Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling

Sungjin Lee, Taejin Kim, Kyungho Kim†, and Jihong Kim

School of Computer Science and Engineering
Seoul National University

†Samsung Electronics

10th USENIX Conference on File and Storage Technologies
February 17, 2012
Flash-based SSDs in Enterprise

- Flash-based SSDs (Solid-State Drives) are becoming an attractive storage solution for enterprise systems.

- The limited lifetime caused by poor write endurance is a main barrier for wider adoption of SSDs in the enterprise market.
SSD Lifetime

- The SSD lifetime is determined by two main factors:

\[
\text{SSD lifetime (days)} = \frac{\text{The total number of bytes that can be written to the SSD}}{\text{The number of bytes written per day}}
\]

- (1) SSD capacity
- (2) Number of program/erase (P/E) cycles
- (3) Incoming write traffic
- (4) Write Amplification Factor (WAF)
  - Efficiency of FTL algorithms
Intensive Write Traffic

- Enterprise systems exhibit high write traffic

- Write not intensive (e.g., Mobile phone and Desktop PC)
  - Bandwidth (MB/sec)
  - Required lifetime

- Write intensive (e.g., Enterprise Server)
  - Bandwidth (MB/sec)

- Cannot guarantee the required lifetime

- Lifetime = Capacity \cdot \# \text{ of P/E cycles} \cdot \text{Write traffic (day)} \cdot \text{WAF}
Decreasing P/E Cycles

- The number of P/E cycles is continuously decreasing as the semiconductor process is scaled-down

(Source: JMicron, Western Digital, Morgan Stanley Research)

Lifetime = \frac{Capacity \times \text{Write traffic (day)}}{\# \text{ of P/E cycles} \times \text{WAF}}
Existing Lifetime-Enhancement Schemes

- Reduce WAF
  - Optimize garbage collection algorithms
  - Optimize wear-leveling algorithms
  - Use more fine-grained mapping schemes

- Reduce incoming write traffic
  - Use lossless data compression
  - Use data deduplication

All those approaches improve the overall SSD lifetime, but cannot guarantee the required SSD lifetime!
Static Throttling (Existing Approach)

- Limit the maximum throughput of SSDs

Disadvantages
- Likely to throttle performance uselessly
  - High performance penalty and high response time variations
- Underutilize the available endurance

Graph showing original write traffic and static throttling to guarantee the required lifetime.

Bandwidth (MB/sec) vs. Required Lifetime graph.
Our Approach (1):

Dynamic Throttling

- Throttle SSD performance dynamically depending on:
  - The characteristics of a given workload
  - The remaining SSD lifetime

- Less performance penalty and response time variations
- Fully utilize the available endurance
Our Approach (2): 
**Exploit Self-Recovery Effect**

- The effective P/E cycles are much larger than the number on datasheets due to the recovery effect.

- Guarantee the SSD lifetime with less throttling overheads.

---

### Bandwidth (MB/sec)

- **Dynamic Throttling**
- **Required Lifetime**
- **Recovery-Aware Dynamic Throttling**
- **An SSD is worn out**
Contribution

- Propose a novel REcovery-Aware DYnamic throttling technique, called READY
  - Throttle the SSD performance to guarantee the required SSD lifetime
  - Exploit the self-recovery property of a flash memory cell to lessen the performance penalty caused by throttling
- Evaluate the proposed READY technique using real-world enterprise traces
  - Guarantee the required SSD lifetime for all evaluated traces
  - Achieve 4.4x higher responses time over a simple static throttling technique
Outline

- Introduction
- Motivation
- Recovery-Aware Dynamic Throttling
- Evaluation Results
- Conclusion
**Design Goals of READY**

- **Design goal 1:** guarantee the required SSD lifetime
  - Throttle the write throughput of SSDs by applying throttling delays to write requests

- **Design goal 2:** minimize average response times
  - Determine a throttling delay as low as possible so that the SSD is completely worn out at the required lifetime

- **Design goal 3:** minimize response time variations
  - Distribute a throttling delay as evenly as possible over every write request
Overall Architecture of READY

- **Host System**: Write
- **Host Interface**: Throttling Layer
- **Throttling Layer**: FTL, NAND Flash Chips
- **SSD**: Write w/ Delay

**Predict future write demands**

**Determine throttling delays**

**Throttle write performance**

**Epoch-Capacity Regulator**: Apply throttling delays

**Write Demand Predictor**: Monitor write demands

**Throttling Delay Estimator**: Estimate how many data will be written
Write Demand Predictor

- The write traffic of enterprise workloads is likely to change significantly over time.
- How to predict future write traffic for throttling?
  - Exploit cyclic behaviors of enterprise applications!
A strong cyclical behavior is frequently observed in enterprise applications.

When a cyclic period is set to 30 min, the write demand difference is less than 30% for 88% periods.

