Biscuit: A Framework for Near-Data Processing of Big Data Workloads

Oct 21, 2016

Duck-Ho Bae

Memory Business, Samsung Electronics
## Outline

- **Biscuit: A Framework for Near-Data Processing of Big Data Workloads, ISCA16**
- **YourSQL: A High-Performance Database System Leveraging In-Storage Computing, VLDB16**
Near-Data Processing (NDP)

- “Moving Computation is Cheaper than Moving Data”

**Traditional data processing**

- Data
- Host interface / Network / ...
- Processing Server
- Client

**Near-data processing**

- Data
- Host interface / Network / ...
- NDP
- Processing Server
- Results
- Client

- Near-data processing moves computation to data
  - Computation is performed right at the data source
  - Efficient when the cost of moving data is very high

*HDFS Architecture Guide*
In–Storage Computing (ISC)

- The ultimate of near–data processing is “In–Storage Computing”

![NDP with ISC Diagram]

- Most prior work focuses on proving the concept of ISC
  - Little attention to designing and realizing a practical framework
  - Realistic large application studies were omitted
Samsung NVMe SSD (PM1725)

800GB/1.6TB/3.2TB

- V-NAND TLC with Epic Controller
- Sequential Read up to 3,100 MB/s
  Sequential Write up to 1,800 MB/s
- 4KB sustained Random Read up to 750K IOPS
  4KB sustained Random Write up to 120K IOPS
- Power-loss Protection
A user-programmable NDP framework for SSDs and data-intensive applications

- The first reported product-strength NDP system
- Modern C++ support (including C++ standard library)
- Dynamic loading of user programs
- Multi-threading, multi-core support
SSD Hardware

- Limitations
  - Low compute power, no cache coherence, a small amount of fast memory, no MMU, and restrictive synchronization primitives

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host interface</td>
<td>PCIe Gen.3 x4 (3.2GB/s)</td>
</tr>
<tr>
<td>Protocol</td>
<td>NVMe 1.1</td>
</tr>
<tr>
<td>Device density</td>
<td>1 TB</td>
</tr>
<tr>
<td>SSD architecture</td>
<td>Multiple channels/ways/cores</td>
</tr>
<tr>
<td>Storage medium</td>
<td>Multi-bit NAND flash memory</td>
</tr>
<tr>
<td>Compute resource for Biscuit</td>
<td>Two ARM Cortex R7 cores @ 750MHz with L1 cache</td>
</tr>
<tr>
<td>On-chip SRAM</td>
<td>&lt; 1 MiB</td>
</tr>
<tr>
<td>DRAM</td>
<td>≥ 1 GiB</td>
</tr>
<tr>
<td>Hardware IP</td>
<td>Key-based pattern matcher per channel</td>
</tr>
</tbody>
</table>
**Biscuit Runtime**

- **Cooperative multi-threading**
  - A limited form of multi-threading (fiber as a scheduling unit)
  - Less context switching overhead
  - Safe resource sharing without locking

- **Shared nothing architecture**
  - All data transmission among threads through I/O ports
  - Enforced by the programming model and APIs
  - C++11 move semantics supported

- **Dynamic loader for user programs**
  - User program as position-independent code (PIC)
  - Symbol relocation to locate each program in a separate address space
**Biscuit System Architecture**

**Host Side**
- **host-side program**
  - `libsisc`
- **NVMe driver**

**Biscuit-enabled SSD**
- **SSD-side module**
  - SSDlet
  - SSDlet
  - SSDlet
  - fiber
  - fiber
  - fiber
- **ISC runtime**
- **SSD firmware**

**Host Interface Controller**
- **CPUs**
- **SRAM**
- **DRAM**
- **FMC**
- **NAND**

**Channels**
- Channel #0
- Channel #(n_{ch}-1)

**Main Memory**
- LLC

**System Agent**
- PCIe I/F
**Biscuit Programming Model**

- **Biscuit follows a data-flow model**
  - The data movement through ISC tasks determines their order of execution
  - On receiving all required inputs, an ISC task produces output and passes it to the next ISC tasks in the data-flow path
An ISC task is a unit of task that would run on an ISC–enabled SSD.

