In-Storage SQL Processing

MIT CSAIL Joint Research Proposal

Shuotao Xu, Xiangyao Yu
CSG Group, DB Group
(shuotao@csail.mit.edu), (yxy@csail.mit.edu)

May 21, 2018

1 Introduction

With the ever-growing data collected and available, technology for fast and efficient data storage is becoming increasingly crucial for supporting complex queries on large datasets. For big-data applications, a large number of analytic platforms rely on relational database systems, such as MySQL and Postgres, to store large datasets and running complex SQL queries. An SQL database system brings data from the flash disks via the interconnecting storage network to the CPU cache to perform analysis of the fetched data.

With today's technology advancement, the storage network I/O bandwidth can be imbalanced with the aggregate secondary storage bandwidth, and become a bottleneck for fast SQL processing. Such a scenario can be true for both a single server and a cluster of servers for SQL processing. On a single server node, a flash disk is typically connected via 4 lanes of PCIe gen3 bus, which provide 4GB/s of bandwidth. The flash chips inside a flash disk can provide a internal aggregate bandwidth which is much greater than the PCIe gen3 bus, if they are organized in a parallel architecture. Therefore, SQL processing on a single node can be bottlenecked by the PCIe bus bandwidth. On the other hand, on the cluster of servers performing distributed SQL analysis, compute and storage components are typically separated on different physical servers, such as the Amazon cloud. With such a compute and storage separation in a cloud, the distributed SQL processing performance can be also held up by the slow server network speed.

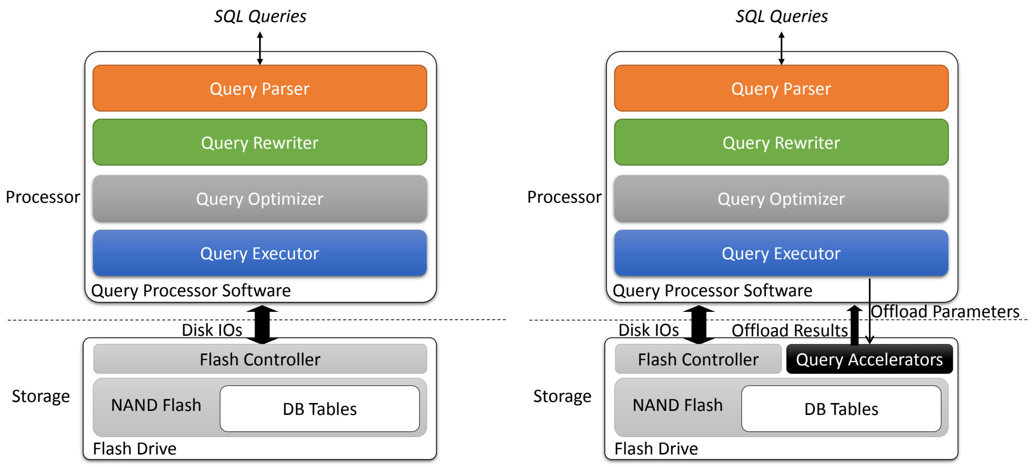
In today's flash storage technology development, industry as well as academia are trying to enable some in-storage computational capabilities, such as ARM processors and FPGAs, to application developers. In this research, we want to investigate how to use such in-storage computation hardware to offload some of the SQL operators inside flash drives to boost SQL query performance. We propose to investigate 5 important SQL operators for the benefits of in-storage processing: *GROUP-BY AGGREGATE*, *JOIN*, *TopK*, *INDEX PROBING* and *SORT*. We will devise new techniques to implement those operators on inside storage due to limited computation and DRAM size. We assume that the IO-intensive operators such as GROUP-BY AGGREGATE, JOIN, TopK, INDEX PROBING will benefit the most from in-storage processing. Operators, such as SORT, is compute-bound, which we assume will not benefit from in-storage computing.

Please note that this MIT internal collaboration between Computer Structure Group and Database Group has been started prior to the Samsung Internship. And we are interested in pursuing this research in a open-source domain.

2 System Architecture

A relational query processor takes a declarative SQL statement, validates it, optimizes it into a procedural dataflow execution plan, and execute that dataflow program on behalf of a client. A relational query processor typically have four major software components 1) Query Parser, 2) Query Rewriter, 3) Query Optimizer and 4)Plan Executor. The first three components generate a query plan, and the Plan Executor then issues disk

IOs to the storage device, and execute the plan as shown in Figure 1a. In this proposal, we want to modify the Plan Executor such that it adds a side channel to the storage device. Via this side channel, IO-intensive operations can be offloaded to the accelerators in the flash drive, as shown in Figure 1b. As a result of such a in-storage accelerator architecture, 1) the total IOs between the processor and storage are dramatically reduced, and 2) accelerators can utilize the entire aggregate internal NAND flash bandwidth.