When a cyclic period is set to 30 min, the write demand difference is less than 20% for 98% periods.
Future Write Demand Estimation

- (1) Divide time into epochs which exhibit similar write demands
- (2) Estimate the similar amount of data written during the latest epoch will be written during the next epoch

The similar amount of data will be written during the i-th epoch

Bandwidth (MB/sec) vs. Time

Cyclical Period (= Epoch)

(i-1)th epoch

i-th epoch

(i+1)th epoch
Overall Architecture of READY

- **Host System**
  - Host Interface
  - Throttling Layer
    - FTL
    - NAND Flash Chips
  - SSD

- **Write Demand Predictor**
  - Monitoring past write demands
  - Predict future write demands

- **Epoch-Capacity Regulator**
  - Apply throttling delays

- **Throttling Delay Estimator**
  - Determine throttling delays
  - Predict future write demands
  - Throttling Delay

- **Recovery Model**
  - Throttling Delay
Throttling Delay Estimator

- Determine a throttling delay
  - (1) The future write demand for the next epoch
    - We already know it
  - (2) The epoch capacity
    - The amount of data allowed to be written during the epoch

\[
\text{Epoch capacity} = \frac{\text{# of remaining P/E cycles} \times \text{SSD capacity}}{\text{# of remaining epochs}}
\]
Change Throttling Delay

- A throttling delay is initially set to 0 and is changed adaptively at the beginning of each epoch.

- Case 1: future write demand = epoch capacity
  - Don’t change a throttling delay

- Case 2: future write demand > epoch capacity
  - Increase a throttling delay

- Case 3: future write demand < epoch capacity
  - Decrease a throttling delay
Exploit Effective P/E Cycles

- P/E operations cause damage to NAND flash memory cells
- This damage is partially recovered during the idle time
- Effective P/E cycles are larger than pre-set P/E cycles

\[
\text{Epoch capacity} = \frac{\# \text{ of remaining P/E cycles} \times \text{SSD capacity}}{\# \text{ of remaining epochs}} < \frac{\# \text{ of effective remaining P/E cycles} \times \text{SSD capacity}}{\# \text{ of remaining epochs}}
\]
Effective P/E Cycles Modeling

- Self-recovery effect validation from real measurements
- Effective P/E cycles modeling

<Effective P/E cycles>

<Self-Recovery Model>

<Damage Model>

FAST '12

S. LEE
The Effective P/E Cycles

- The maximum P/E cycles without the recovery effect are 3K.
- The effective P/E cycles are gradually increased in proportional to the length of the idle time.
Overall Architecture of READY

**Host System**
- Host Interface
- Throttling Layer
- FTL
- NAND Flash Chips

SSD

**FAST '12**

**Throttling Delay Estimator**
- Predict future write demands
- Apply throttling delays

**Write Demand Predictor**
- Monitoring past write demands

**Epoch-Capacity Regulator**
- Throttle write performance
- Apply throttling delays

**Time**
- Past
- Present
- Future
Epoch-Capacity Regulator

- Throttle write performance as evenly as possible
  - To minimize response time variations

- (1) Apply the same throttling delay to every page write
- (2) Increase a throttling delay later to reclaim the over-used capacity

A page write (page size is 8 KB) 32 KB data has been written

Epoch Capacity = 24 KB

Increase a throttling delay slightly to reclaim the over-used capacity
Outline

- Introduction
- Motivation
- Recovery-Aware Dynamic Throttling
- Evaluation Results
- Conclusion
Experimental Setting

- Used the DiskSim-based SSD simulator for evaluations
  - 20 nm 2-bit MLC NAND flash memory with 3K P/E cycles
  - The target SSD lifetime is set to 5 years

- Evaluated four SSD configurations
  - **NT**: No Throttling
    - No performance throttling; No lifetime guarantee
  - **ST**: Static Throttling
  - **DT**: Dynamic Throttling without Recovery
  - **READY**: Recover-Aware Dynamic Dynamic Throttling
### Benchmarks

- Used the traces from MSR-Cambridge and MS-Production benchmarks

<table>
<thead>
<tr>
<th>Trace</th>
<th>Duration</th>
<th>Data written per hour (GB)</th>
<th>WAF</th>
<th>SSD capacity (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>1 week</td>
<td>4.94</td>
<td>1.93</td>
<td>32</td>
</tr>
<tr>
<td>proj</td>
<td>1 week</td>
<td>2.08</td>
<td>1.62</td>
<td>32</td>
</tr>
<tr>
<td>exchange</td>
<td>1 day</td>
<td>20.61</td>
<td>2.24</td>
<td>128</td>
</tr>
<tr>
<td>map</td>
<td>1 day</td>
<td>23.82</td>
<td>1.68</td>
<td>128</td>
</tr>
<tr>
<td>msnfs</td>
<td>6 hours</td>
<td>18.19</td>
<td>2.26</td>
<td>128</td>
</tr>
</tbody>
</table>
Lifetime Analysis

- NT cannot guarantee the required SSD lifetime (except for proj)
- READY achieves the lifetime close to 5 years
- ST and DT exhibit the lifetime much longer than 5 years
Performance Analysis

- NT exhibits the best performance among all the configurations
- READY perform better than ST and DT while guaranteeing the required lifetime
READY shows shorter response times than ST/DT.
ST exhibits significant response time variations.
  - Stop writing if incoming write traffic is higher than a fixed throughput
The write traffic of proj and map changes greatly with time.
- It is hard to predict future write traffic.
- READY and DT exhibit relatively high fluctuation on response times, but is more stable than ST
Conclusion

- We proposed the recovery-aware dynamic throttling technique, called READY
  - Guarantee the SSD lifetime by throttling SSD performance
  - Reduce throttling overheads by exploiting the self-recovery effect of flash memory cells
  - Achieve about 4.4x higher performance over the existing static throttling with less response time variations

- Future works
  - Implement READY in a real SSD platform
  - Support latency-aware performance throttling
Thank you