Alibaba Group

XFUSE: An Infrastructure for Running Filesystem Services in User Space

Qianbo Huai, Windsor Hsu*, Jiwei Lu*, Hao Liang†, Haobo Xu* and Wei Chen**

**Alibaba Group*
Sunnyvale, California
USA

†Alibaba Group
Shenzhen, Guangdong
China

User Space Filesystem

Benefits

- Higher development efficiency and velocity
- Decreased dependency on OS

Concerns

- Performance
- RAS (Reliability, Availability and Serviceability)
- Application and build changes may be required

Related Work

- FUSE: an interface for user-space programs to export a filesystem to the Linux kernel.
  - FUSE-based filesystems are accessible through standard kernel interface
- Large body of work on improving FUSE performance
  - E.g. ExtFUSE allows applications to register “thin” specialized request handlers in the kernel to improve performance
- AVFS uses LD_PRELOAD to intercept libc POSIX API entry and invoke filesystem ops without context switch
- ZUFS leverages persistent memory to have its kernel module directly copy data between source and destination, eliminating the extra copy to/from buffer cache.
- NVFUSE is an embeddable file system as a library running in the user-space incorporated with SPDK library, and supports directly submitting I/O requests to NVMe SSDs.
- Re-FUSE is a framework that provides support for restartable user space filesystems.

Our Contribution: XFUSE

XFUSE

- Backward compatible with FUSE
- Improves performance and RAS for XFUSE-optimized filesystems
- Facilitates large-scale and gradual rollout in production

Designed for user space filesystems that

- Use high speed storage devices
    - PMEM, fast SSDs, distributed storage systems based on high perf network
- Are deployed in production environments
    - With strict RAS requirements

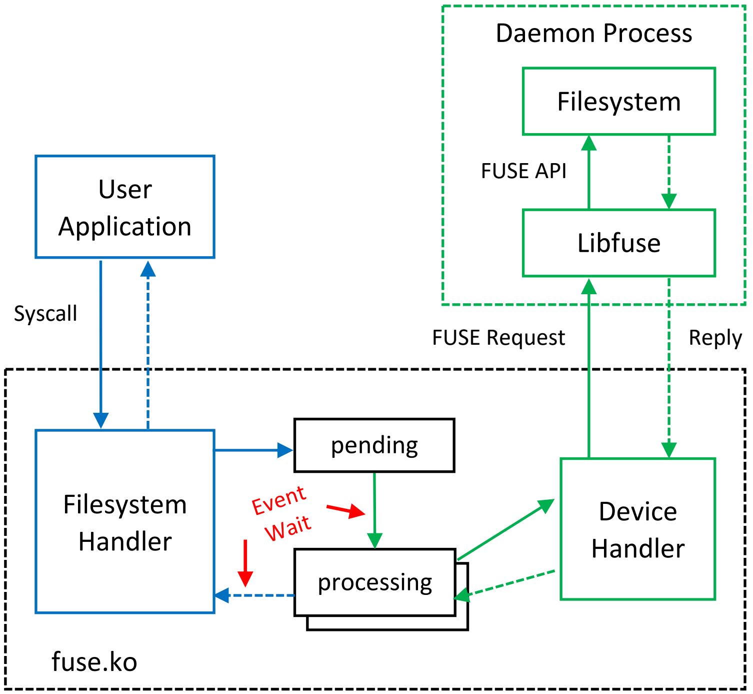
Agenda

- Request Flow in FUSE
- XFUSE Improvements
  - Adaptive waiting to reduce latency
  - Increased parallelism to improve throughput
  - Online upgrade for better RAS
- Performance Evaluation
  - Parametric analysis
  - System-level performance
- Conclusion

FUSE Request Flow

Request flow

- Application makes a syscall (e.g. via POSIX API) to a FUSE-mounted filesystem
- FUSE request travels from the app thread (via fuse.ko) to a filesystem daemon thread
- FUSE reply travels, in reverse direction, from the daemon thread back to the app thread



A synchronous FUSE request may involve two event waits in kernel

- Daemon thread: wait for pending requests if none is available at the time.
- App thread: wait for request completion

