Alleviating Garbage Collection Interference through Spatial Separation in All Flash Arrays

Jaeho Kim, Kwanghyun Lim*, Youngdon Jung, Sungjin Lee, Changwoo Min, Sam H. Noh

*Currently with Cornell Univ.
All Flash Array (AFA)

- Storage infrastructure that contains only flash memory drives
  - Also called Solid-State Array (SSA)

https://images.google.com/
https://www.purestorage.com/resources/glossary/all-flash-array.html
Example of All Flash Array Products (1 brick or node)

<table>
<thead>
<tr>
<th></th>
<th>EMC XtremIO</th>
<th>HPE 3PAR</th>
<th>SKHynix AFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>36 ~ 144 TB</td>
<td>750 TB</td>
<td>552 TB</td>
</tr>
<tr>
<td>Number of SSDs</td>
<td>18 ~ 72</td>
<td>120</td>
<td>576</td>
</tr>
<tr>
<td>Network Ports</td>
<td>4~8 x 10Gb iSCSI</td>
<td>4~12 x 16Gb FC</td>
<td>3 x Gen3 PCIe</td>
</tr>
<tr>
<td>Aggregate Network Throughput</td>
<td>5 ~ 10 GB/s</td>
<td>8 ~ 24 GB/s</td>
<td>48 GB/s</td>
</tr>
</tbody>
</table>

A: EMC XtremIO X2 Specification
B: HPE 3PAR StoreServ Specification
C: Performance Analysis of NVMe SSD-Based All-flash Array Systems. [ISPASS’18]

https://www.flaticon.com/
## Example of All Flash Array Products (1 brick or node)

<table>
<thead>
<tr>
<th></th>
<th>EMC XtremIO</th>
<th>HPE 3PAR</th>
<th>SKHynix AFA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capacity</strong></td>
<td>36 ~ 144 TB</td>
<td>750 TB</td>
<td>552 TB</td>
</tr>
<tr>
<td><strong>Number of SSDs</strong></td>
<td>18 ~ 72</td>
<td>120</td>
<td>576</td>
</tr>
<tr>
<td><strong>Network Ports</strong></td>
<td>4~8 x 10Gb iSCSI</td>
<td>4~12 x 16Gb FC</td>
<td>3 x Gen3 PCIe</td>
</tr>
<tr>
<td><strong>Aggregate Network Throughput</strong></td>
<td>5 ~ 10 GB/s</td>
<td>8 ~ 24 GB/s</td>
<td>48 GB/s</td>
</tr>
</tbody>
</table>

A: EMC XtremIO X2 Specification  
B: HPE 3PAR StoreServ Specification  
C: Performance Analysis of NVMe SSD-Based All-flash Array Systems. [*ISPASS’18*]  
[https://www.flaticon.com/](https://www.flaticon.com/)
## SSDs for Enterprise

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Product Name</th>
<th>Seq. Read Throughput</th>
<th>Seq. Write Throughput</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>DC P4800X</td>
<td>2.5 GB/s</td>
<td>2.2 GB/s</td>
<td>1.5 TB</td>
</tr>
<tr>
<td>Intel</td>
<td>DC D3700</td>
<td>2.1 GB/s</td>
<td>1.5 GB/s</td>
<td>1.6 TB</td>
</tr>
<tr>
<td>Intel</td>
<td>DC P3608</td>
<td>5 GB/s</td>
<td>3 GB/s</td>
<td>4 TB</td>
</tr>
<tr>
<td>Samsung</td>
<td>PM1725b</td>
<td>6.3 GB/s</td>
<td>3.3 GB/s</td>
<td>12.8 TB</td>
</tr>
<tr>
<td>Samsung</td>
<td>PM983</td>
<td>3.2 GB/s</td>
<td>2 GB/s</td>
<td>3.8 TB</td>
</tr>
</tbody>
</table>

Bandwidth Trends for Network and Storage Interfaces

- PCIe: https://en.wikipedia.org/wiki/PCI_Express
- Infiniband: https://en.wikipedia.org/wiki/InfiniBand

Bandwidth (GB/s) over Years:
- SATA2
- SATA3
- SAS-2
- SAS-3
- SATA Exp.
- SAS-4
- 10GbE
- 40GbE
- 100GbE
- 200GbE
- 400GbE
- PCIe 3
- PCIe 4
- PCIe 5

Interfaces: https://en.wikipedia.org/wiki/List_of_interface_bit_rates#Local_area_networks
SATA: https://en.wikipedia.org/wiki/Serial_ATA
PCIe: https://en.wikipedia.org/wiki/PCI_Express
Bandwidth Trends for Network and Storage Interfaces

