Scholar Publishing Group
Distributed Processing System
https://doi.org/10.38007/DPS.2020.010202
ISSN 2790-0916 Vol. 1, Issue 2: 10-17

High Performance Computing for Big Data in Distributed Systems

Antonio Cruz Chavez*

Univ Quebec, Ecole Technol Super, Montreal, PQ, Canada

**corresponding author*

Keywords: Big Data, High Performance Computing, Distributed Systems, Meta Data

Abstract: With the continuous progress of high performance computing technology and application technology, high performance computer has been applied more and more widely. This paper focuses on the research and application of high performance computing for big data in distributed systems. This paper first designs a distributed hybrid storage system based on DRAM and SSD, and then implements a phase-consistent strategy to optimize the performance of client-side cache and optimize the read performance through client-side metadata cache. The simulation results show that the system designed in this paper can be applied to the actual production environment.

1. Introduction

With the continuous development of computer architecture and hardware manufacturing technology, CPU operation speed, random access memory and disk capacity, access speed have made great progress. The number of transistors in DRAM and CPU doubles every 18 months according to Moore's law, while disk and other I/O devices develop at a much slower rate due to the characteristics of their mechanical components. In the past decades, the annual growth rate of CPU performance exceeds 60%, while the performance of disk only grows at a rate of 4% to 7%. The increasing gap between the CPU computing speed and the access speed of the I/O system forms the "I/O wall". The barrel effect in the whole system leads to the I/O system becoming the main bottleneck of the overall system energy, especially in the case of the parallel distributed computing cluster sharing the storage system, the I/O system seriously restricts the overall system performance [1-2]. In order to alleviate the bottleneck of I/O system, many new technologies or researches have been developed to improve the access performance of I/O system. New storage media are used to overcome the performance bottleneck of traditional disk media, such as SSD and other non-mechanical storage media. Improve the network throughput performance of the storage system using high-performance storage networks, such as Infiniband. Research on new storage architecture,

using multi-level storage system to improve the efficiency of data access; The parallel processing technology was introduced into the external storage system, and RAID and other technologies emerged to improve the throughput of disk data access through I/O parallel. In addition, a large number of network storage technologies, such as NAS and SAN, have emerged to provide efficient data storage services through dedicated network storage devices or dedicated storage networks [3-4]. Note that in the high performance computing environment, compute nodes are often configured with large physical memory, and there is often free memory on each compute node in the process of computing. If these idle memory can be aggregated, it can provide a considerable amount of high-speed storage space. At the same time, in high performance computing, it is often the case that all computing nodes frequently share and access some file data. In large-scale distributed rendering applications, a large number of texture materials and object model files are frequently shared and accessed by all computing nodes [5-6]. If the idle memory of the above computing nodes is aggregated, the shared file data frequently accessed by each node is cached in the memory, and the network and disk I/O overhead for accessing this data is shielded. This greatly improves the overall system performance and reduces the access load of the shared file storage system.

With big data era coming, the high performance computing become indispensable tool for scientific research, multiple disciplines in the field of production development needs on high performance computing, these tend to data-intensive application, such as gene sequencing, oil seismic exploration, satellite remote sensing data processing, climate research and air dynamics simulation, etc. [6]. Lustre is a distributed parallel File System developed by Cluster File System formula. It is mature and has excellent performance. Its open source feature makes Lustre widely used in the field of high performance computing. About 60% of supercomputers in the Top500 list use Lustre system, and 6 of the top 10 supercomputers directly use Lustre system [8]. Google File System (GFS) is a distributed parallel File System developed by Google. GFS is designed for a large number of cheap ordinary computers, takes component failure as normal, and provides users with massive storage capacity and high performance services through reliability mechanism [9].

This paper designs and implements a distributed hybrid storage system based on DRAM and SSD for high performance computer, and optimizes the system.

2. Overall Design and Optimization of Distributed Systems for High Performance Computing

2.1. Overall System Design

DMFS is a user-mode distributed memory file system, which is mounted through the user-space file system FUSE, and can be deployed in any operating system that supports FUSE. Centralized metadata management is adopted, and the metadata management process is easy to analyze and optimize. Supports POSIX protocol. Applications can access the file system without modification. Open source system, easy to develop and debug, has the advantages of high performance, high availability, easy deployment and maintenance [10-11].

