Query Processing on Smart SSDs

Kwanghyun Park[1], Yang-Suk Kee[2], Jignesh M. Patel[1],
Jaeyoung Do[3], Chanik Park[2], David J. Dewitt[3]
[1]University of Wisconsin – Madison, [2]Samsung Electronics Corp., [3]Microsoft Corp.

Abstract

Data storage devices are getting smarter. Smart flash storage devices (a.k.a. Smart SSDs) are on the horizon and package a small programmable computer inside the device. Thus, users can run code closer to the data right inside the SSD, on the "other" side of the I/O bus. The focus of this paper is on exploring the opportunities and challenges associated with exploiting this functionality of Smart SSDs for relational analytic query processing. We have implemented an initial prototype of Microsoft SQL Server running on a Samsung Smart SSDs. Our results demonstrate that significant performance and energy gains can be achieved by pushing selected query processing components inside the Smart SSD. We also identify various changes that SSD manufacturers can make to increase the benefits of using Smart SSDs for data processing applications, and suggest possible research opportunities for the database community.

1 Introduction

Modern SSDs pack CPU processing and DRAM storage components inside the SSD to carry out the routine functions (such as managing the FTL logic) for the SSD. Thus, there is a small programmable computer inside a SSD device, presenting an interesting opportunity to move computation closer to the storage.

The key reason for moving computation closer to the data is shown in Figure 1. This figure shows the current and projected internal I/O bandwidth that is inside a Samsung Smart SSD, and the I/O bandwidth of the host interfaces (e.g. SAS and SATA standards). The numbers shown in this figure are relative to the I/O interface speed in 2007 (375 MB/s). Data beyond 2012 are internal projections by Samsung. The key trend highlighted in Figure 1 is that the I/O bus standards (such as SATA and PCIe) evolve slower than the speed of the internal network that is used inside the SSD. Without mechanisms to push computation inside the Smart SSD, we are essentially doomed to be "drinking from a narrow straw" when using SSDs for data intensive workloads.

In this paper, we build on our recent work [7] in this area. The focus of this paper is on exploring how analytical database workloads can exploit Smart SSDs. We have built a prototype version of SQL Server that selectively pushes computation to the Smart SSD, and in this paper we describe the details of our implementation. We also present results that show how using the Smart SSD in this way improves not just the performance of some queries, but also reduces the energy that is required to execute the queries. As energy consumption is a critical factor for database appliance and cloud services, using Smart SSDs also provides an interesting opportunity to design energy-efficient data processing systems.

Copyright 2014 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

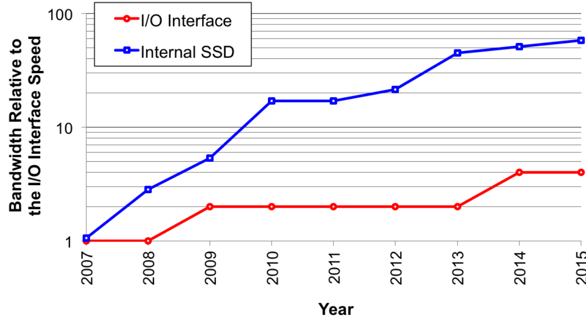

Figure 1: Bandwidth trends for the host I/O interface.

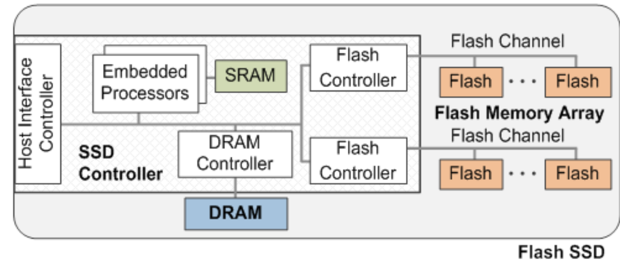

Figure 2: Internal architecture of a modern SSD

On a more subdued note, Smart SSDs are fairly hard to program and use today. In this paper, we also describe the challenges that we had in using the Smart SSDs, and point to potential opportunities to improve the Smart SSDs so that they are easier to use and deploy in the future.