- Storage throughput increases quickly
- Storage isn’t bottleneck anymore

Interfaces: https://en.wikipedia.org/wiki/List_of_interface_bit_rates#Local_area_networks
SATA: https://en.wikipedia.org/wiki/SerialATA
PCIe: https://en.wikipedia.org/wiki/PCI_Express
### Example of All Flash Array Products (1 brick or node)

<table>
<thead>
<tr>
<th></th>
<th>EMC XtremIO</th>
<th>HPE 3PAR</th>
<th>SKHynix AFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>36 ~ 144 TB</td>
<td>750 TB</td>
<td>552 TB</td>
</tr>
<tr>
<td>Number of SSDs</td>
<td>18 ~ 72</td>
<td>120</td>
<td>576</td>
</tr>
<tr>
<td>Network Ports</td>
<td>4~8 x 10Gb iSCSI</td>
<td>4~12 x 16Gb FC</td>
<td>3 x Gen3 PCIe</td>
</tr>
<tr>
<td>Aggregate Network</td>
<td>5 ~ 10 GB/s</td>
<td>8 ~ 24 GB/s</td>
<td>48 GB/s</td>
</tr>
</tbody>
</table>

A: EMC XtremIO X2 Specification
B: HPE 3PAR StoreServ Specification
C: Performance Analysis of NVMe SSD-Based All-flash Array Systems. [ISPASS’18]
### Example of All Flash Array Products (1 brick or node)

<table>
<thead>
<tr>
<th>Product</th>
<th>Capacity</th>
<th>Number of SSDs</th>
<th>Network Ports</th>
<th>Aggregate Network Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMC XtremIO X2</td>
<td>750 TB</td>
<td>120</td>
<td>4~12 x 16Gb FC</td>
<td>8~24 GB/s</td>
</tr>
<tr>
<td>HPE 3PAR StoreServ</td>
<td>552 TB</td>
<td>576</td>
<td>3 x Gen3 PCIe</td>
<td>48 GB/s</td>
</tr>
</tbody>
</table>

Throughput of a few high-end SSDs can easily saturate the network throughput.

A: EMC XtremIO X2 Specification  
B: HPE 3PAR StoreServ Specification  
C: Performance Analysis of NVMe SSD-Based All-flash Array Systems. [ISPASS’18]
Current Trends and Challenges

**Trends**
- Performance of SSDs is fairly high
- Throughput of a few SSDs easily saturates network bandwidth of a AFA node

**Challenges**
- Garbage Collection (GC) of SSD is still performance bottleneck in AFA
- What is an ideal way to manage an array of SSDs with the current trends?
Traditional RAID Approaches

- Traditional RAID employs in-place update for serving write requests
- High GC overhead inside SSD due to random write from the host

Previous solutions:
1) Harmonia [MSST’11]
2) HPDA [TOS’12]
3) GC-Steering [IPDPS’18]
Log-(based) RAID Approaches

- Log-based RAID employs log-structured writes to reduce GC overhead inside SSD
- Log-structured writes involve host-level GC, which relies on idle time
- If no idle time, GC will cause performance drop

Previous solutions:
1) SOFA [SYSTOR’14]
2) SRC [Middleware’15]
3) SALSA [MASCOTS’18]
Performance of a Log-based RAID

- **Configuration**
  - Consist of 8 SSDs (roughly 1TB capacity)

- **Workload**
  - Random write requests continuously for 2 hours
Performance of a Log-based RAID

• Configuration
  • Consist of 8 SSDs (roughly 1TB capacity)

• Workload
  • Random write requests continuously for 2 hours
Performance of a Log-based RAID

- **Configuration**
  - Consist of 8 SSDs (roughly 1TB capacity)

- **Workload**
  - Random write requests continuously for 2 hours

How can we avoid this performance variation due to GC in All Flash Array?
Our Solution (SWAN)

- SWAN (Spatial separation Within an Array of SSDs on a Network)

- Goals
  - Provide sustainable performance up to network bandwidth of AFA
  - Alleviate GC interference between user I/O and GC I/O
  - Find an efficient way to manage an array of SSDs in AFA

- Approach
  - Minimize GC interference through SPATIAL separation

Image: https://clipartix.com/swan-clipart-image-44906/
Our Solution: Brief Architecture of SWAN

- Divide an array of SSDs into front-end and back-end like 2-D array
  - Called, SPATIAL separation
- Employ log-structured writes
- GC effect is minimized by spatial separation
Our Solution: Brief Architecture of SWAN

- Divide an array of SSDs into front-end and back-end like 2-D array
  - Called, SPATIAL separation
  - Employ log-structured writes
  - GC effect is minimized by spatial separation
Our Solution: Brief Architecture of SWAN

