Leveraging Progressive Programmability of SLC Flash Pages to Realize Zero-overhead Delta Compression for Metadata Storage

Xuebin Zhang, Jiangpeng Li, Kai Zhao, Hao Wang and Tong Zhang
xuebinzhang.rpi@gmail.com
ECSE Department, Rensselaer Polytechnic Institute (RPI)
Outline

- Introduction and Background
- Progressive Programmability of SLC Flash Pages
- Metadata Update Solution using Delta Compression
- Evaluation Results and Practical Considerations
- Summary and Future Work
Increasing Adoptions and Decreasing Cost of NAND Flash Memory

Endurance: One of Existing Problems of NAND Flash Memory

- Limited lifetime
- Make it worse: Write amplification.
Why Metadata Storage

A simple example\textsuperscript{[1]} in ext3:

- Write “hello” into text1.txt

  Effective data: \textbf{6 bytes} vs. Flash write: 11 pages*4kB=\textbf{44kB}

- Read “happy” from text2.txt

  Flash write: 3pages*4kB=\textbf{12 kB}

Metadata Update Characteristics in NAND Flash

- Small (< 4KB) and Page-aligned
  - Around 80% write are smaller than 4KB, metadata are even smaller
  
  ![Assign a new page diagram](image)

- Frequent update within a short time period (around 1s)

- Abundant redundancy between consecutive versions of metadata.
  - Metadata is highly structured.
  - Difference would always occur in the same locations.

Delta Compression
Delta Compression: Current Practice

- Need to read multiple pages to recover the latest version.
- Need to keep a mapping table to store the related delta’s location.
NAND Flash Progressive Programming

- Append new data in the same page (for SLC flash pages)
  1. Copy the original part
  2. Append new data
  3. Complement to a full page

<table>
<thead>
<tr>
<th>Physical page</th>
<th>Page content</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>[ \begin{array}{</td>
</tr>
<tr>
<td>$P_0$</td>
<td>[ \begin{array}{</td>
</tr>
<tr>
<td></td>
<td>[ d_0 , d_1 \cdots d_n , 111\cdots11 ]</td>
</tr>
<tr>
<td></td>
<td>[ d_0 , d_1 \cdots d_n , l_l , l_2 \cdots l_m , 1 \cdots 11 ]</td>
</tr>
<tr>
<td></td>
<td>[ d_0 , d_1 \cdots d_n , l_l , l_2 \cdots l_m , 1 \cdots 11 ]</td>
</tr>
</tbody>
</table>
NAND Flash Progressive Programming

- Hardware Platform

![Image of hardware platform with labeled components: Mezzanine Board, NAND Flash Chips, FPGA as SSD Controller, PCIe Connector]
NAND Flash Progressive Programming

- Validity checking of multiple programming before erasing
  - Conventional: one program – one erase
  - Progressive: eight program – one erase

![Graph showing Bit error rate vs Erase count for Conventional and Progressive methods.]
Use SLC to Realize Zero-overhead Metadata Storage

- Procedures to update new version data

1. Input: New version data $D_1$
2. Read original page content $D_0$
3. XOR to generate difference $D_0 \oplus D_1 = L_1$
4. Compress $L_1$
5. Progressive? Y/N
   - Y: Generate $D$ to be programmed
     - After appending
   - N: Write $D_1$ on a new page
6. Programming $D$
7. Page content updated to $D_1$
Use SLC to Realize Zero-overhead Metadata Storage

- Procedures to update new version data

<table>
<thead>
<tr>
<th>Physical page</th>
<th>Initial version $D_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$ $p_0$ $p_1$</td>
<td>Version $D_1$</td>
</tr>
<tr>
<td>$P_0$ $D_0$</td>
<td>Version $D_2$</td>
</tr>
<tr>
<td>$P_0$ $D_0$ $L_1$ $L_2$</td>
<td>Full: Version $D_i$</td>
</tr>
<tr>
<td>$P_0$ $D_0$ $L_1$ $L_i$ $...$</td>
<td>NewPage: Version $D_{i+1}$</td>
</tr>
<tr>
<td>$P_1$ $D_{i+1}$</td>
<td>Allocate a new page</td>
</tr>
</tbody>
</table>
Implement the Delta Compression

- Bit level Run-length compression
- Difference-Index compression

![Diagram showing old data, new data, and compressed data with index sections for different segments.](image-url)
Evaluation Results

- Experiment Setup
  - Use benchmark to generate database/files operations
  - Implement a metaAnalyzer to grasp metadata from file system
  - Analyze the collected consecutive versions of metadata

- Compression Efficiency

![Compression Efficiency Chart]

- Run-length coding
- Diff-index compression
- SQlite Insert
- SQlite Update
- File Append
- File Update
Evaluation Results

- Write Footprint Reduction of Metadata

To store 1000 consecutive versions:

- Required page count

Bar chart showing:
- SQLite Insert
- SQLite Update
- File Append
- File Update
Practical Considerations

- **Read latency brought by decompression**
  - Need multiple decompression to retrieve the latest version.
  - Decompression latency is much smaller (4us) than read latency (41us).
  - Decompression could be done by an ASIC module in controller.

- **Changes on ECC management**
  - Length of compressed delta should be written in the page.

- **SLC mode space overhead compared to MLC mode**
  - When multiple deltas are stored in the page, space overhead will be compensated.
Conclusion

- SLC Flash page can support “Progressive Programming”: different portions of the same flash page can be programmed at different time.

- Proposed solution can save data traffic written into disk.

- Proposed solution can guarantee we do not need to read extra pages to retrieve the latest version.