A host–side program creates and manages ISC tasks.

Both run concurrently in the ISC–enabled SSD and the host, respectively.
Development Process

1. Write codes
2. X86 Compile
3. ARM Cross compile
4. Copy the module into Biscuit SSD
5. Run host-side program

Host-side task

SSD-side task

Host-side program

SSD-side module

Host Computer

ISC
## Experimental Setup

### H/W setup

<table>
<thead>
<tr>
<th>System</th>
<th>Dell PowerEdge R720 server</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2 Intel Xeon(R) CPU E5–2640 (12 threads per socket) @2.50GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>64 GiB DRAM</td>
</tr>
<tr>
<td>OS</td>
<td>64-bit Ubuntu 15.04</td>
</tr>
</tbody>
</table>

### Basic performance results

- Communication latency, data read latency, data read bandwidth

### Application level results

- String search, pointer chasing, DB scan/filtering, TPC–H

### Notations

- **Conv**: system configuration with a default conventional SSD
- **Biscuit**: system configuration with the Biscuit framework on the SSD
Basic Performance Results – Data Read Latency

- **Conv**: Linux pread I/O primitive
- **Biscuit**: internal data read API

<table>
<thead>
<tr>
<th>Read Latency (us)</th>
<th>Conv</th>
<th>Biscuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KiB</td>
<td>90.0</td>
<td>75.9</td>
</tr>
</tbody>
</table>

- Biscuit shows 18% shorter latency
- Biscuit has the shorter round-trip “path” — No data transmission from the device to the host over a host interface
Basic Performance Results – Data Read Bandwidth

- **Conv**: transfer data to the host-side program
- **Biscuit**: transfer data to the SSD-side module (i.e., internal read)

- Biscuit exploits the underutilized internal bandwidth
Application Level Results – Pointer Chasing

- **Conv**: round-trip operation between host and SSD
- **Biscuit**: perform data-dependent logic entirely within SSD

<table>
<thead>
<tr>
<th>Execution time (s)</th>
<th>Conv</th>
<th>Biscuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>20GiB Twitter data</td>
<td>138.6</td>
<td>124.4</td>
</tr>
<tr>
<td>100 starting nodes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Biscuit achieves 11% performance gain
- This gain is comparable to the improvement in read latency with Biscuit
Application Level Results – DB Scan and Filtering

- Data analytics with a real DB engine
  - MariaDB 5.5.42 (XtraDB)
  - We modified the query engine to
    1. identify a candidate table amenable for offloading
    2. estimate its selectivity using a sampling method
    3. determine whether the table is indeed a good target (based on a selectivity threshold)
    4. and finally offload the identified filter to the SSD
Application Level Results – DB Scan and Filtering

Filtering Query

```
SELECT l_orderkey, l_shipdate, l_linenumber
FROM lineitem
WHERE l_shipdate = '1995-1-17'
```

- Biscuit achieves speed-ups of about 11x
- Execution times on Biscuit were very consistent
Application Level Results – Power Consumption

- Filtering Query

- Biscuit consumes more power during query processing
- Biscuit achieves significantly lower energy consumption thanks to its reduced execution time

<table>
<thead>
<tr>
<th></th>
<th>Conv</th>
<th>Biscuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Energy</td>
<td>60.5</td>
<td>12.2</td>
</tr>
</tbody>
</table>
Running all queries, Conv takes nearly two days, while Biscuit takes about 13 hours (3.6x speed-up)

Top 5 queries take 70+% of total execution time
Conclusions

- We presented the design and implementation of Biscuit, an NDP framework built for high-speed SSDs.

- With Biscuit, we pursued achieving high programmability on distributed resources including processing units of SSDs as well as host CPUs.

