MyFlashSQL: Flash is more than faster-harddisk

Sang-Won Lee
SKKU, Korea

Contributors:
Gihwan Oh, Dasom Whang, Mijin Ahn, Donghyun Kang, and Samsung Electronic Memory Division
Sang-Won Lee: Who am I?

- **Professor at SKKU (Sungkyunkwan Univ.), Korea since 2002**
  - Research staff, Oracle Korea (1999 – 2001)

- **Interest: DBMS and OS for NVM (Flash and NVDIMM)**
  - 10+ papers in SIGMOD, VLDB, USENIX FAST and ATC
  - Almost every research was carried out using real DBMS engines (e.g. Oracle, Postgres, MySQL, Couchbase, SQLite)

- **Interested in making research results open-source**
  - MyFlashSQL StarLab (funded by Korean government: 2015 – 2023)
  - Psync [VLDB ‘12], FaCE [VLDB ‘12], SHARE [SIGMOD ‘16]
  - SQLite Optimization [SIGMOD ‘13, VLDB ‘15, In-Progress]
# Table of Contents

- Introduction to Flash and its Opportunities
- Index scan opt. using parallelism [VLDB 2012]
- Share-based DWB opt. [SIGMOD 2016]
- From WAR to RAW [Work-In-Progress]
MySQL/InnoDB on All-Flash

VS.

Sang-Won Lee (swlee@sckku.edu)
**SSD Architecture**

**Figure 1.** An illustration of SSD architecture [3].
# Flash Characteristics and its Implications

<table>
<thead>
<tr>
<th></th>
<th>Storage Mgmt</th>
<th>Buffer Mgmt</th>
<th>Index &amp; QP</th>
<th>Transaction Mgmt</th>
<th>Cache Mgmt</th>
<th>File Consistency / DB Space Mgmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymmetric Read/Write</td>
<td>IPL [SIGMOD07]</td>
<td>CFLRU</td>
<td></td>
<td>tIPL [ICDE2011]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No overwrite / Addr. Mapping Layer</td>
<td>X-FTL [SIGMOD13], SHARE [SIGMOD2016]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X-FTL, SHARE</td>
</tr>
<tr>
<td>No mechanics (Seq RD ~ Rand RD)</td>
<td></td>
<td></td>
<td>SIDX / IDX-based QP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Write &gt;&gt; Random Write</td>
<td>SFS [FAST12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FaCE [VLDB12]</td>
</tr>
<tr>
<td>SSD Architecture (Parallelism et. al.)</td>
<td></td>
<td></td>
<td></td>
<td>Psync [VLDB12]</td>
<td>DuraSSD [SIGMOD2014]</td>
<td></td>
</tr>
<tr>
<td>SSD Architecture (Beyond block device)</td>
<td></td>
<td>Trim, X-FTL, Share; In-Storage Computing; Unit of IO in DB; Multi-streamed IO, NVMe Multi-Queue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sang-Won Lee (swlee@skku.edu)
MySQL/InnoDB on Flash SSDs

- SSDs are not just faster HDD
  - More parallelism (8 ~ 16 degree)
  - Asymmetric read/write speed
  - Computing power and new interfaces
    - e.g. 8 cores and NVMe

- Opportunities for optimizations

- Why not using 4KB instead of 16KB???
  - 16KB ← 5 min rule paper by J. Gray for DISK
  - 16KB → 4KB: 2.5X
Index-Scan Opt. by Exploiting Parallelism
Overview

- Non-clustered index scan causes random I/Os.
- And, leaf nodes in primary index are read one by one.
  - This leads to severe SSD underutilization.
    - Do not believe IOSTAT metrics.
  - The same is true for almost every tree-based indexes.

