QoS-aware Flash Memory Controller

Bryan S. Kim and Sang Lyul Min

Seoul National University
Flash memory ubiquity

<table>
<thead>
<tr>
<th>NAND flash memory</th>
<th>Flash-based storage</th>
<th>Devices and applications</th>
</tr>
</thead>
</table>

* Images from various sources via google search
Flash memory eccentricities

Hello World RTAS 2016

<table>
<thead>
<tr>
<th>“Hello”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“World”</td>
</tr>
<tr>
<td>“RTAS”</td>
</tr>
<tr>
<td>“2016”</td>
</tr>
</tbody>
</table>
Flash memory eccentricities

Hello World RTAS 2017

*Invalidate*

*Update map*

“Hello”

“World”

“RTAS”

“2016”

“2017”

Background  Design  Evaluation  Conclusion
Flash memory eccentricities

Hello World RTAS 2017

"Hello"
"World"
"RTAS"
"2016"

*Copy valid data*

"2017"
"Hello"
"World"
"RTAS"
Flash memory eccentricities

Hello World RTAS 2017

*Erase a block*

```
“Hello”
“World”
“RTAS”
“2016”
```

```
“2017”
“Hello”
“World”
“RTAS”
```
Flash translation layer

- Host request handling
- Garbage collection
- Mapping table management
- Sudden power off recovery
- Wear-leveling
- Bad block management
- Error handling
- Read scrubbing
Performance of flash storage

SSD Performance States - Normalized IOPS

* Graph from SNIA solid state storage performance test specification

Memory & Storage Architecture Lab.
@ Seoul National University
Challenge 1: scheduling

- Mapping table management
- Read scrubbing
- Wear-leveling
- Error handling
- Bad block management
- Sudden power off recovery

Host request handling:
- Host req #0: Prog(chip 0, block 4, page 18)
- Host req #1: Prog(chip 1, block 2, page 17)
- Host req #2: Read(chip 0, block 0, page 46)

Garbage collection:
- GC req #0: Read(chip 0, block 1, page 55)
- GC req #1: Read(chip 0, block 1, page 78)

Flash memory subsystem:

Background  Design  Evaluation  Conclusion
Challenge 2: changes in importance

Host request handling
- Host req #0: Prog(chip 0, block 4, page 18)
- Host req #1: Prog(chip 1, block 2, page 17)
- Host req #2: Read(chip 0, block 0, page 46)
- Host req #3: Read(chip 1, block 6, page 03)

Garbage collection
- GC req #0: Read(chip 0, block 1, page 55)

Mapping table management
- Wear-leveling
- Read scrubbing
- Error handling
- Bad block management
- Sudden power off recovery

Flash memory subsystem

Host req #2: Read(chip 0, block 0, page 46)
Challenge 2: changes in importance

Host request handling
- Mapping table management
- Read scrubbing
- Wear-leveling
- Error handling
- Bad block management
- Sudden power off recovery

Garbage collection
- GC req #0: Read(chip 0, block 1, page 55)
- GC req #1: Read(chip 0, block 1, page 78)
- GC req #2: Read(chip 0, block 1, page 99)
- GC req #3: Erase(chip 0, block 1)

Flash memory subsystem
- Host req #0: Prog(chip 0, block 4, page 18)
- Host req #1: Prog(chip 1, block 2, page 17)
Challenge 3: load balancing

- Mapping table management
- Read scrubbing
- Wear-leveling
- Error handling
- Bad block management
- Sudden power off recovery

Host request handling

Garbage collection

Flash memory subsystem

- Prog
- Read
- Read
- Read
- Read
- Read

Memory & Storage Architecture Lab.
@ Seoul National University
QoS-aware flash controller

- Host request handler
- Non-binding request handler
- Wait queues
- Fair share scheduler
- Dynamic share allocator
- Non-binding request handler
- Garbage collector
- Shared state (Map, # of free blocks, block list, utilization, etc)
- Block allocation
- Update
- Look up
- # free blocks
- Set
- Flash channel
- Flash chip
- Flash channel
- Flash chip
- Flash chip

Memory & Storage Architecture Lab.
@ Seoul National University
**Fair share scheduler**

- Keep track of the state of resources
- Select request to service based on share
- Interface with the low-level controller

