# 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 Laboratory+


# Outline

- **Background & Motivation**
- Design
- Performance Evaluation
- Conclusion

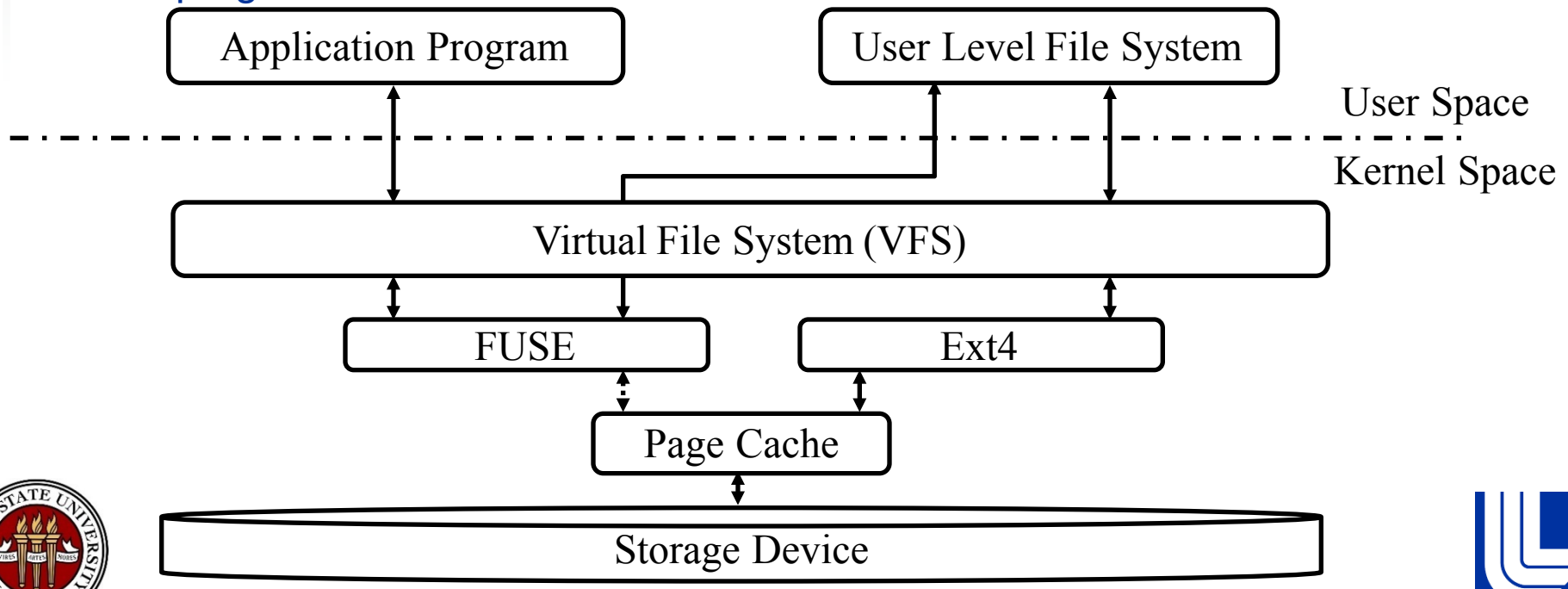

# Introduction

- High-performance computing (HPC) systems needs efficient **file system** for supporting large-scale scientific applications
  - Different file systems are used for different kinds of data in a single job
  - Both kernel- and user-level file systems can be used in the applications
  - Due to kernel-level file systems' development complexity, reliability and portability issues, user-level file systems are more leveraged for particular I/O workloads with special purpose

- Filesystem in UserSpacE (FUSE)
  - A software interface for Unix-like computer operating systems
  - It allows non-privileged users to create their own file systems without modifying kernel code
  - User defined file system is run as a separate process in user-space
  - Example: SSHFS, GlusterFS client, FusionFS(BigData'14)