2 Background: SSD Architecture

The SSD controller has four key subcomponents: a host interface controller, an embedded processor(s), a DRAM controller, and flash memory controllers. The host interface controller implements a bus interface protocol such as SATA, SAS, or PCI Express (PCIe). The embedded processors are used to execute the SSD firmware code that runs the host interface protocol, and also runs the Flash Translation Layer (FTL), which maps Logical Block Address (LBA) in the host OS to the Physical Block Address (PBA) in the flash memory. Time-critical data and program code are stored in the SRAM. Today, the processor of choice is typically a low-powered 32-bit RISC processor, like an ARM series processor, which typically has multiple cores. The controller also has on-board DRAM memory that has higher capacity (but also higher access latency) than the SRAM. The flash memory controller is in charge of data transfer between the flash memory and DRAM. Its key functions include running the Error Correction Code (ECC) logic, and the Direct Memory Access (DMA). To obtain higher I/O performance from the flash memory array, the flash controller uses chip-level and channel-level interleaving techniques. All the flash channels share access to the DRAM. Hence, data transfers from the flash channels to the DRAM (via DMA) are serialized. The NAND flash memory array is the persistent storage medium. Each flash chip has multiple blocks, each of which holds multiple pages. The unit of erasure is a block while the read and write operations in the firmware are done at the granularity of pages.

3 API for Query Processing

We have implemented a Smart SSD runtime framework that implements the common functionality that is needed to run user-defined programs in the Smart SSDs. We have developed a simple session-based protocol that is compatible with the standard SATA/SAS interfaces (but could be extended for PCIe). The protocol consists of three commands: OPEN, GET, and CLOSE.

- OPEN, CLOSE: A session starts with an OPEN command and terminates with a CLOSE command. Once the session starts, runtime resources including threads and memory that are required to run a user-defined program are granted, and a unique session id is then returned to the host.
- GET: The host can monitor the status of the program and retrieve results that the program generates via a GET command. This command is mainly designed for the traditional block devices (based on SATA/SAS

interfaces), in which case the storage device is a passive entity and responds only when the host initiates a request.

Using these extensions, we can now invoke specific database operator code that runs inside the Smart SSD. Essentially, the query operation to be performed is passed as parameters to the OPEN call, and corresponding result tuples are fetched into SQL Server using the GET call. Finally, the state information inside the Smart SSD is cleared using the CLOSE call.

4 Evaluation

In this section, we present selected results from an empirical evaluation of Smart SSD with Microsoft SQL Server. For our experiments, we push down scan, simple hash join, and aggregation operations into the Smart SSD. Note that in this paper we largely focus on presenting results that go beyond the initial results, which we presented in [7]. In other words, we refer the interested reader to [7] for additional results.

4.1 Experimental Setup

4.1.1 Workloads

For our experiments, we use the LINEITEM and the PART tables defined in the TPC-H benchmark [3]. Our modifications to the original LINEITEM and PART table specifications are as follows:

1. We use a fixed-length char string for the variable-length column,
2. All decimal numbers are multiplied by 100 and stored as integers,
3. All date values are converted to the number of days since the last epoch.

The LINEITEM and the PART tables are populated at a scale factor of 100. Thus, the LINEITEM table has 600M tuples (∼90GB), and the PART table has 20M tuples (∼3GB).

For our experiments, we also use two synthetic tables called Synthetic64_R and Synthetic64_S. Both these tables have 64 integer columns. The Synthetic64_R table has 1M tuples (∼300MB) and the Synthetic64_S table has 400M tuples (∼120GB). The first column of the Synthetic64_R (R.Col_1) is the primary key, and the second column of the Synthetic64_S (S.Col_2) is the foreign key pointing to R.Col_1.

To insert data into the table, we created a SQL Server heap table (without a clustered index). By default, tuples in these tables are stored in slotted pages using the traditional N-ary Storage Model (NSM). For the Smart SSDs, we also implemented the PAX layout [5] in which all the values of a column are grouped together within a page.