- Divide an array of SSDs into front-end and back-end like 2-D array
  - Called, SPATIAL separation
  - Employ log-structured writes
  - GC effect is minimized by spatial separation
Our Solution: Brief Architecture of SWAN

- Divide an array of SSDs into front-end and back-end like 2-D array
  - Called, SPATIAL separation
- Employ log-structured writes
- GC effect is minimized by spatial separation
Our Solution: Brief Architecture of SWAN

- Divide an array of SSDs into front-end and back-end like 2-D array
  - Called, SPATIAL separation
- Employ log-structured writes
- GC effect is minimized by spatial separation

Random writes
SWAN
- Log-structured write
Front-end
Back-end

Reduced GC effect
Our Solution: Brief Architecture of SWAN

**Log-based RAID:** Temporal separation between GC and user I/O

**SWAN:** Spatial separation between GC and user I/O
Architecture of SWAN

• Spatial separation
  • Front-end: serve all write requests
  • Back-end: perform SWAN’s GC

• Log-structured write
  • Segment based append only writes, which is flash friendly
  • Mapping table: 4KB granularity mapping table

• Implemented in block I/O layer
  • where I/O requests are redirected from the host to the storage
Example of Handling I/O in SWAN

Block I/O Interface

Logical Volume  ...

Physical Volume  ...

Front-end  Back-end  Back-end
SSD  SSD  SSD
Example of Handling I/O in SWAN

Block I/O Interface

Logical Volume

Physical Volume

Segment

Front-end

Back-end

Back-end

SSD

W

R

Write req.

Read req.

... W1 W3 R7 R8 ...

... W1 W3 ...
Example of Handling I/O in SWAN

Block I/O Interface

Logical Volume

Physical Volume

like RAID parallelism
Example of Handling I/O in SWAN

Block I/O Interface

Logical Volume

Physical Volume

like RAID parallelism

like RAID parallelism
Procedure of I/O Handling (1/3)

- **Front-end** absorbs all write requests in *append-only* manner
  - To exploit **full performance** of SSDs

(a) First - phase
Procedure of I/O Handling (1/3)

- **Front-end** absorbs all write requests in *append-only* manner
- To exploit **full performance** of SSDs

(a) First - phase
Procedure of I/O Handling (1/3)

- **Front-end** absorbs all write requests in *append-only* manner
- To exploit **full performance** of SSDs

(a) First - phase

- **Front-end** absorbs all write requests in *append-only* manner
- **Parity** unit
- **Parallelism unit**

- **Write Req.**

- **Back-end**
  - **SSD**
  - **SSD**
  - **SSD**

- **Append only**
Procedure of I/O Handling (2/3)

• When the front-end becomes full
  • Empty back-end becomes front-end to serve write requests
  • Full front-end becomes back-end
  • Again, new front-end serves write requests

(a) Second - phase
Procedure of I/O Handling (2/3)

• When the front-end becomes full
  • Empty back-end becomes front-end to serve write requests
  • Full front-end becomes back-end
  • Again, new front-end serves write requests

(a) Second - phase
Procedure of I/O Handling (2/3)

• When the front-end becomes full
  • Empty back-end becomes front-end to serve write requests
  • Full front-end becomes back-end
  • Again, new front-end serves write requests

(a) Second-phase
Procedure of I/O Handling (2/3)

- When the front-end becomes full
  - Empty back-end becomes front-end to serve write requests
  - Full front-end becomes back-end
  - Again, new front-end serves write requests

(a) Second - phase
Procedure of I/O Handling (2/3)

- When the front-end becomes full
  - Empty back-end becomes front-end to serve write requests
  - Full front-end becomes back-end
  - Again, new front-end serves write requests

(a) Second-phase
Procedure of I/O Handling (2/3)

- When the front-end becomes full
  - Empty back-end becomes front-end to serve write requests
  - Full front-end becomes back-end
  - Again, new front-end serves write requests
Procedure of I/O Handling (2/3)

- When the front-end becomes full
  - Empty back-end becomes front-end to serve write requests
  - Full front-end becomes back-end
  - Again, new front-end serves write requests

(a) Second-phase
Procedure of I/O Handling (3/3)

- When there is no more empty back-end
  - SWAN’s GC is triggered to make free space
  - SWAN chooses a victim segment from one of the back-ends
  - SWAN writes valid blocks within the chosen back-end
  - Finally, the victim segment is trimmed

(a) Third - phase
Procedure of I/O Handling (3/3)