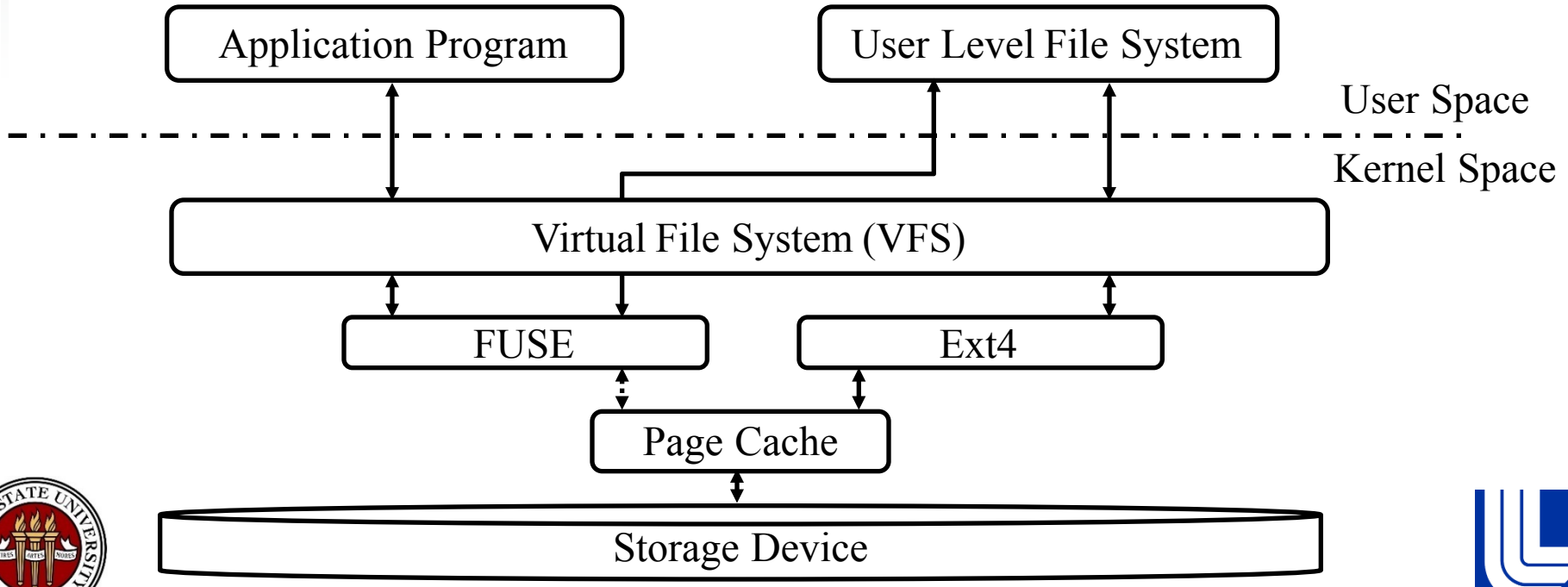

# How does FUSE Work?

- Execution path of a function call
  - Send the request to the user-level file system process
    - App program → VFS → FUSE kernel module → User-level file system process
  - Return the data back to the application program
    - User-level file system process → FUSE kernel module → VFS → App program



# FUSE File System vs. Native File System

| | FUSE File System | Native File System |
|---|---|---|
| # User-kernel Mode Switch | **4** | 2 |
| # Context Switch | **2** | 0 |
| # Memory Copies | **2** | 1 |



# Number of Context Switches & I/O Bandwidth

- The complexity added in FUSE file system execution path causes performance degradation in I/O bandwidth
  - *tmpfs*: a file system that stores data in volatile memory
  - *FUSE-tmpfs*: a FUSE file system deployed on top of tmpfs
  - *dd* micro-benchmark and *perf* system profiling tool are used to gather the I/O bandwidth and the number of context switches
  - Experiment method: continually issue 1000 writes

| | Write Bandwidth | | # Context Switches | |
|---|---|---|---|---|
| Block Size (KB) | **FUSE-tmpfs (MB/s)** | tmpfs (GB/s) | FUSE-tmpfs | tmpfs |
| 4 | **163** | 1.3 | 1012 | 7 |
| 16 | **372** | 1.6 | 1012 | 7 |
| 64 | **519** | 1.7 | 1012 | 7 |
| 128 | **549** | 2.0 | 1012 | 7 |
| 256 | **569** | 2.4 | **2012** | 7 |


# Breakdown of Metadata & Data Latency

- The actual file system operations (i.e. metadata or data operations) only occupy a small amount of total execution time
  - Tests are on tmpfs and FUSE-tmpfs
  - **Real Operation** in metadata operation: the time of conducting operation
  - **Data Movement**: the actual time of write in a complete write function call
  - **Overhead**: the cost besides the above two, e.g. the time of context switches