Notes:

- Certain details (such as background queue, async io) are omitted and the omission does not impact our discussion

XFUSE improvements

Adaptive Waiting

Problem

- Kernel event-wait and notification take a few μs to deliver
- High perf storages: data may become available sooner

Add an initial busy-wait period

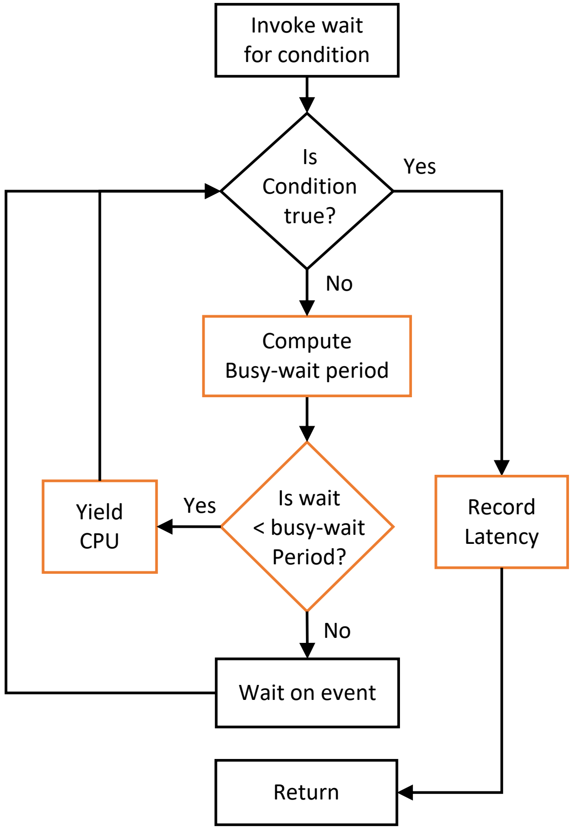
- End-to-end latency can be as low as 3~4 μs
  (vs. 8~9 μs under event-wait)

Effectiveness of busy-wait

- Performance characteristics of filesystem and storage
- Thread placement (same vs. different CPUs)
- Workload

Adaptive busy-event wait (or adaptive waiting)

- Dynamically predict if busy-wait is beneficial, and
- Turn on/off busy-wait accordingly



Increased Parallelism

FUSE

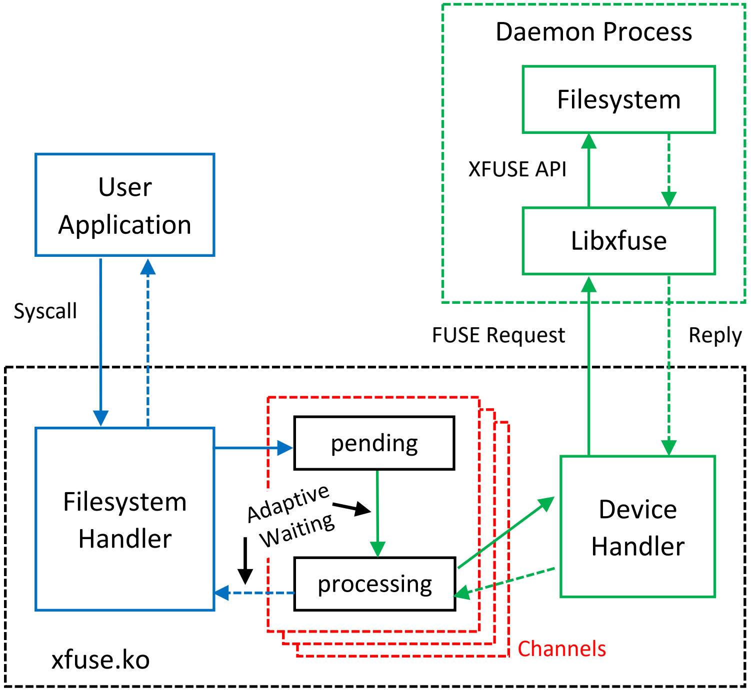
- New request → pending queue (one per mount)
- Request fetched → processing queue (one per FD)

