Optimizing OLAP Cube Processing on Solid State Drives

Zhibo Chen
University of Houston
Houston, TX 77204, USA

Carlos Ordonez
University of Houston
Houston, TX 77204, USA

ABSTRACT

Hardware technology has improved to a point where a solid state drive (SSD) can read faster than a traditional hard disk drive (HDD). This unique ability to retrieve data quickly combines perfectly with OLAP cube processing. In this paper, we study how to improve performance of OLAP cube processing on SSDs. The main novelty of our work is that we do not alter the internal subsystems of the DBMS. Instead, the DBMS treats the SSD as though it was a regular HDD. We propose optimizations for SQL queries to enhance their performance on SSDs. An experimental evaluation with the TPC-H database compares performance of our optimizations on SSDs and HDDs. We found that even though SSDs have slower write speeds than HDDs, their excellent read speed more than overcomes this limitation.

1. INTRODUCTION

On-Line Analytical Processing (OLAP) involves the efficient evaluation of queries on a multidimensional cube [4, 2]. Cube processing requires scanning large tables, computing aggregations at various granularity levels, filtering rows and writing query results back to the database. Performance ultimately depends on CPU speed, RAM speed and the speed of storage devices. In general, the storage device is the limiting factor for performance. With the amount and diversity of data increasing each year (i.e., big data), there is a need to overcome this bottleneck.

Nowadays most DBMSs uses a group of HDDs that connect directly to the motherboard using an IDE, SATA, or SCSI connection interface. The read and write speeds of HDDs depends on both the speed at which disks spin and the density of stored data. Currently, most HDDs spin at a rate around 7200 RPM, a speed that has not changed much in the past decade. The main issue is that the disks inside a HDD have a practical limit on how fast they can spin before other factors, such as safety and power usage. As such, even with improvements in media density, there has been little improvement in read and write speeds. On the other hand, one of the latest technologies in storage devices, solid state drives (SSDs), have the promise to solve many of the issues faced by today's magnetic hard disk drives (HDDs). SSDs have no mechanical parts and have performance that exceeds or matches that of HDDs. Moreover, SSDs consume less power, which is an issue for data centers. We do not consider the lower limit on the number of writes per block an issue because it improves every year.Also, OLAP cube processing is not heavily dependent on write speed, like transaction processing. Lastg but not least, SSD access and write speeds keep improving each year [5].

We focus on analyzing whether SSDs have improved to a point where it is feasible to perform OLAP cube operations on SSDs instead of HDDs. The novelty of our work comes from the fact that we do not alter the internal architecture of the DBMS, choosing instead to use "as-is". The motivation behind our research lies in observing whether improvements in read and access speeds for SSDs has overcome slow writing speed. We show SSDs are promising for OLAP cube processing because DBMS physical operators are generally read intensive, with a lesser emphasis on writing.

http://dx.doi.org/10.1145/2513190.2513197

2. DEFINITIONS

We compute aggregations a fact table F with n records having d cube dimensions [4], and e measure attributes, where $D = \{D_1, \ldots, D_d\}$ and $E = \{E_1, \ldots, E_e\}$. The lattice is a mathematical structure that represents all subsets of dimensions and their containment with each other. The size of the lattice depends on d and it is computed as $\prod_{1 \leq p \leq d} c_p$, where c_p is the cardinality of dimension p. Data values stored within the nodes inside the lattice areobtained from the measure attributes. We call each subset of dimensions a lattice node, which can be further divided into lattice groups, each of which represents one specific combination of values for those dimensions.

3. CUBE PROCESSING ON SSDS

In this section, we will explain some operations within a database, which features of the storage device they utilize, and provide a comparison between solid state drives and hard drives. We first explain some basic database operations (Insert, Table Scan, Join, and Group-By) that are vital to fast OLAP processing. Then, we will analyze how SSD technology affects OLAP operations, such as slicing and pivoting. Next, we will show an optimization that affects SSDs in a different way than HDs. Then, we will analyze whether these storage drives will affect the execution speed of User-Defined Functions (UDFs) [1, 8]. Finally, we will analyze how SSD technology affects cube exploration operations, such as slicing and pivoting.