In order to achieve the design goal of high system reliability, this paper uses SSD to persist data blocks in DRAM space, eliminating the multi-copy mechanism and avoiding the use of valuable DRAM storage space by duplicates.

In order to achieve the high performance of the hybrid storage system, we design a data layout strategy that combines the characteristics of the two storage media, and migrates the data of the two storage Spaces reasonably.

The SSD storage capacity constitutes most of the storage space of the hybrid storage system, and the change of node memory capacity has a small impact on the data service of the system. Therefore,

we no longer use CG mode, and the data server is a separate Chunkserver, which simplifies the management in the Chunkserver. The communication and data overhead between Chunkservers are reduced, and the load of Chunkserver is reduced [12].

(1) Metadata management service module Master

The metadata management service module still uses the active/standby mode to improve the availability of the metadata service, detects anomalies through Keepalived and performs active/standby switchover. The metadata management module no longer maintains the block replicas and does not need to manage the migration of replicas between Chunkservers.

(2) Data server Chunkserver

The function of data server is to provide data storage and data access services. The data server manages the hybrid storage space internally and provides a unified storage view externally. The hybrid storage space is transparent to applications. The data server is also responsible for marking the data block value and migrating the data block based on the data block value and storage space usage.

(3) Client Client

The client retains POSIX protocol support and is mounted through FUSE.

The system implements a distributed hybrid storage system in user mode. The storage media are free memory resources in compute nodes and mounted SSDS. The system mainly provides large-capacity storage services with high bandwidth and low latency for data-intensive applications. To improve metadata access performance, the Master loads metadata to the memory of a node. Therefore, the Master must be deployed on a node with low CPU usage and sufficient memory resources. Currently, the compute nodes are not equipped with SSDS. Therefore, the Chunkserver is deployed on the compute nodes mounted with SSDS. The Client runs on any compute node that supports FUSE [13].

(4) Functional modules

The data store location module of the Client locates the data server where the access block resides based on the full path name and access offset of the accessed file. If it is the first time, you need to communicate with the Master to obtain metadata information and file layout information. If the file layout information is available, the system calculates the data server where the data block resides and establishes a connection for data transmission.

Space management module of the Chunkserver: The space management module is responsible for allocating and reclaiming DRAM and SSD storage space. The space management module feeds spatial information to the data migration module, and triggers data migration when DRAM space is insufficient [14]. The implementation of data migration needs the support of space management module. Storage space management determines whether the performance of the two storage media can be fully utilized, and directly affects the read and write process of hybrid storage.

Data migration module: Migration is based on data blocks. The data migration module migrates data blocks based on their value.

2.2. Cache Optimization Strategy

In the distributed file system used in this paper, metadata of all files is centrally managed by the metadata server, and each data block of a file has corresponding metadata information in the master. When an application's I/O request arrives at the client of the file system, it will first be added to the corresponding request queue. After accessing a file data block object, it will go to the metadata server master to obtain the metadata information corresponding to the data block. After the client

obtains the metadata information corresponding to the data block, Directly communicate with the corresponding DS to obtain the corresponding data. Every time the client requests for data block access, it needs to access the metadata server. When the client is in high concurrency I/O access mode, it will exert great access pressure to the metadata server [15-16].

The process analysis of the metadata service shows that every data read operation of the client communicates with the master. When multiple clients request metadata at the same time, it will cause great pressure on the master, which will affect the client I/O performance of the application. In high performance applications, the high performance of applications is usually the first priority. In typical high performance applications, the client access mode to files is 1:1, that is, a file is read and written by only one process, and most files are read and written consecutively at one time. Therefore, in order to relieve the pressure of the metadata server in the file system, prevent the client from requesting frequent access to the metadata server, and consider the series of I/O characteristics of high-performance applications, the metadata access management of the file system client can be optimized [17-18].

In order to alleviate the metadata service bottleneck when the client reads and writes data, this paper proposes to set up file metadata cache on the client of the file system. Optimizing metadata management mainly includes the following two aspects:

(1) Metadata cache on the client (inode corresponds to metadata information of all data blocks) Figure 1 shows the processing flow of metadata read request after metadata optimization. Whenever the chunk of a file is read for the first time, metadata information of all data blocks of the file corresponding to this chunk is cached on the client. Subsequently, the chunk metadata information does not need to access the master, thereby reducing the access load of the master.

