Flash Talk

Flash Memory Database Systems: Challenges and Opportunities

Bongki Moon
Department of Computer Science
University of Arizona
Tucson, AZ 85721, U.S.A.
bkmoon@cs.arizona.edu

In collaboration with Sang-Won Lee (SKKU), Chanik Park (Samsung)
With technical assistance from M-Tron and Samsung Electronics
Magnetic Disk vs Flash SSD

Champion for 50 years

Seagate ST340016A
40GB, 7200rpm

New challengers!

Samsung FlashSSD
128 GB 2.5/1.8 inch

Intel X25-M Flash SSD
80GB 2.5 inch
Technology Trend

- **NAND flash density increases faster than Moore’s law**
  - Predicted *twofold annual increase* of NAND flash density until 2012 [Hwang, ProcIEEE’03]
  - purSilicon announced 2.5” Nitro SSD with 1-TB capacity (CES’09)
    - Double-stacked 128 chips (2 x 64 x 64Gb), 32-channel, 512 MB RAM, SATA-II

- **Bandwidth catches up and throughput excels**
  - Bandwidth in range of 200-300 MB/sec and 80-150 MB/sec for R/W
  - Throughput in range of 10k-30k and 1k-3k for R/W
Flash SSD for Databases?

- Not inconceivable to run a full database server
  - Computing platforms with TB-scale Flash SSD

- Immediate benefit for some DB operations
  - Reduce commit-time delay by fast logging
  - Reduce read time for multi-versioned data
  - Flash-friendly I/O patterns in temp table spaces

- Still, random scattered I/O is an issue
  - Slow random writes by flash SSD can handle this?
Transactional Log

SQL Queries

System Buffer Cache

Database
Table space

Transaction
(ReDo) Log

Temporary
Table Space

Rollback
Segments
Commit-time Delay by Logging

• Write Ahead Log (WAL)
  - A committing transaction *force-writes* its log records
  - Makes it hard to hide latency
  - With a separate disk for logging
    - No seek delay, but …
    - *Half a revolution of spindle* on average
    - 4.2 msec (7200RPM), 2.0 msec (15k-RPM)
  - With a Flash SSD: about 0.4 msec

• Commit-time delay remains to be a significant overhead
  - Group-commit helps but the delay doesn’t go away altogether.

• How much commit-time delay?
  - On average, 8.2 msec (HDD) vs 1.3 msec (SDD) : *6-fold reduction*
    - TPC-B benchmark with 20 concurrent users.
HDD vs SSD for Logging

- With SSD for log
  - CPU better utilized
    - By shortening commit-time, and serving more active transactions.
  - Leads to higher TPS
- TPC-B to stress-test logging
  - Transaction commit rate higher than TPC-C
  - Logging exaggerated by caching entire DB in memory
Temporary Table Space

SQL Queries

System Buffer Cache

Database
Table space

Transaction
(Redo) Log

Temporary
Table Space

Rollback
Segments
Temp Data and Query Time

- Query processing often generates temp data
  - Sorts, joins, index creation, etc.
  - Typically bulky, performed in foreground;
    Direct impact on query processing time
- Typically stored in separate storage devices

- Ask the same question
  - What happens if SSD replaces HDD for temporary table spaces?
External Sort: I/O Pattern

- External Sort algorithm runs in two phases
  - Sorted run generation
    - Partitioned to chunks, sorted separately and, saved in sorted runs
    - Read sequentially from table space, written sequentially into temp space
  - Merging sorted runs
    - Read randomly from temp space, written sequentially into table space

- Dominant I/O patterns are **sequential write** followed by **random read**
  - No-in-place-update limitation is avoided.
  - These are **flash-friendly** I/O patterns!!
External Sort: Performance

- HDD vs SSD as a medium for a temp table space
  - Sort a table of 2 M tuples (200 MB), with 2 MB buffer cache
- SSD is good at **sequential write + random read**
  - Almost an order of magnitude reduction in merge times
Hash Join: Performance

- HDD vs SSD as a medium for a temp table space
  - Hash-join two tables of 2 M tuples (200 MB) each, with 2 MB buffer cache
  - About 3-fold reduction in join time
Rollback Segments
MVCC Rollback Segments

- Multi-version Concurrency Control (MVCC)
  - Alternative to traditional Lock-based CC
  - Support read consistency and snapshot isolation
  - Oracle, PostgreSQL, Sybase, SQL Server 2005, MySQL

- Rollback Segments
  - Each transaction is assigned to a rollback segment
  - When an object is updated, its current value is recorded in the rollback segment sequentially (in *append-only* fashion)
  - To fetch the correct version of an object, check whether it has been updated by other transactions
**MVCC Write Pattern**

- **Write requests from TPC-C workload**
  - Concurrent transactions generate multiple streams of append-only traffic in parallel (apart by approximately 1 MB)
  - HDD moves disk arm very frequently
  - SSD has no negative effect from no in-place update limitation
MVCC Read Performance

- To support MV read consistency, I/O activities will increase
  - A long chain of old versions may have to be traversed for each access to a frequently updated object

- Read requests are scattered randomly
  - Old versions of an object may be stored in several rollback segments
  - With SSD, 10-fold read time reduction was not surprising
Database Table Space

SQL Queries

System Buffer Cache

Database Table space
Transaction (Redo) Log
Temporary Table Space
Rollback Segments
Workload in Table Space