We separate OLAP cube operations into two main categories: generation and exploration. In the cube generation phase, entire or partial OLAP cubes are created through ag-

gregations. This phase involves the traversal of the entire or a part of the dimensional lattice. In the cube exploration phase, database operations involve retrieving subgroups or subnodes from cubes that have already been computed.

In terms of the storage drives, we consider read speed to be the speed at which the DBMS is able to retrieve blocks of data from the storage drive. At the same time, write speed involves the speed at which the DBMS is able to write blocks of data to the storage drive. Since we are working on a naive DBMS, we do not assume either sequential or random data blocks.

3.1 Cube Generation with Queries

Since we are only using a DBMS, the generation phase must be entirely composed of SQL queries. One method of accomplish this is by performing a Group-By statement on each combination of the dimensions. These results are then inserted into a final table that holds all computed values of the OLAP cube. Since one aggregation is performed per combination of dimensions, we would need to perform 2^d aggregations followed by an equal number of Insert statements [9]. This method combines the best and worst aspects of a solid state drive. The aggregations involve mostly read speed, which is the best feature of a SSD. However, the insert portion involves many small writes, which is the worst situation for a SSD. When comparing the best with the worst, we find that SSDs still should hold a small advantage over HDs.

3.2 Cube Generation with CUBE Operator

Another approach to cube generation lies with the CUBE operator, a built-in routine that helps calculate OLAP cubes giving the dimensions as input. For CUBE operations, most computations are performed in main memory, which greatly improves the performance of cube generation. However, while the CUBE operator definitely helps performance, it does not utilize the benefits of a solid state drive because of two main reasons: (1) most of the execution time is spent in memory processing. As a result, this portion of the execution is independent of the storage device. (2) the entire operation only requires one table scan and one large write. Comparing SSDs and HDDs, this approach should have similar results on both because the CUBE operator does not rely much on the read/write speeds of the storage device. In addition, the CUBE operator is not available in all commercial DBMSs, making portability problematic. This is an example of a query involving the CUBE operation on three dimensions of the TPC-H Orders table:

```
SELECT O_ORDERSTATUS, O_SHIPPRIORITY, O_ORDERPRIORITY
      ,MAX(O_TOTALPRICE)
FROM TPCH_ORDERS
GROUP BY O_ORDERSTATUS, O_SHIPPRIORITY, O_ORDERPRIORITY
WITH CUBE;
```

3.3 Optimization of Cube Generation

For both cube generation approaches mentioned above, the advantage of a solid state drive are not exploited. In general, any queries involving solid state drives should focus on its ability to access data at a fast rate while avoiding writes as much as possible. Should writes not be avoidable, then it is better to write a large amount of data at once as opposed to writing smaller chunks many times. Thus, our goal for optimizing cube generation is to minimize the number of writes while maximizing the number of reads.

We found that the query operation UNION ALL can help optimize OLAP cube generation in solid state drives by doing exactly what is mentioned before. UNION ALL allows several SELECT statements to be stacked together, and, when combined with an Insert statement, allows for the information from several SELECTs to be grouped and written at the same time. In addition to this, an Insert statement of several Select statements with union is faster than obtaining the same result using several Inserts. This is due to the addition of an overhead for every query statement that is processed by the database. For example, if we compare one hundred Insert statements with one Insert statement containing one hundred unions, then the latter would be faster because it only involves one overhead as opposed to one hundred.

