Write-Aware Buffer Management Policy for Performance and Durability Enhancement in NAND Flash Memory

Xin Jin, Student Member, IEEE, Sanghyuk Jung, Student Member, IEEE and Yong Ho Song, Member, IEEE
the Department of Electronics and Computer Engineering, Hanyang University, Seoul, Korea
Outline

• Introduction
• Related Work
• Proposed Scheme
• Evaluation
• Conclusion
Introduction (1/3)

• attractive features of NAND flash memory
  – light weight
  – low power consumption
  – small size
  – shock resistance

• drawbacks of NAND flash:
  – erase-before-write
  – asymmetric read/write operation time

<table>
<thead>
<tr>
<th></th>
<th>SLC</th>
<th>MLC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Page size</strong></td>
<td>2K Bytes</td>
<td>4K Bytes</td>
</tr>
<tr>
<td><strong>Block size</strong></td>
<td>64 pages</td>
<td>128 pages</td>
</tr>
<tr>
<td><strong>Page read (μs)</strong></td>
<td>25</td>
<td>60</td>
</tr>
<tr>
<td><strong>Page write (μs)</strong></td>
<td>200</td>
<td>800</td>
</tr>
<tr>
<td><strong>Block erase (ms)</strong></td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td><strong>Block endurance</strong></td>
<td>100K Cycles</td>
<td>10K Cycles</td>
</tr>
</tbody>
</table>
Currently, the mainstream SSD products are usually equipped with a 64MB or larger **DRAM** buffer.

- because DRAM is fast and cheap

- absorbing and clustering the write operations to reduce the total number of write commands issued to the NAND flash memory
• write-intensive page
  – some pages are written over and over again
  – should not be evicted early

• Write Intensive Page Preserving Algorithm (WIPPA)
  – to keep write-intensive pages in the DRAM buffer to the fullest extend
  – simulation results shows that up to 30% of the write count can be reduced.
Related Work (1/4)

• **CFLRU (Clean First Least Recently Used)**
  - evicts all of the clean pages within a window in LRU fashion
  - **windows size** is determined carefully so that the hit ratio of main memory can be kept in a bearable level.
  - postpones the eviction of dirty pages

-- considers only recency but not respects the frequency
Related Work (2/4)

- FARS (Flash Aware Replacement Strategy)
  - distinguishes from the non-intensive access pages by using **four states** for a page
  - in the eviction process, the **access intensive pages** are granted **higher priority** for staying in the buffer

![State transition diagram of FARS]

Fig. 2. State transition diagram of FARS

WI, RI: Write Intensive, Read Intensive
Related Work (3/4)

- **FARS (Cont.)**
  - uses a **ghost list** to store the metadata of the recently evicted pages
    - can keep track of more pages without sacrificing much buffer space
    - make the detection of access intensive pages more accurate

  - when the page is found in the **ghost list**, fetch into the **L1 list**
    - when **L1 list** is full: **LRU**
    - when the buffer is full
      - **victim page** is selected in **L2**
      - **eviction priority** is set according to the page status:
        - \( R \rightarrow RI \rightarrow W \rightarrow WI \)
      - metadata of victim page is stored in ghost list
    - when the **ghost list** is full: **LRU**

![Diagram of L1, L2, and ghost lists in FARS](image)
Related Work (4/4)

• FARS (Cont.)
  – a write-intensive page could become a read page after two successive read operations, which makes it likely to be evicted
  – for example:

    • **eviction priority:** R → RI → W → WI

    ![Flowchart](image)

    **Fig. 4. Imaginary workload trace**

    **Fig. 2. State transition diagram of FARS**

    • state of **Sector 1** changes from **write, write-intensive, write** to **read**
Proposed Scheme (1/4)

- **WIPPA** (Write Intensive Page Preserving Algorithm)
  - Add a single bit **Intensive Flag (IF)** to the metadata of a page during its residence in the buffer
  - prevent access intensive pages from becoming read pages

![Diagram of WIPPA](image)

- Fig. 4. Imaginary workload trace
  - state of Sector 1 changes from write, write-intensive, write to read-intensive

- Fig. 5. State transition diagram of WIPPA

WI, RI: Write Intensive, Read Intensive
Proposed Scheme (2/4)

- **Eviction Policy of WIPPA**
  - when a new page is brought into the buffer, it enters L2 and its IF flag is set to ‘0’
  - L1 list stores WI, RI, and the pages fetched from the ghost list
  - L2 stores R and W pages,
  - the summation of the length of the two lists equals the total amount of buffer space, while each individual length is variable
    - L2 = 0, if the buffer is full of access intensive pages
  - the victim page could be in either list L1 or L2

