request into queue without lock competition
Wait-free queue (Based on userspace-rcu)
Fetching request from queue with low lock competition
Slave thread 1
Leader
Slave thread 2
Member
Slave thread 3
Member
Slave thread N
Member
Slave thread pool

Figure 3: Workflow of wait-free queue slave.

4 EVALUATION

4.1 Environment and Methodology

The SEPVS is deployed at the National Supercomputing Centre in Jinan, China, and uses a many-core processor named SW26010Pro. Each processor contains 6 core groups and can support 6-way I/O concurrent processes, and the nodes are interconnected using the Sunway high performance network named SWnet. The storage system conforms to the supercomputing storage system architecture, where the data forwarding node is configured with a dual-way server CPU, 256GB of memory, each node has a 200Gbps SWnet network interface. The data forwarding system was constructed using LWFS, and the Master/Slave optimization method proposed in this paper was used to improve the performance of LWFS.

In this paper we compare the performance of the original LWFS before optimization, LWFS after multi-threaded master optimization, and LWFS with both multi-threaded master optimization and no-wait queue slave optimization stacked on top of each other in a high concurrency scenario. The benchmark we used is IOR(IOR,), and the compute node sizes set to 1, 4, 8, 16, 24, 32 and 64, where each compute node starts 4 concurrent processes. According to the data block size distribution commonly used in supercomputing applications, a sequential read/write bandwidth of 1M block size and a random read/write bandwidth of 4K block size were selected to represent two scenarios, large block sequential and small block random for comparative analysis, respectively.

4.2 Evaluation Results and Analysis

The evaluation results of the 1MB block sequential write are shown in Figure 4. At a concurrent scale of 16 compute nodes, the optimized LWFS read and write bandwidths of the multi-threaded master and the wait-free queue slave (stacked on top of the multi-threaded master) are all higher than the original LWFS. And as the concurrency processes number continues to increase, the performance of the multi-threaded master optimized LWFS gradually decreases, in contrast to the wait-free queue slave optimized LWFS where the performance gradually increases and reaches peak performance with 32 compute nodes accessed concurrently. In terms of peak write bandwidth, the multi-threaded master achieves an 85% improvement in peak performance compared to the original LWFS, and the stacked with wait-free queue slave optimization achieves a further 2% improvement compared to the multi-threaded optimized master. The two optimizations add up to a 90% improvement in peak performance compared to the original LWFS. For the six different node sizes in our evaluation, the multi-threaded optimized master improved by an average of 53% compared to the original LWFS, the stacked wait-free queue optimized slave improved by an average of 10%.

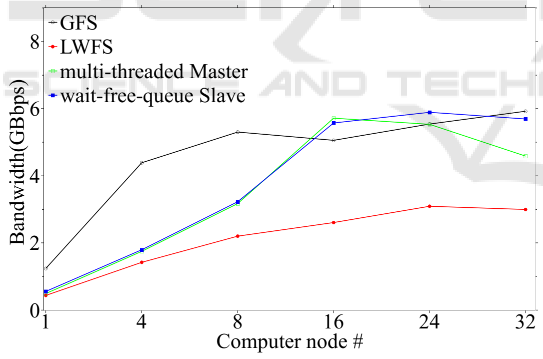

Figure 4: Results of 1MB Block Sequential Write Aggregate Bandwidth Test.

The evaluation results of the 1MB block sequential read are shown in Figure 5. The variation pattern of the sequential read bandwidth is similar to that of the 1MB sequential write bandwidth. The stack of the two optimizations not only improves performance, but also provides more stable performance in high concurrency scenarios. The peak read aggregation bandwidth of the multi-thread master is 14% higher than the original LWFS, and after stacked with wait-free queue optimization it achieves a further 2% improvement compared to the multi-threaded optimized master. The average increase of the multi-threaded master under different node sizes is only 6%. The main reason is that the master is not a performance bottleneck in the small-scale concurrent scenario, and the multi-threaded master introduces overhead in the epoll part when receiving network packets, which leads to a certain performance overhead. After adding the wait-free queue slave optimization, the average increase is 8%.

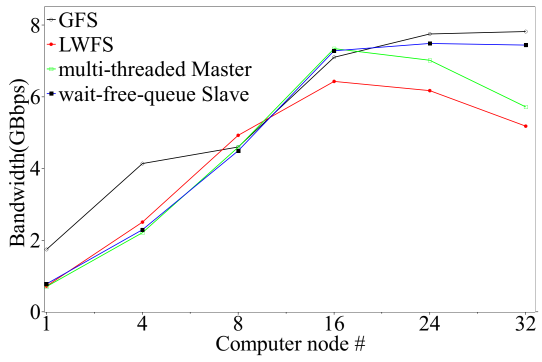

Figure 5: Results of 1MB Block Sequential Read Aggregate Bandwidth Test.

The evaluation results of the 4KB block random write are shown in Figure 6. The results show that as the number of compute nodes increased, the original LWFS, the multi-threaded master, the stacked multi-threaded master and the wait-free queue slave all achieved performance increases, with the peak performance of the multi-threaded master improving by 21% and the peak performance of the stacked wait-free queue slave further improving by 21%. The peak performance of the two optimization methods is 48% higher than that of the original LWFS. The average performance improvement is 9% for the multi-threaded master, and 12% for the wait-free queue slave at different node sizes.