Due to the diminishing of overheads, lattice generation using union statements instead of multiple INSERT-SELECT statements should result in an improvement regardless of the type of storage device that is being used. However, having stated this, this optimization should have a much higher effect on solid state drives than on hard drives. The rationale is that by reducing overhead and collecting multiple small writes into one large write, we are able to diminish the negativity caused by SSDs slow write speed. If we use the union operation to combine all the Select statements of a cube generation, then we have a total of 2^d aggregations with only one Insert statement. While this INSERT will contain many records, a solid state drive is able to handle this situation better than 2^d INSERTs. In addition, UNION ALL is a standard SQL statement that is portable among DBMSs.

3.4 Cube Generation with UDFs

In our previous research, we showed that OLAP operations could be efficiently processed using UDFs [1]. For this paper, we also analyzed the effect of solid state drives on the speed of the UDFs. This situation is not as clear cut as its SQL counterparts. For the algorithm used in the UDF, as the dataset is scanned, the dimensional lattice is held in main memory. Once the dataset has been scanned once, the lattice is complete and then written to the storage device. Since the read speed within a UDF is often very fast, the bottlenecks for the UDF are the main memory speed and the write speed for storing the final results. Which of these two speeds are more important to the final execution times depends on on the number of dimensions in the dataset. If the dimensionality is low, then the main memory speed comprises most of the execution time. This is due to the fact that low dimensionality datasets do not create large dimensional lattices, which means there will not be much data to be written to disk. In this case, the storage devices would not have a large influence on the final execution time since we consider all other computer parts, including main memory, to the same between the SSD and the HD. As the dimensionality increases, the size of the final lattice also increses, causing more data to be written to disk. As a result, when the dimensionality of the dataset is higher (12-15 dimensions), the write speed of the disk becomes more important than the main memory speed. In these situations, the SSD would not perform as well as the HDD because of its slower write speed.

3.5 Cube Exploration

OLAP exploration techniques assists the user in traversing the OLAP cube and extract specific information from a larger set of data. This is one of the perfect situations for solid state drives because these techniques mostly involve many reads with only a few writes. In these sets of operations, we assume that the entire or partial lattice has been computed. Thus, in order to obtain results, the query needs only to return the appropriate tuples.

An example of the an exploration technique is slicing and dicing, which allow for retrieval of certain groups from the OLAP cube. A slicing of the cube in question involves the removal of a dimension while dicing generally involves removing two or more dimensions. When translated to SQL, these can be represented by one or more Select statements with predicates that constraint the amount of data returned. More specifically, the predicates and columns returned reflects the dimensions that are sought by the user. Since we have explained that Select statements are faster in SSD than HD, we can also predict that the solid state drive will also out-perform the hard drive in these exploration techniques.

Additional techniques such as pivoting can also be represented by sets of Select statements. One of the methods involves the use of CASE statements within the SQL query to facilitate the pivoting. Each of these statements would represent one column of the final table. Though this operation is more complex that slicing and requires more writes than a retrieval of groups, the total number of reads still far exceeds the amount of writes. As such the fast read speed of a solid state drive allows this storage device to excel beyond the performance that a normal hard drive can attain.

4. EXPERIMENTAL RESULTS

We conducted our experiments with a commercial DBMS installed in a server with 2.2GHz CPU and 1GB of RAM. In the following subsections, we will present experiments that will compare the performance of a Seagate 160GB magnetic Hard Drive and a Patriot 64GB Solid State Drive.

4.1 Large Input Table

We used a large table to compare the speed of an HDD versus an SSD. Thus our fact table is a pre-processed version of the ORDERS table from the TPC-H benchmark database, created with default parameters (scale factor 1). We used total price attribute as the measure attribute and the remaining columns as dimensions. In addition, we also processed the date attribute in order to increase the number of dimensions. As a result, our final TPC-H ORDERS table had the following characteristics: n=1.5 million, d=9, and e=1. Finally, we also used the TPCH-CUSTOMER table as a referenced dimension table to compute joins.

4.2 OLAP Cube Generation

We first conducted experiments on cube generation using the following three approaches: (1) Multiple Individual Aggregations, (2) CUBE operator, and (3) Merging Aggregations using UNION ALL.