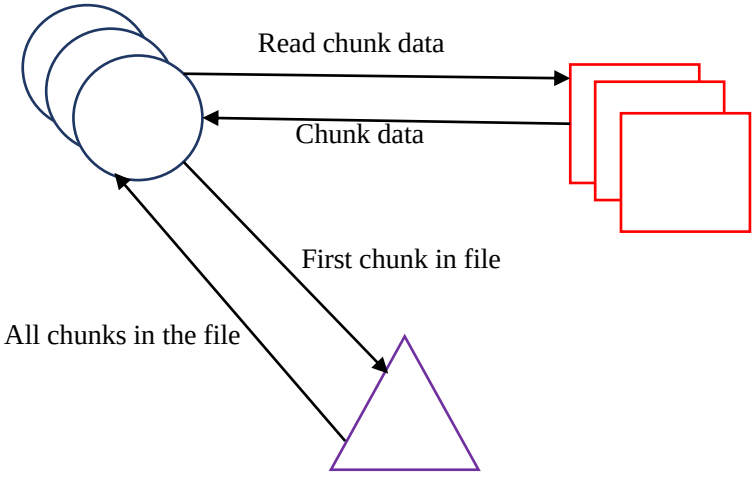

Figure 1. Read request process after metadata optimization

(2) Metadata cache is weakly consistent

The I/O process of a typical high-performance application is usually staged. To maintain the consistency of the metadata cache, the metadata cache is periodically released according to the phase duration of the high-performance application, so that the metadata cache is consistent with the master in the current file operation phase.

3. Simulation Experiment

In the design of the system, the block used to represent the data region is the basic unit of DMFS

file segmentation. If the block size is too large, the limited memory space will be wasted due to the large segmentation granularity. If the block size is too small, the amount of metadata will be large and the management efficiency will be reduced. As the basic unit of MC data area check and operation, sub-block is directly related to the speed of data check and operation. The selection of block and sub-block sizes has an important impact on system read/write performance, metadata scale, and management efficiency. Traditional distributed file systems such as GFS, HDFS, and MooseFS all use 64MB block size to meet the application scenarios of cloud computing and big data, while MooseFS supports 4MB block size and 4KB subblock size to meet the processing requirements of small files. Since the storage capacity of DMFS is in the tens of terabytes, the metadata size is not a primary concern, so the blocks can be relatively small, between 2MB and 8MB by test selection.

