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

Developing kernel file systems presents challenges due to the complexity of the environment. The inflexibility of this environment discourages exploration of alternatives; most file systems research is focused on improving the storage management aspects, rather than the application facing interfaces. While userspace file systems mitigate the development complexity, they do so with a significant performance penalty and little flexibility for exploring novel access models. Finesse addresses these two concerns by adding both a client side library and a FUSE file system library. Finesse implements a kernel bypass mechanism for key operations and enables making new interfaces available to applications. Improving performance while balancing compatibility against flexibility to explore alternative interfaces is compelling for userspace file systems development.

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 [2], is typically performance. Recent work has pointed this out and looked at various ways of improving performance [1, 3].

Prior work has explored various ways of improving performanc, including interception libraries [4] and kernel mode extensions for optimizing data copy [**?**]. The idea of implementing a hybrid combination of both kernel bypass and fallback kernel support has not been explored.

2 Model

Finesse explores such a hybrid environment by introducing a client side shared library that intercepts a subset of calls and provides an alternative implementation, as well as an enhanced FUSE library model. This combination permits Finesse to provide an efficient kernel bypass mechanism. This is achieved using an efficient message passing interface between the client application and the FUSE file system library. Operations that are not implemented by the Finesse library fall back to system calls and standard FUSE behavior.

The Finesse application library is a user-facing library that allows the explicit invocation of operations implemented in the Finesse library and the implicit invocation of operations through LD_PRELOAD, that loads the Finesse library before other shared libraries such as *libc*. The Finesse FUSE extension lies between the FUSE file system and the FUSE library. It listens to incoming file system operations from the Finesse library and redirects them to the FUSE file system.

3 Evaluation

Finesse is still in development, and we are working at porting a bigger number of file system operations to use the Finesse library instead of defaulting to regular FUSE behavior. Preliminary evaluations have showed that the implementation of *unlink* using the Finesse library led to considerable performance improvements for a number of operations. For example, when deleting 4 million preallocated 4KB files over many directories, the existing FUSE library we tested showed a 15% decrease in performance. The same test done with the Finesse+FUSE library yielded a 29% performance gain.

4 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 44% better performance on specific operations, and we anticipate further positive results as we continue expanding our work. In future, we will explore adding file systems API enhancements that will yield both better performance as well as novel functionality.

References

[1] 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* (Huntsville, Ontario, Canada) *(SOSP 19)*. Association for Computing Machinery, New York, NY, USA, 478493. https://doi.org/10.1145/3341301.3359637

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

[3] 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.

[4] 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.

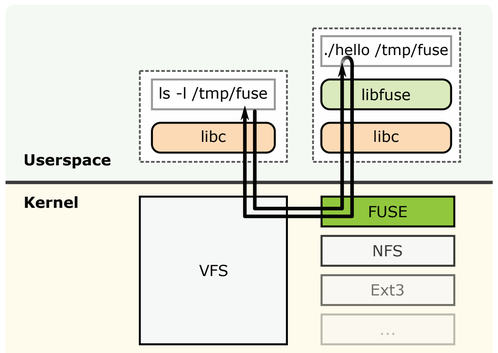
*Student, Presenter
†Student

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



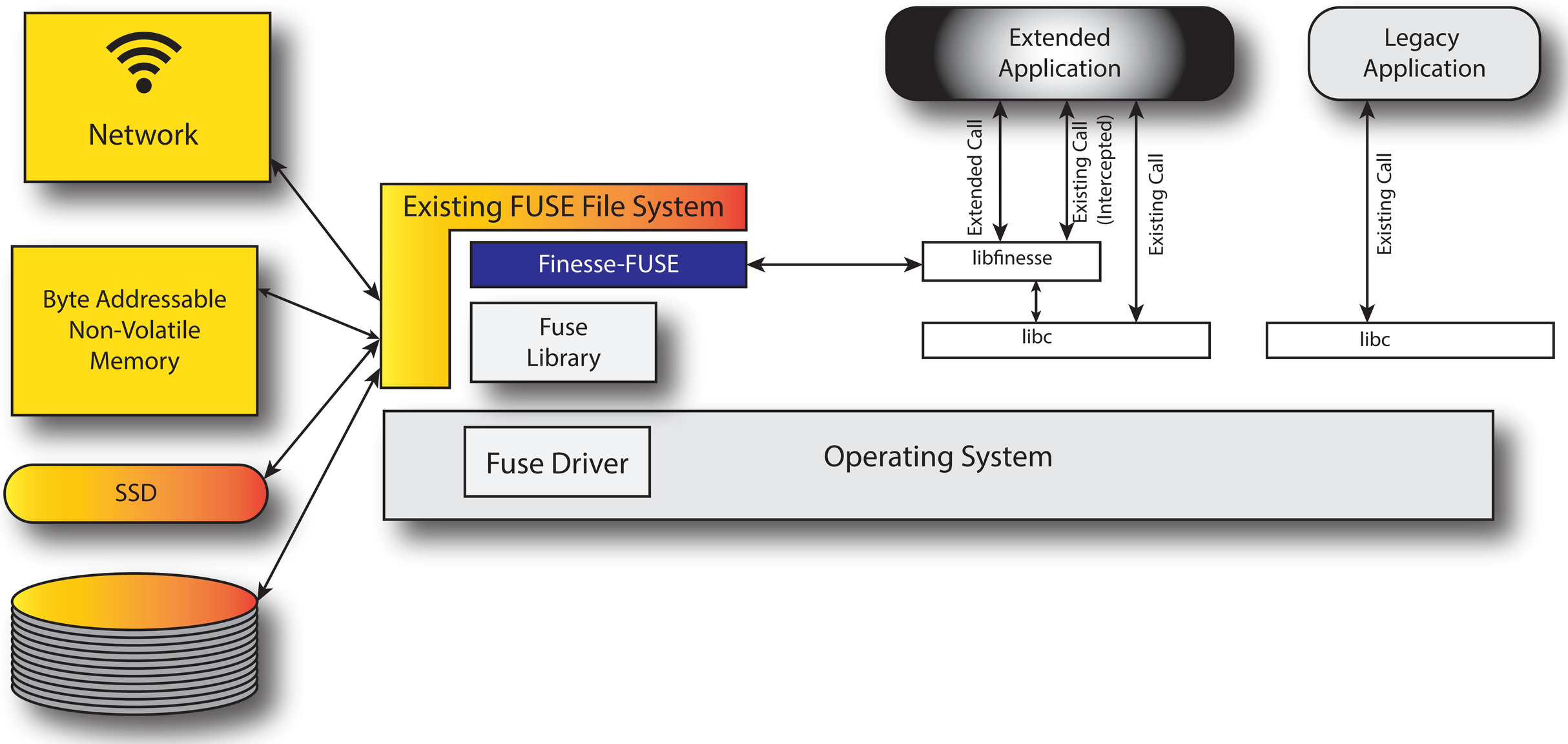
Source: https://commons.wikimedia.org/w/index.php?curid=3009564

Challenges:

- Large body of existing applications
- Rigid interfaces
- Implicit state information
- Performance
- Existing semantics

Advantages:

- Large body of existing applications
- 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

- Optimize meta-data paths
- Extend support for more calls
- Enhance message passing performance
- Integrate with other FUSE extensions (e.g., ext-FUSE)
- More kernel bypass