Flash Memory & Data Structures: What they don't want you to know

SeungBum Jo
PhD candidate
Seoul National University, Republic of Korea

Vineet Pandey
Undergraduate
BITS Pilani, India

Srinivasa Rao Satti
Assistant Professor
Seoul National University, Republic of Korea

Flash 101

aka Flash memory is the future.

1. **Advantages:** Increased Throughput, Increased Robustness, Reduced Latency
2. **Characteristics:** Erase before write, Out-of-place update, Weariness, Garbage collection
3. **Used as:** Raw flash and SSDs in Cache, primary tier
4. **Problems:** Slow random writes, Expensive

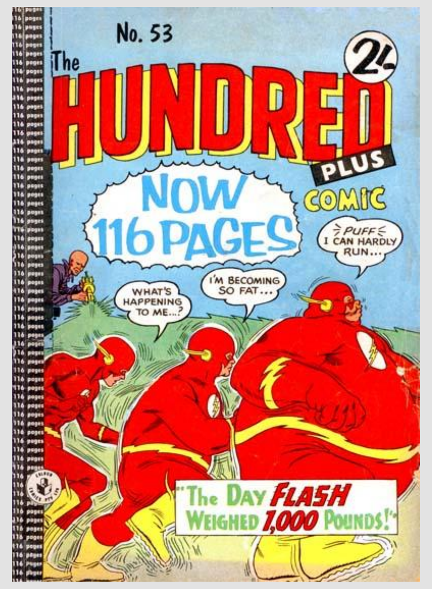


Figure 1. Flash getting fat [AusReprints.com]

Indexing problem on flash memory

Standard B+-tree sucks: Write-amplification, faster usage of flash disk.

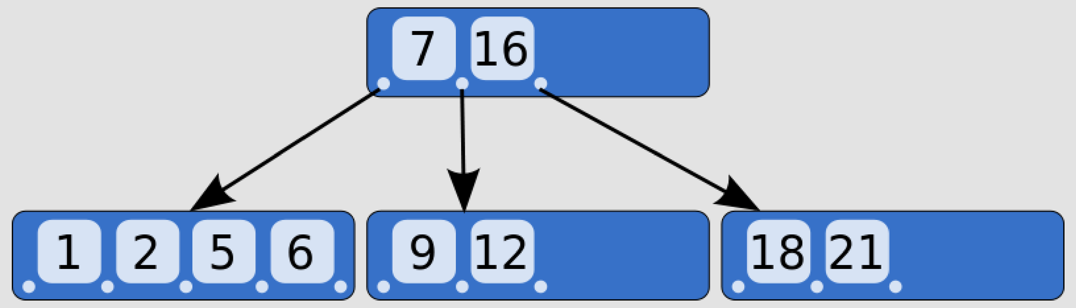


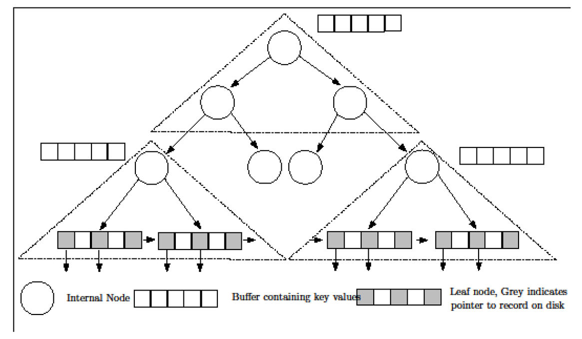
Figure 2. Sample B-tree [wikimedia.org]

How to construct and evaluate new data structures to choose the best?

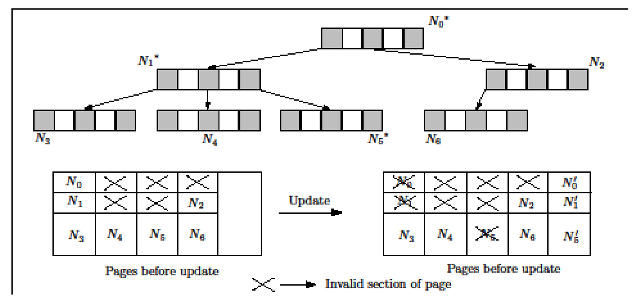
Cost analysis

We theoretically evaluate the performance for insert, delete and modify operations.

Use general-cost flash memory model [2].



LA-tree [1]: Periodic buffering of I/O operations in the tree



μ-tree [4]: Variable-sized nodes

FD-tree [5]: Converts random writes to sequential writes

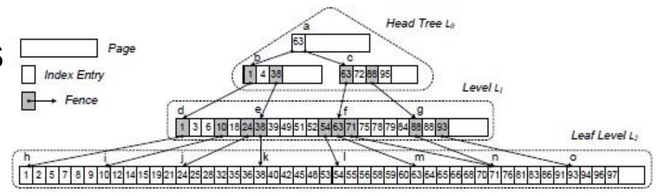


Figure 3. Proposed indexing structures

Further trends:
1. Block-aware layout
2. Self-tuning structure of the node
3. *Intelligent* FTL

Comparison results

Table 1: Complexity of operations in various data structures in the general-cost model. (h = height of the tree)

Data Structure	Search cost (read I/Os)	Update cost (write I/Os)	Reference
B^+-tree (FTL)	$O(\log_{B\_R} N)$	$O(1)$	[3]
LA-tree	$O((1 + \frac{B_U}{k\,B_R}) \log_{B\_R} N)$	$O(\frac{\log_{B\_R} N}{k})$	[1]
FD-tree	$O(\log_{B\_R} N)$	$O(\frac{B_R}{B_W} \log_{B\_R} N)$	[5]
μ-tree	$O(h)$	$O(1)$	[4]

N = Number of elements in tree
B_R = Read block size
B_W = Read block size
B_U = Size of buffer

Conclusions

1. FTL is useful but causes space and time overhead
2. Data structures like FD-tree can reduce write-amplification even without FTL, but are complex to implement
3. Simple workload-aware data structures (like FlashDB) can be very useful

Recommendations

- Need for developing more flash-aware algorithms
- Standardize performance claims over common workloads

Literature cited

[1] Devesh Agrawal, Deepak Ganesan, Ramesh Sitaraman, Yanlei Diao, and Shashi Singh. Lazy-adaptive tree: an optimized index structure for flash devices. Proc. VLDB Endow., 2:361-372, 2009

[2] Deepak Ajwani, Andreas Beckmann, Riko Jacob, Ulrich Meyer, and Gabriel Moruz. On computational models for ash memory devices. In SEA, Volume 5526 of Lecture Notes in Computer Science, pages 1627. Springer, 2009.

[3] Douglas Comer. Ubiquitous b-tree. ACM Computing Surveys, 11:121 -137, 1979.

[4] Dongwon Kang, Dawoon Jung, Jeong-Uk Kang, and Jin-Soo Kim. µ-tree: an ordered index structure for nand ash memory. In Proceedings of the 7th ACM & IEEE international conference on Embedded software, EMSOFT '07, pages 144-153. ACM, 2007.

[5] Yinan Li, Bingsheng He, Robin Jun Yang, Qiong Luo, and Ke Yi. Tree indexing on solid state drives. Proc. VLDB Endow., 3:1195-1206, 2010.

Presentation template: http://colinpurrington.com

Acknowledgments

- BITS Pilani for allowing undergraduate thesis at Seoul National University
- **NetApp Advanced Technology Group** for sponsoring the registration for event **(Woohoo!)**

For further information

vineet.pandey@netapp.com or ask for my card from Vineet.

Online pdf:
http://www.hipc.org/hipc2011/studsym-papers/1569513637.pdf