WAFTL:
A Workload Adaptive Flash Translation Layer with Data Partition

Qingsong Wei †
Bozhao Gong€, Suraj Pathak€, Bharadwaj Veeravalli €, Lingfang Zeng€ and Kanzo Okada†

† Data Storage Institute, A-STAR, Singapore
€ National University of Singapore

MSST 2011
May 26, 2011
Flash Fills the Performance Gap

- Much faster than Disk (10X-100X)
- Non-volatile (Disk like)
- Low power consumption
- Light weight
- Resistance to shock

Flash memory fills the performance gap by:

- Processor
- DRAM memory
- Hard Disk
NAND Flash Memory

- Different access unit
  - Read/write in pages (us)
  - Erase in blocks (very slow, ms)

- Out-place-Update: Does not allow overwrite.

- Limited number of erase per cell. 100-300K for SLC and 10-30K for MLC.

- Poor random write performance.

\[
m=64/128/256
\]
**Flash Translation Layer (FTL) in SSD**

- **Host Interface**
  - SATA, SCSI, FC, PCI etc

- **Host I/F Layer**

- **Flash Translation Layer**
  - LBA
  - PBA
  - Data Buffer

- **NAND I/F Layer**
  - NAND Flash

- **Control Bus**
  - Embedded Processor
  - SRAM
  - NAND Flash
  - Flash Bus

**Introduction**
State-of-the-Art

Page-based FTL
- High performance
- High space usage
- Low garbage collection overhead
- Large mapping table
- Not scalable

Address Mapping table

<table>
<thead>
<tr>
<th>LPN</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Flash

Block 0

Block 1

Block 2

Block 3

Host → Logical Page No (LPN)

Physical Page No (PPN)

LPN: 4
**State-of-the-Art**

**Block-based FTL**
- Small mapping table
  - Low performance
  - Low space usage (internal fragmentation)
  - High garbage collection overhead

### Address Mapping table

- **Host**
  - Logical Page No (LPN)
  - Logical Block NO (LBN)
  - Physical Block No (PBN)

<table>
<thead>
<tr>
<th>LBN</th>
<th>PBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

LPN: 7
LBN: 7/2 = 3
Offset: 1

**Flash**

- Block 0
- Block 1
- Block 2
- Block 3

PBN: 1
Offset: 1
Log-buffer based Hybrid FTL

- Small mapping table
- Improved performance comparing to Block-based FTL
- Poor performance for random writes because of expensive merge operations

Association

- One to one (BAST)
- One to Many (FAST, LAST)
- Many to Many (KAST)
DFTL: Demand-based FTL

+ Partial buffered page-based FTL
+ Good Performance (but lower than pure page-based FTL)
+ Reduced memory requirement
- Large mapping table
## State-of-the-Art

<table>
<thead>
<tr>
<th></th>
<th>Page-based FTL</th>
<th>DFTL</th>
<th>Block-based FTL</th>
<th>Hybrid FTL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>High</td>
<td>Good</td>
<td>Low</td>
<td>Middle</td>
</tr>
<tr>
<td><strong>Garbage Collection Overhead</strong></td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Middle</td>
</tr>
<tr>
<td><strong>Favored Workload</strong></td>
<td>Random</td>
<td>Random</td>
<td>Sequential</td>
<td>Sequential</td>
</tr>
<tr>
<td><strong>Mapping table Size</strong></td>
<td>Large</td>
<td>Large</td>
<td>Small</td>
<td>Small</td>
</tr>
<tr>
<td><strong>Workload adaptive</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Scalable with SSD capacity</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Generally limited in one way or another, either in memory requirement, performance, garbage collection overhead or scalability.
Design Rationale

Sequential Data (Large, inactive)

Block-level Mapping Blocks (BMB)

Random Data (Small, active)

Page-level Mapping Blocks (PMB)

Flash Memory
WAFTL: Workload Adaptive FTL

Design Rationale

Buffer Zone

Page-level Mapping

Data Partition

Sequential Data (Large, inactive)

Random Data (Small, active)

Data Partition

Block-level Mapping Blocks (BMB)

Page-level Mapping Blocks (PMB)
WAFTL: Workload Adaptive FTL

Two-level Address Translation Process

Global Mapping Table (GMT)

<table>
<thead>
<tr>
<th>LBN</th>
<th>PBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>-1</td>
</tr>
<tr>
<td>2010</td>
<td>3201</td>
</tr>
<tr>
<td>1031</td>
<td>1201</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Page Mapping Table (PMT)

<table>
<thead>
<tr>
<th>LPN</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1300</td>
</tr>
<tr>
<td>3200</td>
<td>4300</td>
</tr>
<tr>
<td>0010</td>
<td>2300</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

LPN: Logical Page Number
PPN: Physical Page Number
LBN: Logical Block Number
PBN: Physical Block Number

Block Offset: LPN mod Blk_Size
WAFTL: Workload Adaptive FTL

Layout and In-memory Data Structure

Request

SRAM
Buffer Zone Mapping Tree (BZMT)

Global Mapping Table (GMT)

Cached Page Mapping Table (CPMT)

LBN  PBN
1200  -1
3200  4300
4200  2300
...

LPN  PPN
4120  1301
1032  2300
1120  3101
...

