Flash Device Support for Database Management

Oracle – January 12, 2010

Philippe Bonnet, ITU
Copenhagen, Denmark

Luc Bouganim, INRIA,
Paris – Rocquencourt, France

This work is partially supported by the Danish Strategic Research Council.
Outline

• Motivation
• Flash device behavior
• The Good, the Bad and the FTL
• Minimal FTL
• Bimodal FTL
• Example: Hash join on Bimodal FTL
• Conclusion
DBMS on (or using) flash devices

• NAND flash performance is impressive
  ▪ Flash devices is part of the memory hierarchy
  ▪ Replace or complement hard disks

• DBMS design = 3 decades of optimization based on the (initial) hard disk behavior

• Revisit the DBMS design wrt. flash device behavior?

Need to understand the behavior of flash devices
Some examples of behavior (Samsung)

SR, SW and RR have similar (good) performance

RW, not shown, are much more expensive, 10-30ms
Some examples of behavior (Samsung)

Average performance can vary of an order of magnitude depending on the device state
Some examples of behavior (Intel X25-E)

SR, SW and RW have similar performance.
RR are more costly!

RW (16 KB) performance varies from 100 µs to 100 ms!! (x 1000)
Some examples of behavior (Fusion IO)

• Capacity vs Performance **tradeoff**
• Sensitivity to device state

**Graph:**
- **x-axis:** 
  - Low level formatted
  - Fully written
- **y-axis:** Response time (µs)
- **Legend:**
  - SR
  - RR
  - SW
  - RW
- **IO Size:** 4KB
Flash device behavior (1)

• Understanding flash behavior [uFLIP, CIDR 2009]
  ▪ Flash devices (e.g., SSDs) do not behave as flash chips
  ▪ Flash devices performance is difficult to measure (device state)
    – Need for an adequate methodology
  ▪ We proposed a wide benchmark to cover current and future devices.
  ▪ We also observed a common behavior and deduced design hints
    – Not true anymore on recent devices!

• Making assumptions about flash behavior
  ▪ Consider the behavior of flash chips (embedded context)
  ▪ Consider the behavior of a given device or of a class of devices
Flash device behavior (2)

• What is actually the behavior of flash devices?
  □ Update in place are inefficient?
  □ Random writes are slower than sequential ones?
  □ Better not filling the whole device if we want good performance?

➡️ Behavior varies across devices and firmware updates

Should we continue running after the flash technology?

In this talk, we propose another way to include flash devices in the DBMS landscape
Flash devices performance is impressive!

- A **single** flash chip offers **great performance**
  - e.g., 40 MB/s Read, 10 MB/s Write
  - Random access is **as fast as** sequential access
  - Low energy consumption

- A flash device contains **many** (e.g., 16, 32) flash chips and provides inter-chips parallelism

- Flash devices include some (power-failure resistant) **cache**
  - e.g., 16-32 MB of RAM
Flash chips have severe constraints!

- **C1**: Write granularity:
  - Writes must be performed at flash page granularity (e.g. 4 KB)
- **C2**: Must erase a block (e.g., 64 pages) before rewriting a page
- **C3**: Writes must be sequential within a flash block
- **C4**: Limited lifetime (from $10^4$ up to $10^6$ erase operations)
The Flash Translation Layer (FTL) emulates a classical block device, handling flash constraints

- Distribute erase across flash (wear leveling)
  - Address C4 (limited lifetime)

- Make out-of-place updates (using reserved flash blocks)
  - Address C2 (erase before write) and C1 (writes smaller than a page updates)

- Maintain a logical to physical address mapping
  - Necessary for out-of-place updates and wear leveling, address C3 (seq. writes)

- A garbage collector is necessary!
Beside these two extremes, many techniques were designed, using temporal/spatial locality, caching, detecting “hotness” of data, distinguishing RW and SW, grouping blocks, etc.

**FTL is a complex piece of software, generally kept secret by flash device manufacturers**
FTL designers vs DBMS designers goals

• **Flash device designers goals:**
  - Hide the flash device constraints (usability)
  - Improve the performance for most common workloads
  - Make the device auto-adaptive
  - Mask design decision to protect their advantage (black box approach)

• **DBMS designers goals:**
  - Have a model for IO performance (and behavior)
    - Predictable
    - Clear distinction between efficient and inefficient IO patterns
  - To design the storage model and query processing/optimization strategies
  - Reach best performance, even at the price of higher complexity (having a full control on actual IOs)

These goals are conflicting!
Minimal FTL: Take the FTL out of equation!