XFUSE

- Introduces multiple request pending queues
- Groups each pair of pending and processing queues as a channel
- New request → channel (per selection policy)

Benefits

- Daemon threads work on their own pending and processing queues
- Reduced kernel lock contention between daemon threads



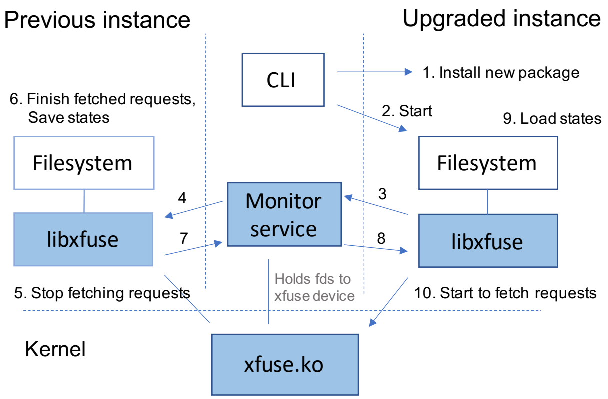
Online Upgrade

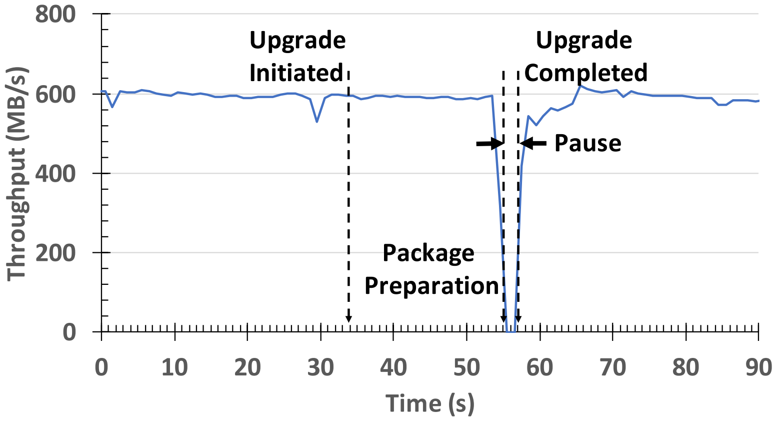
Business needs

- Fast paced rollout of new features and bug fixes for user space filesystems
- Minimal disruption to tens or hundreds of mounts and apps on each host during upgrade

Online upgrade solution

- Extend libfuse to support online upgrade workflow and state transition
- Monitor Service
  - Coordinates the interactions between two filesystem daemons
  - Assists the transfer of filesystem internal states, including FDs (to special fuse device)





Performance Evaluation

Parametric Analysis

Objectives

- Understand the effects of policy choices and tuning params
- Project potentially achievable performance

Method

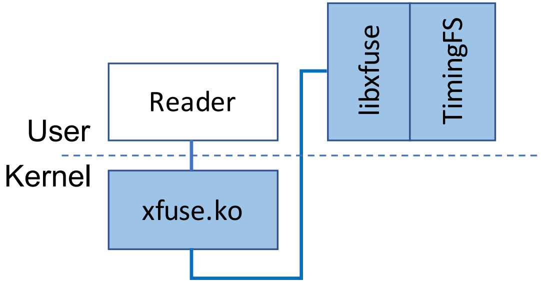
- Explore aspects of XFUSE individually

Aspects

- Waiting strategy in adaptive waiting
- Placement of app and daemon threads
- Channel selection for new FUSE request

Test setup

- Dedicated Linux 4.19.91 servers on Alibaba Cloud
- 24 channels in XFUSE
- 24 threads in TimingFS
- Threads can configured to affine to CPUs



TimingFS

- User space filesystem, via FUSE lowlevel API
- Optimized to probe aspects of XFUSE individually
- Can emulate timing characteristics storage systems
- E.g. READ copies 4KB randomly from a large file
  - PMEM-like: reply to XFUSE.ko immediately
  - SSD-like: delays 100 μs before replying

Parametric Analysis: Waiting Strategy

How I/O performance is impacted by

- Varying busy-wait period (note: “0μs” disables busy-wait, is essentially event-wait only)
- Wait-decision algorithm; threshold for turning on/off busy-wait