- Biscuit is the first reported product-strength NDP system implementation.

- We successfully ported Biscuit on small and large data-intensive applications including MariaDB.

- Biscuit accomplished the performance improvement of up to 166x for TPC-H queries (average 6.1x improvement).
YourSQL: A High-Performance Database System Leveraging In-Storage Computing
YourSQL – ISC-enabled Database System

- Realizes very early-filtering of data by offloading data scanning of a query to ISC-enabled SSDs

- Why early-filtering?
  - Early-filtering is data-intensive, non-complex query operations
  - I/O reduction from the optimized join order and irrelevant data elimination is dramatic!

<table>
<thead>
<tr>
<th>Join order</th>
<th>Table name</th>
<th>Access method</th>
<th># of read requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Region</td>
<td>All</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Nation</td>
<td>Ref</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Supplier</td>
<td>Ref</td>
<td>36,867</td>
</tr>
<tr>
<td>4</td>
<td>Partsupp</td>
<td>Ref</td>
<td>2,842,639</td>
</tr>
<tr>
<td>5</td>
<td>Part</td>
<td>Eq_ref</td>
<td>651,525</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>3,531,060</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Join order</th>
<th>Table name</th>
<th>Access method</th>
<th># of read requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Part</td>
<td>Ref</td>
<td>245</td>
</tr>
<tr>
<td>2</td>
<td>Partsupp</td>
<td>Ref</td>
<td>98,520</td>
</tr>
<tr>
<td>3</td>
<td>Supplier</td>
<td>Eq_ref</td>
<td>45,679</td>
</tr>
<tr>
<td>4</td>
<td>Nation</td>
<td>Eq_ref</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Region</td>
<td>All</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>144,453</strong></td>
</tr>
</tbody>
</table>

(a) MySQL w/o ICP

(b) MySQL w/ ICP

- TPC-H Q.2 on TPC-H dataset with a scale factor of 100
YourSQL Architecture

YourSQL Query Engine

1. Parser
2. YourSQL Query Planner
3. Host-side Sampler
4. YourSQL Query Executor
5. Host-side Filter
6. Prefetcher

YourSQL Storage Engine

ISC FrameWORK

- Sampler Task
- Filter Tasks
- Internal Sequential Read
- ISC-enabled SSD

Bulk Random Read
- **Early-filtering target table** is placed first in the join order
  - YourSQL assigns a **limiting score** for each filter predicate, which represents how restrictive its filter predicates are
  - The table with the highest limiting score is determined as the early filtering target

- For the remaining join order, it follows MySQL's decision
**YourSQL Query Engine – Join Order Optimization**

List all the tables with filter predicates

Eliminate small tables

Calculate the limiting score of each remaining table

Eliminate the tables whose limiting score is below a given threshold

Select the table with the highest limiting score as the candidate

**TPC-H Query 2**

```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'
ORDER BY s_acctbal DESC, n_name, s_name, p_partkey
LIMIT 100;
```

- Region table: Single predicate
- Part table: Two predicate
TPC-H Query 2

```
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;
```
YourSQL Query Engine – Join Order Optimization

TPC-H Query 2

```
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;
```

- Add a limiting score of each filter predicate
- A filter predicate gets a higher score as its type of operation is more restrictive (e.g., EQUAL)
**YourSQL Query Engine – Join Order Optimization**

**TPC-H Query 2**

```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;
```

- Region table: Single predicate
- Part table: Two predicate
YourSQL Query Engine – Join Order Optimization

- Query Plan of YourSQL for TPC-H Query 2

Intermediate row sets can significantly be reduced at the earliest stage of join!

