DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings

Aayush Gupta  Youngjae Kim  Bhuvan Urgaonkar

The Pennsylvania State University, University Park

ASPLOS 2009

Presented by Wang Yuanxuan
Outline

1. Introduction
2. Background
3. Design of DFTL
4. The FlashSim Simulator
5. Experimental Results
6. Concluding Remarks
1 Introduction

- Flash Memory
- The Flash Translation Layer
- Contributions
Advantages

- Significantly cheaper than main memory
- Price-per-byte is falling
- Improve the performance of disk-based systems via caching and buffering
Defects

- Not always superiors in performance to a disk (in sequential accesses)
- Poor performance with random writes
FTL
Maintains a mapping table of virtual addresses from upper layers to physical addresses

Problem
*Erases are at a much coarser spatial granularity than pages*
Out-of-Place Updates

1. chooses an already erased page
Out-of-Place Updates

1. chooses an already erased page
2. writes to it
Out-of-Place Updates

1. chooses an already erased page
2. writes to it
3. invalidates the previous version of page
Out-of-Place Updates

1. chooses an already erased page
2. writes to it
3. invalidates the previous version of page
4. updates its mapping table

Problem

*GC is needed*
- DFTL
DFTL

FlashSim
Contributions

- DFTL
- FlashSim
- Evaluations of realistic enterprise-scale workloads
Outline

2 Background

- Basics of Flash Memory Technology
- Flash Translation Layer
Granularity

- Blocks
- Pages

Out-of-Band Area

- Error Correctness Code
- Logical page number
- State
  1. valid
  2. invalid
  3. free/erased

Basics of Flash Memory Technology
### Basics of Flash Memory Technology

#### NAND Flash Organization and Access Time Comparision

<table>
<thead>
<tr>
<th>Flash Type</th>
<th>Data Unit Size</th>
<th>Access Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Page (Bytes)</td>
<td>Block (Bytes)</td>
</tr>
<tr>
<td>Small Block</td>
<td>512</td>
<td>16</td>
</tr>
<tr>
<td>Large Block</td>
<td>2048</td>
<td>64</td>
</tr>
</tbody>
</table>

**Other considerations**
- wear-leveling
Two extremes

Granularity

- Page-level
  - Fully associative cache
  - Compact and efficient utilization
  - Requires a large mapping table (32MB for 16GB)
Two extremes

Granularity

- **Page-level**
  - Fully associative cache
  - Compact and efficient utilization
  - Requires a large mapping table (32MB for 16GB)

- **Block-level**
  - Set-associative cache
  - Reduce the size of the mapping table
  - LPN offset within the block is fixed
  - A grow in GC overhead
Hybrid FTL Scheme

- Two groups of blocks
  - Data Blocks: block-level
  - Log/Update Blocks: page-level
- Any update on the data blocks are performed by writes to the log blocks
Hybrid FTL Scheme

- Latest version of 'X' in Data Block?
  - Yes: Logical Address: LPN
    - LBN, offset
    - PBN, offset
    - Block-level FTL
  - No: Physical Address: PPN
    - PPN
    - Page-level FTL
    - Logical Address: LPN
      - LBN, offset
      - PBN, offset

- FLASH:
  - Data Blocks
  - Log Blocks
Various Merge Operations

(a) Switch Merge

(b) Partial Merge

(c) Full Merge
Expensive Full Merge

1. Data Block D1 (Victim)
   - LPN=0, V
   - LPN=1, V
   - LPN=2, I
   - LPN=3, V

2. Data Block D2
   - LPN=4, I
   - LPN=5, I
   - LPN=6, I
   - LPN=7, V

3. Log Block L4
   - LPN=82, V
   - LPN=2, V
   - LPN=30, V
   - LPN=45, V

4. Log Block L2
   - LPN=91, V
   - LPN=9, V
   - LPN=12, I

5. Log Block L1
   - LPN=0, V
   - LPN=5, V
   - LPN=1, V

6. Free/Erased Block
   - LPN=0, F
   - LPN=0, F
3 Design of DFTL
- DFTL Architecture
- Logical to Physical Address Translation
- R/W Operations and GC
Contention

Doing away with log-blocks!