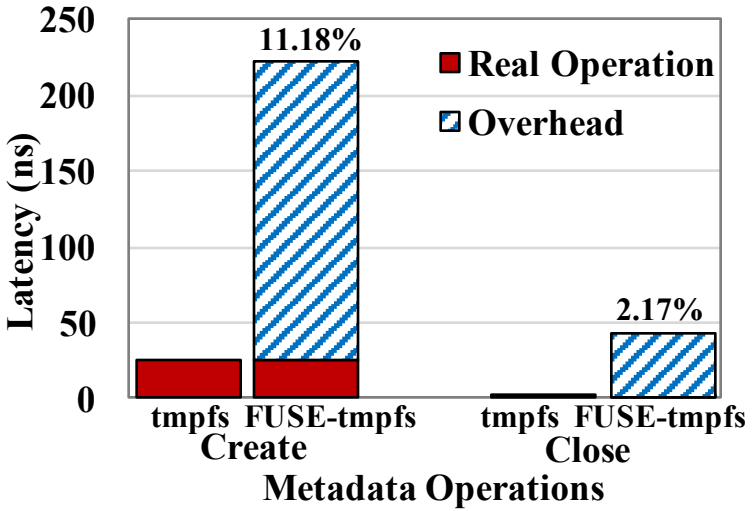



Fig. 1. Time Expense in Metadata Operations

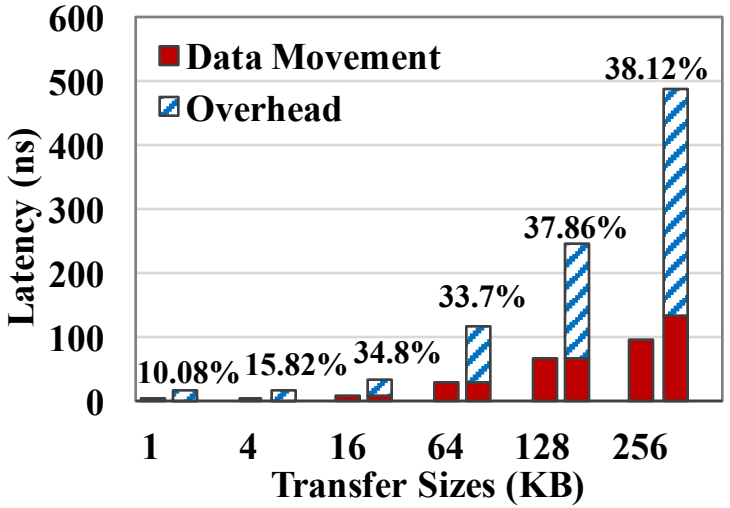



Fig. 2. Time Expense in Data Operations

# Existing Solution and Our Approach

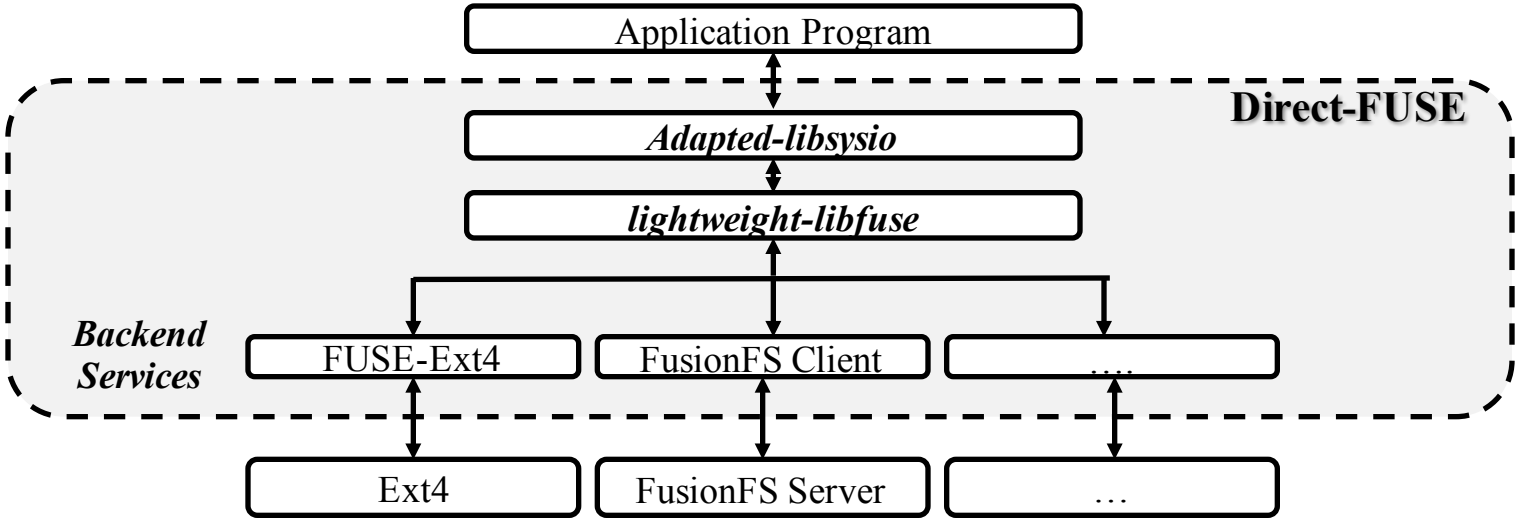
- How to reduce the overheads from FUSE?
  - Build an independent user-space library to avoid going through kernel (e.g., IndexFS (SC'14), FusionFS)
  - However, this approach cannot support multiple FUSE libraries with distinct file paths and file descriptors
- We propose Direct-FUSE to support multiple backend I/O services to an application
  - We adapted *libsysio* to our purpose in Direct-FUSE
    - ***libsysio*** *i*s developed by Scalability team of Sandia National Lab):
      - a POSIX-like file I/O, and name space support for remote file systems from an application's user-level address space.