- Need to change index-scan so as to utilize the abundant parallelism in SSDs.
MySQL InnoDB Engine

- Secondary Index Scan

  Secondary index tree
  (Non-clustered index)

  ![Secondary Index Tree Diagram]

  ![Primary Index Tree Diagram]

  Primary index tree
  (Clustered index)

  Example:
  ```sql
  SELECT * FROM tab
  WHERE a BETWEEN 10 AND 13;
  ```
Prefetch in MySQL InnoDB

Secondary index tree

Primary index tree

Submit asynchronous I/Os (sorted, for prefetching)
Experimental Setup

- **Server Specification**
  - Ubuntu 14.04, Intel Core i5, 3.40G Hz, 8GB(RAM)
  - Two SSDs: Samsung 850Pro (256GB) / Intel SSD P3700 NVMe (400GB)

- **DBMS: MySQL 5.6**

- **Parallel read factor: from 8 to 256**

- **“Orders” table in TPC-H(scale factor 10)**
  - Range query
    
    ```sql
    SELECT * FROM table FORCE INDEX (idx)
    WHERE column_a BETWEEN min AND MAX;
    ```
Experimental Result

- Samsung 850 Pro
  - 16KB Page: ~3.1X with 256 parallel reads
Experimental Result

- **Samsung 850 Pro**
  - 4KB Page: ~4.5X in case of 256 parallel reads

![Graph showing execution time vs. selectivity]
Experimental Result

- PCIe Intel SSD P3700 NVMe
  - 4KB page: **10x** in case of 256 parallel reads
(current) Limitations

- Performs better only in direct IO mode
  - Submit_io() does not operate parallel in buffered IO mode
Share-based DWB Opt.
InnoDB DWB for Atomic Page Write

**Database Buffer**
- **Main LRU List**
  - Head: 
  - Double Write Buffer

**Database on Flash SSD**
- Double Write Buffer

**Free list**
- Scan LRU List from tail
- **Dirty Page Set**
- Flush Dirty Pages
InnoDB Extension with SHARE

- Page-mapping FTL inside SSD

- DWB with SHARE
  - Call SHARE instead of writing data to DB files
  - No redundant writes
  - $\frac{1}{2}$ WAF
  - $2x \uparrow$ performance
SHARE Interface for Flash Storage

**SHARE Interface**
- **Explicit semantic interface** beyond read/write operations

### Page Mapping Table (L2P)

<table>
<thead>
<tr>
<th>LPN</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Physical Address in Flash Memory

- - A B C D E - - -

SHARE (A\_LPN, D\_LPN)
Experimental Result - Jasmine

- MySQL/InnoDB 5.7.5 using LinkBench
  - Page size: 4KB, 8KB, 16KB

(a) Throughput

(b) Total amount of written data
Experimental Result - 960 Pro

- MySQL/InnoDB performance evaluation with LinkBench
  - Benchmark is in progress
  - 24 cores/ 48 threads Intel Server
  - 128 LinkBench Users
  - DWB-on vs. DWB-on with SHARE

![Graph showing the comparison between DWB-on and DWB-SHARE operations per second for 4KB operations. The graph indicates a 2.4x increase with DWB-SHARE.](image)
SHARE Interface for File Systems

- Three types of file systems
  - Journaling: Ext4
  - LFS: F2FS
  - Copy-on-Write: BTRFS

- Runtime overheads for guaranteeing consistency
  - Ext4: Double-writes for metadata/data (like DWB)
  - F2FS: Segment cleansing (like Couchbase Compaction)
  - BTRFS: Tree-wandering (like Couchbase Write)
Experimental Result

- **LinkBench on MySQL: Original vs. AFS(SDJ)**

![Graphs showing performance, write amount, and disk cache flush count for different configurations.](image)

**Figure 1:** OLTP benchmark results of Sysbench and LinkBench using MySQL. The original MySQL versions were tested in three different configurations: 1) DWB-0FF/0D, 2) DWB-0N/0D (default), and 3) DWB-0FF/DJ, while MySQL on AFS were in DWB-0FF/SDJ. SysBench in an OLTP mode (10 GB database (20 files) with 40 million rows for 1,000,000 operations. LinkBench: we ran 4,800,000 operations for a 50 GB database (24 files) after a two minute warm-up. In both experiments, MySQL/InnoDB engine was configured to use 5 GB as a buffer pool with sixteen concurrent threads, and all under buffered I/O mode.
From WAR to RAW
MySQL Buffer Manager: Read