**Diagram:**

- Host request handler
- Non-binding request handler
- Wait queues
- Dynamic share allocator
- Fair share scheduler
- Low-level flash controller
- Flash channel
- Flash chip
- Shared state (Map, # of free blocks, block list, utilization, etc)
- Block allocation
- Look up
- Response
- Update
- Non-binding request handler
- Wait queues
- Low-level flash controller
- Flash channel
- Flash chip
- Flash chip
- Flash chip
Fair share scheduler

- Keep track of the state of resources
- Select request to service based on share
- Interface with the low-level controller
Fair share scheduler

- Keep track of the state of resources
- Select request to service based on share
- Interface with the low-level controller

Flash chip

Host progress

40% share

@ 0us

100us

Read

0us + 100us / 40% = 250us

GC progress

60% share

Memory & Storage Architecture Lab. @ Seoul National University
Fair share scheduler

- Keep track of the state of resources
- Select request to service based on share
- Interface with the low-level controller

Flash chip

Host progress
- 40% share
- @ 0us
- @ 150us
- 100us

Read

GC progress
- 60% share
- @ 150us
Fair share scheduler

- Keep track of the state of resources
- Select request to service based on share
- Interface with the low-level controller

**Flash chip**
- Host progress
  - @ 0us: 100us
  - @ 150us: 240us
- GC progress
  - @ 150us: 150us + 240us / 60% = 550us

**GC program**
- 60% share
- 40% share
- 0us + 100us / 40% = 250us

**Conclusion**
Fair share scheduler
- Keep track of the state of resources
- Select request to service based on share
- Interface with the low-level controller

---

**Fair share scheduler**

<table>
<thead>
<tr>
<th>Host progress</th>
<th>Flash chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>40% share</td>
<td>Read @ 0us</td>
</tr>
<tr>
<td></td>
<td>100us</td>
</tr>
</tbody>
</table>
|               | GC program | 240us
|               | 100us      |
|               | 390us + 100us/40%
|               | = 640us    |

<table>
<thead>
<tr>
<th>GC progress</th>
<th>Flash chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>60% share</td>
<td>Read @ 150us</td>
</tr>
<tr>
<td></td>
<td>150us</td>
</tr>
</tbody>
</table>
|               | GC program | 240us
|               | 150us + 240us/60%
|               | = 550us    |
Dynamic share allocator

- # of free blocks as representation of state
- Adjust share to control # of free blocks
Dynamic share allocator

- # of free blocks as representation of state
- Adjust share to control # of free blocks

Scheduler

Host requests
GC requests

Dynamic Share Allocator

Target # of free blocks

Host share
GC share

# of free blocks

Scheduled requests
Non-binding request handler

- Estimate queue delay for each chip
- Re-assign non-binding requests to a new chip
- Notify FTL task of the selection
Non-binding request handler

- Estimate queue delay for each chip
- Re-assign non-binding requests to a new chip
- Notify FTL task of the selection

Host: 80%
GC: 20%

Host program

100us
200us
100us
200us

Chip 0
Chip 1
Evaluation methodology

- **Storage system configuration**
  - FTL tasks: host request handling & garbage collection
    - Generate a stream of asynchronous flash memory requests
    - Inter-arrival time for requests to model processing overhead for tasks
  - FTL with 4KB mapping granularity
  - Garbage collection with greed policy (select block with minimum # of valid data)
  - ~14% over-provisioning factor
Evaluation methodology

- **Storage system configuration**
  - FTL tasks: host request handling & garbage collection
    - Generate a stream of asynchronous flash memory requests
    - Inter-arrival time for requests to model processing overhead for tasks
  - FTL with 4KB mapping granularity
  - Garbage collection with greed policy (select block with minimum # of valid data)
  - ~14% over-provisioning factor

- **Workload configuration**
  - Issue rate: 5K IOPS
    - Such that both host request handling & garbage collection run concurrently
    - While not causing requests to queue up unboundedly
  - Duration: 1 hour simulation time (up to 18M IOs)
## Experiment 1: Establishing baseline

<table>
<thead>
<tr>
<th>QoS-unaware</th>
<th>: schedule in order of arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throttling</td>
<td>: limit bandwidth use</td>
</tr>
<tr>
<td>QoS-aware (FSS)</td>
<td>: 50:50 share</td>
</tr>
</tbody>
</table>
Experiment 1: Establishing baseline

QoS-unaware: schedule in order of arrival
Throttling: limit bandwidth use
QoS-aware (FSS): 50:50 share