# Outline

- Background & Motivation
- **Design**
- Performance Evaluation
- Conclusion


# The Overview of Direct-FUSE

- Direct-FUSE mainly consists of three components
  1. Adapted-libsysio
     - Intercept file path and file descriptor for backend services identification
     - Simplify metadata and data execution path in original libsysio
  2. lightweight-libfuse (not real *libfuse)*
     - Abstract file system operations from backend services to unified APIs
  3. Backend services
     - Provide defined file system operations (e.g., FusionFS)



# Path and File Descriptor Operations

- To facilitate the interception of file system operations for multiple backends, the operations are categorized into two:
  1. File path operations
     i. Intercept prefix and path (e.g., sshfs:/sshfs/test.txt) and return mount information
     ii. Look up corresponding inode based on the mount information, and redirect to defined operations
  2. File descriptor operations
     i. Find *open-file* record based on given file descriptor
        - *Open-file record* contains pointers to inode, current stream position, etc
     ii. Redirect to defined operations based inode info in *open-file* record


# Requirements for New Backends

- Interact with FUSE high-level APIs
- Separated as an independent user-space library
  - The library contains the fuse file system operations, initialization function, and also the unmount function
  - If a backend passes some specialized data to the fuse module via fuse_mount(), then the data has to be globalized for later file system operations
- Implemented in C/C++ or has to be binary compatible with C/C++


# Outline

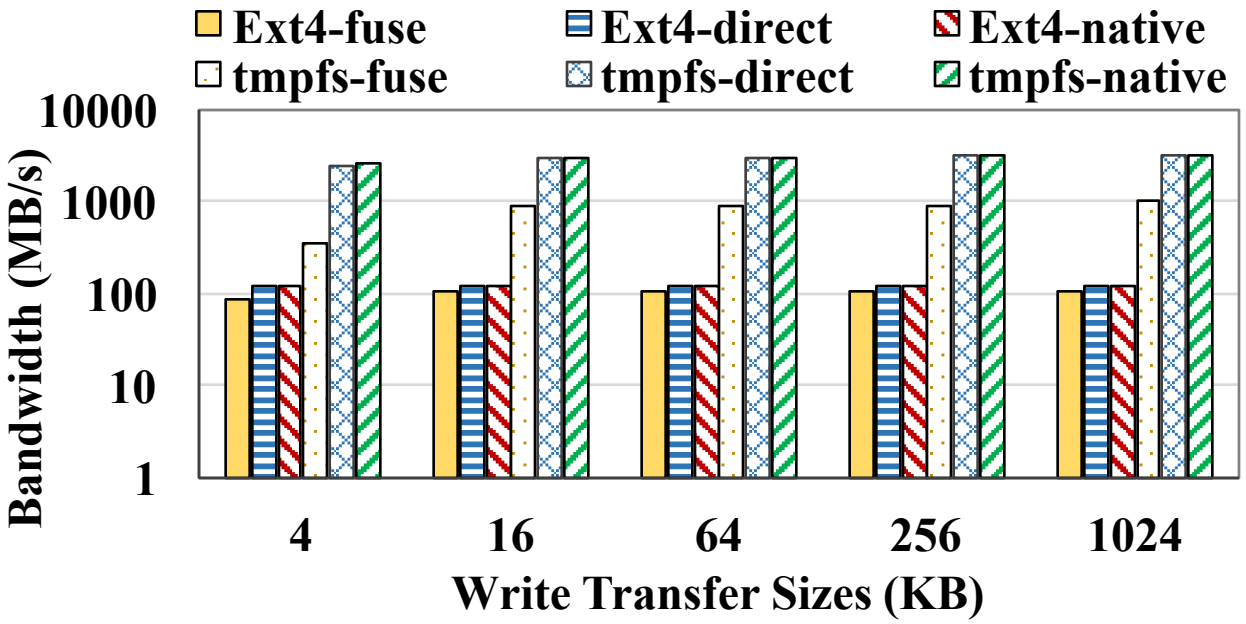
- Background and Challenges
- Design
- **Performance Evaluation**
- Conclusion