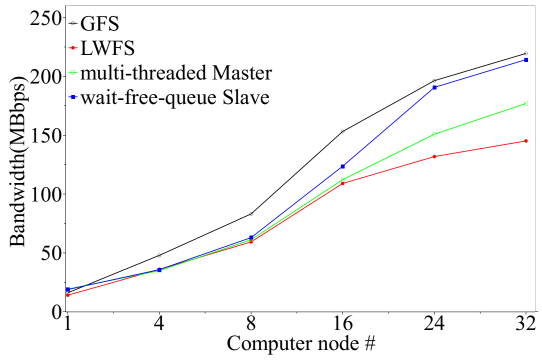

Figure 6: Results of 4KB Block Random Write Aggregate Bandwidth Test.

The results of the 4KB block random read are shown in Figure 7. The peak performance of the multi-threaded master is improved by 35% and further improved by 33% with the addition of the wait-free queue slave. The peak performance of the two optimizations is improved by 80% compared to the original LWFS. The average performance improvement is 13% for the multi-threaded master and 19% for the wait-free queue slave.

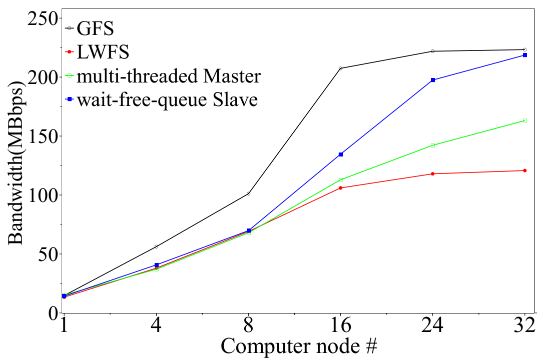

Figure 7: Results of 4KB Block Random Read Aggregate Bandwidth Test.

The above results show that the Master/Slave model of LWFS after the stack of the two optimization methods is more stable and efficient in terms of large block sequential write performance in high concurrency scenarios, and there is almost no performance loss in peak aggregation bandwidth compared to GFS in high concurrency scenarios, achieving a lossless global storage performance output.

5 CONCLUSIONS

The The Master/Slave model is a common parallel mechanism for global storage in supercomputing, but the limitations of this mechanism are becoming more and more prominent in today's increasing network performance and concurrency of supercomputing applications. In this paper, we propose two performance optimization methods to address this issue: multi-threaded optimization for master and wait-free queue optimization for slave. Evaluations of the two optimization methods in the Sunway E-class Prototype Verification System show that the peak bandwidth of 1M block sequential read/write is improved by 16% and 90% respectively, and the peak bandwidth of 4K block random read/write is improved by 80% and 48% respectively. The performance of global storage is more stable under concurrent scenarios after the two optimization methods are stacked. The evaluation results prove that the optimization mechanism proposed in this paper can effectively solve the problem of excessive performance overhead of the traditional Master/Slave model, and can better adapt to the I/O requirements of applications under high concurrency scenarios.

REFERENCES

Arbel, M. and Attiya, H. (2014). Concurrent updates with rcu: Search tree as an example. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*.

Bez, J. L., Miranda, A., Nou, R., Boito, F. Z., Cortes, T., and Navaux, P. (2021). Arbitration policies for on-demand user-level i/o forwarding on hpc platforms. In *International Parallel and Distributed Processing Symposium*.

Chen, Q., Chen, K., Chen, Z.-N., Xue, W., Ji, X., and Yang, B. (2020). Lessons learned from optimizing the sunway storage system for higher application i/o performance. *Journal of Computer Science and Technology*, 35:47–60.

Chen, Y. and Lu, Y. (2019). Scalable rdma rpc on reliable connection with efficient resource sharing. pages 1–14.

Duan, H. C., Xian-Liang, L. U., and Song, J. (2004). Analysis and design of communication server based on epoll and sped. *Computer Applications*.

Gammo, L., Brecht, T., Shukla, A., and Pariag, D. (2004). Comparing and evaluating epoll, select, and poll event mechanisms. *Proceedings of the 6th Annual Ottawa Linux Symposium*.

IOR. HPC IO benchmark repository. https://github.com/hpc/ior.

Ji, X., Yang, B., Zhang, T., Ma, X., Zhu, X., Wang, X., El-Sayed, N., Zhai, J., Liu, W., and Xue, W. (2019). Automatic, Application-Aware I/O forwarding resource allocation. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 265–279, Boston, MA. USENIX Association.

Jiangang Gao, Hongsheng Lu, W. H. e. a. (2021). The interconnection network and message machinasim of sunway exascale prototype system. *Chinese Journal of Computers*, 44:222–234.

Leung, J. and Zhao, H. (2008). Scheduling problems in master-slave model. *Annals OR*, 159:215–231.

Rao, N., Imam, N., Hanley, J., and Oral, S. (2018). Wide-area lustre file system using lnet routers. pages 1–6.

Yamazaki, H., Nagatani, M., Hamaoka, F., Kanazawa, S., Nosaka, H., Hashimoto, T., and Miyamoto, Y. (2017). Discrete multi-tone transmission at net data rate of 250 gbps using digital-preprocessed analog-multiplexed dac with halved clock frequency and suppressed image. *Journal of Lightwave Technology*, PP:1–1.

Yildiz, O., Dorier, M., Ibrahim, S., Ross, R., and Antoniu, G. (2016). On the root causes of cross-application i/o interference in hpc storage systems. pages 750–759.