Throughput (IOPS)
Time (sec)
Cumulative probability
Response time (us)

99.9%: <47ms
@10ms: 56%
Experiment 1: Establishing baseline

QoS-unaware: schedule in order of arrival
Throttling: limit bandwidth use
QoS-aware (FSS): 50:50 share

- Throughput (IOPS) vs. Time (sec)
- Cumulative probability vs. Response time (us)

99.9%: <28ms
@10ms: 74%
@10ms: 56%

QoS-unaware vs. SW throttling
Experiment 1: Establishing baseline

QoS-unaware: schedule in order of arrival
Throttling: limit bandwidth use
QoS-aware (FSS): 50:50 share

99.9%: <3.2ms
@10ms: 100%

99.9%: <3.2ms
@10ms: 74%

99.9%: <3.2ms
@10ms: 56%
Experiment 2: Effects of share weight

H80G20: Static share of 80% for host, 20% for GC
H20G80: Static share of 20% for host, 80% for GC
Experiment 2: Effects of share weight

H80G20: Static share of 80% for host, 20% for GC
H20G80: Static share of 20% for host, 80% for GC

Random read/write workload

99.9%: <1.3ms
99.9%: <15ms
Experiment 2: Effects of share weight

**H80G20**: Static share of 80% for host, 20% for GC

**H20G80**: Static share of 20% for host, 80% for GC

Random read/write workload

- 99.9%: <1.3ms
- 99.9%: <15ms

Composite workload

- 99.9%: <19ms
- 99.99%: <32ms
Experiment 2: Effects of share weight

H80G20: Static share of 80% for host, 20% for GC
H20G80: Static share of 20% for host, 80% for GC

Sequential write
Random read/write
Experiment 3: Dynamically adjusting weight

- **H80G20**: Static share of 80% for host, 20% for GC
- **H20G80**: Static share of 20% for host, 80% for GC
- **FSS+DSA**: Share dynamically allocated

Cumulative probability and Read response time (us) graphs are shown, indicating:
- 99.9%: <4.2ms
- 99.9%: <15ms

Graphs show performance metrics for FSS.H80G20 Read and FSS.H20G80 Read.

Memory & Storage Architecture Lab.
@ Seoul National University
Experiment 3: Dynamically adjusting weight

H80G20: Static share of 80% for host, 20% for GC
H20G80: Static share of 20% for host, 80% for GC
FSS+DSA: Share dynamically allocated

99.9%: <3.8ms
99.9%: <4.2ms
99.9%: <15ms

Cumulative probability vs. Read response time (us)

# of free blocks vs. Time (sec)
Experiment 4: Putting it all together

H80G20 : Static share of 80% for host, 20% for GC
H20G80 : Static share of 20% for host, 80% for GC
FSS+DSA : Share dynamically allocated
FSS+DSA+NRH : All techniques applied
Experiment 4: Putting it all together

H80G20 : Static share of 80% for host, 20% for GC
H20G80 : Static share of 20% for host, 80% for GC
FSS+DSA : Share dynamically allocated
FSS+DSA+NRH : All techniques applied

99.9%: <4.2ms
99.9%: <3.8ms
99.9%: <15ms

Cumulative probability vs. Read response time

Conclusion

Background

Design

Evaluation

Conclusion
Experiment 4: Putting it all together

H80G20 : Static share of 80% for host, 20% for GC
H20G80 : Static share of 20% for host, 80% for GC
FSS+DSA : Share dynamically allocated
FSS+DSA+NRH : All techniques applied

Cumulative probability

Read response time

99.9%: <1.5ms
99.9%: <4.2ms
99.9%: <3.8ms
99.9%: <15ms
99.9%: <3.8ms

Throttle
99.9%: <26ms
QoS-unaware
99.9%: <46ms
Conclusion

- Design for reducing response time variation
  - Fairly schedule requests at the controller-level according to share
  - Dynamically adjust the share weight depending on the state of the system
  - Balance the load across multiple chips

**QoS-aware Flash Memory Controller**

Improves average response time
- by 12 ~ 38 for reads
- by 1.4 ~ 6.9 for writes

Improves 99.9% response time
- by 29 ~ 56 for reads
- by 2.0 ~ 8.5 for writes
Future directions

- Generalizing the scheduler
  - Implement other FTL tasks
  - Support any combination of tasks

- Hardware prototyping
  - Implement on an FPGA development board
  - Run real workloads in real-time

- And much more!