Features

- Making use of temporal locality
- Page-based mapping table on the flash device
- Data area to store the image (2MB for 1GB)
Data Pages and Translation Pages

- Stores active address mappings
- Stores logical to physical address translations
- Fetch mapping entry
- Evict mapping entry for synchronization
- Consult location of translation pages on flash
- Global Translation Directory
  - MVPN MPPN
  - Directory Entries
- Tracks translation pages on flash
- Stores real data from I/O requests
- Cached Mapping Table
  - DLPN DPPN
  - Mapping Entries
- SRAM
- Translation Blocks
- Flash
- Data Blocks
Algorithm

Input: Request's Logical Page Number ($request_{lpn}$), Request's Size ($request_{size}$)
Output: NULL

while $request_{size} \neq 0$ do
  if $request_{lpn}$ miss in Cached Mapping Table then
    if Cached Mapping Table is full then
      /* Select entry for eviction using segmented LRU replacement algorithm */
      victim$_{lpn}$ ← select.victim.entry()
    if victim$_{last_mod.time} \neq victim_{load_time}$ then
      /*victim$_{type}$ : Translation or Data Block
      Translation_Page$_{victim}$ : Physical
      Translation-Page Number containing victim entry */
      Translation_Page$_{victim}$ ← consult_GTD(victim$_{lpn}$)
      victim$_{type}$ ← Translation Block
      DFTL_Service_Request(victim)
    end
  end
  erase_entry(victim$_{lpn}$)
end
Translation_Page$_{request}$ ←
consult_GTD($request_{lpn}$)
/* Load map entry of the request from flash into Cached Mapping Table */
load_entry(Translation_Page$_{request}$)
end
request$_{type}$ ← Data Block
request$_{ppn}$ ← CMT_lookup($request_{lpn}$)
DFTL_Service_Request(request)
request$_{size}$--
end
Logical to Physical Address Translation

### Algorithm

1. \(D_{LPN} = 1280\)
2. Look up in the Cached Mapping Table:
   - \(DLPN = 3\) maps to \(DPNN = 150\)
3. Check in the Global Translation Directory:
   - \(MVPN = 0\) maps to \(MPNN = 21\)
4. Find the Translation Page:
   - \(MPNN = 21\) maps to \(DLPN = 1024, DPPN = 576\)
5. Assign a victim entry:
   - \(MVPN = 2\) maps to \(MPNN = 15\)
6. Assign a victim entry:
   - \(MVPN = 0\) maps to \(MPNN = 23\)
7. Update the Global Translation Directory:
   - \(MPNN = 23\) maps to \(DLPN = 1280, DPPN = 660\)

Data Block:
- \(DLPN = 1280, F \rightarrow V\)
- \(DPPN = 661\)
- \(OOB\)

Translation Block:
- \(MVPN = 2, F \rightarrow V\)
- \(MVPN = 0, V \rightarrow I\)
- \(MVPN = 0, F \rightarrow V\)
Overhead

- Worst-case: 2 translation page reads and 1 translation page write
- Rooted deeply in temporal locality
- Batch updates optimization
- Current Data/Translation Block for writing
- Invoke GC when $GC_{threshold}$ is crossed
Outline

4 The FlashSim Simulator
   - FlashSim
- Built by enhancing Disksim
- Able to simulate different storage sub-system components
- Core FTL engine
Experimental Results

- Setup
- Results
## Evaluation Setup

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Avg. Req. Size (KB)</th>
<th>Read (%)</th>
<th>Seq. (%)</th>
<th>Inter-arrival Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial [25]</td>
<td>4.38</td>
<td>9.0</td>
<td>2.0</td>
<td>133.50</td>
</tr>
<tr>
<td>Cello99 [10]</td>
<td>5.03</td>
<td>35.0</td>
<td>1.0</td>
<td>41.01</td>
</tr>
<tr>
<td>TPC-H [28]</td>
<td>12.82</td>
<td>95.0</td>
<td>18.0</td>
<td>155.56</td>
</tr>
</tbody>
</table>

Baseline: Pure page-based FTL
Results

Average System Response Time

(a) Financial Trace (OLTP)
(b) Cello99
(c) TPC-H
(d) Web-Search
Results

Average System Response Time

(a) Workload A

(b) Workload B
Concluding Remarks
DFTL Offers

- improved performance
- reduced GC overhead
- improved overload behavior
- free from any tunnable parameters