SSDAlloc: Hybrid SSD/RAM
Memory Management Made Easy

Anirudh Badam
Vivek S. Pai
Princeton University

03/31/2011
Memory in Networked Systems
Memory in Networked Systems

- As a cache to reduce pressure on the disk
  - Memcache like tools
  - Act as front-end caches for Web data back-end
Memory in Networked Systems

- As a cache to reduce pressure on the disk
  - Memcache like tools
  - Act as front-end caches for Web data back-end
- As an index to reduce pressure on the disk
  - Indexes for proxy caches, WAN accelerators and inline data-deduplicators
  - Help avoid false positives and use the disk effectively
Problem: Memory Density
Problem: Memory Density

Total DRAM

$/GB (Total Cost)
Problem: Memory Density

$\text{$/GB (Total Cost)}$

<table>
<thead>
<tr>
<th>Total DRAM</th>
<th>$\text{$/GB (Total Cost)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>37.5</td>
</tr>
<tr>
<td>16</td>
<td>75</td>
</tr>
<tr>
<td>32</td>
<td>112.5</td>
</tr>
<tr>
<td>64</td>
<td>150</td>
</tr>
<tr>
<td>128</td>
<td>187.5</td>
</tr>
<tr>
<td>256</td>
<td>225</td>
</tr>
<tr>
<td>512</td>
<td>262.5</td>
</tr>
<tr>
<td>1024</td>
<td>300</td>
</tr>
</tbody>
</table>
Problem: Memory Density

![Bar chart showing memory density costs with a breaking point at 256 GB.](image)

- Breaking Point at 256 GB
Problem: Memory Density

$ / GB (Total Cost)

Breaking Point

8 16 32 64 128 256 512 1024

0 37.5 75 112.5 150

3
Problem: Disk Speed Limits
Problem: Disk Speed Limits

• Magnetic disk speed is not scaling well
  • Capacity is increasing but seek latency is not decreasing
  • About 200 seeks/disk/sec
Problem: Disk Speed Limits

- Magnetic disk speed is not scaling well
  - Capacity is increasing but seek latency is not decreasing
  - About 200 seeks/disk/sec
- High speed disk arrays: many smaller capacity drives
  - Total cost about 10X more compared to similar capacity 7200 rpm drives
  - Use more rack space per byte
Proposal: Use Flash as Memory
Proposal: Use Flash as Memory

- Address DRAM density limitation
  - Overcome per system DRAM limits via flash memory
  - Provide a choice -- more servers or a single server + flash memory
Proposal: Use Flash as Memory

- Address DRAM density limitation
  - Overcome per system DRAM limits via flash memory
  - Provide a choice -- more servers or a single server + flash memory
- Reduce total cost of ownership
  - “Long-tailed” workloads are important
  - DRAM too expensive and disk too slow
  - CPU under-utilized due to DRAM limit
Proposal: Use Flash as Memory

- Address DRAM density limitation
  - Overcome per system DRAM limits via flash memory
  - Provide a choice -- more servers or a single server + flash memory
- Reduce total cost of ownership
  - “Long-tailed” workloads are important
  - DRAM too expensive and disk too slow
  - CPU under-utilized due to DRAM limit
- How to ease application development with flash memory?
Flash Memory Primer
Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
- Writes happen after an erase
  - Limited lifetime and endurance
Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
- Writes happen after an erase
  - Limited lifetime and endurance
- No seek latency (only read/write latency)
Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
- Writes happen after an erase
  - Limited lifetime and endurance
- No seek latency (only read/write latency)
- Large capacity (single 2.5” disk ~ 512GB)
  - PCIe 10.2 TB - Fusion-io io-octal drive
Question of Hierarchy
Question of Hierarchy

Memory

Disk
## Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
</tbody>
</table>
## Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
</tr>
</tbody>
</table>