Findings

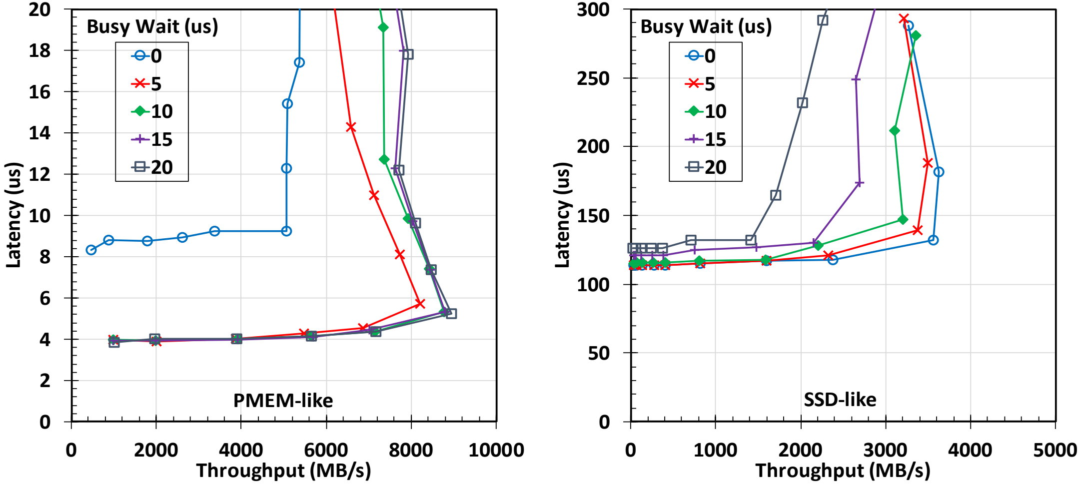
- PMEM-like: 10μs busy-wait, good balance between performance and CPU usage.
- SSD-like: last latency value is sufficient to predict the latency for the current request
- SSD-like: adaptive waiting outperforms busy-wait-only when system is under load

Wait-decision:

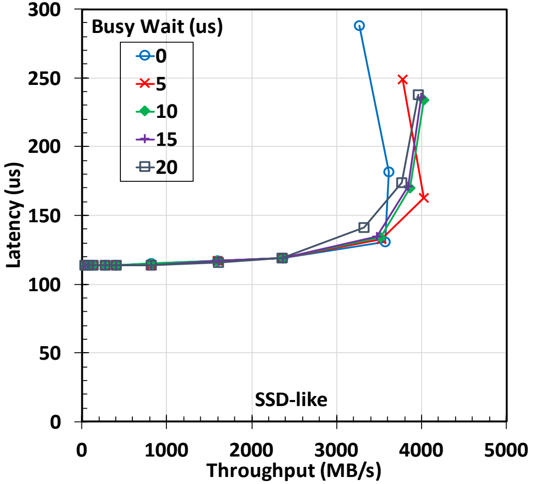
```
threshold = busy_wait_period + event_wait_overhead
          = 10μs + 5μs = 15μs

if observed_latency < threshold
        do busy-event wait
else
        do event wait
```

