# OSDI 2021 and ATC 2021 joint preview session on storage systems

Saurabh Kadekodi
Postdoc @ Google Research

# Introduction to storage systems

- **Objective: efficient storage and retrieval of data**
- Popular storage systems that we interact with regularly:
  - File systems
  - Databases
- We are going to focus on **file systems**
- File systems primarily deal with the storage and retrieval of files
- There are two main classes of file systems:
  - Local (store data locally on your computer / device): **Ext4, Btrfs, NTFS**
  - Distributed (data in cloud or datacenter): **GFS, Lustre, HDFS**

# Research challenges in local file systems

**Metadata management**

- Efficiently managing file block locations, directory entries, permissions, etc.
- Designing efficient indexing structures and fast lookup strategies

**Data management**

- Efficiently storing and retrieving data to / from the storage devices
- Designing favorable on-device data layouts

**Crash consistency**

- Ensuring correctness of data across arbitrary system crashes and reboots

**Garbage collection**

- Reclaiming space occupied by deleted data

**File system device interface**

- Developing the right interface for higher flexibility, high device utilization, reducing redundant work

**Division of labor**

- Negotiating between host and device about best entity to perform certain tasks

# Research challenges in distributed file systems

**Fault tolerance**

- Ensuring adequate data reliability and availability typically via redundancy
- Efficient reconstruction of lost data due to failed disk, decommissioned nodes, etc.

**Metadata management**

- Similar to local file system metadata management
- Additional layer of indirection involving nodes, each of which typically contains a local file system

**Load balancing**

- Maintaining high and fair utilization of the nodes / devices for high throughput and predictable latency

**Scalability and concurrency**

- Ensuring high scalability as the size of the distributed file system increases via devices or nodes (or both)

# ZNS: Avoiding the Block Interface Tax for Flash-based SSDs

FS Device Interface | Data Management | Division of Labor

**Motivation:**

- Traditional block interface is unfit for SSDs and SMR devices — underlying technology is append-only
- SSDs and SMRs thus have complex firmwares with heavy internal IO and high over-provisioning
- This increases the cost-per-usable-GB (called the block interface tax) by a lot
- The block interface tax is projected to only get worse with larger devices

**Approach:**

- ZNS — Zoned Namespace standard that identifies append-only ranges of logical block addresses
- Does not perform complex in-device features; leaves that to the host file system

**Result:** Higher flexibility, higher throughput, lower cost-per-usable-GB with no block interface tax

ATC 2021: July 15th (Thu) 8:45 am - 10:15 am PDT (Track 2)

# ZNS+: Advanced Zoned Namespace Interface For Supporting In-Storage Zone Compaction

FS Device Interface | Data Management | Division of Labor | Garbage Collection

**Motivation:**

- ZNS divides logical blocks into fixed-sized append-only zones — favorable for SSDs and SMR disks
- Requires revamping the file system completely to incorporate zone-awareness
- Increased host file system overhead since cost of data copying initiated by host >> initiated by device

**Approach:**

- Offload the data copying operations to the device
- Enable sparse sequential writing within zone for more flexible garbage collection (segment cleaning)
- Device can decide where to copy data in the device based on copy costs to different locations

**Result:** Higher throughput observed by leveraging device support due to reduced, more efficient IO

OSDI 2021: July 14th (Wed) 10:30 am - 12:00 pm PDT

# Optimizing Storage Performance with Calibrated Interrupts

FS Device Interface | Metadata Management

**Motivation:**

- Instant interrupts on IO completion minimize latency, but batching IOPS per interrupt increases throughput
- Adaptive heuristics of whether to fire an interrupt or wait lack semantic knowledge of latency-sensitive ops
- Mistaking a latency-sensitive op as a throughput op can large latency spikes

**Approach:**

- Inform the device of which ops are latency-sensitive to enable interrupt-calibration in the device
- A two-bit addition to requests sent to device can make this happen

**Result:** Tiny change to app to issue hints increases RocksDB throughput by 37% and reduces latency by 86%

OSDI 2021: July 14th (Wed) 10:30 am - 12:00 pm PDT

# Modernizing File System through In-Storage Indexing

Metadata Management | FS Device Interface | Division of Labor

**Motivation:**

- Current SSD exposes a series of logical blocks, which are mapped to physical blocks by device firmware
- Paper argues that this is an inflexible interface that puts lot of responsibility on the host file system
- File system aging also causes fragmentation resulting in performance slowdown

**Approach:**

- Paper proposes a interface and division of labor for better device management, lesser IO
- Key-value interface with device with a key-value store in the device
- Pushes file and directory indexing to the device
- Pushes transaction support to the device for crash consistency

**Result:** Simpler file system-device interface and lower IO amplification

OSDI 2021: July 14th (Wed) 10:30 am - 12:00 pm PDT

# Rearchitecting Linux Storage Stack for $\mu$s Latency and High Throughput

Scalability and Concurrency | Load Balancing

**Motivation:**

- For multiple apps Linux storage stack can either provide low latency or high throughput, but not both
- Multi-tenant deployments cause head-of-line blocking despite there being multiple IO queues (per-core)