- **Read-blocked-by-Write problem**
  - Read is blocked until the dirty page is safely written to the storage

- Considering the **asymmetric R/W speed of flash**, read operations cannot fully utilize its performance because of reads blocked by write operation
MySQL Buffer Manager: Read

Single page flush
⇒ CPU/IO utilization, throughput ↓
From WAR to RAW

- **Benefits**
  - Better read latency
  - Higher CPU and SSD utilization → Higher throughput

- **For source code (@InnoDB 5.6), contact me at swlee@skku.edu**
Experimental Setup

- **System Configuration**
  - Intel(R) Core(TM) i5-4670 CPU @ 3.40GHz
  - Linux kernel 4.10.1 (Ubuntu 14.04.4 LTS)
  - Data devices
    - 850 PRO SSD / 960 PRO NVMe / PM961 NVMe (Samsung)
    - 845 DC battery-backed SSD

- **Workloads:** TPC-C / LinkBench / SysBench

- **InnoDB:** 5.6
TPC-C Benchmark Result

- Page size 16KB / DB 200GB / Buffer 4GB / 64 users

### HDD

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>RAW</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpmC (Transactions per minute Count)</td>
<td>121</td>
<td>123</td>
</tr>
</tbody>
</table>

Sang-Won Lee (swlee@skku.edu)
TPC-C Benchmark Result

- Page size 16KB / DB 200GB / Buffer 4GB / 64 users

- NVMe SSD:
  - Original: 14023 TpmC
  - RAW: 32070 TpmC
  - Improvement: 2.3x

- DC SSD (battery-backed):
  - Original: 26544 TpmC
  - RAW: 33305 TpmC
  - Improvement: 1.3x

- Samsung 850 PRO SSD:
  - Original: 8468 TpmC
  - RAW: 20269 TpmC
  - Improvement: 2.4x
TPC-C Benchmark Result

- Samsung 960 PRO NVMe

![Graphs showing TpmC (Transactions per minute Count) and CPU Utilization (%)]
TPC-C Benchmark Result

- Samsung 845DC EVO SSD (battery-backed SSD)
TPC-C Benchmark Result

- Samsung 850 PRO SSD

- TpmC (Transactions per minute Count)
  - Original: 8468, 8549, 8565, 8727
  - Modified: 20269, 18126, 17715, 17232

- CPU Utilization (%)
  - Original: 512, 1024, 2048, 4096
  - Modified: 512, 1024, 2048, 4096

- IO Utilization (%)
  - Original: 512, 1024, 2048, 4096
  - Modified: 512, 1024, 2048, 4096
LinkBench Result

- Samsung 850 PRO / PM961 NVMe
- 16KB page / DB 59GB / Buffer 1GB / 128 users
- Read up to 3.7x, Write up to 3.8x
LinkBench Result

LinkBench Transaction Latency (p99)

LinkBench Transaction Latency (max)

Sang-Won Lee (swlee@skku.edu)
SysBench using Further Optimized RAW

- Samsung 960 PRO NVMe for data
- Page size 16KB / DB 188GB / Buffer 4GB / 200 users
Other topics

- **Completed:**
  - FaCE: Flash as Cache (done)

- **On-going**
  - QP and QO for Index Scan
  - Insert Buffer as Write Reduction Mechanism
  - Advanced FTL for Database Applications
  - Applying multi-stream SSD to MySQL
  - NVDIMM-based InnoDB Logging and CC opt.
Q & A

- Homepage: [http://flashsql.skku.ac.kr](http://flashsql.skku.ac.kr);
  - Papers, tech. blogs

- Github: [https://github.com/FlashSQL](https://github.com/FlashSQL)
  - Source codes: AIO-prefetch / SHAREd MySQL / MySQL/FaCE
  - RAW-enabled MySQL (upon request)
    - E-mail: swlee@skku.edu