FTL provides only wear leveling, using block mapping to address C4 (limited lifetime)

• **Pros**
  - Maximal performance for SR, RR, SW
  - Semi-Random Writes
  - Maximal control for the DBMS

• **Cons**
  - All complexity is handled by the DBMS
  - All IOs must follow C1-C3
    - The whole DBMS must be rewritten
    - The flash device is dedicated
Semi-random writes (uFLIP [CIDR09])

- Inter-blocks: Random
- Intra-block: Sequential
- Example with 3 blocks of 10 pages:
Bimodal FTL: a simple idea …

• Bimodal Flash Devices:
  - Provide a *tunnel* for those IOs that respect constraints C1-C3 ensuring maximal performance
  - Manage other *unconstrained* IOs in best effort
  - Minimize *interferences* between these two modes of operation

• Pros
  - Flexible
  - Maximal performance and control for the DBMS for constrained IOs

• Cons
  - No behavior guarantees for unconstrained IOs.

**DBMS**

- unconstrained patterns
- constr. patterns (C1, C2, C3)

**Update mgt, Garb. Coll.**
(C1, C2, C3)

**Block map., Wear Leveling**
(C4)

**Flash chips**

(C1) Write granularity
(C2) Erase before write
(C3) Sequential writes within a block
(C4) Limited lifetime
Bimodal FTL: easy to implement

- Constrained IOs lead to **optimal blocks**

  ![Diagram]

  - Optimal blocks can be trivially
    - mapped using a small map table in safe cache
    - detected using a flag and cursor in safe cache

- No interferences!

- No change to the block device interface:
  - Need to expose two constants: block size and page size
Bimodal FTL: better than Minimal + FTL

- Non-optimal block can become optimal (thanks to GC)
Bimodal FTL does not exist yet!

- A simple test
- Device must support TRIM operation → Only recent SSDs
- Results on Intel X25-M
Impact on DBMS Design

Using bimodal flash devices, we have a solid basis for designing efficient DBMS on flash:

• What IOs should be constrained?
  ▪ i.e., what part of the DBMS should be redesigned?

• How to enforce these constraints? Revisit literature:
  ▪ Solutions based on flash chip behavior enforce C1-C3 constraints
  ▪ Solutions based on existing classes of devices might not.
Example: Hash Join on HDD

Tradeoff: IOSize vs Memory consumption

- **IOSize should be as large as possible**, e.g., 256KB – 1 MB
  - To minimize IO cost when writing or reading partitions

- **IOSize should be as small as possible**
  - To minimize memory consumption: One pass partitioning needs $2 \times \text{IOSize} \times \text{NbPartitions}$ in RAM
  - Insufficient memory $\rightarrow$ multi-pass $\rightarrow$ performance degrades!
Hash join on SSD and on bimodal SSD

• With non bimodal SSDs
  ▪ No behavior guarantees but…
  ▪ Choosing IOSize = Block size (128 – 256 KB) should bring good performance

• With bimodal SSDs
  ▪ Maximal performance are guaranteed (constrained patterns)
  ▪ Use semi-random writes
  ▪ IOSize can be reduced up to page size (2 – 4 KB) with no penalty
    ➔ Memory savings
    ➔ Performance improvement
Conclusion

• Adding bimodality is necessary to support efficiently DBMS on flash devices
  ▪ DBMS designer retains control over IO performance
  ▪ DBMS leverages performance potential of flash chips

• Adding bimodality to FTL does not hinder competition between flash device manufacturers, they can
  ▪ bring down the cost of constrained IO patterns (e.g., using parallelism)
  ▪ bring down the cost of unconstrained IO patterns without jeopardizing DBMS design

• This study is very preliminary – many issues to explore
  ▪ More complex storage systems (e.g., RAID, ASM, etc)
  ▪ What abstraction for flash device?
    – Memory abstraction (block device interface)
    – Network abstraction (two systems collaborating)
More information


- **Authors**: Luc.Bouganim@inria.fr, phbo@itu.dk