Performance with Busy-Event Wait



Performance with Adaptive Busy-Event Wait

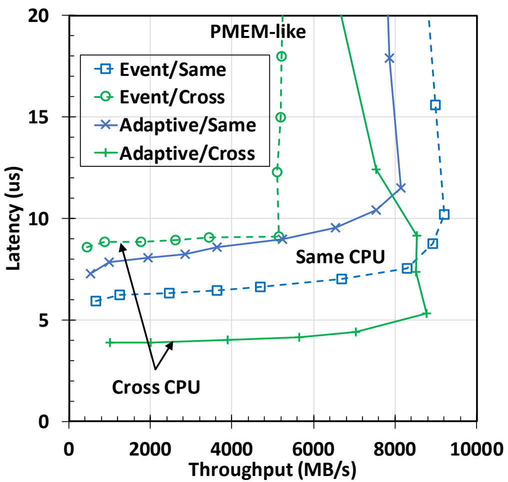


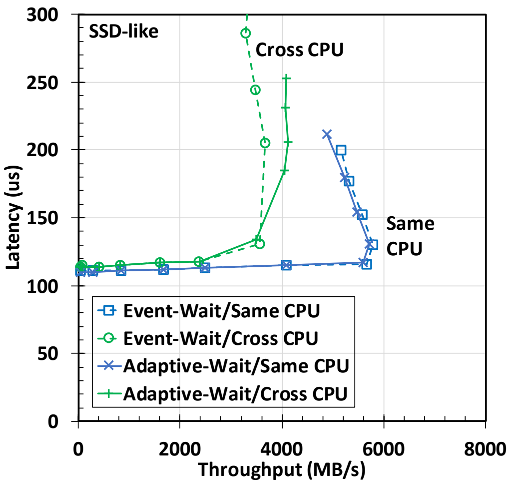
Parametric Analysis: Thread Placement

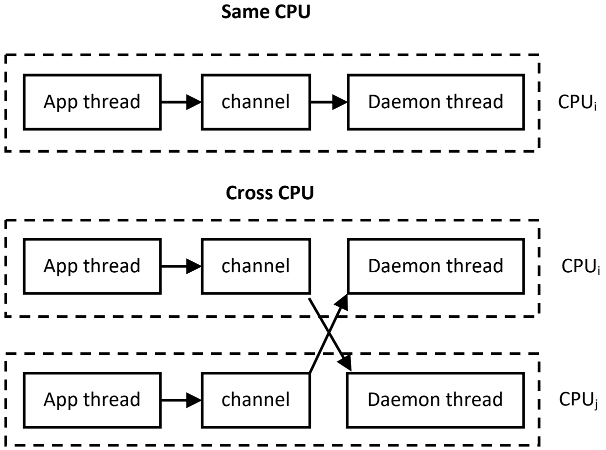
In production environments where thread placement can be controlled

Placement of app thread and corresponding daemon thread:

- PMEM-like storage, different CPUs
  - Two threads affined on the same CPU cannot busy-wait for each other.
- SSD-like storage: same CPU
  - Event notification on local CPU is faster than that across CPUs.





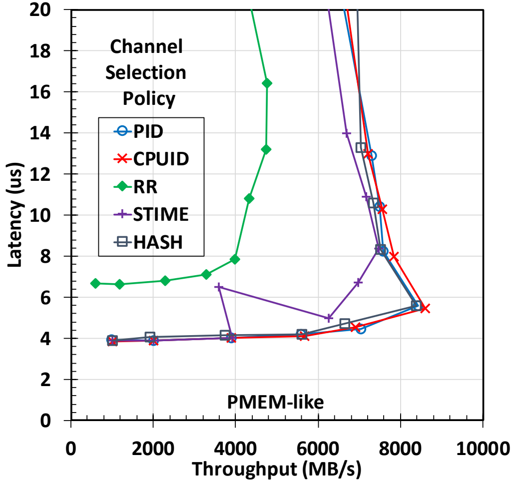


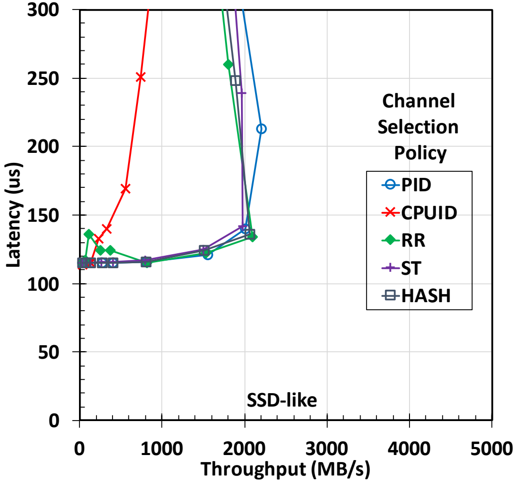
Parametric Analysis: Channel Selection

Findings

- Best strategy: evenly distribute requests across all channels
- Avoid policies that keep on switching to an idle channel, which renders busy-wait ineffective (see the RR line in PMEM-like figure).