- TPC-C workload (wholesale supplier queries)
  - Exhibit little locality and sequentiality
    - Mix of small/medium/large read-write, read-only (join)
  - Highly skewed
    - 84% (75%) of accesses to 20% of tuples (pages)
- Write caching not as effective as read caching
  - Physical read/write ratio is much lower than logical read/write ratio
- All bad news for flash memory SSD
  - Due to the No in place update and Asymmetric read/write speeds
Industry Response

- Common in Enterprise Class SSDs
  - Multi-channel, inter-command parallelism
    - Thruput than bandwidth, write-followed-by-read pattern
  - Command queuing (SATA-II NCQ)
  - Large RAM Buffer (with super-capacitor backup)
    - Even up to 1 MB per GB
    - Write-back caching, controller data (mapping, wear leveling)

- Samsung EC SSD Prototype
  - Fat provisioning (up to ~20% of capacity)

- Intel X-25M/E
  - Claims a very low (~1.1) write amplification factor
Impressive Improvement

- **Samsung EC SSD**
  - 10x/100x higher R/W IOPS than early prototypes
  - 20x/8x higher R/W IOPS than a 15k-RPM disk
  - 1.4x~2x higher transaction rate than RAID0 (eight 15k-RPM disks) for R/W TPC-C workload

- **Intel X-25M**
  - Bandwidth: 240/80 (MB/sec) for R/W
  - Throughput: 20000/1200 (IOPS) for R/W
Still, Not There Yet …

- Write still lags behind
  - \( \text{IOPS}_{\text{Disk}} < \text{IOPS}_{\text{SSD-Write}} << \text{IOPS}_{\text{SSD-Read}} \)
  - \( \frac{\text{IOPS}_{\text{SSD-Read}}}{\text{IOPS}_{\text{SSD-Write}}} = 4 \sim 17 \)

<table>
<thead>
<tr>
<th>Prototype/Product</th>
<th>EC SSD</th>
<th>X-25M</th>
<th>15k-RPM Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (IOPS)</td>
<td>10500</td>
<td>20000</td>
<td>450</td>
</tr>
<tr>
<td>Write (IOPS)</td>
<td>2500</td>
<td>1200</td>
<td>450</td>
</tr>
</tbody>
</table>
In-Page Logging (IPL)

- Some academics believe
  - Improving SSD alone cannot do the job

- Key Ideas of the IPL Approach
  - Changes written to log instead of updating them in place
    - Avoid frequent write and erase operations
  - Log records are co-located with data pages
    - No need to write them sequentially to a separate log region
    - Read current data more efficiently than sequential logging
  - DBMS buffer and storage managers work together
Design of the IPL

- Logging on Per-Page basis in both Memory and Flash

- An **In-memory log sector** can be associated with a buffer frame in memory
  - Allocated on demand when a page becomes dirty

- An **In-flash log segment** is allocated in each erase unit

The log area is shared by all the data pages in an erase unit
IPL Write

- Data pages in memory
  - Updated in place, and
  - Physiological log records written to its in-memory log sector
- In-memory log sector is written to the in-flash log segment, when
  - Data page is evicted from the buffer pool, or
  - The log sector becomes full
- When a dirty page is evicted, the content is *not written* to flash memory
  - The previous version remains intact
- Data pages and their log records are physically co-located in the same erase unit
IPL Read

- When a page is read from flash, the current version is computed on the fly

When a page is read from flash, the current version is computed on the fly.

Apply the “physiological action” to the copy read from Flash.

(CPU overhead)

Read from Flash

- Original copy of $P_i$
- All log records belonging to $P_i$

(IO overhead)

Buffer Pool

$P_i$

Re-construct the current in-memory copy

Flash Memory

- data area (120KB): 15 pages
- log area (8KB): 16 sectors
IPL Merge

- When all free log sectors in an erase unit are consumed
  - Log records are applied to the corresponding data pages
  - The current data pages are copied into a new erase unit
    - Consumes, erases, and releases only one erase unit

![Diagram of IPL Merge]

- Physical Flash Block
  - Can be Erased
  - B\textsubscript{old}
  - log area (8KB): 16 sectors

- Merge
  - 15 up-to-date data pages
  - B\textsubscript{new}
  - Clean log area
Evaluation of IPL

- **IPL simulation with TPC-C workload**
  - Average length of a log record: 20 ~ 50 Bytes
    - A single log sector can absorb more than 10 updates
  - An order of magnitude improvement in write time

- **TPC-C *Write* frequencies are highly skewed**
  - Blocks containing hot pages consume log sectors quickly, causing frequent erase operations
  - Trade space for improved write performance
    - Use a larger log segment in blocks for less frequent merges

- **Zero (or negative) write amplification possible**
Concluding Remarks

- **Flash Memory SSD will stay here** …
  - Co-exist or even replace Magnetic Disk
  - Significant performance boost for enterprise systems
  - Cost recovery from energy savings in large-scale TPC-C systems, data centers, HEC systems, etc.

- **Flash-Aware DBMS Design**
  - Need fresh new look at almost everything: Buffer management, B-trees, Sorting and Hashing, Self-Tuning, File Systems, etc.

- **DBMS-Aware SSD Architecture (?)**
  - Address mapping, channel parallelism, command queuing, etc.
Questions

For more information about Bongki’s work,

www.cs.arizona.edu/~bkmoon