# Experimental Methodology

- We compare the bandwidth of Direct-FUSE with local FUSE file system and native file system on disk and memory by Iozone
  - Disk
    - Ext4-fuse: FUSE file system overlying Ext4
    - Ext4-direct: Ext4-fuse bypasses the FUSE kernel
    - Ext4-native: original Ext4 on disk
  - Memory
    - tmpfs-fuse, tmpfs-direct, and tmpfs-native are similar to the three tests on disk
- We also compare the I/O bandwidth of distributed FUSE file system with Direct-FUSE
  - FusionFS: a distributed file system that supports metadata- and write-intensive operations

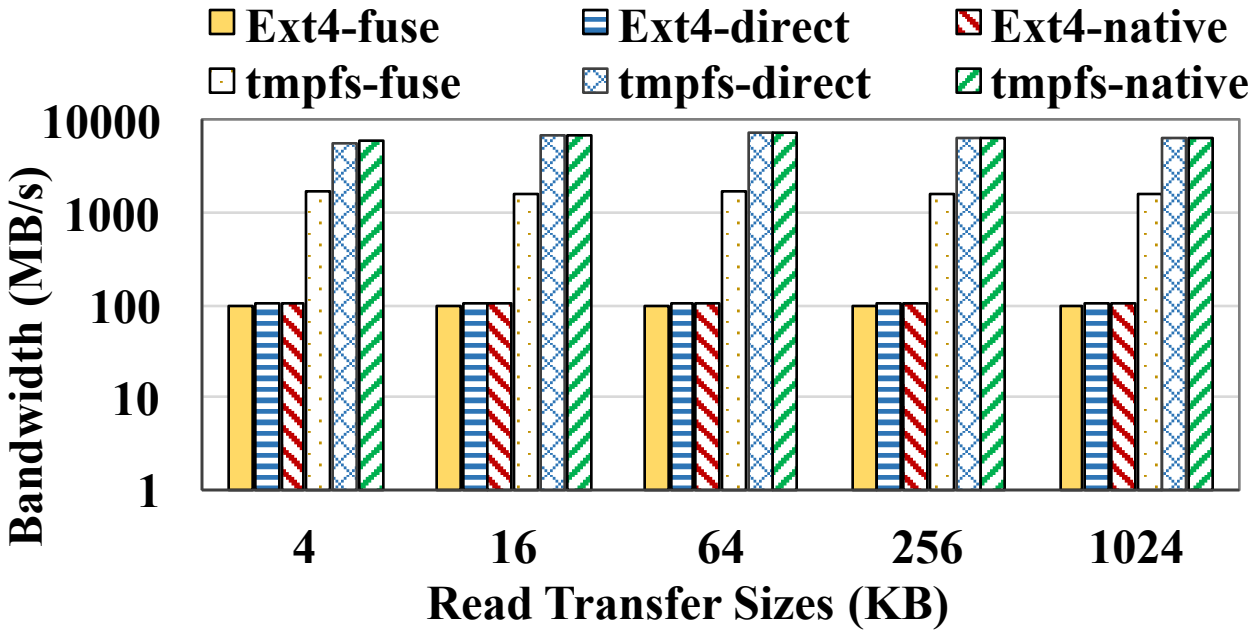

# Sequential Write Bandwidth

- Direct-FUSE achieves comparable bandwidth performance to the native file system
  - Ext4-direct outperforms Ext4-fuse by 16.5% on average
  - tmpfs-direct outperforms tmpfs-fuse at least 2.15x



