Characterizing I/O Workloads of HPC Applications Through Online Analysis

Wenrui Dong*, Guangming Liu*†, Jie Yu*, You Zuo*
*School of Computer Science
National University of Defense Technology, Changsha, Hunan, China
†National Supercomputer Center in Tianjin, Tianjin, China
*Email: {dongwr, liugm, yujie, zuoyou}@nscc-tj.gov.cn

***Abstract*—The performance of storage subsystem of supercomputers can not meet the demands of complex applications running on them. One of its major causes is that the bandwidth of storage hardware has not been utilized efficiently due to the complex and changing application I/O behavior. Therefore, I/O characterization tools are vital to application development and orchestration of storage system.**

This paper proposes an I/O characterization tool called FTracer. It captures I/O traces and performs traces analysis at runtime. In order to provide more flexible analysis, this FTracer allows users to vary the analysis instances at runtime. This mechanism ensures users get what exactly they want about the I/O characteristics of their applications when applications are running. In this work, we characterize MADbench2 benchmark to demonstrate the ability of FTracer.

I. Introduction

I/O characterization tools are vital to application development and the orchestration of storage system[1], [2]. Many studies[3], [4] have proved that I/O characteristic is useful to improve caching efficiency and accuracy of prediction. Tuning file system parameters to achieve good performance can get profit from I/O characteristic as well. The tools to perform I/O characterization of applications can be classified into two categories. For the first type, tracing tools capture I/O traces and write them to files, such as Recorder, RIOT and //TRACE etc. For the second one, profiling tools provide statistical summaries by using counter at runtime trading off detailed traces for low storage and runtime overhead, such as Darshan[5].

There are three reasons promote us to design a new tracing and analyzing framework. First, many applications running on supercomputer are not written in MPI. We observed from TH-1A that some data-intensive applications in oil seismic exploration area and climate simulate are not written in MPI. Darshan can only profile file operations of MPI applications. For the initialization work of tracing I/O operation needed are done in `MPI_Init` operations. Second, the software I/O stack of many current supercomputers is abundant and is composed of several levels as shown in Figure 1. Each level provides end users portable data abstractions and performance optimization. Unfortunately, the inadequate combination of optimization strategy at each level can cause performance decrease of applications. Capture the I/O traces at all levels and do analysis refers to large amount of work, designers needs to re-implement all the interfaces of all levels. And the resulting trace files are big, which incurs extra overheads on storage system. Actually, capture I/O traces at all levels is unnecessary, all the I/O calls invoked in each level are ultimately processed by the VFS layer of the operating system and underlying file systems. Only tracing I/O calls at VFS level is sufficient to get the I/O access characteristics of applications and to check the effect of combination of optimization strategy. Third, Most analysis tools are post-processed after applications running, which incurs extra I/O workloads and computation.

In this paper, we present a I/O characterizing tool called FTracer. It captures I/O traces of applications at VFS level by using FUSE framework and perform on-line analysis on traces to get I/O characteristics of parallel applications. FTracer is flexible for the following reasons. In tracing, the traces can be kept to storage or discarded at runtime by modifying a parameter in configure file. In online analysis, the analysis instance can be replaced when application is running, therefore users can adjust the perspective of on-line analysis.

II. Design and Implementation of FTracer

A. *Fuse-based tracer*

FTracer is implemented in user-space with FUSE. FUSE is an easy-to-use framework for users of Unix-like operating systems to create their own file system in user space without modifying kernel code. The FUSE framework is comprised of two mayor components as shown in Figure 1 with orange blocks: the fuse kernel module and the libfuse library. The fuse kernel module interacts with the VFS interface, intercepts the I/O operations and handles them with the kernel level interface. The libfuse library provides a high level API to develop file system in user space.

Figure 1 illustrates the design of our Fuse-based trace tool, called FTracer. In essence, FTracer is a file system and should be mounted on the operating system firstly before used. After mounted, I/O operations to files on FTracer will be handled by the routines we defined in user-space. The I/O routines FTracer implemented are quite simple, it first records the I/O traces in a buffer allocated when FTracer is mounted, then redirects the I/O requests to other file systems to carry on the real open, read or write operations.