- When there is **no more empty back-end**
  - SWAN’s GC is triggered to make free space
  - SWAN chooses a victim segment from one of the back-ends
  - SWAN writes valid blocks **within the chosen back-end**
  - Finally, the victim segment is **trimmed**

(a) Third - phase
Procedure of I/O Handling (3/3)

- When there is no more empty back-end
  - SWAN’s GC is triggered to make free space
  - SWAN chooses a victim segment from one of the back-ends
  - SWAN writes valid blocks within the chosen back-end
  - Finally, the victim segment is trimmed

(a) Third - phase
Procedure of I/O Handling (3/3)

- When there is no more empty back-end
  - SWAN’s GC is triggered to make free space
  - SWAN chooses a victim segment from one of the back-ends
  - SWAN writes valid blocks within the chosen back-end
  - Finally, the victim segment is trimmed

Ensure writing a segment sequentially inside SSDs

(a) Third-Phase
Procedure of I/O Handling (3/3)

- When there is no more empty back-end
  - SWAN’s GC is triggered to make free space
  - SWAN chooses a victim segment from one of the back-ends
  - SWAN writes valid blocks within the chosen back-end
  - Finally, the victim segment is trimmed

All write requests and GC are spatially separated
Feasibility Analysis of SWAN

- How many SSDs in front-end?
- How many back-ends in AFA?
- Analytic model of SWAN GC

Please refer to our paper for details!
Evaluation Setup

• Environment
  • Dell R730 server with Xeon CPUs and 64GB DRAM
  • Up to 9 SATA SSDs are used (up to 1TB capacity)
  • Open channel SSD for monitoring internal activity of an SSD

• Target Configurations
  • RAID0/4: Traditional RAID
  • Log-RAID0/4: Log-based RAID
  • SWAN0/4: Our solution

• Workloads
  • Micro-benchmark: Random write request
  • YCSB C benchmark

<table>
<thead>
<tr>
<th>No parity</th>
<th>1 parity per stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAID0</td>
<td>RAID4</td>
</tr>
<tr>
<td>Log-RAID0</td>
<td>Log-RAID4</td>
</tr>
<tr>
<td>SWAN0</td>
<td>SWAN4</td>
</tr>
</tbody>
</table>

Please refer to paper for more results!
Random Write Requests for 2 Hours (8KB Sized Req.)

Log-RAID0

SWAN0
Random Write Requests for 2 Hours (8KB Sized Req.)

Interference between GC I/O and user I/O

Log-RAID0

SWAN0
Analysis of Log-RAID’s Write Performance

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>SSD1</th>
<th>SSD2</th>
<th>SSD3</th>
<th>SSD4</th>
<th>SSD5</th>
<th>SSD6</th>
<th>SSD7</th>
<th>SSD8</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

User observed throughput

GC starts here
Analysis of Log-RAID’s Write Performance

GC starts here: All SSDs involved in GC

Log-RAID0

User observed throughput
Analysis of Log-RAID’s Write Performance

GC starts here: All SSDs involved in GC

Red lines increases while blue lines drop down since GC incurs read and write operations

User observed throughput

Log-RAID0
Analysis of Log-RAID’s Write Performance

GC starts here: All SSDs involved in GC

Red lines increases while blue lines drop down since GC incurs read and write operations

Performance fluctuates as all SSDs are involved in GC

User observed throughput

Log-RAID0

Throughput (MB/sec)

Time (sec)
Analysis of SWAN’s Write Performance

- SWAN has 1 front-end and 4 back-ends
- Front/back-ends consists of 2 SSDs

User observed throughput

GC starts here

Throughput (MB/sec)

Time (sec)

Configuration:

- SSD 1
- SSD 2
- SSD 3
- SSD 4
- SSD 5
- SSD 6
- SSD 7
- SSD 8
Analysis of SWAN’s Write Performance

- **Front-end**
  - SSD 1
  - SSD 2
  - SSD 3
  - SSD 4
  - SSD 5
  - SSD 6
  - SSD 7
  - SSD 8

- **Write throughput**
  - User observed throughput

- **Configuration**
  - SWAN has 1 front-end and 4 back-ends
  - Front/back-ends consists of 2 SSDs

- GC starts here

> Write throughput and Read throughput

**Time (sec)**
- Throughput (MB/sec)
Analysis of SWAN’s Write Performance

SWAN has 1 front-end and 4 back-ends
Front/back-ends consists of 2 SSDs

GC starts here
Analysis of SWAN’s Write Performance

- **Front-end** consists of 2 SSDs
- SWAN has 1 front-end and 4 back-ends
- Front/back-ends consists of 2 SSDs

GC starts here

User observed throughput

![Diagrams showing throughput and time for various SSDs and configurations]
Analysis of SWAN’s Write Performance