- PID and HASH policies perform well in repeated tests
- PID-policy is computationally cheaper. HASH-policy consistently avoids skewed request distribution





Channel selection

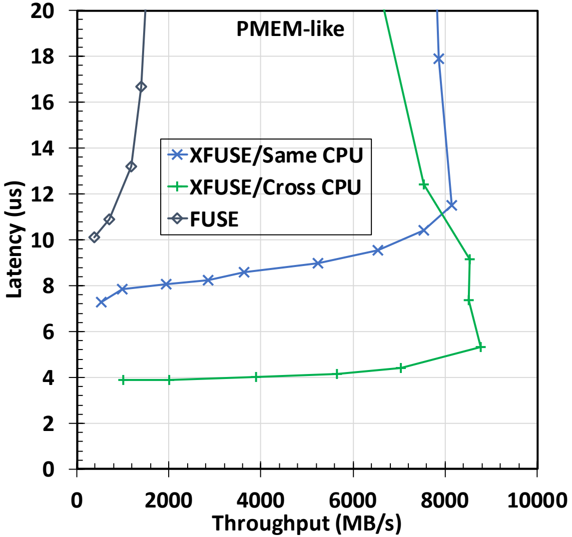
- channel_index = val % channel_num

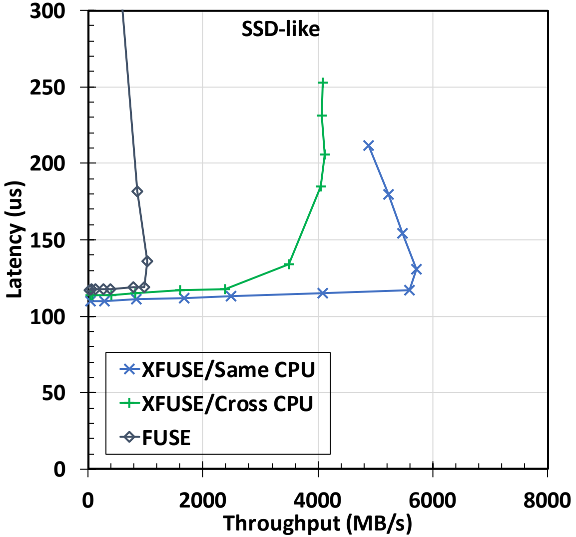
Where val is

- PID: thread id
- CPUID: id of CPU
- RR: round-robin, i.e. val = ++channel_index
- ST: thread start time
- HASH: hash of thread id
  Compute 3 different hashes
  Select the channel with the shortest queue

Parametric Analysis: XFUSE vs FUSE

- Project the best-case performance that XFUSE can achieve
- XFUSE configuration:
  - Adaptive busy-event wait: busy-wait period $10\mu s$. event-wait overhead $5\mu s$
  - 24 channels. 24 threads in TimingFS, one for each physical core.





System-Level Performance

Setup a common basis for comparing XFUSE, FUSE and regular kernel-mode EXT4

- Err on the side of being conservative for XFUSE

Evaluate the performance potential of XFUSE

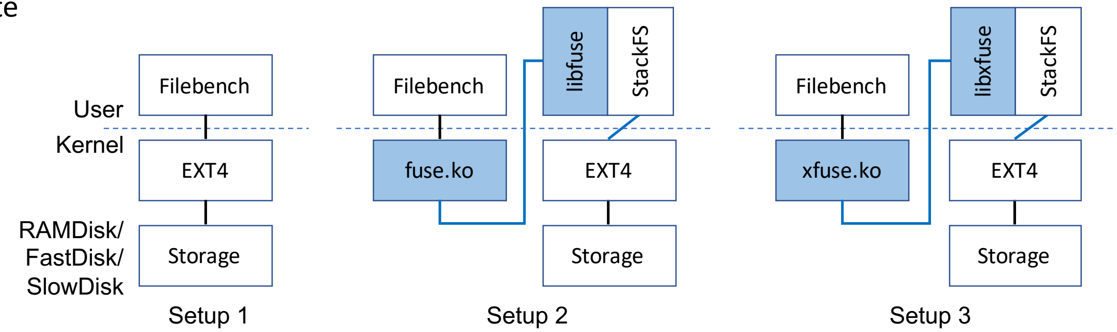
- In cases where FUSE has a significate gap with EXT4