4.1.2 Hardware/Software Setup

All experiments were performed on a system running 64bit Windows 7 with 32GB of DRAM (24 GB of memory is dedicated to the DBMS). The system has two Intel Xeon E5430 2.66GHz quad core processors, each of which has a 32KB L1 cache, and two 6MB L2 caches shared by two cores. For the OS and the transactional log, we used two 7.5K RPM SATA HDDs, respectively. In addition, we used a LSI four-port SATA/SAS 6Gbps HBA (host bus adapter) [1] for the three storage devices that we used in our experiments. These three devices are:

1. A 146GB 10K RPM SAS HDD,
2. A 400GB SAS SSD, and
3. A Smart SSD prototyped on the same SSD as above.

We implemented code for simple selection, aggregation, and selection with join queries, and uploaded that code into the Smart SSD. We also modified some components in SQL Server 2012 [2] to recognize and communicate with the Smart SSD. For each query that is used in this empirical evaluation, we have a special path in SQL Server that we created to communicate with the SSD using the API described in Section 3. Note that the results presented here are from a prototype version of SQL Server that only works on a selected class of queries.

All the results presented here are for cold experiments; i.e., there is no data cached in the buffer pool prior to running each query.

4.2 Experimental Results

To aid the analysis of the results that are presented below, the I/O characteristics of the (regular) SAS SSD and the Smart SSD are shown in Table 2.

	SAS SSD	(internal) Smart SSD
Seq.Read (MB/sec)	550	1,560

Table 2: Maximum sequential read bandwidth with 32-page (256KB) I/Os.

As can be seen in Table 2, the internal sequential read bandwidth of the Smart SSD is 2.8X faster than that of the SSD. This value can be used as the upper bound of the performance gains that this Smart SSD could potentially deliver. As described in Figure 1, over time it is likely that the gap between the SSD and the Smart SSD will grow to a much larger number than 2.8X. We also note that the improvement here (of 2.8X) is far smaller than the gap shown in Figure 1 (about 10X). The reason for this gap is that the access to the DRAM is shared by all the flash channels, and currently in this SSD device, only one channel can be active at a time, which then becomes the bottleneck. One could potentially address this bottleneck by increasing the bandwidth to the DRAM or adding more DRAM buses. As we discuss below, this and other issues must be addressed to realize the full potential of the Smart SSD vision.

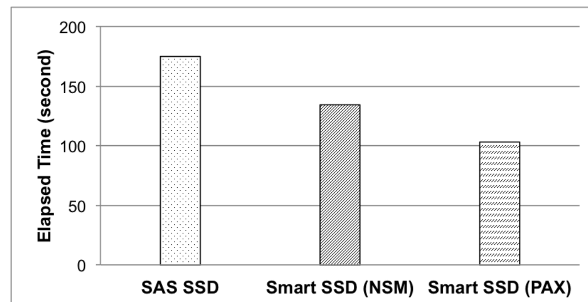

Figure 3: Elapsed time for the TPC-H query 6 on the LINEITEM table (100SF).

4.2.1 Single Table Scan Query: TPC-H Query 6

In this section, we present results for a single table scan query using TPC-H Query 6. (Additional results for single table scan queries for varying tuple sizes, scan selectivities, with and without aggregation can be found

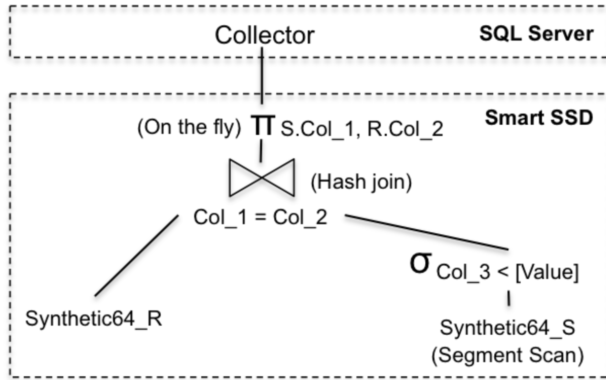

Figure 4: Query plan for the selection with join query in the Smart SSD.