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

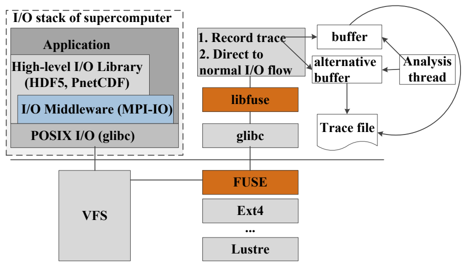

Fig. 1. The software stack of supercomputer and design overview of FTracer

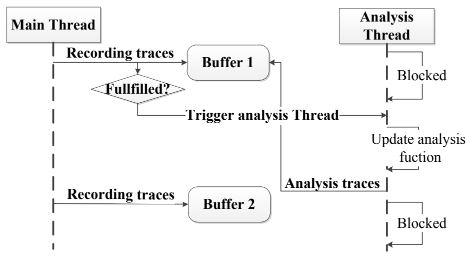

Fig. 2. The cooperation of main thread and analysis thread to finish tracing and analysis

B. Online Analysis

The on-line analysis to traces is taken by a daemon thread. When FTracer starts, it creates a thread and blocks it. FTracer keeps two equal size buffers to store traces. As soon as one trace buffer is full-filled, the other trace buffer is used to store traces. Meanwhile, master thread signals the daemon thread and arises it to analyze traces in the fullfilled buffer. After analysis finished, the buffer is cleared and ready to record traces. Figure 2 illustrates the cooperation of main thread and analysis thread to finish tracing and on-line analysis. The buffer size can be configured before FTracer is mounted, in our implementation is 16MB and about 50000 traces accommodated. The two buffers mechanism guarantees the concurrency of the recording and analysis of traces. The two threads have no effect on each other, therefore, analyzing the traces to obtain I/O characteristic of the running application on-line do not incur extra overhead of tracing. In addition, the analyzing thread is blocked while there is no full-filled buffer to be processed. In other word, the analyzing thread is blocked most of the time and the waste of CPU resources is avoided.

FTracer provides a mechanism to enable users to replace or add their new trace analysis algorithm at runtime. The analysis algorithms are stored to a dynamically loaded library. The analysis thread first open this library by call `dlopen` function, then update the analysis algorithm with `dlsym` function. When users finish new analysis algorithms, they can compile them into the dynamically loaded library(in our implementation is `libanalysis.so`) and replace the old version of this library with the new one. In the subsequent analysis stage, the new algorithm is used to perform analysis. FTracer currently provides three types of analysis routine for users who do not want to write analysis function themselves, respectively under three perspectives, which is from process, node and file.

III. Benchmark Study

MADBench2 benchmark is based on MADspec code which is a cosmology application. In this study, we perform MADBench2 with 16 compute nodes on TH-1A, and MADBench2 accesses an single shared file.

The runtime analysis to I/O traces provided by FTracer can used to get bandwidth related information. As Figure 3 illustrate, the first write phase of MADBench2 takes about

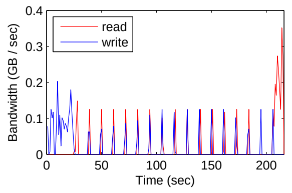

Fig. 3. The read/write bandwidth of MADBench2 benchmark

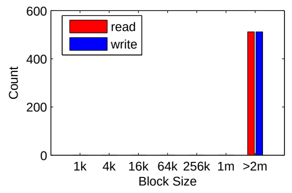

Fig. 4. The distribution of I/O access size of MADBench2 benchmark

25s, and the maximum write bandwidth is about 200MB/s. The reason of low bandwidth is that the written file is not striped among multiple OSTs of Lustre, therefore the 200MB/s is achieved by one OST. The distribution of I/O size of MADBench2 is shown in Figure 4, all the read/write sizes are larger than 2MB, actually, the I/O size can be configured by users.

IV. Conclusion

We have designed and implemented a FUSE-based I/O characterization tool called FTracer. FTracer captures the I/O traces of applications and perform online traces analysis meanwhile. The functions used to analyze traces can be replaced at runtime when application is still running. This mechanism ensures that users can get what exactly they want about the I/O characteristics.

References

[1] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, “Understanding and improving computational science storage access through continuous characterization,” *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, p. 8, 2011.

[2] S. A. Wright, S. D. Hammond, S. J. Pennycook, R. F. Bird, J. Herdman, I. Miller, A. Vadgama, A. Bhalerao, and S. A. Jarvis, “Parallel file system analysis through application i/o tracing,” *The Computer Journal*, vol. 56, no. 2, pp. 141–155, 2013.

[3] Y. Zhao, K. Yoshigoe, and M. Xie, “Pre-execution data prefetching with inter-thread i/o scheduling,” in *Supercomputing*. Springer, 2013, pp. 395–407.

[4] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X.-H. Sun, “I/o acceleration with pattern detection,” in *Proceedings of the 22nd international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2013, pp. 25–36.

[5] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, “24/7 characterization of petascale i/o workloads,” in *Cluster Computing and Workshops, 2009. CLUSTER’09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.