4. Analysis of Simulation Experiment Results

4.1. Sequential Read and Write Performance

Table 1. Sequential read performance comparison of different block sizes

	16	32	64	128
2MB	587	854	832	825
4MB	632	960	992	1041
8MB	596	971	1014	1176

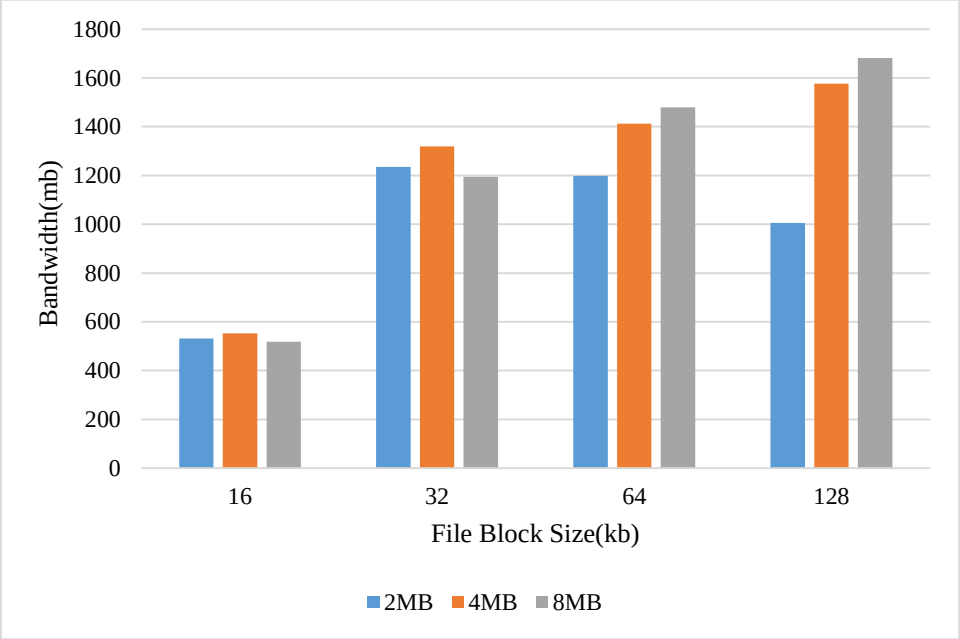

Figure 2. Sequential write performance comparison of different block sizes

As shown in Table 1 and Figure 2, the sequential read/write performance is compared. For sequential read, the performance of 2MB, 4MB and 8MB is close when the file block size is small, but when the file block size is large, 4MB and 8MB are better than 2MB. For sequential writes,

2MB and 4MB perform close to each other and are balanced across various block sizes, while 8MB performs better when the file block is large.

4.2. Random Read/Write Performance

Table 2. Random read performance comparison of different block sizes

	16	32	64	128
2MB	71	186	435	495
4MB	73	203	468	574
8MB	72	205	471	596

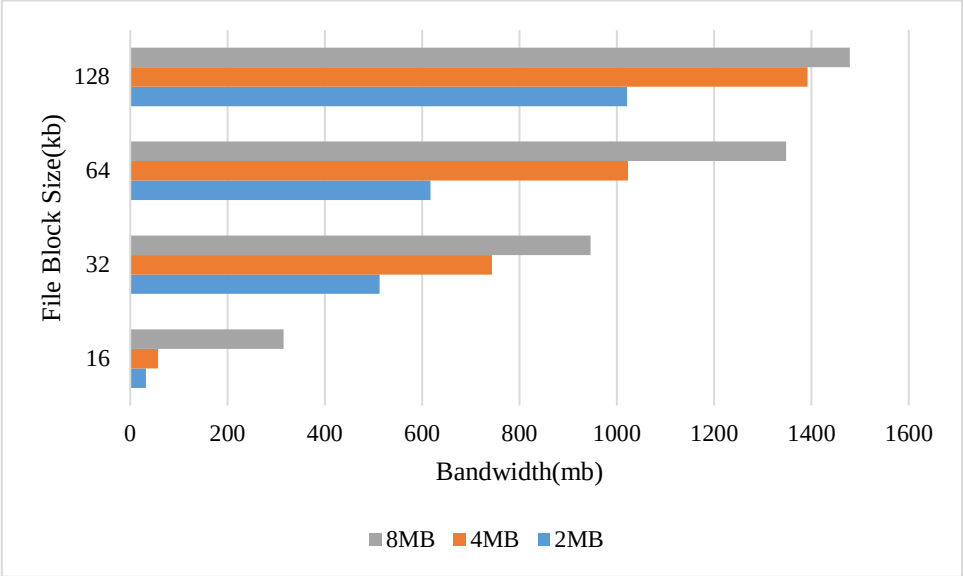

Figure 3. Random write performance comparison of different block sizes

As shown in Table 2 and Figure 3, the performance of random read/write is compared. For random read, the performance of 2MB, 4MB, and 8MB are close to each other. For random write, 8MB is more advantageous when the file block is small. Based on the above analysis, 8MB provides the most comprehensive performance, so the block size of the system is determined to be 8MB.

5. Conclusion

Aiming at the I/O bottleneck problem of high performance computer storage system, this paper presents the structure characteristics and idle resources oriented to high performance computer, and designs and implements a distributed memory file system. A distributed memory file system (DMFS) for high performance computer is designed and implemented. It implements metadata service with high availability, data grouping for dynamic expansion, object storage based on memory block and data transfer mechanism RBB based on RDMA. In this paper, the key technologies of distributed memory file system for high performance computer have been studied and some achievements have been made. It can make use of free memory and bandwidth resources,

relieve the pressure of existing storage systems to a certain extent, and improve the execution efficiency of I/ O-intensive applications. However, since distributed file system has always been a complex system engineering, in view of the time factor, some of the design or implementation of DMFS has been simplified, there are still a lot of valuable problems need to be further studied.

Funding

This article is not supported by any foundation.

Data Availability

Data sharing is not applicable to this article as no new data were created or analysed in this study.

Conflict of Interest

The author states that this article has no conflict of interest.