**Approach:**

- Key insight: per-core IO queues + multi-queue storage + network hardware $\approx$ network switches
- Apply classical techniques of queueing and priority scheduling from networking literature

**Result:** tens of latency sensitive apps can run alongside throughput bound apps with $\mu$s latency

OSDI 2021: July 14th (Wed) 10:30 am - 12:00 pm PDT

# LODIC: Logical Distributed Counting for Scalable File Access

Scalability and Concurrency | Metadata Management

**Motivation:**

- Reference counting is pervasive in kernel; page frames, nodes, file descriptors, etc.
- Millions of counters are not uncommon
- Existing distributed counters are per-core and suffer from excessive memory pressure and query latencies

**Approach:**

- A per-process counter instead of a per-core counter
- Counters only in processes that actually share the entity
- Per-process counters able to capture skew in popularity (degree of sharing); per-core counters cannot

**Result:** Up to 65x throughput increase in read shared file block; 124% increase in NGINX (web-server)

ATC 2021: July 16th (Fri) 10:15 am - 11:30 am PDT (Track 1)

# Z-Journal: Scalable Per-Core Journaling

Scalability and Concurrency | Crash Consistency

**Motivation:**

- Journaling is a performance bottleneck in file systems, especially atop modern hardware (SSD, NVM)
- Per-core journaling exists, but suffers from complex journal coherence mechanisms and write ordering

**Approach:**

- A per-core journal that overcomes the journal coherence mechanism while preserving parallelism
- Shared blocks in parallel journals form order-preserving write graphs (akin to transactions)
- Each per-core journal maintains UUIDs of other per-core journal transactions and commit order
- Can replace external journal like JBD2; i.e. no change required to file system

**Result:** Journal-bottlenecked workloads see an improvement of up to 4000%

ATC 2021: July 16th (Fri) 10:15 am - 11:30 am PDT (Track 1)

# MapperX: Adaptive Metadata Maintenance for Fast Crash Recovery of DM-Cache Based Hybrid Storage Devices

Metadata Management | Crash Consistency

**Motivation:**

- Device-Mapper Cache (DM-Cache) maps SSDs and HDDs as virtual block devices
- It enables high-performance SSDs to act as a cache for HDDs
- To ensure high performance DM-Cache asynchronously persists dirty bits of data in SSD
- Thus all data in SSD has to be considered dirty on system crash, causing astronomical recovery times (hrs)

**Approach:**

- Design a smarter data structure — an adaptive bit-tree (ABT) to allow fast synchronous dirty bit updates
- ABT exploits spatial locality of updates and uses single bit to account for range of consecutive blocks

**Result:** Orders of magnitude reduction in DM-Cache recovery times with negligible metadata persistence cost

ATC 2021: July 15th (Thu) 8:45 am - 10:15 am PDT (Track 2)

# Max: A Multicore-Accelerated File System for Flash Storage

Metadata Management | Data Management | Scalability and Concurrency

**Motivation:**

- File systems do not scale to the parallelism provided by modern SSDs
- Lock contention via unscalable data structure is a major source of overhead
- File system and block layer need to exploit multiple cores to keep up with device parallelism

**Approach:**

- Reducing cache coherence by having per-core locks and counters
- Adding an abstraction (*file cell*) which repacks data and metadata to allowing multiple indexing entities
- Dividing allocation into multiple independent regions to facilitate concurrent file operations

**Result:** Order of magnitude higher throughput and significantly higher device utilization via high parallelism

ATC 2021: July 16th (Fri) 10:15 am - 11:30 am PDT (Track 1)

# XFUSE: An Infrastructure for Running Filesystem Services in User Space

Metadata Management | Division of Labor

**Motivation:**

- Frequent messaging between user-space and kernel causes user-space file systems to be slow
- FUSE also suffers from lower reliability, availability and security (RAS) concerns
- As devices get faster (persistent memory), RAS and performance problems of FUSE become worse

**Approach:**

- Understand and address RAS concerns of FUSE
- Lower latency using techniques like busy-wait instead of signals, parallelism via multiple daemon threads

**Result:** XFUSE enables user space file systems to have very low latency (4 $\mu$s) and throughput > 8GB/s

ATC 2021: July 16th (Fri) 10:15 am - 11:30 am PDT (Track 1)

# Boosting Full-Node Repair in Erasure-Coded Storage

Fault Tolerance | Scalability and Concurrency | Load Balancing

**Motivation:**

- Existing erasure coding repair optimizations do not consider full-node failure
- Do not utilize full-duplex bandwidth for repair due to unbalanced repair solutions
- Cannot incorporate different priorities of repair owing to different erasure coding schemes requiring repair

**Approach:**

- Design a scheduling framework that can optimize for full-node recovery
- Construct a repair directed acyclic graph (RDAG) to capture data routing and chunk dependencies

**Result:** Accelerated repair times of full-node failures significantly

ATC 2021: July 15th (Thu) 8:45 am - 10:15 am PDT (Track 2)

Thank you and enjoy the talks!