Disk Block Addressable

Non-volatile

High Latency

Directly Addressed

Virtually Addressed

Low Latency

Byte Addressable

Memory

Virtual Addressable
# Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
</tr>
<tr>
<td>Low Latency</td>
<td>High Latency</td>
</tr>
</tbody>
</table>
## Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
</tr>
<tr>
<td>Low Latency</td>
<td>High Latency</td>
</tr>
<tr>
<td>Volatile</td>
<td>Non-volatile</td>
</tr>
</tbody>
</table>
# Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
</tr>
<tr>
<td>Low Latency</td>
<td>High Latency</td>
</tr>
<tr>
<td>Volatile</td>
<td>Non-volatile</td>
</tr>
</tbody>
</table>
# Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
<th>SSDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
<td>Non-volatile</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
<td></td>
</tr>
<tr>
<td>Low Latency</td>
<td>High Latency</td>
<td></td>
</tr>
<tr>
<td>Volatile</td>
<td>Non-volatile</td>
<td></td>
</tr>
</tbody>
</table>
Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
</tr>
<tr>
<td>Low Latency</td>
<td>High Latency</td>
</tr>
<tr>
<td>Volatile</td>
<td>Non-volatile</td>
</tr>
</tbody>
</table>

Flash has low latency
Question of Hierarchy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Addressable</td>
<td>Block Addressable</td>
</tr>
<tr>
<td>Virtually Addressed</td>
<td>Directly Addressed</td>
</tr>
<tr>
<td>Low Latency</td>
<td>High Latency</td>
</tr>
<tr>
<td>Volatile</td>
<td>Non-volatile</td>
</tr>
</tbody>
</table>

SSDs

Flash has low latency
Transparent Tiering Today
Transparent Tiering Today

• Use it as memory via swap or mmap
  • Application need not be modified
  • Pages transparently swapped in and out based on usage in DRAM
• Use it as memory via swap or mmap
  • Application need not be modified
  • Pages transparently swapped in and out based on usage in DRAM
  • Native pager delivers only 10% of the SSD’s performance
  • Flash aware pager delivers only 30% of the SSD’s performance
  • OS pager optimized for seek limited disks and was designed as a “dead page storage”
Transparent Tiering Today

RAM

SSD

RAM

SSD
## Transparent Tiering Today

### RAM

| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

### SSD

| 1 | 2 | 3 | 4 |
| 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 |
| 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 |
| 53 | 54 | 55 | 56 |

- **O**: read

---

RAM

SSD
Transparent Tiering Today

RAM

SSD

1 2 3 4
5 6 7 8
9 10 11 12
29 30 31 32

17 18 19 20
21 22 23 24
25 26 27 28
41 42 43 44

33 34 35 36
37 38 39 40
41 42 43 44
57 58 59 60

49 50 51 52
53 54 55 56
57 58 59 60
61 62 63 64

read
Transparent Tiering Today

RAM

SSD

read

Circle 31
Transparent Tiering Today

RAM

SSD

RAM: 读

SSD: 写

读操作示意图
Transparent Tiering Today

RAM

SSD
Transparent Tiering Today

RAM

SSD

Transitioning from RAM to SSD

1-4: RAM
5-8: RAM
9-12: RAM
29-32: SSD
33-36: SSD
37-40: SSD
41-44: SSD
45-48: SSD
49-52: SSD
53-56: SSD

Read operations are indicated by the green circle, while write operations are indicated by the red circle.
Transparent Tiering Today

RAM

SSD
Transparent Tiering Today

RAM

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>32</th>
</tr>
</thead>
</table>

SSD

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
</tbody>
</table>
Transparent Tiering Today

RAM

SSD

1 2 3 4
5 6 7 8
9 10 11 12
29 30 31 32

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

17 18 19 20
21 22 23 24
25 26 27 28
29 30 31 32

33 34 35 36
37 38 39 40
41 42 43 44
45 46 47 48

49 50 51 52
53 54 55 56
57 58 59 60
61 62 63 64
Transparent Tiering Today

RAM

- Read
- Write
- Free()

SSD
Transparent Tiering Today

RAM

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
</tbody>
</table>

SSD

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
</tbody>
</table>
Transparency Tiering Today

**RAM**

- **Read**
- **Write**
- **Free()**

Indirection Table → In the OS or in the FTL

**SSD** (log structured page store)
Non-Transparent Tiering
Non-Transparent Tiering

- Redesign application to be flash aware
  - Custom object store with custom pointers
  - Reads, writes and garbage collection at an application object granularity
  - Avoid in-place writes (objects could be small)
  - Obtain the best performance and lifetime from flash memory device
Non-Transparent Tiering