Fig. 6. Page immigration between lists in WIPPA
Proposed Scheme (3/4)

- Eviction Policy (Cont.)
  - Once the buffer is full, the victim page selection process starts.
  - the selection order
    - R in L2 → RI in L1 → W in L2 → LRU of L1
    - make the best use of buffer space to store write-intensive pages

![Diagram](image-url)

**Algorithm 1. WIPPA Eviction**

```plaintext
PagePointer* Eviction()
{
    if (read page found in L2)
        return the page;
    else if (read-intensive page found in L1)
        return the page;
    else if (write page found in L2)
        return the page;
    else
        return the tail page of L1;
}
```

**Fig. 6. Page immigration between lists in WIPPA**
Overhead Discussion

- WIPPA searches both L1 and L2 for a victim page
  - time consuming
  - use a hash-based technique to reduce the computational overhead
  - for example:
    - 100,000 sectors and 1,000 hash entries
    - each entry is responsible for 100 pages
    - can quickly return a hit or a NULL

Fig. 7. Indexing pages in buffer
Evaluation (1/4)

• Simulation Environment
  – two workload traces
  • Application Loading
  • Local Disk Search

<table>
<thead>
<tr>
<th></th>
<th>Application Loading</th>
<th>Local Disk Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total requests</td>
<td>1,040,719</td>
<td>1,819,822</td>
</tr>
<tr>
<td>Write requests</td>
<td>563,908 (54.2%)</td>
<td>1,033,970 (56.8%)</td>
</tr>
<tr>
<td>Read requests</td>
<td>476,811 (45.8%)</td>
<td>785,852 (43.2%)</td>
</tr>
<tr>
<td>Sequential access (%)</td>
<td>44.8%</td>
<td>8.85%</td>
</tr>
</tbody>
</table>

Fig. 8. Trace characteristics of Application Loading

Fig. 9. Trace characteristics of Local Disk Search
Evaluation (2/4)

- Simulation Environment (Cont.)
  
  - the **window size** of CFLRU is set to **0.5**

  - the **L1** list in FARS is set to **half** of the total length of the buffer space

<table>
<thead>
<tr>
<th>TABLE IV</th>
<th>SIMULATED NAND CHIP SPECIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Value</td>
</tr>
<tr>
<td>Number of blocks per chip</td>
<td>8192 (1GBytes)</td>
</tr>
<tr>
<td>Number of pages per block</td>
<td>64</td>
</tr>
<tr>
<td>Page size</td>
<td>2112 Bytes</td>
</tr>
<tr>
<td>Number of sectors per page</td>
<td>4</td>
</tr>
</tbody>
</table>
• Evaluation Result

- WIPPA retained more write-intensive pages at the cost of issuing more read requests to FTL, but it is well worth it.
Evaluation (4/4)

• Impact on FTL

- WIPPA incurred significantly fewer write, erase, and copy operations than the other two schemes.
- Application Loading trace has far more sequential accesses. FARS does not handle sequential accesses well.
- Considering the erase cycle restriction of NAND flash, the reduction in the erase count extends its lifetime and durability.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Read</th>
<th>Write</th>
<th>Erase</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFLRU</td>
<td>763,876,528</td>
<td>3,793,244</td>
<td>47,321</td>
<td>80,803</td>
</tr>
<tr>
<td>FARS</td>
<td>764,107,284</td>
<td>3,960,712</td>
<td>51,226</td>
<td>102,102</td>
</tr>
<tr>
<td>WIPPA</td>
<td>763,881,631</td>
<td>2,569,299</td>
<td>29,075</td>
<td>56,793</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Read</th>
<th>Write</th>
<th>Erase</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFLRU</td>
<td>166,305,497</td>
<td>10,413,965</td>
<td>353,611</td>
<td>5,778,182</td>
</tr>
<tr>
<td>FARS</td>
<td>167,852,548</td>
<td>9,086,582</td>
<td>314,615</td>
<td>5,292,840</td>
</tr>
<tr>
<td>WIPPA</td>
<td>167,880,741</td>
<td>8,995,072</td>
<td>308,878</td>
<td>5,141,419</td>
</tr>
</tbody>
</table>

Fig. 14. Normalized NAND flash operations

Fig. 15. Normalized NAND flash operations
Conclusion

• WIPPA focuses on retaining write-intensive pages to the fullest extent, and the results are better than FARS and CFLRU

• not much extra time overhead accompanies improvements in the performance and durability

• future work
  – make better use of the pages fetched from the ghost list
  – further investigate the page status transition diagram to retain more write-intensive pages