Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices

Yanlei Diao

University of Massachusetts, Amherst

Collaborators: Devesh Agrawal, Deepak Ganesan, Ramesh Sitaraman and Shashi Singh
Flash Based Databases

“Disk is Tape, Flash is Disk”
Jim Gray, 2006

Advantages Over Disk
- Fast Random Reads
- Energy Efficient
- Small Size
- Robust...

Indexing Over Flash
- Crucial for Efficient Retrieval
- Challenging on Flash

64GB SSD
1GB NAND chip for Mote
Flash Characteristics

Raw NAND chips

NAND flash

- Erase before rewrite
- Can’t delete individual pages
**Flash Characteristics**

**Memory**
- Modify *single* page in-memory
- Load *entire* block into memory
- Write *whole* block back

**Flash**
- Erase block on flash

**Existing Solutions**
- *In-Place Updates*
- *Out-of-Place Updates (FTL)*

Raw NAND chips
Flash Characteristics

Existing Solutions
- **In-Place Updates**
- **Out-of-Place Updates (FTL)**

Page updates expensive
Goal: Design a flash friendly index

- Minimize total number of I/Os
- Minimize page updates for efficient index updates
- Perform efficient index lookups at the same time
Lazy Adaptive Tree (LA-Tree)

B+ Tree augmented with flash resident buffers to hold updates

K Levels

Lookup Key

Flash resident buffer to hold updates
Idea 1: Lazy Updates

Buffer Empties push down updates in a batch
Lazy Updates: Non Leaf Buffer Empty

Advantages
- Amortizes node reads
- Buffers cheap to maintain
Lazy Updates: Leaf Buffer Empty

1. Distribute Down

10, 11, 12, 14, ...
11, 14, 14, 18, 23, 25
14, 25, ...
Sorted Buffer
2. Standard Bottom up node updates

18, 22, 23, 25, ...

Advantages

- Amortizes node writes
Problem with Lazy Updates

- Critical to adapt buffer size to workload
  - Lookup Dominated $\rightarrow$ Small Buffer
  - Update Dominated $\rightarrow$ Large Buffer
  - Different subtrees can see different workloads
Idea 2: Buffer Size Control Algorithm

An *Online Algorithm* (ADAPT)

- Observes current lookup-insert workload on each buffer
- Adapts each buffer independently
ADAPT : Intuition

Buffer Size vs. Time

Lookups: L1, L2, L3, L4, L5
Scan Cost: 15, 50, 80, 110, 150
Empty Cost: 70, 140, 200, 260, 340
ADAPT: Intuition

Emptying at L2 Gain:
- Cost Spent: 140
- Future Savings: 150
- Profit: 10

Empty Cost:
- L1: 70
- L2: 140
- L3: 200
- L4: 260
- L5: 340

Profit of emptying:
- L1: -10
- L2: 10
- L3: -40
- L4: -150
- L5: -340
Can’t predict future savings

Reasoning in hindsight:
Empty the buffer at L5 after seeing L3-L5 savings

Buffer Size

Time

Lookup: L1

L2

L3

L4

L5

Savings in Hindsight: 150
Empty Cost at L2: 140
Profit in Hindsight: 10

Optimal Online Algorithm
- Proved 2-competitive
- Best possible competitive ratio
Implementation Challenges

- Reduce Buffer Fragmentation
- Reduce Garbage Collection overhead

Fragmentation increases buffer read cost

Buffer maintained as linked list on flash

Flash
Implementation Challenges

- Reduce Buffer Fragmentation
- Reduce Garbage Collection overhead

Flash increases GC costs
LA-Tree System Architecture

- **Memory Manager**
  - Write Coalescing Buffer Pool
  - LRU node cache
  - Reduces Buffer Fragmentation

- **Flash Manager**
  - Flash written as a log
  - Empty buffers before GC
  - Reduces GC overhead
Evaluation Setup

- Flash devices
  - Toshiba TC58DVG02A1FT00 1Gb NAND flash
  - MTRON PRO 7000 16GB SATA SSD

- Data sets
  - Synthetic: Uniformly random key distributions
  - Scientific DB: meteorological radar data traces
  - Transactional: TPC-C Index Trace
MicroBenchmarks: ADAPT evaluation

- Raw NAND flash
- 128KB RAM
- Synthetic workload with 1M updates (1/3 deletes)
- LTU: Lookup to Update Ratio
- Response time/operation (us)

- Fixed buffer sizes do not scale with workload
- ADAPT matches any workload
MicroBenchmarks: versus B+ Tree

LA-Tree outperforms B+ tree across all workloads

( B+ tree implemented as LA-Tree with buffering turned off )
### Versus Existing Schemes (NAND)

Response Time Per Operation in micro-seconds

<table>
<thead>
<tr>
<th></th>
<th>Synthetic</th>
<th>Sci-DB</th>
<th>TPC-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTU</td>
<td>10%</td>
<td>200%</td>
<td>0.01%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8% (Customer)</td>
</tr>
<tr>
<td>FlashDB</td>
<td>1270</td>
<td>844</td>
<td>1385</td>
</tr>
<tr>
<td></td>
<td>550</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>BFTL</td>
<td>1322</td>
<td>2073</td>
<td>1478</td>
</tr>
<tr>
<td></td>
<td>862</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>IPL</td>
<td>2489</td>
<td>1702</td>
<td>2205</td>
</tr>
<tr>
<td></td>
<td>999</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>LA-tree</td>
<td>113</td>
<td>254</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Gain</td>
<td><strong>11.2 x</strong></td>
<td><strong>3.3 x</strong></td>
<td><strong>8.5 x</strong></td>
</tr>
<tr>
<td></td>
<td><strong>4.6 x</strong></td>
<td><strong>2.2 x</strong></td>
<td></td>
</tr>
</tbody>
</table>

- High gains (over the best) across spectrum of workloads
- Gains are higher for update-heavy workloads

[FlashDB: IPSN 07], [IPL: SIGMOD 07], [BFTL: TECS 07]
# Versus Existing Schemes (SSD)

Response Time Per Operation in micro-seconds

<table>
<thead>
<tr>
<th></th>
<th>Synthetic</th>
<th>Sci-DB</th>
<th>TPC-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTU</td>
<td>10%</td>
<td>200%</td>
<td>0.01%</td>
</tr>
<tr>
<td>FlashDB</td>
<td>1487</td>
<td>1083</td>
<td>642</td>
</tr>
<tr>
<td>BFTL</td>
<td>1463</td>
<td>1192</td>
<td>1175</td>
</tr>
<tr>
<td>IPL</td>
<td>2791</td>
<td>2289</td>
<td>6705</td>
</tr>
<tr>
<td>LA-tree</td>
<td>315</td>
<td>351</td>
<td>12</td>
</tr>
<tr>
<td>Gain</td>
<td>4.6 x</td>
<td>3 x</td>
<td>52 x</td>
</tr>
</tbody>
</table>

LA-Tree significantly reduces random writes

[FlashDB: IPSN 07], [IPL: SIGMOD 07], [BFTL: TECS 07]
Summary and Ongoing Work

LA-Tree is a new flash optimized index

- **Idea 1:** Lazy Updates
- **Idea 2:** Adaptive buffer size control
- Efficient implementation for flash and memory constraints
- Large gains over many workloads and flash types

Ongoing Work

- Extend to a broad family of Indexes
- Extend to other storage devices (SSD, HDD)
- Use SSDs in MapReduce-based large scale data analysis
Thank You

Questions ?