- Redesign application to be flash aware
  - Custom object store with custom pointers
  - Reads, writes and garbage collection at an application object granularity
  - Avoid in-place writes (objects could be small)
  - Obtain the best performance and lifetime from flash memory device
  - Intrusive modifications needed
  - Expertise with flash memory needed
Non-Transparent Tiering

malloc + SSD-swap

MyObject* obj = malloc( sizeof( MyObject ) );
obj->x = 0;
obj->y = 1;
obj->z = 2;
free( obj );

MyObjectID oid = createObject( sizeof( MyObject ) );
MyObject* obj = malloc( sizeof( MyObject ) );
readObject( oid, obj );
obj->x = 0;
obj->y = 1;
obj->z = 2;
writeObject( oid, obj );
free( obj );
Our Goal
Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
- Use the DRAM effectively
  - Use it as an object cache (not as a page cache)
Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
- Use the DRAM effectively
  - Use it as an object cache (not as a page cache)
- Use the SSD wisely
  - As a log-structured object store
Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
- Use the DRAM effectively
  - Use it as an object cache (not as a page cache)
- Use the SSD wisely
  - As a log-structured object store
- Reorganize virtual memory allocation to discern object information
SSDAlloc Overview

- Application
- Virtual Memory (Object per page - OPP)
- Physical Memory
- SSD
SSDAlloc Overview

Application

Virtual Memory (Object per page - OPP)

Physical Memory

SSD

Memory Manager: Creates 64 objects of 1KB size
SSDAlloc Overview

<table>
<thead>
<tr>
<th>Application</th>
<th>Memory Manager: Creates 64 objects of 1KB size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Memory (Object per page - OPP)</td>
<td></td>
</tr>
<tr>
<td>Physical Memory</td>
<td></td>
</tr>
<tr>
<td>SSD</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
</tbody>
</table>
SSDAloc Overview

Application

Virtual Memory (Object per page - OPP)

Memory Manager: Creates 64 objects of 1KB size

Physical Memory

SSD

Page Buffer
SSDAlloc Overview

<table>
<thead>
<tr>
<th>Application</th>
<th>Memory Manager: Creates 64 objects of 1KB size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Memory (Object per page - OPP)</td>
<td>1</td>
</tr>
<tr>
<td>Physical Memory</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>33</td>
</tr>
<tr>
<td>SSD</td>
<td>Page Buffer</td>
</tr>
</tbody>
</table>
SSDAlloc Overview

Memory Manager: Creates 64 objects of 1KB size

Virtual Memory (Object per page - OPP)

Physical Memory

SSD

Log structured object store
SSDAlloc Options
# SSDAlloc Options

<table>
<thead>
<tr>
<th>Data Entity</th>
<th>Object Per Page (OPP)</th>
<th>Memory Page (MP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application Defined Objects</td>
<td>4KB objects (like pages)</td>
</tr>
</tbody>
</table>
# SSDAlloc Options

<table>
<thead>
<tr>
<th></th>
<th>Object Per Page (OPP)</th>
<th>Memory Page (MP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Entity</strong></td>
<td>Application Defined Objects</td>
<td>4KB objects (like pages)</td>
</tr>
<tr>
<td><strong>Memory Manager</strong></td>
<td>Pool Allocator</td>
<td>Coalescing Allocator</td>
</tr>
</tbody>
</table>

- **Physical Memory**: Separate Page Buffer & RAM Object Cache
- **Virtual Memory**: No. of objects * page_size
- **No. of pages** * page_size

- **Object Per Page**: Application Defined Objects
- **Memory Page**: 4KB objects (like pages)

- **Code Changes**: Minimal changes restricted to memory allocation
- **SSD Usage**: Log-structured Object Store, Log-structured Page Store
- **SSDAlloc Options**: Minimal changes restricted to memory allocation, No changes needed
## SSDAlloc Options

<table>
<thead>
<tr>
<th>Data Entity</th>
<th>Object Per Page (OPP)</th>
<th>Memory Page (MP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application Defined Objects</td>
<td>4KB objects (like pages)</td>
</tr>
<tr>
<td>Memory Manager</td>
<td>Pool Allocator</td>
<td>Coalescing Allocator</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>No. of objects * page_size</td>
<td>No. of pages * page_size</td>
</tr>
</tbody>
</table>
## SSDAlloc Options