References

[1] Kumar H, Chauhan N K, Yadav P K. A High Performance Model for Task Allocation in Distributed Computing System Using K-Means Clustering Technique. International Journal of Distributed Systems and Technologies, 2018, 9(3):1-23. https://doi.org/10.4018/IJDST.2018070101

[2] Sierra R, Carreras C, Caffarena G. Witelo: Automated generation and timing characterization of distributed-control macroblocks for high-performance FPGA designs. Integration, 2019, 68(SEP.):1-11. https://doi.org/10.1016/j.vlsi.2019.05.001

[3] CJ Barrios Hernández, Gitler I, Klapp J. [Communications in Computer and Information Science] High Performance Computing Volume 697 || Distributed Big Data Analysis for Mobility Estimation in Intelligent Transportation Systems. 2017, 10.1007/978-3-319-57972-6(Chapter 11):146-160.

[4] Brinkmann A, Mohror K, Yu W, et al. Ad Hoc File Systems for High-Performance Computing. Journal of Computer Science and Technology, 2020, 35(1):4-26.

[5] Czarnul P, Proficz J, Drypczewski K. Survey of Methodologies, Approaches, and Challenges in Parallel Programming Using High-Performance Computing Systems. Scientific Programming, 2020, 2020(5):1-19. https://doi.org/10.1155/2020/4176794

[6] Mathe Z, Haen C, Stagni F. Monitoring performance of a highly distributed and complex computing infrastructure in LHCb. Journal of Physics Conference, 2017, 898(9):092028.

[7] Titov M, G Zaruba, De K, et al. A study of the applicability of recommender systems for the Production and Distributed Analysis system PanDA of the ATLAS Experiment. Journal of Physics Conference Series, 2018, 1085(4):042028.

[8] Roberto, Diversi, Andrea, et al. Thermal Model Identification of Computing Nodes in High-Performance Computing Systems. IEEE Transactions on Industrial Electronics, 2019, 67(9):7778-7788.

[9] Cuomo S, Galletti A, Marcellino L. A GPU parallel optimised blockwise NLM algorithm in a distributed computing system. International Journal of High Performance Computing and Networking, 2018, 11(4):304.

[10] Michal, Janczykowski, Wojciech, et al. Large-scale urban traffic simulation with Scala and high-performance computing system - ScienceDirect. Journal of computational science, 2019, 35(C):91-101. https://doi.org/10.1016/j.jocs.2019.06.002

[11] Nathan H, Vishal A, Farrens M K, et al. A Survey of End-System Optimizations for High-Speed Networks. ACM Computing Surveys, 2018, 51(3):1-36. https://doi.org/10.1145/3184899

[12] CaoNgocNguyen, SoonwookHwang, Jik-SooKim. Making a case for the on-demand multiple distributed message queue system in a Hadoop cluster. Cluster Computing, 2017, 20(3):2095–2106.

[13] Borghesi A, Molan M, Milano M, et al. Anomaly Detection and Anticipation in High Performance Computing Systems. IEEE Transactions on Parallel and Distributed Systems, 2020, PP(99):1-1.

[14] P L ópez, Baydal E. Teaching high-performance service in a cluster computing course. Journal of Parallel and Distributed Computing, 2018, 117(jul.):138-147. https://doi.org/10.1016/j.jpdc.2018.02.027

[15] Ko H, Pack S. Distributed Device-to-Device Offloading System: Design and Performance Optimization. IEEE Transactions on Mobile Computing, 2020, PP(99):1-1.

[16] Yokota R, Weiland M, Keyes D, et al. [Lecture Notes in Computer Science] High Performance Computing Volume 10876 || Zeno: A Straggler Diagnosis System for Distributed Computing Using Machine Learning. 2018, 10.1007/978-3-319-92040-5(Chapter 8):144-162. https://doi.org/10.1007/978-3-319-92040-5_8

[17] Reuther A, Byun C, Arcand W, et al. Scalable system scheduling for HPC and big data. Journal of Parallel and Distributed Computing, 2018, 111(jan.):76-92. https://doi.org/10.1016/j.jpdc.2017.06.009

[18] Han M, Park J, Baek W. Design and Implementation of a Criticality- and Heterogeneity-Aware Runtime System for Task-Parallel Applications. IEEE Transactions on Parallel and Distributed Systems, 2020, PP(99):1-1