Filebench simulates workloads

- Web-Server, Random-Read, File-Create

Storage types

- RAMDisk: PMEM-like
- FastDisk: SSD-like cloud disk. Avg 4KB read latency: 115μs. Max 80K IOPS
- SlowDisk: Cloud disk. Avg 4KB read latency: 250μs. Max 5K IOPS



System-Level Performance: Results

RAMDisk (PMEM-like)

- XFUSE closes the perf gap with kernel-mode EXT4.
- For random-read, XFUSE achieves 3x throughput over FUSE

FastDisk (SSD-like)

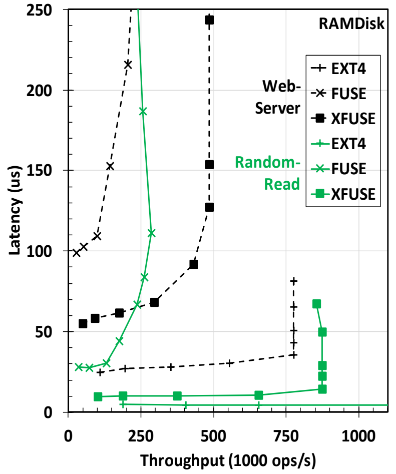
- XFUSE offers significant benefit over FUSE.
- For random read, XFUSE delivers full throughput of the FastDisk, maxed at 80K IOPS.

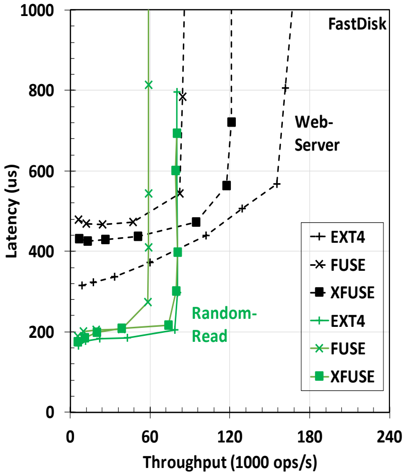
SlowDisk

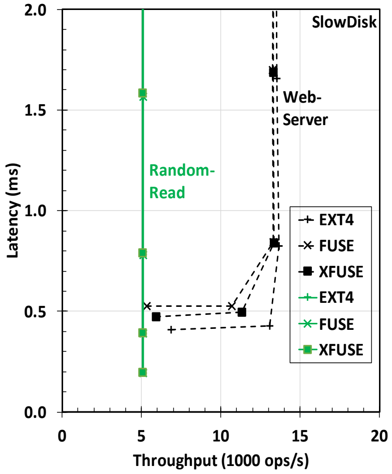
- Performance is bottlenecked by the storage than by conduit to user space

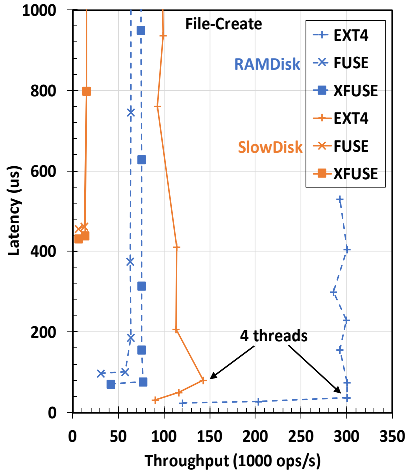
File-Create

- XFUSE outperforms FUSE for RAMDisk and FastDisk but by a smaller margin
- Benefit of XFUSE over FUSE is limited by the scalability of StackFS and EXT4









XFUSE

A FUSE-compatible framework for filesystem in user space

Enables significantly higher performing user space filesystems

- Delivers round-trip latency in the 4 μs range, offers throughput exceeding 8 GB/s

Supports filesystems with strict RAS requirements in production

Alibaba Group

Thank You

Questions: qianbo.huai@alibaba-inc.com