<table>
<thead>
<tr>
<th>Data Entity</th>
<th>Object Per Page (OPP)</th>
<th>Memory Page (MP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application Defined Objects</td>
<td>4KB objects (like pages)</td>
</tr>
<tr>
<td>Memory Manager</td>
<td>Pool Allocator</td>
<td>Coalescing Allocator</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>No. of objects * page_size</td>
<td>No. of pages * page_size</td>
</tr>
<tr>
<td>Physical Memory</td>
<td>Separate Page Buffer &amp; RAM Object Cache</td>
<td>No such separation</td>
</tr>
</tbody>
</table>

**Code Changes**

Minimal changes restricted to memory allocation.

**No changes needed**

**SSDAlloc Options**

14
## SSDAlloc Options

<table>
<thead>
<tr>
<th></th>
<th>Object Per Page (OPP)</th>
<th>Memory Page (MP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Entity</strong></td>
<td>Application Defined Objects</td>
<td>4KB objects (like pages)</td>
</tr>
<tr>
<td><strong>Memory Manager</strong></td>
<td>Pool Allocator</td>
<td>Coalescing Allocator</td>
</tr>
<tr>
<td><strong>Virtual Memory</strong></td>
<td>No. of objects * page_size</td>
<td>No. of pages * page_size</td>
</tr>
<tr>
<td><strong>Physical Memory</strong></td>
<td>Separate Page Buffer &amp; RAM Object Cache</td>
<td>No such separation</td>
</tr>
<tr>
<td><strong>SSD Usage</strong></td>
<td>Log-structured Object Store</td>
<td>Log-structured Page Store</td>
</tr>
</tbody>
</table>
## SSDAlloc Options

<table>
<thead>
<tr>
<th></th>
<th>Object Per Page (OPP)</th>
<th>Memory Page (MP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Entity</strong></td>
<td>Application Defined Objects</td>
<td>4KB objects (like pages)</td>
</tr>
<tr>
<td><strong>Memory Manager</strong></td>
<td>Pool Allocator</td>
<td>Coalescing Allocator</td>
</tr>
<tr>
<td><strong>Virtual Memory</strong></td>
<td>No. of objects * page_size</td>
<td>No. of pages * page_size</td>
</tr>
<tr>
<td><strong>Physical Memory</strong></td>
<td>Separate Page Buffer &amp; RAM Object Cache</td>
<td>No such separation</td>
</tr>
<tr>
<td><strong>SSD Usage</strong></td>
<td>Log-structured Object Store</td>
<td>Log-structured Page Store</td>
</tr>
<tr>
<td><strong>Code Changes</strong></td>
<td>Minimal changes restricted to memory allocation</td>
<td>No changes needed</td>
</tr>
</tbody>
</table>
SSDAlloc Overview

- Application
- Virtual Memory
- RAM Object
- Cache
- SSD
SSDAlloc Overview

- A small set of pages in core
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using *mprotect*
SSDAlloc Overview

- A small set of pages in core
- Pages materialized on demand from RAM object cache/SSD
- Restricted in size to minimize RAM wastage (from OPP)
- Implemented using mprotect
SSDAlloc Overview

- A small set of pages in core
- Pages materialized on demand from RAM object cache/SSD
- Restricted in size to minimize RAM wastage (from OPP)
- Implemented using `mprotect`
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using mprotect
  - Page materialized in seg-fault handler
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using mprotect
  - Page materialized in seg-fault handler
- RAM Object Cache continuously flushes dirty objects to the SSD in LRU order
SSD Maintenance
SSD Maintenance

Virtual Memory

Object Tables

RAM Object Cache

Dirty Objects

SSD
SSD Maintenance
SSD Maintenance

- Copy-and-compact garbage-collector/log-writer
  - Seek optimizations not needed
SSD Maintenance

- Copy-and-compact garbage-collector/log-writer
  - Seek optimizations not needed
- Read at the head and write live and dirty objects
  - Use Object Tables to determine liveness
SSD Maintenance