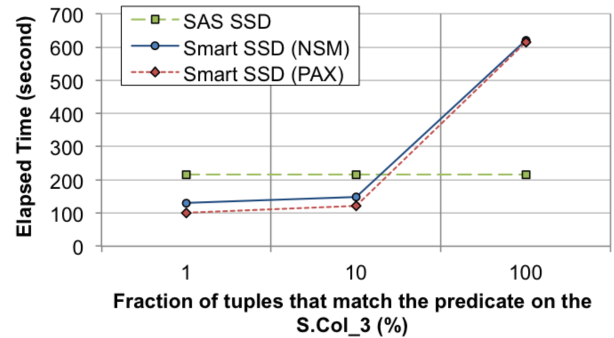

Figure 5: Elapsed time for the join query on the Synthetic64_R and the Synthetic64_S tables at various selectivity factors.

in [7].) This query is [3]:

> **SELECT** SUM(l_extendedprice * l_discount)
> **FROM** LINEITEM
> **WHERE** l_shipdate >= 1994-01-01 AND l_shipdate <1995-01-01
> AND l_discount >0.05 AND l_discount <0.07 AND l_quantity <24

Figure 3 shows the results with the (regular) SSD and the Smart SSD (with the NSM and the PAX layouts). As can be seen in this figure, the Smart SSD with the PAX layout improves overall the query response time by 1.7X over the SSD case. As explained in [7], the number of tuples in a data page, and the selectivity factor have a big impact on the performance improvement that is achieved using the Smart SSD. The selectivity factor (0.6%) of this query is small enough to enjoy the benefits of the Smart SSD. However, its high computation complexity (five predicates, 51 tuples per data page) saturates the CPU and the memory resources in the Smart SSD. As a result, we only achieved 1.7X speedup for this query instead of 2.8X.

4.2.2 Two-way Join Queries

In this section, we evaluate the impact of using the Smart SSD to run simple join queries.

4.2.2.1 Selection with Join Query For this experiment, we use the Synthetic64_R, the Synthetic64_S tables, and the following SQL query:

> **SELECT** S.Col_1, R.Col_2
> **FROM** Synthetic64_R R, Synthetic64_S S
> **WHERE** R.Col_1 = S.Col_2 AND S.Col_3 <[VALUE]

Figure 4 shows the query plan for this query. Since the size of the Synthetic64_R table is far smaller than the Synthetic64_S table, (i.e., |S| = 400|R|), and the hash table for the Synthetic64_R table fits in memory, we used a simple hash join algorithm that builds a hash table on the Synthetic64_R table. As shown in Figure 4, the core computation of this query is carried out inside the Smart SSD and the host only collects the output from the Smart SSD. In the case of the regular SSD, we used the same query plan as the Smart SSD, but the plan was run entirely in the host.

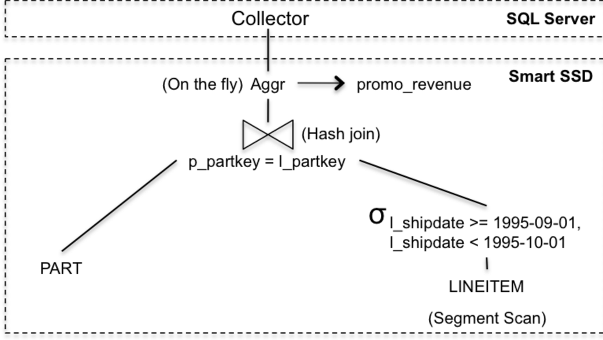

Figure 6: Query Plan for the TPC-H query 14 in the Smart SSD.

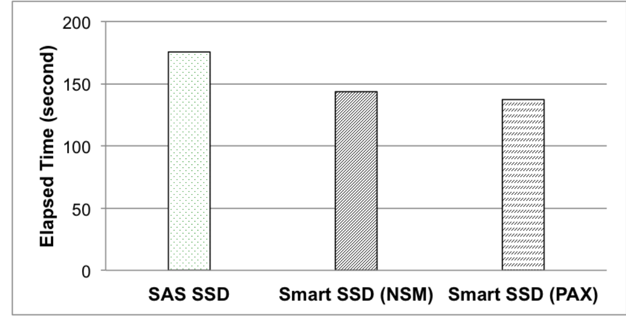

Figure 7: Elapsed time for the TPC-H query 14 on the LINEITEM and the PART table (100SF).