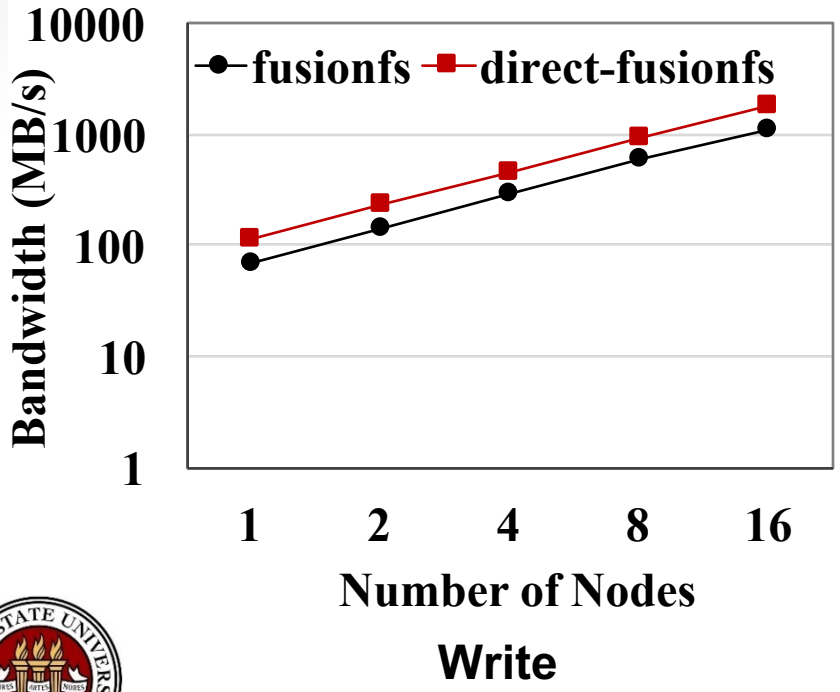
# Sequential Read Bandwidth

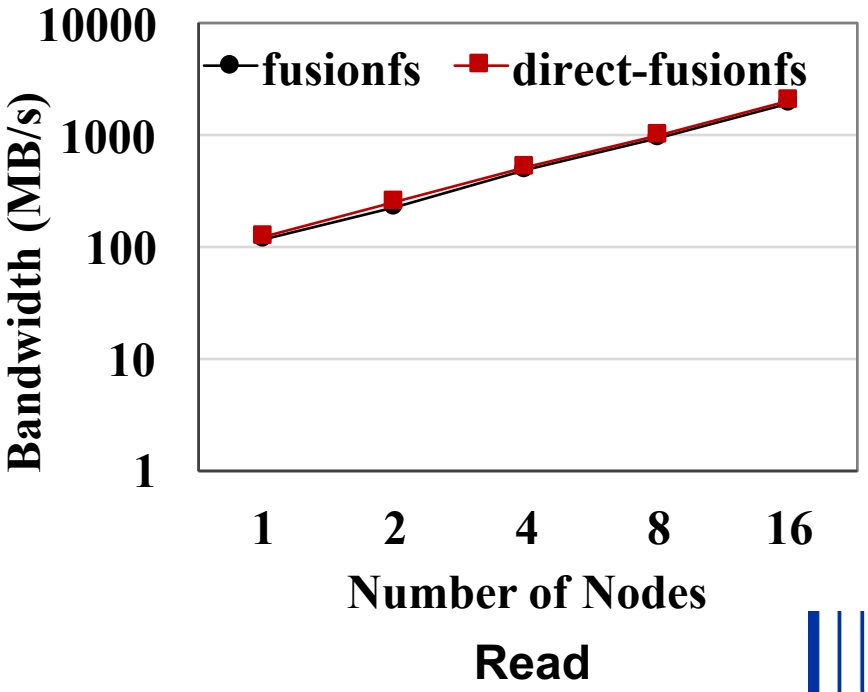
- Similar to the sequential write bandwidth, the read bandwidth of Direct-FUSE is comparable to the native file system
  - Ext4-direct outperforms Ext4-fuse by 2.5% on average
  - tmpfs-direct outperforms tmpfs-fuse at least 2.26x