- Copy-and-compact garbage-collector/log-writer
  - Seek optimizations not needed
- Read at the head and write live and dirty objects
  - Use Object Tables to determine liveness
- Garbage is disposed
  - Objects written elsewhere are garbage
  - OPP object which is “free” is garbage
Implementation
Implementation

- 11,000 lines of C++ code (runtime library)
  - Implemented using `mprotect`, `mmap`, and `madvise`
  - `SSDAlloc-OPP` pool and array allocator
  - `SSDAlloc-MP` coalescing allocator (array allocations)
  - `SSDFree` frees the allocated data
  - Can coexist with `malloc` pointers
SSD Usage Techniques
## SSD Usage Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Write Logging</th>
<th>Access &lt; 4KB</th>
<th>Finegrained GC</th>
<th>Avoid DRAM Pollution</th>
<th>High Performance</th>
<th>Programming Ease</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD Swap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>SSD Swap (Write Logged)</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>Application Rewrite</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>
## SSD Usage Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Write Logging</th>
<th>Access &lt; 4KB</th>
<th>Finegrained GC</th>
<th>Avoid DRAM Pollution</th>
<th>High Performance</th>
<th>Programming Ease</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD Swap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSD Swap (Write Logged)</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Application Rewrite</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SSDAlloc</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
SSDAlloc Runtime Overhead
SSDAlloc Runtime Overhead

- Overhead for SSDAlloc runtime intervention

<table>
<thead>
<tr>
<th>Overhead Source</th>
<th>Max Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB Miss (DRAM Read)</td>
<td>0.014 µSec</td>
</tr>
<tr>
<td>Object Table Lookup</td>
<td>0.046 µSec</td>
</tr>
<tr>
<td>Page Materialization</td>
<td>0.138 µSec</td>
</tr>
<tr>
<td>Page Dematerialization</td>
<td>0.172 µSec</td>
</tr>
<tr>
<td>Signal Handling</td>
<td>0.666 µSec</td>
</tr>
<tr>
<td>Combined Overhead</td>
<td>0.833 µSec</td>
</tr>
</tbody>
</table>
SSDAlloc Runtime Overhead

- Overhead for SSDAlloc runtime intervention

<table>
<thead>
<tr>
<th>Overhead Source</th>
<th>Max Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB Miss (DRAM Read)</td>
<td>0.014 μSec</td>
</tr>
<tr>
<td>Object Table Lookup</td>
<td>0.046 μSec</td>
</tr>
<tr>
<td>Page Materialization</td>
<td>0.138 μSec</td>
</tr>
<tr>
<td>Page Dematerialization</td>
<td>0.172 μSec</td>
</tr>
<tr>
<td>Signal Handling</td>
<td>0.666 μSec</td>
</tr>
<tr>
<td>Combined Overhead</td>
<td>0.833 μSec</td>
</tr>
</tbody>
</table>

- NAND Flash latency ~ 30-50 μSec
SSDAlloc Runtime Overhead

- Overhead for SSDAlloc runtime intervention

<table>
<thead>
<tr>
<th>Overhead Source</th>
<th>Max Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB Miss (DRAM Read)</td>
<td>0.014 μSec</td>
</tr>
<tr>
<td>Object Table Lookup</td>
<td>0.046 μSec</td>
</tr>
<tr>
<td>Page Materialization</td>
<td>0.138 μSec</td>
</tr>
<tr>
<td>Page Dematerialization</td>
<td>0.172 μSec</td>
</tr>
<tr>
<td>Signal Handling</td>
<td>0.666 μSec</td>
</tr>
<tr>
<td>Combined Overhead</td>
<td>0.833 μSec</td>
</tr>
</tbody>
</table>

- NAND Flash latency ~ 30-50 μSec
- Can reach 1 Million IOPS
Experiments
Experiments

• Comparing three allocation methods
  • `malloc replaced with SSDAlloc-OPP`
  • `malloc replaced with SSDAlloc-MP`
  • `Swap`
Experiments

• Comparing three allocation methods
  • `malloc` replaced with `SSDAlloc-OPP`
  • `malloc` replaced with `SSDAlloc-MP`
  • `Swap`