MySQL w/ICP performs early filtering by secondary indexes on filter columns. In contrast, YourSQL performs early filtering with the ISC filters, which scan the early filtering target.
YourSQL Storage Engine – Filtering Condition Pushdown (FCP)

- An optimization for the case where YourSQL retrieves rows from a table using filter predicates

- YourSQL’s filter leverages the H/W pattern matcher
  - Transforms filter predicates into binary patterns and feeds them to the ISC Filter task
  - E.g., in TPC-H Query2, p_type LIKE ‘%BRASS’ is converted into binary keys, ‘42 52 41 53 53’

- Match hints: a byte array whose element is set to 1 if the corresponding page satisfies filtering conditions.
YourSQL checks match hints first, and fetches pages whose match hint is set to one with “normal host read”

- **YourSQL** checks match hints first, and fetches pages whose match hint is set to one with “normal host read”

![Diagram](image)

(a) Sequential processing

- **Early filtering task and the remaining tasks (i.e., match page reads and row processing) run concurrently in the ISC-enabled SSD and the host**

![Diagram](image)

(b) Concurrent processing
Optimization – Sampling-driven FCP

- List all the tables with filter predicates
- Eliminate small tables
- Calculate the limiting score of each remaining table
- Eliminate the tables whose limiting score is below a given threshold
- Select the table with the highest limiting score as the candidate
- Estimate the filtering ratio of the candidate by the ISC sampler
- Determine the candidate as the target if the estimated filtering ratio is sufficiently high

- The limiting score is a simple heuristic, but not quantitatively correlated with filtering ratio
- An ISC task called “sampler” is used to provide a quantitative estimation of filtering ratio
  - Sampler is the same as the filter functionality-wise, but scans the sampling region only
- The estimated filtering ratio enables YourSQL to check further if early filtering for a candidate table would really be beneficial in terms of execution time
- Hardware matcher only performs byte-granular matching and the filtered data may still contain false positives depending on the filtering conditions
  - e.g., shipdate > `'1995-09-01' and shipdate < `1995-09-01' + INTERVAL 1 MONTH
    -> `8F 97 21' and `8F 97 41' -> `8F 97' (extract common two byte sequence)
  - '8F 97' would match sequences from '8F 97 00' through '8F 97 FF'
**Optimization – Highly Accurate Bulk Prefetch**

### (a) Synchronous reads of single-page units

1. **Start**
2. Read of match page
3. CPU exec.
4. Read of match page
5. CPU exec.

**ISC-enabled SSD**

**Host-side YourSQL**

**Early filtering**

**Early filtering**

### (b) Synchronous reads of multi-page units

1. **Start**
2. Read of match page
3. CPU exec.
4. Read of match page
5. CPU exec.

**ISC-enabled SSD**

**Host-side YourSQL**

**Early filtering**

**Early filtering**

### (c) Asynchronous reads of multi-page units

1. **Start**
2. Bulk read of match pages
3. CPU exec.
4. Bulk read of match pages
5. CPU exec.

**ISC-enabled SSD**

**Host-side YourSQL**

**Prefetcher**

**Early filtering**

**Early filtering**
Experimental Setup

- H/W setup

<table>
<thead>
<tr>
<th>System</th>
<th>Dell PowerEdge R720 server</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2 Intel Xeon(R) CPU E5–2640 (12 threads per socket) @2.50GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GiB DRAM</td>
</tr>
<tr>
<td>SSD</td>
<td>Samsung PM1725 1TB (ISC-enabled)</td>
</tr>
<tr>
<td>OS</td>
<td>64-bit Ubuntu 15.04</td>
</tr>
</tbody>
</table>

- Baseline system and workload
  - MariaDB 5.5.42 was integrated with Biscuit framework
  - TPC–H with a scale factor of 100 was chosen
Out of 22 queries, eight queries were FCP-enabled:
- The rest queries had no filter predicates or YourSQL did not expect speed-up for FCP.
- The average speed-up of the top five queries reached 15x.
- 3.6x reduction of the overall execution time was achieved.
Evaluation Results – Optimization Techniques

- More optimizations yield higher speed-up, since each optimization scheme is orthogonal to one another.

