Direct-FUSE: Removing the Middleman for High-Performance FUSE File System Support

Yue Zhu† Teng Wang† Kathryn Mohror‡ Adam Moody‡ Kento Sato‡ Muhib Khan† Weikuan Yu†
†Florida State University ‡Lawrence Livermore National Lab
{yzhu, twang, khan, yuw}@cs.fsu.edu {kathryn, moody20, sato5}@llnl.gov

I. Introduction

An efficient file system is important for high-performance computing (HPC) systems in supporting large-scale scientific applications. In general, kernel-level file systems are generalized for broader Unix-like environments, while user-level file systems are designed with special-purpose features for particular I/O workloads. The Filesystem in Userspace (FUSE) [3] is a software framework for Unix-like file systems, which allows non-privileged users to create their file systems without kernel-based file system implementations.

Since most user-level file systems are designed to support particular I/O workloads, different file systems can be used for different kinds of data in a single job. However, interacting with multiple FUSE file systems is a complicated task, due to the communication round trip between an application program and file system processes, and the need of root privilege to mount file systems. Our approach is Direct-FUSE, a framework to support user-level file systems without crossing the kernel boundary. It is built on top of *libsysio* [1], which is developed by Sandia Scalable I/O team and provides a POSIX-like interface redirecting I/O function calls to particular file systems.

II. The Overview of Direct-FUSE

Direct-FUSE is designed to support various user-level file systems in one job with a unified POSIX-like interface, while bypassing the FUSE kernel for less overheads. Direct-FUSE mainly consists of three layers, i.e., *adapted-libsysio*, *lightweight-libfuse*, and *backend services*.

First, adapted-libsysio is adapted from the original libsysio to support multiple FUSE file systems, in which we enable different methods for file path and file descriptor operations to identify various backend services for applications. Second, unlike the original libfuse that includes block devices registrations and helps bridging file system processes and kernel module, our lightweight-libfuse exposes only abstract file system operations to the underlying backend services. Third, the backend services incorporate multiple user-level file systems that can deal with different kinds of data. For example, we can enable both direct access to remote volumes through SSHFS and the straight communication from application to GlusterFS deamons. In addition, all three layers are kept in user space, which helps Direct-FUSE avoid the inter-process communication of the original FUSE file system resulted from the use of FUSE kernel module.

III. Evaluation

In our tests, we compare the I/O performance of a FUSE file system, the Direct-FUSE, and a native file system on different storage medias. The experiments are conducted on machines that are equipped 64 GB memory, and a 1 TB Seagate ST91000640NS SATA disk with 64 MB cache. Ext4 and tmpfs are used as the underlying file system on disk and RAMDisk, correspondingly.

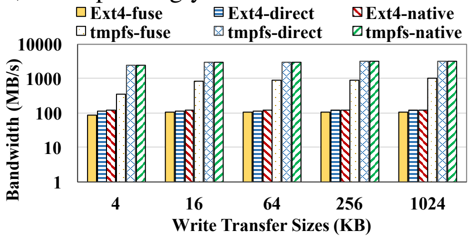

Fig. 1: Sequential write bandwidth measured via Iozone

As shown in Fig. 1, *Ext4-fuse* is a FUSE file system on top of the native Ext4 file system, *Ext4-direct* is our Direct-FUSE on top of Ext4, and *Ext4-native* is the original Ext4. Similarly, *tmpfs-fuse*, *tmpfs-direct*, and *tmpfs-native* denote a FUSE file system on tmpfs, Direct-FUSE on tmpfs, and the original tmpfs on RAMDisk, respectively. Due to the space limitation, we only show the sequential bandwidth in this section. The sequential write bandwidths are measured by Iozone benchmark [2].

Fig. 1 shows that, *Ext4-direct* outperforms *Ext4-fuse* by 11.9% on average. In addition, our design only loses at most 2.5% of I/O bandwidth when compared with *Ext4-native*. A similar trend is also observed on tmpfs results due to relatively low latency of RAMDisk. Therefore, the overhead of Direct-FUSE on RAMDisk is slightly higher (at most 5.7%) than the overhead on disk. Overall, our Direct-FUSE delivers higher I/O bandwidth compared to the FUSE-native file systems on both disk and RAMDisk. It also demonstrates comparable performance to the underlying file system.

Acknowledgment This work is performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-ABS-740686

References

[1] The SYSIO library. https://sourceforge.net/projects/libsysio/.
[2] William D Norcott and Don Capps. Iozone Filesystem Benchmark, 2003.
[3] Miklos Szeredi et al. Fuse: Filesystem in Userspace. *Accessed on*, 2010.