In-Storage Compute: an Ultimate Solution for Accelerating I/O-intensive Applications

13 August 2015

YANG SEOOK KI, Director, Memory Solutions Lab, Samsung Electronics
Disclaimer: The contents provided in this material are based on concepts and early research results, and are for technical discussions only. This material does not reflect any product-level plan of records.
Contributors

Korea HST Team
- Boncheol GU
- Duckho BAE
- Sungho YOON

US ADS Team
- Yangwook KANG
- Heekwon PARK

Special thanks to:
Yoonho Chung, Insoon Jo, Minwook Jung, Jungwook Kang, Moonsang Kwon, Truong Nguyen, Dongchul Park, Prem Paulson, Jonghyun Yoon
Outline

1. Background
2. In-Storage Compute Concept
3. ISC Prototype
4. Case Studies
5. Summary
**Data Processing Market**

- **Constant growth of business intelligence & analytics market**

**Worldwide BI and Analytics Tools Revenue by Segment**

**BI and analytics tools are I/O hungry!!**

- They usually access terabytes (sometimes even petabytes) of data on slow storage device

<table>
<thead>
<tr>
<th>Event</th>
<th>Waits</th>
<th>Time(s)</th>
<th>Avg. wait (ms)</th>
<th>% DB time</th>
<th>Wait class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct path read</td>
<td>4,604,339</td>
<td>567.141</td>
<td>123</td>
<td>63.67</td>
<td>User I/O</td>
</tr>
<tr>
<td>Direct path read temp</td>
<td>1,955,162</td>
<td>147,298</td>
<td>75</td>
<td>16.54</td>
<td>User I/O</td>
</tr>
<tr>
<td>DB CPU</td>
<td>38,874</td>
<td></td>
<td></td>
<td>4.36</td>
<td></td>
</tr>
<tr>
<td>DB file sequential read</td>
<td>117,944</td>
<td>16,399</td>
<td>139</td>
<td>1.84</td>
<td>User I/O</td>
</tr>
<tr>
<td>Direct path write temp</td>
<td>597,138</td>
<td>13,507</td>
<td>23</td>
<td>1.52</td>
<td>User I/O</td>
</tr>
</tbody>
</table>

Source: HUAWEI, Accelerate Oracle Performance, Sep 2012 [2]
CPU-centric Computing Model (Von Neumann)

SSDs

Data 1

Data 2

Data 3

Long journey of data

Final data

Host

DRAM

CPU

Confidential
Moving Data is Expensive!!

Moving Computation is Cheaper than Moving Data
Source: HDFS Architecture Guide [3]

Reducing data movement can help improve both energy and performance

The energy consumed by data movement is starting to exceed the energy consumed by computation
Near Data Processing Technology

Intelligent SSD [NxGnData]

Exadata [Oracle]

SPU (Storage Processing Unit) [Seagate]

Netezza S-blade [IBM]

Closer to source
The ultimate of close-to-data compute for high performance & low power is

“In-Storage Compute (ISC)”
What is ISC (In-Storage Compute)?

ISC SSDs

Host

DRAM

CPU

Final data
ISC is an ultimate approach to IO reduction/avoidance.
SSD is a complete computer with high performance low power processor.
Why? Bandwidth Gap

Superfluous internal bandwidth

- To hide processing overhead of host interface and FTL
Why? Resource Utilization

- Storage resource is underutilized
How? ISC Application Development Process

1. **C/C++ - Support C++11/STL**

2. **X86 Compile**

3. **ISC SSD Emulation**

4. **ARM Cross compile**

5. **Download /isc/myprogram/ssdlet**

6. **Run /isc/myprogram/host**
ISC Dataflow Programming Model

Host ISC Application

Storage

Device ISC Application

SSDlet

Input port

Output port

Pipe

get()

put()

read()

write()
ISC Multiple Device Model

Host ISC Application

- Input port
- Output port
- Pipe
int main(int argc, char *argv[]) {
    SSD ssd("/dev/nvme0n1p1");
    module_id_t mid = ssd.loadModule(File(ssd, "./libkvstore.so");
    Application app(ssd);

    SSDLet kvstore(app, mid, "KVStore");
    auto out_command = app.connectTo<String>(kvstore.in(0));
    auto out_key = app.connectTo<String>(kvstore.in(1));
    auto out_value = app.connectTo<String>(kvstore.in(2));
    auto in_result = app.connectTo<String>(kvstore.out(0));

    app.start();

    string command, key, value;
    while (std::cin >> command) {
        if (command == "get") {
            out_command.put(command);
            std::cin >> key;
            out_key.put(key);
            in_result.get(value);
            std::cout << value << std::endl;
        } else if (command == "put") {
            out_command.put(command);
            std::cin >> key >> value;
            out_key.put(key);
            out_value.put(value);
        } else break;
    }
    return 0;
}
ISC Runtime Framework