# Distributed I/O Bandwidth

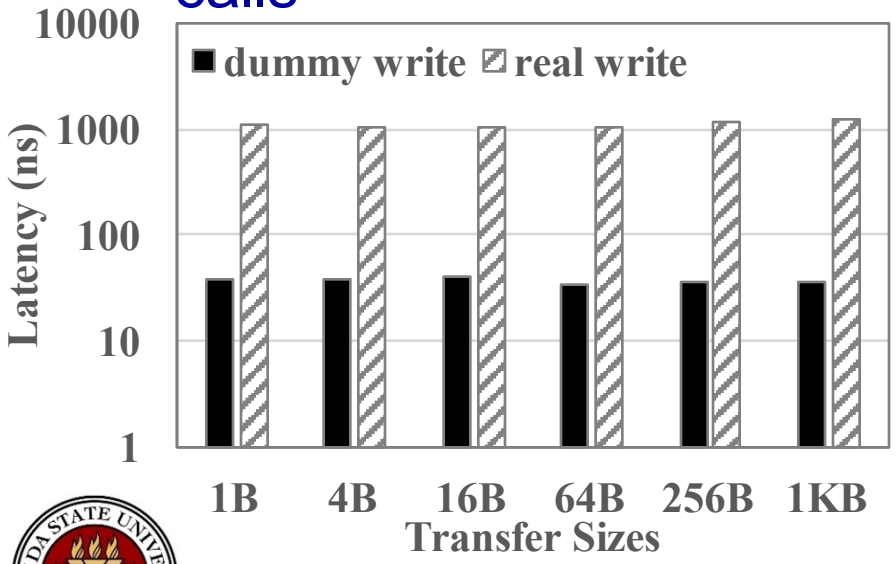
- Direct-FUSE outperforms FusionFS in write bandwidth and shows comparable read bandwidth
  - Writes benefit more from the FUSE kernel bypassing
- Direct-FUSE delivers similar scalability results as the original FusionFS

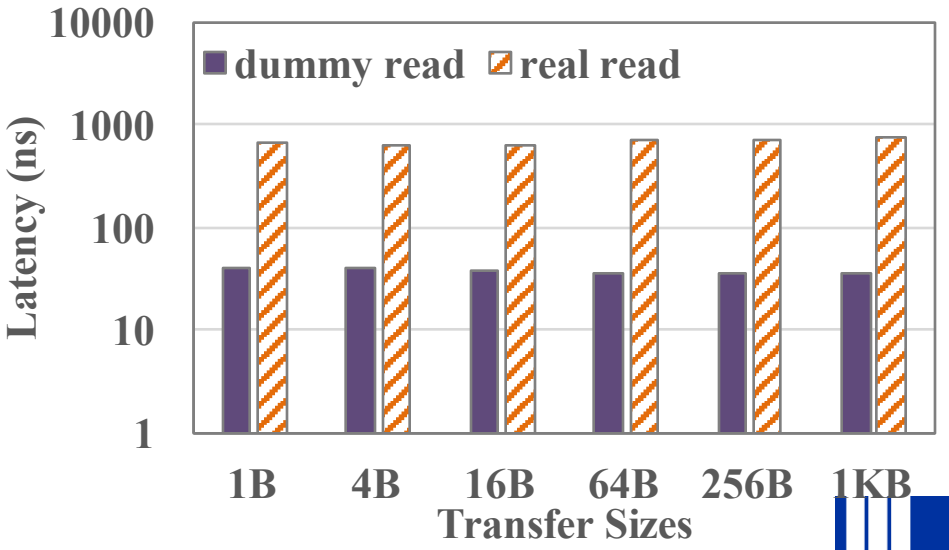


# Overhead Analysis

- The dummy read/write occupies less than 3% of the complete I/O function time in Direct-FUSE, even when the I/O size is very small
  - Dummy write/read: no actual data movement, directly return once reach the backend service
  - Real write/read: the actual Direct-FUSE read and write I/O calls



# Conclusions

- We have revealed and analyzed the context switches count and time overheads in FUSE metadata and data operations
- We have designed and implemented Direct-FUSE, which can avoid crossing kernel boundary and support multiple FUSE backends simultaneously
- Our experimental results indicate that Direct-FUSE achieves significant performance improvement compared to original FUSE file systems


# Sponsors of This Research