- SWAN has 1 front-end and 4 back-ends
- Front/back-ends consists of 2 SSDs
### Analysis of SWAN’s Write Performance

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Throughput (MB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>SSD1, SSD2, SSD3, SSD4, SSD5, SSD6, SSD7, SSD8</td>
</tr>
<tr>
<td>1200</td>
<td>SSD1, SSD2, SSD3, SSD4, SSD5, SSD6, SSD7, SSD8</td>
</tr>
<tr>
<td>1800</td>
<td>SSD1, SSD2, SSD3, SSD4, SSD5, SSD6, SSD7, SSD8</td>
</tr>
<tr>
<td>2400</td>
<td>SSD1, SSD2, SSD3, SSD4, SSD5, SSD6, SSD7, SSD8</td>
</tr>
<tr>
<td>3000</td>
<td>SSD1, SSD2, SSD3, SSD4, SSD5, SSD6, SSD7, SSD8</td>
</tr>
<tr>
<td>3600</td>
<td>SSD1, SSD2, SSD3, SSD4, SSD5, SSD6, SSD7, SSD8</td>
</tr>
</tbody>
</table>

- **GC starts here**: Only one back-end is involved in GC.
- **Configuration**:
  - SWAN has 1 front-end and 4 back-ends.
  - Front/back-ends consist of 2 SSDs.

### User observed throughput

- Read throughput: 85 MB/sec.
Throughput (MB/sec) | Time (sec)
---|---
600 | 1200
1800 | 2400
3000 | 3600

SSD1 | SSD2 | SSD3 | SSD4 | SSD5 | SSD6 | SSD7 | SSD8

GC starts here

Only one back-end is involved in GC

- This pattern continues
- SWAN separates write requests and GC

**Configuration**
- SWAN has 1 front-end and 4 back-ends
- Front/back-ends consists of 2 SSDs

User observed throughput

**Analysis of SWAN's Write Performance**

**Write throughput**

**Read throughput**
Read Tail Latency for YCSB-C

- SWAN4 shows the shortest read tail latency
- RAID4 and Log-RAID4 suffers long tail latency

Spatial separation is effective for handling read requests as well
Benefits with Simpler SSDs

• SWAN can save cost and power consumption w/o compromising performance by adopting simpler SSDs
  1) Smaller DRAM size
  2) Smaller over-provisioning space (OPS)
  3) Block or segment level FTL instead of page-level FTL

SWAN sequentially writes data to segments and TRIMs a large chunk of data in the same segment at once
Conclusion

- Provide full write performance of an array of SSDs up to network bandwidth limit

- Alleviate GC interference through separation of I/O induced by application and GC of All Flash Array

- Introduce an efficient way to manage SSDs in All Flash Array

Thanks for attention!

Q&A
Backup slides
Handling Read Requests in SWAN

• Recent updated data might be served at page cache or buffer

• Falling in front-end
  • Give the highest priority to read requests

• Falling in GC back-end
  • Preempt GC then serve read requests

• Falling in idle back-ends
  • Serve immediately read requests
GC overhead inside SSDs

- GC overhead should be very low inside SSDs
  - SWAN writes all the data in a segment-based append-only manner
  - Then, SWAN gives TRIMs to ensure writing a segment sequentially inside SSDs
## Previous Solutions

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Write Strategy</th>
<th>How Separate User &amp; GC I/O</th>
<th>Disk Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harmonia [MSST’11]</td>
<td>In-place write</td>
<td>Temporal (Idle time)</td>
<td>RAID-0</td>
</tr>
<tr>
<td>HPDA [IPDPS’10]</td>
<td>In-place write</td>
<td>Temporal</td>
<td>RAID-4</td>
</tr>
<tr>
<td>GC-Steering [IPDPS’18]</td>
<td>In-place write</td>
<td>Temporal</td>
<td>RAID-4/5</td>
</tr>
<tr>
<td>SOFA [SYSTOR’14]</td>
<td>Log write</td>
<td>Temporal</td>
<td>Log-RAID</td>
</tr>
<tr>
<td>SALSA [MASCOTS’18]</td>
<td>Log write</td>
<td>Temporal</td>
<td>Log-RAID</td>
</tr>
<tr>
<td>Purity [SIGMOD’15]</td>
<td>Log write</td>
<td>Temporal</td>
<td>Log-RAID</td>
</tr>
<tr>
<td><strong>SWAN (Proposed)</strong></td>
<td><strong>Log write</strong></td>
<td><strong>Spatial</strong></td>
<td><strong>2D Array</strong></td>
</tr>
</tbody>
</table>