Host Program

ISC Host Library

Input port

Output port

Pipe
Samsung ISC SSD Prototype

- **Commodity SSD**
  - Samsung PM1725 NVMe with the ISC feature
  - PCIe 3.0 x4
  - 800 GB

- **Software**
  - C++11
  - C++ STL
  - g++
  - Software emulator
MySQL determines data pages to fetch according to relevance hints from SSD

- MySQL gets relevance hints for pages in a given range all at once
- Filter out access to pages with irrelevant data
**Data Analytics Query**

### Elapsed time of TPC-H query 2

- An analytic query to find a minimum cost supplier

```sql
SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND
  p_size = 15 AND p_type LIKE '%BRASS' AND
  s_nationkey = n_nationkey AND n_regionkey = r_regionkey AND r_name = 'EUROPE'
  AND ps_supplycost = (  
    SELECT MIN(ps_supplycost)  
    FROM partsupp, supplier, nation, region  
    WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND  
    s_nationkey = n_nationkey AND n_regionkey = r_regionkey AND r_name = 'EUROPE')
ORDER BY s_acctbal desc, n_name, s_name, p_partkey
LIMIT 100;
```

- ISC reduces the query time to less than 1/40

<table>
<thead>
<tr>
<th># of pages read w/ MySQL (baseline)</th>
<th>Table name</th>
<th># of pages read w/ ISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,325,978</td>
<td>Total</td>
<td>22,317</td>
</tr>
<tr>
<td>325,386</td>
<td>Part</td>
<td>7,525</td>
</tr>
<tr>
<td>15,229</td>
<td>Supplier</td>
<td>4,582</td>
</tr>
<tr>
<td>985,354</td>
<td>Partsupp</td>
<td>10,201</td>
</tr>
<tr>
<td>5</td>
<td>Nation</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Region</td>
<td>4</td>
</tr>
</tbody>
</table>

The most efficient plan is to put part table first in the join order and filter out its irrelevant pages!
3.6X TPC-H Query Processing Speed Up

A representative TPC-H benchmark subset is expected to reveal over 3.6x performance gains

<table>
<thead>
<tr>
<th>Host server</th>
<th>Dell PowerEdge R720</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Intel(R) Xeon(R) CPU E5-2640 0 @ 2.50GHz x2</td>
</tr>
<tr>
<td></td>
<td>- 3G of DRAM</td>
</tr>
<tr>
<td></td>
<td>- OS device: Samsung MZ-6ER100T SAS 100GB SSD</td>
</tr>
<tr>
<td></td>
<td>- Data device: PM1725 480GB NVMe SSD (SR=3GB/s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS</th>
<th>Ubuntu 15.04 (3.19.0 kernel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Mariadb-5.5.42 &amp; TPC-H 2.17.0</td>
</tr>
<tr>
<td>TPC-H dataset</td>
<td>20G of dataset (with scale factor of 10)</td>
</tr>
</tbody>
</table>

**Speed-up by ISC**

![Bar chart showing speed-up by ISC for different query numbers, with Geo mean highlighted.

Geo mean: 3.6
Case Study 2: Storage Compaction

**LevelDB**

- One of popular embedded databases
- Open-source, embedded key/value store by Google
- Base database system for other open source projects
  - RocksDB (LevelDB+HBase), HyperLevelDB
  - Riak, Ceph storage backend

- **Log**: Max size of 4MB then flushed into a set of Level 0 SST files
- **Level 0**: Max of 4 SST files then one file compacted into Level 1
- **Level 1**: Max total size of 10MB then one file compacted into Level 2
- **Level 2**: Max total size of 10 x Level 1 then one file compacted into Level 3
- **Level 3+**: Max total size of 10 x previous level then one file compacted into next level
New LevelDB with Compaction Powered by ISC

- **Insert/update**
  - memtable
  - Immutable table
- **Append**
  - Log
- **Flush**
- **Compact**
- **Reads**
- **Writes**
  - Compaction Read/Merge/Write

Host

SSD
Up to 10X Throughput Improvement

No Read

More Flush
Take-Away Messages

- Computing paradigm shift from CPU-centric to data-centric for I/O intensive applications
- Samsung ISC realizes heterogeneous computing framework across general purpose CPU and SSD.
- IO intensive applications can benefit from low power high performance of embedded processors and high internal bandwidth of SSDs.

Samsung ISC prototype

- ISC-aware MySQL achieves performance improvement up to 80x or 3.6x on average with TPC-H
- ISC-aware LevelDB achieves up to 10x throughput improvement with dbbench (default benchmark)
Meet our engineers at booth 307
4-7PM (Tue), 12-7PM (Wed), 10:30AM-2PM (Thu)

Thank You!

yangseok.ki@samsung.com