Finesse: Kernel Bypass for File Systems

Matheus Stolet*
University of British Columbia
stolet@cs.ubc.ca

Tony Mason†
University of British Columbia
fsgeek@cs.ubc.ca

Abstract

Finesse implements a kernel bypass technique for enhancing the performance of the popular cross-platform FUSE file system framework, without requiring intrusive changes to existing applications *or* user-mode file systems. Further, Finesse provides a more extensible model that permits enhancement of applications by shifting expensive meta-data intensive operations into the FUSE file system. Thus, Finesse offers a compelling improvement for FUSE file system developers: existing applications benefit from improved performance without program changes and without requiring file system source code changes, as well as permitting application developers to exploit new functionality that is exposed by the Finesse-enhanced file system, as well as functionality extensions implemented by file systems developers.

1 Introduction

Kernel programming is notorious for being a challenging development environment. Despite this complexity, production use file systems are frequently implemented for in-kernel execution because they offer the best performance. Programming in user space is more forgiving and has a broader range of well-supported languages, libraries, and options that are not available in the kernel environment. The cost of using userspace file systems development tools, such as FUSE [3], is typically performance. Recent work has pointed this out and looked at various ways of improving performance [2, 4].

Prior work has explored various ways of improving performance, including interception libraries [5] and kernel mode extensions for optimizing data copy [1]. The idea of implementing a hybrid combination of both kernel bypass and fallback kernel support has not been explored.

2 Description

Finesse explores this hybrid environment by modifying the behavior of the existing system libraries, which enables enhancing existing applications without requiring any program changes, and by modifying the FUSE file system library, which enables supporting existing FUSE file systems with a simple recompilation.

Finesse uses a simple message passing model for implementing a kernel bypass between the applications and user mode FUSE file systems. Existing applications can eschew using Finesse by using the original system libraries without any loss of functionality. Currently, we achieve replacement of existing system libraries on Linux by using the LD_PRELOAD mechanism, which ensures the Finesse application library is invoked first. When Finesse determines that the call is directed to a mounted FUSE file system, it converts that call to a message, which is passed directly to the FUSE file system.

We have observed up to a 61% performance improvement with Finesse with the *unlink* system call, versus using the standard FUSE interface. In this test, we added *unlink* support to the FUSE pass-through file system, preallocated four million 4KB files in multiple directories, and measured the time to delete them. Finesse was 29% *faster* than the native file system, and 61% faster than FUSE alone. Finesse is in active development and we continue to increase the number of file system operations supported by Finesse.

3 Conclusion

Finesse combines a kernel bypass message passing architecture with a fallback traditional file systems support layer. Preliminary results for this approach have achieved up to 62% better performance on one operations. We anticipate further positive results as we continue expanding our work. In future, we will also explore adding file systems API enhancements that will yield both better performance as well as novel functionality.

References

[1] Ashish Bijlani and Umakishore Ramachandran. 2019. Extension Framework for File Systems in User space. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 121–134. https://www.usenix.org/conference/atc19/presentation/bijlani

[2] Mingkai Dong, Heng Bu, Jifei Yi, Benchao Dong, and Haibo Chen. 2019. Performance and Protection in the ZoFS User-Space NVM File System. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP 19)*. Association for Computing Machinery, New York, NY, USA, 478493. https://doi.org/10.1145/3341301.3359637

[3] Nikolaus Rath. [n.d.]. FUSE: Filesystem in Userspace. https://github.com/libfuse/libfuse.

[4] Bharath Kumar Reddy Vangoor, Vasily Tarasov, and Erez Zadok. 2017. To FUSE or Not to FUSE: Performance of User-Space File Systems.. In *FAST*. 59–72.

[5] Steven A Wright, Simon D Hammond, Simon J Pennycook, Iain Miller, John A Herdman, and Stephen A Jarvis. 2012. LDPLFS: improving I/O performance without application modification. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 1352–1359.