(a) Traditional Query processing stack

(b) Query processing stack with In-storage Accelerators

Figure 1: DBMS Query Processing Architectures

In particular, we propose to offload two important IO-intensive SQL functions inside the in-storage accelerators, Aggregate and JOIN.

2.1 Wire-speed Aggregate Accelerator

Aggregate functions are often used with GROUPBY database operations, where rows are grouped based on column values and each group of rows is summarized by applying the aggregate functions. When number of groups are much smaller than total number of rows in a table, storage IO can be greatly reduced by offloading aggregate functions with GROUPBY to the flash drive.

Due to the large bandwidth of the internal flash drive, multiple rows of data can arrive at the same clock cycle. Previous work on aggregate accelerators typically uses a binary tree to aggregate results, in which case only a row can be processed per clock cycle.

We propose a novel wire-speed aggregate accelerator on wide datapath, in a bump-in-the-wire fashion (See Figure 2). This is achieved by first striping the wide input across an array of single-aggregators, each of which can perform aggregations on a pair of elements every cycle. The resulting streams of aggregated elements are merged together using a hierarchy of multi-aggregators, which use sorting network-based high-throughput merge sorters and an sorting-network-like aggregator network that operate on the duplicate keys that may exist between the streams. All components in the accelerator are pipelined, so that high performance can be achieved without requiring significant on-chip buffers. The details of the accelerator work can be found in the submitted FPL paper attached.

2.2 Hash-Join Accelerator

An SQL JOIN operation combines columns from one or more tables in a relational database. A popular implementation of JOIN operations uses the hash join algorithm. JOIN is typically IO-intensive operations because all rows of the joining tables have to be read from the secondary storage.

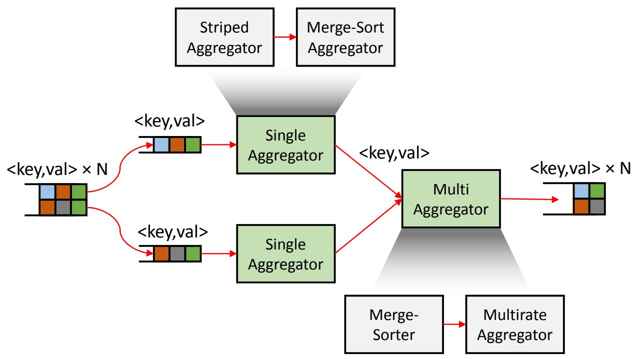
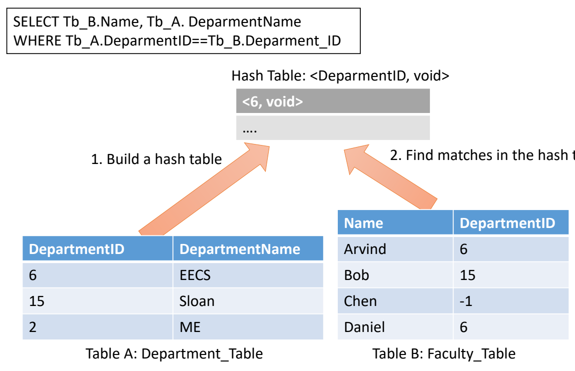


Figure 2: Accelerator architecture overview

As show in Figure 3a, to hash-join Table A and B, a hash table is first constructed with values of the departmentID column in table A. Then the values of the departmentID column in table B are fetched to find a match in the hash table. Such an algorithm requires all the data records to be retrieved from the secondary storage to CPU.

We propose accelerator for hash-join which implements a in-storage bloom-filter in datapath as shown in Figure 3b. When rows of table A is read from flash to CPU for the hash table construction, it constructs a bloom-filter in the datapath. When the rows of table B are streamed from flash to join the hash table, they are filtered with the constructed bloom filter before sending to CPU. If the bloom-filter selectivity is high, i.e., only a relatively few rows need to be joined, such a mechanism can greatly reduced required IO from storage to CPU.


SELECT Tb_B.Name, Tb_A. DeparmentName
WHERE Tb_A.DeparmentID==Tb_B.Deparment_ID

Hash Table: <DeparmentID, void>
<6, void>
....
1. Build a hash table
2. Find matches in the hash table

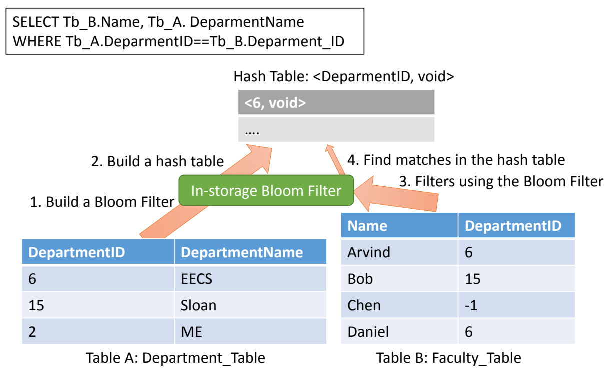
DepartmentID	DepartmentName
6	EECS
15	Sloan
2	ME

