CS 744: SPLIT-FS

Shivaram Venkataraman

Fall 2021
- Course Project: Check in: Today! → Instructions on Piazza
- Midterm 2 next week!

→ From SQL to TPU paper

1 page → Canvas
Serverless Computing

Compute Accelerators

Infiniband Networks

Non-Volatile Memory
PERSISTENT MEMORY

Prior benchmarking study

<table>
<thead>
<tr>
<th>Property</th>
<th>SSD</th>
<th>DRAM</th>
<th>Intel PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential read latency (ns)</td>
<td>~1000</td>
<td>81</td>
<td>169 (2.08×)</td>
</tr>
<tr>
<td>Random read latency (ns)</td>
<td>81</td>
<td>305  (3.76×)</td>
<td></td>
</tr>
<tr>
<td>Store + flush + fence (ns)</td>
<td>86</td>
<td>91   (1.05×)</td>
<td></td>
</tr>
<tr>
<td>Read bandwidth (GB/s)</td>
<td>120</td>
<td>39.4 (0.33×)</td>
<td></td>
</tr>
<tr>
<td>Write bandwidth (GB/s)</td>
<td>80</td>
<td>13.9 (0.17×)</td>
<td></td>
</tr>
</tbody>
</table>

much higher than PCIe SSDs
WHAT IS DIFFERENT?

Traditional / Existing

Optane DIMM | DRAM
---|---
Optane DIMM | DRAM
Optane DIMM | DRAM
Direct-mapped Cache (4KB Block)

(a) Optane Platform Modes (Memory and AppDirect)

OSTEP → Andrea Renzi
BACKGROUND: FILE SYSTEM API

POSIX API for files

```c
int fd = open(char *path, int flag, mode_t mode) → open a file
read(int fd, void *buf, size_t nbyte) → read a FD
write(int fd, void *buf, size_t nbyte) →
close(int fd)

rename(char *old, char *new) → move a file to a
diff location

fsync(int fd) → flush any
state that has been buffered
```
**MOTIVATION: OVERHEADS**

**DAX:** mmap pages into virtual memory.

No page caches!

<table>
<thead>
<tr>
<th>File system</th>
<th>Append Time (ns)</th>
<th>Overhead (ns)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext4 DAX</td>
<td>9002</td>
<td>8331</td>
<td>1241%</td>
</tr>
<tr>
<td>PMFS</td>
<td>4150</td>
<td>3479</td>
<td>518%</td>
</tr>
<tr>
<td>NOVA-Strict</td>
<td>3021</td>
<td>2350</td>
<td>350%</td>
</tr>
<tr>
<td>SPLITFS-Strict</td>
<td>1251</td>
<td>580</td>
<td>86%</td>
</tr>
<tr>
<td>SPLITFS-POSIX</td>
<td>1160</td>
<td>488</td>
<td>73%</td>
</tr>
</tbody>
</table>
SPLIT-FS: GOALS

Low software overhead → compared to time spent on device

Transparency → compatible with existing applications

Minimal data-copy/IO

Flexible semantics
SPLIT FS DESIGN: READ/WRITES

POSIX Application
- read()
- write()
- [append]
- open()
- close()

U-Split
- mmaps

K-Split
- File on PM
- Staging File
- Op log

PM Device
- File on PM
- Staging File
- Op log

fs is split into
part in userspace
and part in kernel space

Data operations will
be in U-Split
Metadata operations
will be in K-Split

→ don't pay system call overheads
→ File on PM
→ Staging File
→ Op log

File on PM
Staging File
Op log
**SPLIT FS DESIGN: READ/WRITES**

- File is already open
- `mmaps` relevant region of the file
- Reads happen using `LOAD` instructions which go to PM
- Write do a `STORE` or `MOV` instructions
- "Non-temporal" store
SPLIT FS DESIGN: APPEND

Appends are forwarded to a staging file. fsync() → relinked to the original file. mmaps also tracks pre-allocated staging files used for appends.
Avoid copying data from staging file to the original file.
# SPLIT-FS MODES

<table>
<thead>
<tr>
<th>Mode</th>
<th>Sync. Data Ops</th>
<th>Atomic Data Ops</th>
<th>Sync. Metadata Ops</th>
<th>Atomic Metadata Ops</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>ext4-DAX</td>
</tr>
<tr>
<td>sync</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>Nova-Relaxed, PMFS</td>
</tr>
<tr>
<td>strict</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NOVA-Strict, Strata</td>
</tr>
</tbody>
</table>

- Metadata
- Directory
- Sizer
- Sizes
- ACL etc.

Sync = all returns changes are visible

Atomic = all changes happen simultaneously

append (10), append (4), fsync (fd) → Atomicity 10 and 4
**SPLIT-FS: LOGGING**

Atomicity → ⊥ overwrites

Logical redo logging
Log entry: 64B in size! 4B checksum!
sfence to ensure ordering

Fixed length log: 128 MB per-application

Replay entire log on recovery!

logical | physical logging
L → write down what operation needs to be done in the log

Redo | Undo logging
L → the operations are idempotent

Empty log entries
Persistent Memory: New opportunities, new challenges

Split-FS: split Pipelining to use CPU, GPU
Partition buffer, BETA ordering

Split FS: Partition FS between User / Kernel
Data / Metadata
Support for efficient appends, atomic ops using staging files
DISCUSSION

https://forms.gle/8TwGgqXhVyuiRCpx8
<table>
<thead>
<tr>
<th>System call</th>
<th>Strict</th>
<th>Sync</th>
<th>POSIX</th>
<th>ext4 DAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>2.09</td>
<td>2.08</td>
<td>1.82</td>
<td>1.54</td>
</tr>
<tr>
<td>close</td>
<td>0.78</td>
<td>0.69</td>
<td>0.69</td>
<td>0.34</td>
</tr>
<tr>
<td>append</td>
<td>3.14</td>
<td>3.09</td>
<td>2.84</td>
<td>11.05</td>
</tr>
<tr>
<td>fsync</td>
<td>6.85</td>
<td>6.80</td>
<td>6.80</td>
<td>28.98</td>
</tr>
<tr>
<td>read</td>
<td>4.57</td>
<td>4.53</td>
<td>4.53</td>
<td>5.04</td>
</tr>
<tr>
<td>unlink</td>
<td>14.60</td>
<td>13.56</td>
<td>14.33</td>
<td>8.60</td>
</tr>
</tbody>
</table>

Table 6: SPLITFS system call overheads. The table compares the latency (in us) of different system calls for various modes of SPLITFS and ext4 DAX.
In what ways can SplitFS improve performance of Big Data frameworks like MR/Spark?

→ Faster read / writes to input / output files
→ Better performance
→ Appends being optimized is good for MR/Spark

→ Persistence
→ Helps fault tolerance and reduce recomputation on failure
→ Capacity can also help increase the RDDs in "cache"
Next class: TPU
Project check-ins tonight!
DISCUSSION

Staging files in DRAM?

Page faults are expensive on open()