CS 744: SPLIT-FS

Shivaram Venkataraman
Fall 2021
ADMINISTRIVIA

- Course Project: Check in: Today!
- Midterm 2 next week!
Serverless Computing

Compute Accelerators

Infiniband Networks

Non-Volatile Memory
# Persistent Memory

<table>
<thead>
<tr>
<th>Property</th>
<th>DRAM</th>
<th>Intel PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential read latency (ns)</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>
</tr>
<tr>
<td>Store + flush + fence (ns)</td>
<td>86</td>
<td>91 (1.05×)</td>
</tr>
<tr>
<td>Read bandwidth (GB/s)</td>
<td>120</td>
<td>39.4 (0.33×)</td>
</tr>
<tr>
<td>Write bandwidth (GB/s)</td>
<td>80</td>
<td>13.9 (0.17×)</td>
</tr>
</tbody>
</table>
WHAT IS DIFFERENT?

(a) Optane Platform Modes (Memory and AppDirect)
BACKGROUND: FILE SYSTEM API

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

rename(char *old, char *new)

fsync(int fd)
```
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

Transparency

Minimal data-copy/IO

Flexible semantics
SPLIT FS DESIGN: READ/Writes
SPLIT FS DESIGN: APPEND

POSIX Application

read() write() [append]
open() close()

mmaps

U-Split

 POSIX Application

read() write() [append]
open() close()

mmaps

U-Split

User space
Kernel space

K-Split

File on PM Staging File Op log

File on PM Staging File Op log

PM Device
RELINK

1. Logical blocks
   - Target file
     - Physical blocks
   - Staging file
     - Staging 1
     - Staging 2

   Init state: Staging file has a `mmap()` region of 2 pre-allocated physical block available for appends.

2. Logical blocks
   - Target 2
   - Staging 2
   - Physical blocks

   An append to the target file is routed to the staging file block. Later reads to the appended region are also routed to the block.

3. Logical blocks
   - Staging 2
   - Physical blocks

   On `fsync`, the newly written block in the staging file is logically linked to the target file, while retaining its `mmap()` region and physical block.
# 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>
SPLIT-FS: LOGGING

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!
SUMMARY

Persistant Memory: New opportunities, new challenges

Split-FS: split Pipelining to use CPU, GPU
  Partition buffer, BETA ordering
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: SPLiTSFS system call overheads. The table compares the latency (in us) of different system calls for various modes of SPLiTSFS and ext4 DAX.
In what ways can SplitFS improve performance of Big Data frameworks like MR/Spark?
NEXT STEPS

Next class: TPU
Project check-ins tonight!
Staging files in DRAM?

Page faults are expensive on open()