Similar to the previous result, the Smart SSD shows significant performance over the regular SSD case. As seen in Figure 5, the Smart SSD (with the PAX layout) improves performance for the highly selective queries by up to 2.2X over the (regular) SSD case when 1% of the tuples in the Synthetic64_S table satisfy the selection predicate. Similar to the case of single table scan queries, when the selectivity of the query is low (i.e., 100%), the performance of the Smart SSD is saturated because the query cost is dominated by the cost of the high volume of data that has to be transferred between the host and the Smart SSD.

4.2.2.2 TPC-H Query 14 For this experiment, we used the LINEITEM, the PART table, and Query 14 from the TPC-H benchmark [3]. This query is:

SELECT 100 * SUM(case when p_type LIKE 'PROMO%' then l_extendedprice *
(1 - l_discount) else 0 end) / SUM(l_extendedprice * (1 - l_discount)) AS promo_revenue
FROM LINEITEM, PART
WHERE l_partkey = p_partkey AND l_shipdate >= date '1995-09-01'
AND l_shipdate <date '1995-09-01' + interval '1' month

Figure 6 shows the query plan for this query when run in the Smart SSD. Overall, this query plan is the same as the plan that was used in Section 4.2.2.1 (except that the selection was replaced by an aggregation), since the size of the PART table is also much smaller than the LINEITEM table, (i.e., $|\text{LINEITEM}| \simeq 30|\text{PART}|$), and the hash table for the PART table fits in memory.

Figure 7 shows the results with the (regular) SSD, and the Smart SSD (with the NSM and the PAX layouts). The Smart SSD with the PAX layout improves the overall query response time by 1.3X over the (regular) SSD case. As discussed in [7], the Smart SSD achieves greater benefits when the query requires fewer computations (number of CPU cycles) per data page. However, this query requires a large number of CPU cycles to be executed per page compared to the query used in Section 4.2.2.1, which saturates the performance of the Smart SSD. That is why the performance improvement of this query is lower than the selection with join query in Section 4.2.2.1 (1.3X vs. 2.2X).

4.2.3 Energy Savings

In this section, we show the energy consumption profile with the TPC-H Query 6 [3]. Table 3 shows the results with the HDD, the (regular) SSD, and the Smart SSD (with the NSM and the PAX layouts). We note that compared to the Smart SSD case with PAX, the HDD case consumes 11.6X more energy at the whole server/system level, and about 14.3X more energy in just the I/O subsystem. In addition, using the Smart SSD (with PAX) is 1.9X and 1.4X more energy efficient than the regular SSD in the entire system and I/O subsystem

	SAS HDD	SAS SSD	Smart SSD (NSM)	Smart SSD (PAX)
Elapsed time (seconds)	1,186	175	134	103
Entire System Energy (kJ)	430	69	48	37
I/O Subsystem Energy (kJ)	200	19	18	14

Table 3: Energy Consumptions for the TPC-H Query 6

perspectives. The energy gains are likely to be much bigger with more balanced host machines than our test-bed machine. As explained, we observed 11.6X and 1.9X energy gains for the entire system, over the HDD and the SSD, respectively. However, if we only consider the energy consumption over the base idle energy (235W), then these gains become 12.4X and 2.3X over the HDD and the SSD, respectively.

4.3 Discussion

On the DBMS side, the implication of using a Smart SSD for query processing has other ripple effects. One key area is around caching in the buffer pool. If there is a copy of the data in the buffer pool that is more current than the data in the SSD, pushing the query processing to the SDD may not be feasible. Similarly, queries with any updates cannot be processed in the SSD without appropriate coordination with the DBMS transaction manager. If the database is immutable then some of these problems become easier to handle.