- The biggest improvement seen in Opt-PSH implies that the host-side read operation was the limiting factor in accelerating the overall performance.
As the memory size decreases, the resulting speed-up becomes higher.

When the memory usage becomes tighter, the relative cost of read I/O is increased and the impact of its reduction becomes more prominent.
Conclusions

- We presented the design and implementation of **YourSQL**, an ISC-enabled database system.
- With YourSQL, we pursued accelerating data-intensive queries with the help of additional in-storage computing capabilities.
- We seamlessly integrated query offloading to SSDs into one of the most popular database systems, MySQL.
- YourSQL accomplished the 3.6x reduced execution time for TPC-H queries.
Wordcount Example

![Diagram of wordcount example]

- Host-side program
- Mapper
- Mapper
- Mapper
- Shuffler
- Reducer
- Reducer

Input: filename
Output: <word, count>
int main(int argc, char *argv[]) {
    // create an instance of the SSD class that corresponds to an Smart SSD
    SSD ssd("/dev/nvme0n1p1");

    // load an SSDlet stored on the Smart SSD
    File file(ssd, "/var/isc/slets/libwordcount.so");
    module_id_t mid = ssd.loadModule(std::move(file));

    // create an instance of the Application class to manage SSDlets on the Smart SSD
    Application wordcount(ssd);

    // create instances of necessary SSDlet classes included in the loaded module
    auto args = std::make_tuple(File(ssd, argv[1]));
    SSDLet mapper(wordcount, mid, "idMapper", std::move(args));
    SSDLet shuffler(wordcount, mid, "idShuffler");
    SSDLet reducer(wordcount, mid, "idReducer");
// make connections between SSDlets
wordcount.connect(mapper.out(0), shuffler.in(0));
wordcount.connect(shuffler.out(0), reducer.in(0));

auto port = wordcount.connectTo<std::pair<std::string, uint32_t>>(reducer.out(0));

// starting application would make all SSDlets begin execution
wordcount.start();

std::pair<std::string, uint32_t> value;
// keep reading as long as output is available
while (port.get(value))
    std::cout << value.first << "\t" << value.second << std::endl;

ssd.unloadModule(mid);
return 0;
```cpp
class Mapper
    : public SSDLet<OUT_TYPE<std::pair<std::string, uint32_t>>,
        ARG_TYPE<File>> {
public:
    // SSDlet start function
    void run() {
        // get filename as argument from host-side code
        auto& file = getArgument<0>();
        // get outputPort connected with Shuffler SSDlet
        auto output = getOutputPort<0>();

        // do Mapper tasks
        FileInputStream fs(std::move(file));
        while (true) {
            sstring line;
            if (!readline(fs, line))
                break;

            line.tokenize();
            sstring::const_iterator word;
            while ((word = line.next_token()) != line.cend()) {
                // send results to Shuffler SSDlet through pipe
                if (!output.put({std::string(word), 1}))
                    return;
            }
        }
    }

    // register 'Mapper' SSDlet
    RegisterSSDLet(idMapper, Mapper)
```

**Wordcount Example – SSDlet: Mapper**

- **Arg:** File
- **Out:** <str, uint32_t>
Biscuit I/O Ports

Communication through ports

(a) Inter-SSDlet ports: among SSDlet instances belonging to a single Application instance

(b) Host-to-device ports: between an SSDlet instance and a host program

(c) Inter-application ports: between two SSDlets from different Application instances
ISC-enabled Database System

- **Design Considerations**
  - Partitioning host/ISC tasks
  - Defining interfaces between a host and ISC tasks
  - Optimizing query planner for ISC
  - Reorganizing datapath for ISC database system

Traditional

- Parser
- Query Planner
- Query Executor
- Storage Engine
- Host I/F
- Normal SSD

ISC-enabled DBMS

- Parser
- ISC-aware Query Planner
- ISC-aware Query Executor
- Host-side ISC module
- Storage Engine
- Host I/F
- ISC-enabled SSD
- ISC task
- ISC task