Table A: Department_Table

Name	DepartmentID
Arvind	6
Bob	15
Chen	-1
Daniel	6

Table B: Faculty_Table

(a) Hash-join Operations without accelerators


SELECT Tb_B.Name, Tb_A. DeparmentName
WHERE Tb_A.DeparmentID==Tb_B.Deparment_ID

Hash Table: <DeparmentID, void>
<6, void>
....
2. Build a hash table
4. Find matches in the hash table
3. Filters using the Bloom Filter
1. Build a Bloom Filter
In-storage Bloom Filter

DepartmentID	DepartmentName
6	EECS
15	Sloan
2	ME

Table A: Department_Table

Name	DepartmentID
Arvind	6
Bob	15
Chen	-1
Daniel	6

Table B: Faculty_Table

(b) Accelerated Hash-join Operations with In-storage Bloom filter

Figure 3: Hash-join Operations

3 Evaluation Benchmarks

To evaluate the work, we propose to run some synthetic SQL queries with aggregate functions and join, as well as some queries from TPC-H, and TPC-DS. By comparing the same query on the same dataset running on the traditional relational database systems, such as MySQL and Postgres, we can illustrate the performance benefits of the new architecture with accelerators.

Below are examples of SQL queries we plan to evaluate.

1. Query A: Synthetic query for evaluating aggregate accelerator

```
SELECT department, years in company, SUM(salary) as sal sum
FROM employee GROUP BY department, years_in_company
```

2. Query B: Modified Q14 of TPC-H for evaluating hash-join accelerator

   ```
   SELECT sum(l_extendedprice * (1-l_discount)) as promo_revenue
   FROM lineitem, part
   WHERE l_partkey = p_partkey
   and l_shipdate >= 1995-09-01
   and l_shipdate < 1995-10-01
   ```

3. Query C: Q52 of TPC-DS for evaluating complex queries by both accelerators

   ```
   SELECT dt.d_year, item.i_brand_id brand_id, item.i_brand brand,
   SUM(ss_ext_sales_price) ext_price
   FROM date_dim dt, store sales, item
   WHERE dt.d date sk = store sales.ss sold date sk AND
   store sales.ss item sk = item.i item sk AND
   item.i manager id = 1 AND dt.d moy = 11 AND dt.d year = 2000
   GROUP BY dt.d year, item.i brand, item.i brand id
   ORDER BY dt.d year, ext price DESC, brand id
   ```

4 Development Environment Requirement

- Bluespec: HLS Compiler to generate hardware specifcations. bluespec.com/
- Connectal: software/hardware co-design library which enables RPC-like interfaces between host and FPGA. https://github.com/cambridgehackers/connectal
- Vivado: Synthesis and implementation tools for Xilinx FPGAs

5 Milestone

In this project, we need to implement hardware and software components, as well as integrating those system components as a SQL database system. For the hardware component development, I have a fairly clear grasp of the required domain-specific knowledge and expertise. For software component development, especially regarding to the integration with hardware accelerators, help from the experts in the database domain in Samsung will be greatly appreciated.

Also, please note that the entire work of the proposed project might not be able to fit in the 3-month internship period. So continuing collaboration with Samsung after the internship is expected.

The tasks of the internship can be broken into four phases:

1. In the first phase, we will need to set up Bluespec, and the software/hardware codesign library, connectal, on Samsung's development platform. Knowledge and help about the development platform from the Samsung Semiconductor will be needed.
2. In second phase, I will implement the standalone aggregate accelerator. And a hand-coded Query A from Section 3 will be used to evaluate to accelerator.
3. In third phase, I will implement the standalone hash-join accelerator. And a hand-coded Query B from Section 3 will be used to evaluate to accelerator.

4. In final phase, the accelerators needed to be integrated with software query processor. Query C from Section 3 will used to evaluate the overall system. For this phase, help from experts from the software domain will be of great help.

Below is the detailed timeline of the project.

	Weeks (05.2018 – 8.2018)												
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**Post-Intern**
Development Environment Setup	🟦	🟦	🟦										
Aggregate Accelerator Implementation			🟦	🟦	🟦	🟦							
Aggregate Accelerator Evaluation							🟦						
Hash-Join Accelerator Implementation								🟦	🟦	🟦	🟦		
Hash-Join Accelerator Evaluation												🟦	
SW/HW Integration and Evaluation												🟦	🟦

Figure 4: Project Timeline