Disk Schedulers for Solid State Drives

Jaeho Kim
School of Computer Science University of Seoul Seoul, Korea kjhnet07@uos.ac.kr

Yongseok Oh
School of Computer Science University of Seoul Seoul, Korea ysoh@uos.ac.kr

Eunsam Kim
School of Computer & Inform. Eng. Hongik University Seoul, Korea eskim@hongik.ac.kr

Jongmoo Choi
Division of Information and CS Dankook University Seoul, Korea choijm@dankook.ac.kr

Donghee Lee*
School of Computer Science University of Seoul Seoul, Korea dhl_express@uos.ac.kr

Sam H. Noh
School of Computer & Inform. Eng. Hongik University Seoul, Korea samhnoh@hongik.ac.kr

EMSOFT’09, October 12–16, 2009, Grenoble, France
Outline

• Introduction
• Flash Translation Layer
• Linux Disk I/O Scheduler
• Logical Block
• Design of New Disk Schedulers
• Experimental Results
• Conclusions
Introduction

• The SSD (Solid State Drive) has been introduced and is gaining popularity in embedded systems and laptops
• Because of the differences in device characteristics, the current schedulers may not adequately schedule requests for SSDs
Flash Translation Layer (1/3)

• Solve the constraint that page overwrite is forbidden
• Maintain a pool of writable pages by pre-erasing the redundant blocks
• Approach
  – Page mapping
  – Block mapping
Flash Translation Layer (2/3)

• **Page mapping**
  - The map translates a sector number to a combination of page number and block number where the sector data exists
  - Be excellent random write performance
  - Increase garbage collection overhead after randomly write
Flash Translation Layer (3/3)

• **Block mapping**
  
  • The map translates a sector number to a combination of page number and block number where the sector data exists
  
  • To modify data, the block mapping FTL reserves some redundant blocks called **log blocks**
Linux Disk I/O Scheduler

• Noop
  – Merge adjacent requests to a larger one

• Deadline
  – If the waiting time of a request exceeds its deadline, the request is served immediately

• Anticipatory
  – Waits for new in-coming read requests for a predetermined period

• CFQ (Complete Fair Queuing)
  – Separate queue for each process and serves requests of queues in round-robin order
Logical Block (1/4)

- Read performance of an SSD is almost consistent and independent of the ordering and geometrical distance between read requests.
- Sequential writes to a logical block is much more efficient than random writes going to various logical blocks.
Logical Block (2/4)

(a) Samsung SSD read
(b) Mtron SSD read
(c) Samsung SSD write
(d) Mtron SSD write
Logical Block (3/4)

• LBA-bundle (Logical Block Aligned-bundle)
  – A way to maximize write performance in an SSD
  – Arrange write requests into bundles the size of a logical block so that write requests falling in a logical block belong to the same bundle

• The writing order of the bundles themselves will not be important if all requests within the bundle are written sequentially at a time
When the merger occurs:

- **Case 1**: Involve one logical block for a merge operation.
  - Block 1: Write requests 1, 4, 3
  - Case 1: 1, 4, 3

- **Case 2**: Involve two logical blocks for a merge operation.
  - Block 1: Write requests 1, 4, 3
  - Block 2: Write requests 1, 3, 4
  - Case 2: 1, 3, 4
Design of New Disk Schedulers (1/2)

• The IRBW-FIFO (Individual Read Bundled Write FIFO) Scheduler
  – Apply FIFO ordering to the bundles of write requests and individual read requests.
Design of New Disk Schedulers (2/2)

• The IRBW-FIFO-RP (Individual Read Bundled Write FIFO with Read Preference) Scheduler
  – Maintains separate FIFO ordering among read requests and among the bundles of write requests
  – Gives higher priority to read requests than to the bundles of write requests
  – To avoid write starvation, allow each LBA-bundle to yield to a read request only once
Experimental Results (1/3)

- Throughput results
- Response time results
- Experimental environment

<table>
<thead>
<tr>
<th>Type</th>
<th>Specifics</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU/RAM</td>
<td>Intel Core2 Duo E4500 2.2GHz / 2GB DRAM</td>
</tr>
<tr>
<td>SSD</td>
<td>Samsung 64GB MCCOE64G5MPP SATA-2</td>
</tr>
<tr>
<td></td>
<td>Mtron 16GB MSD-SATA6025 SATA-1</td>
</tr>
<tr>
<td>OS</td>
<td>Linux-kernel 2.6.23 / Ext3 File system</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Postmark</td>
</tr>
<tr>
<td></td>
<td>Type-A: 1KB~32KB file size</td>
</tr>
<tr>
<td></td>
<td>Type-B: 1MB~16MB file size</td>
</tr>
<tr>
<td></td>
<td>IOmeter</td>
</tr>
<tr>
<td></td>
<td>File server access pattern</td>
</tr>
<tr>
<td>Targets</td>
<td>IRBW-FIFO, IRBW-FIFO-RP, and existing Linux I/O schedulers</td>
</tr>
</tbody>
</table>
Experimental Results (2/3) - Throughput Results

(a) Postmark Type-A
Figure 8. Throughput of Postmark benchmarks on Samsung SSD

(b) Postmark Type-B

(a) Postmark Type-A
Figure 9. Throughput of Postmark benchmarks on Mtron SSD
Experimental Results (3/3) - Response Time Results

- Average response times of the two schedulers are 0.475 and 0.458

Figure 11. Response time of Postmark benchmark on Samsung SSD
Conclusions

• SSDs have much faster read service times than the magnetic disks with the service times being almost constant, while write request service times are more complex.

• IRBW-FIFO and IRBW-FIFO-RP arrange write requests into LBA-bundles while reads are independently scheduled.