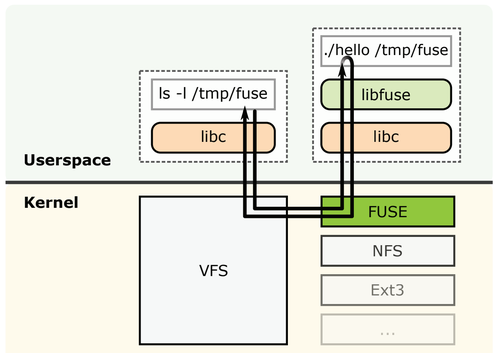
*Student, Presenter
†Student

Finesse: Kernel Bypass for FUSE File Systems

Matheus Stolet
stolet@cs.ubc.ca
University of British Coumbia, Vancouver BC

Tony Mason
fsgeek@cs.ubc.ca
University of British Coumbia, Vancouver BC

File Systems In User Space



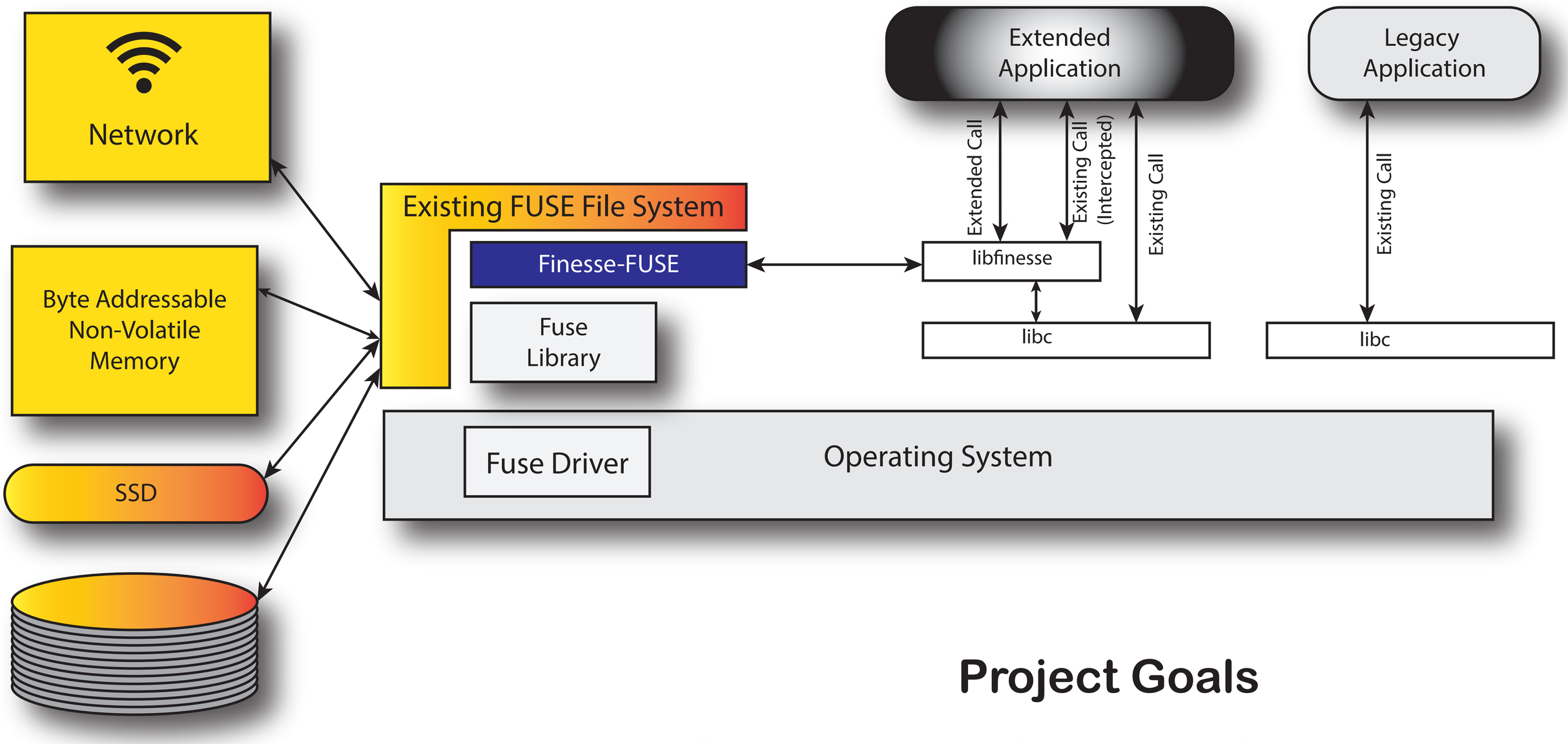
Source: https://commons.wikimedia.org/w/index.php?curid=3009564

Challenges

- Large body of existing applications (numerous APIs to support)
- Rigid interfaces
- Implicit state information
- Performance
- Existing semantics

Advantages

- Large body of existing applications (easy to benefit from Finesse)
- Cross-platform support
- User mode development (easier)
- Easy to prototype/build
- Well understood semantics



Preliminary Results

Workload	I/O Size (KB)	Ext4 HDD (ops/sec)	StackfsBase HDD (%Diff)	StackfsOpt HDD (%Diff)	Finesse HDD (%Diff)	EXT4 SSD (ops/sec)	StackfsBase SSD (%Diff)	StackfsOpt SSD (%Diff)	Finesse SSD (%Diff)
files-cr-1th	4	16406	-155.16	-247.01	-151.96	25622	-21.04	-13.79	-268.03
files-cr-32th	4	20186	-1122.04	-169.25	-1654.03	46183	-17.00	-32.26	-3826.01
files-rd-1th	4	164	-22.17	-107.40	-1.03	5803	-25.87	-25.66	-480.56
files-rd-32th	4	432	-174.39	-86.68	-375.78	65779	-18.99	-18.73	-6218.78
files-del-1th	-	685	-164.54	-235.32	+97.77	21465	-11.38	-14.94	+29.21
files-del-32th	-	301	+80.28	-18.00	+98.36	20000	-14.93	-20.12	-13.34

Green class (marked with +) indicates that the performance either degraded by less than 5% or actually improved; Yellow class includes results with the performance degradation in the 5–25% range; Orange class indicates that the performance degradation is between 25–50%; And finally, the Red class is for when performance decreased by more than 50%

Workload Name	Description
files-cr-Nth	Nthreads (1,32) create 4 million 4KB files over many directories
files-rd-Nth	Nthreads (1,32) read from 1 million preallocated 4KB files over many directories
files-del-Nth	Nthreads (1,32) delete 4 million preallocated 4KB files over many directories

Project Goals

- **Transparent support for existing applications**
- **No source code changes to existing FUSE file systems**
- **Optimize meta-data paths**
- **Extend support for more calls**
- **Enhance message passing performance**
- **Integrate with other FUSE extensions (e.g., extFUSE)**
- **Increase kernel bypass**