• 2.4Ghz Quadcore CPU with 16GB RAM
  • RiData, Kingston, Intel X25-E, Intel X25-V and Intel X25-M
# Results Overview

<table>
<thead>
<tr>
<th>Application</th>
<th>Original LOC</th>
<th>Modified LOC</th>
<th>SSDAlloc-OPP’s gain vs Swap</th>
<th>SSDAlloc-MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>11,193</td>
<td>21</td>
<td>5.5 - 17.4x</td>
<td>1.4 - 3.5x</td>
</tr>
<tr>
<td>B+Tree Index</td>
<td>477</td>
<td>15</td>
<td>4.3 - 12.7x</td>
<td>1.4 - 3.2x</td>
</tr>
<tr>
<td>Packet Cache</td>
<td>1,540</td>
<td>9</td>
<td>4.8 - 10.1x</td>
<td>1.3 - 2.3x</td>
</tr>
<tr>
<td>HashCache</td>
<td>20,096</td>
<td>36</td>
<td>5.3 - 17.1x</td>
<td>1.3 - 3.3x</td>
</tr>
</tbody>
</table>
## Results Overview

<table>
<thead>
<tr>
<th>Application</th>
<th>Original LOC</th>
<th>Modified LOC</th>
<th>SSDAlloc-OPP's gain vs Swap</th>
<th>SSDAlloc-MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>11,193</td>
<td>21</td>
<td>5.5 - 17.4x</td>
<td>1.4 - 3.5x</td>
</tr>
<tr>
<td>B+Tree Index</td>
<td>477</td>
<td>15</td>
<td>4.3 - 12.7x</td>
<td>1.4 - 3.2x</td>
</tr>
<tr>
<td>Packet Cache</td>
<td>1,540</td>
<td>9</td>
<td>4.8 - 10.1x</td>
<td>1.3 - 2.3x</td>
</tr>
<tr>
<td>HashCache</td>
<td>20,096</td>
<td>36</td>
<td>5.3 - 17.1x</td>
<td>1.3 - 3.3x</td>
</tr>
</tbody>
</table>

- SSDAlloc applications write up to 32 times less data to the SSD than when compared to the traditional VM style applications.
Microbenchmarks
Microbenchmarks

- 32GB array of 128 byte objects (32GB SSD, 2GB RAM)
Microbenchmarks

- 32GB array of 128 byte objects (32GB SSD, 2GB RAM)

Throughput Gain Factor

All Reads 25% Reads 50% Reads 75% Reads All Writes

SSDAlloc-OPP over Swap

SSDAlloc-OPP over SSDAlloc-MP
Memcached Benchmarks
Memcached Benchmarks

- 30GB SSD, 4GB RAM, 4 memcache clients
- Memcache server slab allocator modified to use SSDAlloc
Memcached Benchmarks

- 30GB SSD, 4GB RAM, 4 memcache clients
- Memcache server slab allocator modified to use SSDAlloc

![Graph showing throughput (req/sec) vs. average object size]
Memcached Benchmarks
Memcached Benchmarks

- Performance for 50% reads and 50% writes
Memcached Benchmarks

- Performance for 50% reads and 50% writes

![Bar chart showing throughput (req/sec) for different storage options: SSD-swap, SSDAlloc-MP, SSDAlloc-OPP. The y-axis represents throughput (req/sec) ranging from 0 to 30000, and the x-axis represents different storage options. The chart shows varying performance based on the type of storage used.]
Summary

• SSDAlloc migrates SSD naturally into VM system

• RAM as a compact object cache

• Virtual memory addresses are used

• Only memory allocation code changes (9 to 36 LOC)

• Other approaches need intrusive modifications
Summary

• SSDAlloc migrates SSD naturally into VM system
  • RAM as a compact object cache
  • Virtual memory addresses are used
  • Only memory allocation code changes (9 to 36 LOC)
  • Other approaches need intrusive modifications

• SSD as log-structured object store
  • Can obtain 90% raw SSD random read performance
  • Other transparent approaches deliver only 10--30%
  • Reduce write traffic by up to 32 times
Thanks

Anirudh Badam
abadam@cs.princeton.edu
http://tinyurl.com/ssdalloc