Data access

Buffer Zone

BMB

PMB

Data

Mapping replacement

M_{PPN}=11

M_{PPN}=15

GMT

PMT

LBN  PBN
0010  -1
2010  3201
M_{VPN}=0

LBN  PBN
1110  0012
3210  1320
M_{VPN}=1
Buffer Zone Mapping Tree

Key: Block Number

Root node

Interior node

Leaf node

Block Node

Page count

Block Popularity

Block Dirty

Pointer Array

Page-level mapping

Buffer Zone

LPN | PPN | Dirty
---|---|---
1201 | 2311 | 0
1202 | 1578 | 1

WAFTL: Workload Adaptive FTL
WAFTL: Workload Adaptive FTL

Buffer Zone Migration

Block No: 0
- Popularity: 3
- Page count: 4

Block No: 2
- Popularity: 2
- Page count: 3

Block No: 4
- Popularity: 2
- Page count: 3

Most Popular Block

Page count \( \geq \) Threshold

Active BMB

Active PMB

Active Page

BMB

PMB
Determining Migration Threshold

- **Performance oriented threshold**
  - Threshold is statically set as full block size and only full block data is stored into BMB.
  - *BEST performance, but Big Mapping table*

- **Dynamic threshold considering memory constraint**
  - Initially, THR is set to Block size
  - THR decreases to a smaller value if performance degradation is smaller than D, and mapping table reduction is larger than D. D is a control parameter defined by users.

\[
\frac{\Delta P}{P_{\text{Blk\_size}}} = \frac{P_{\text{THR}} - P_{\text{Blk\_size}}}{P_{\text{Blk\_size}}} \leq D
\]
\[
\frac{\Delta M}{M_{\text{Blk\_size}}} = \frac{M_{\text{Blk\_size}} - M_{\text{THR}}}{M_{\text{Blk\_size}}} \geq D
\]

\(P_{\text{THR}}\) and \(P_{\text{Blk\_size}}\) represent performance when threshold is set as \(THR\) and full block, respectively. \(M_{\text{THR}}\) and \(M_{\text{Blk\_size}}\) represent mapping table size when threshold is set as \(THR\) and full block, respectively.

*Mapping table size is more sensitive to threshold than performance.*
Garbage Collection Policy

- Idle period \( t_{\text{predict}}^i \) is predicted as
  \[
  t_{\text{predict}}^i = \alpha t_{\text{real}}^{i-1} + (1 - \alpha) t_{\text{predict}}^{i-1}
  \]

- WAFTL use predicted idle length to calculate and determine how many invalid blocks \( N_i \) to be erased according to following Equation

\[
N_i = \begin{cases} 
  \frac{t_{\text{predict}}^i}{T_{\text{erasure}}} & \text{if } t_{\text{predict}}^i \geq T_{\text{erasure}} \\
  0 & \text{if } t_{\text{predict}}^i < T_{\text{erasure}}
\end{cases}
\]
Evaluation

Setup
- SSD Simulator
- FTL schemes: WAFTL, Pure Page-based, DFTL, FAST
- 7 enterprise workloads

Evaluation Metrics
- Average response time
- Erase count
- Page read/write operations

Configuration

<table>
<thead>
<tr>
<th>Workload</th>
<th>Avg. Req. Size(KB)</th>
<th>Write(%)</th>
<th>Seq.(%)</th>
<th>Avg. Req. Inter-arriv Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fin1</td>
<td>8.291986</td>
<td>35%</td>
<td>0.6%</td>
<td>8.189</td>
</tr>
<tr>
<td>Fin2</td>
<td>7.776166</td>
<td>17%</td>
<td>0.7%</td>
<td>11.081</td>
</tr>
<tr>
<td>Exchange</td>
<td>54.593399</td>
<td>74%</td>
<td>0.5%</td>
<td>1.3148</td>
</tr>
<tr>
<td>DevDiv91</td>
<td>36.168616</td>
<td>91%</td>
<td>0.9%</td>
<td>2.0435</td>
</tr>
<tr>
<td>DevDiv</td>
<td>88.983830</td>
<td>72%</td>
<td>1.5%</td>
<td>2.6882</td>
</tr>
<tr>
<td>Sequential Read</td>
<td>67.678195</td>
<td>19%</td>
<td>60%</td>
<td>49.857298</td>
</tr>
<tr>
<td>Sequential Write</td>
<td>67.552513</td>
<td>80%</td>
<td>70%</td>
<td>49.741739</td>
</tr>
</tbody>
</table>
**Evaluation**

**Fin1 Trace** (Static Migration threshold=64pages,256KB)

**Fin2 Trace** (Static Migration threshold=64pages,256KB)
Evaluation

**Exchange Trace** (Static Migration threshold=64pages,256KB)

**DevDiv Trace** (Static Migration threshold=64pages,256KB)
**Evaluation**

**DevDiv91 Trace** (Static Migration threshold=64pages,256KB)

**Seq_Read Trace** (Static Migration threshold=64pages,256KB)
**Evaluation**

**Workload adaptive**

WAFTL is adaptive to workloads (32GB SSD)

Mapping table sizes under different workloads (256MB buffer zone, 32GB SSD)
Evaluation

Effect of Threshold

- **Average Response Time (ms)**
- **Mapping Table Size (MB)**

Buffer Zone: 256MB
SSD: 32GB, 64 Pages/Block

Buffer Zone: 256MB
SSD: 32GB, 64 Pages/Block

60%
40%
WAFTL explores

- either page-level or block-level address mapping for data blocks based on access patterns in order to balance the mapping efficiency of a block-based approach and the GC efficiency of a page-based approach.

- a small part of NAND flash space for the buffer area to log data sequentially until it is full and then migrate them into either a page mapping block or block mapping block, depending on the migration threshold.

- different scheme to determine the value of migration threshold

- thereby, improving performance while reducing the mapping table size at the same time.
Thanks & Question