In addition, there are other implications for the internals of existing DBMSs, including query optimization. If all or part of the data is already cached in the buffer pool, then pushing the processing to the Smart SSD may not be beneficial (from both the performance and the energy consumption perspectives). In addition, even when processing the query the usual way is less efficient than processing all or part of the query inside the Smart SSD, we may still want to process the query in the host machine as that brings data into the buffer pool that can be used for subsequent queries.

Finally, using a Smart SSD can change the way in which we build database servers/appliances. For example, if the issues outlined above are fixed, and Smart SSDs in the near future have both significantly more processing power and are easier to program, then one could build appliances that have far fewer compute and memory resources in the host server than what typical servers/appliances have today. Thus, pushing the bulk of the processing to Smart SSDs could produce a data processing system that has higher performance and potential a lower energy consumption profile than traditional servers/appliances.

At the extreme end of this spectrum, the host machine could simply be the coordinator that stages computation across an array of Smart SSDs, making the system look like a parallel DBMS with the master node being the host server, and the worker nodes in the parallel system being the Smart SSDs. The Smart SSDs could basically run lightweight isolated SQL engines internally that are globally coordinated by the host node. Of course, the challenges associated with using the Smart SSDs (e.g. buffer pool caching and transactions as outlined above) must be addressed before we can approach this end of the design spectrum.

5 Conclusion

The results in this paper show that Smart SSDs have the potential to play an important role when building high-performance database systems/appliances. Our end-to-end results using SQL Server and a Samsung Smart SSD demonstrated significant performance benefits and a significant reduction in energy consumption for the entire server over a regular SSD. While we acknowledge that these results are preliminary (we only tested a limited class of queries and on only one server configuration), we also feel that there are potential new opportunities for crossing across the traditional hardware and software boundaries with Smart SSDs.

A significant amount of work remains. On the SSD vendor side, the existing tools for development and debugging must be improved if Smart SSDs are to have a bigger impact. We also found that the hardware inside our Smart SSD device is limited, and that the CPU quickly became a bottleneck as the Smart SSD that we used was not designed to run general purpose programs. The next step must be to add in more hardware (CPU, SRAM and DRAM) so that the DBMS code can run more effectively inside the SSD. These enhancements are absolutely crucial to achieve the 10X or more benefit that Smart SSDs have the potential of providing (see Figure 1). The hardware vendors must, however, figure out how much hardware they can add to fit both within their manufacturing budget (Smart SSDs still need to ride the commodity wave) and the associated power budget for each device. On the software side, the DBMS vendors need to carefully weigh the pros-and-cons associated with using smart SSDs. Significant software development and testing time will be needed to fully exploit the functionality offered by Smart SSDs. There are many interesting research and development issues that need to be further explored, including extending the query optimizer to push operations to the Smart SSD, designing algorithms for various operators that work inside the Smart SSD, considering the impact of concurrent queries, examining the impact of running operations inside the Smart SSD on buffer pool management, considering the impact of various storage layout, etc. To make these longer-term investments, DBMS vendors will likely need the hardware vendors to remove the existing roadblocks.

Overall, the computing hardware landscape is changing rapidly and Smart SSDs present an interesting additional new axis for thinking about how to build future high-performance database servers/appliances. Our results indicate that there is a significant potential benefit for database hardware and software vendors to come together to explore this opportunity.

6 Acknowledgements

This research was supported in part by a grant from the Microsoft Jim Gray Systems Lab, and by the National Science Foundation under grant IIS-0963993.

References

[1] Lsi, sas 9211-4i hba. http://www.lsi.com/channel/products/storagecomponents/Pages/LSISAS9211-4i.aspx.

[2] Microsoft sql server 2012. http://www.microsoft.com/sqlserver.

[3] Tpc benchmark h (tpc-h). http://www.tpc.org/tpch.

[4] A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. Technical report, Oracle Corp, 2012.

[5] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *VLDB*, pages 169–180. Morgan Kaufmann, 2001.

[6] P. Francisco. The netezza data appliance architecture: A platform for high performance data warehousing and analytics. In *IBM Redbook*, 2011.

[7] J. Do, Y-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt. Query processing on smart SSDs: opportunities and challenges In *SIGMOD Conference*, pages 1221–1230. ACM, 2011.