4.2.1 Multiple Aggregation Queries

Table 1 shows the time it takes to create the entire lattice when the number of dimensions is increased for different size datasets. These experiments involve the writing of the final cube into a larger result table. However, notice that the large writes do not seem to affect the SSD as much as in our experiment on Insert statements. This is because the writes that are involved in this experiment involves a few thousand records as opposed to the millions that we had previously tested. We can also observe that HDDs are not as fast as SSDs for all the dimensions that we ran tests on. We can see that solid state drives average over 15% faster than hard drives. The reasoning behind such improvements is that in the generation of a full lattice, the most common operation that was used is aggregations, or Group-By statements. As the dimensionality increases, there are more groups to write to disk, causing the SSD performance to decrease until it is nearly equal to that of the HD.

Table 1: Times for Lattice Generation.

d	HDD n=2M	SSD n=2M	HDD n=4M	SSD n=4M	HDD n=8M	SSD n=8M
1	3.1	2.2	11.6	6.0	14.0	12.3
2	7.9	7.1	24.6	18.9	50.0	38.3
3	17.8	16.9	50.8	44.9	94.2	90.9
4	38.7	37.5	105.1	98.3	202.5	197.5
5	81.9	81.0	216.1	208.6	423.9	415.7
6	172.7	170.5	445.9	436.1	874.9	862.2
7	365.3	361.0	926.8	912.2	1812.2	1785.4
8	804.9	797.6	2046.4	2027.5	3883.1	3829.4

Table 2: Performance of CUBE Varying d.

d	HDD n=2M	SSD n=2M	HDD n=4M	SSD n=4M	HDD n=8M	SSD n=8M
1	2.6	0.9	5.6	1.8	13.5	4.0
2	3.1	2.6	6.5	5.0	13.6	10.1
3	3.6	2.9	7.2	5.9	13.7	11.9
4	4.1	3.4	7.6	6.9	15.2	13.8
5	4.4	4.0	8.2	7.9	16.7	15.8
6	4.8	4.4	9.8	9.4	19.2	18.9
7	5.8	5.4	10.6	10.4	20.9	20.7
8	11.2	11.1	16.7	16.4	27.6	27.3
9	18.4	18.2	24.6	24.1	37.0	36.5

4.2.2 CUBE Operator

We now test lattice generation with a built-in function, CUBE, that, when given a set of dimensions, will calculate all supersets of the dimensions. While this function is restricted to a maximum number of dimensions of 10, we will not be testing up to that limit in our experiments. Instead, we used CUBE to generate the full OLAP cube for both hard drives and solid state drives. We also varied two parameters: the number of records and the number of dimensions. Table 2 shows that solid state drives are faster regardless of an increase in the number of dimensions. This trend also continues when the number of records is varied. The performance gap between the SSD and the HDD is not large because a majority of the execution time depends on CPU and memory speed. In addition, only one table scan and one Insert step is used by this operator. As a result, the SSD

Table 3: Performance of CUBE Varying n.

n	HDD d=1	SSD d=1	HDD d=4	SSD d=4	HDD d=9	SSD d=9
500K	0.9	0.3	1.0	0.9	13.5	13.1
1M	1.8	0.5	1.9	1.8	15.1	14.2
2M	3.0	0.9	3.6	3.4	18.4	17.9
4M	6.3	1.6	7.5	6.9	24.6	24.0
8M	14.2	3.1	14.8	13.7	37.6	36.1

Table 4: Breakdown of UDF Execution Times Varying n. n=1.5M.

n	HDD d=4	SSD d=4	HDD d=9	SSD d=9
1M	1.2	1.1	83.4	83.5
2M	2.2	1.8	84.2	84.2
4M	4.1	3.2	86.8	85.4
8M	7.2	5.9	89.1	88.1

Table 5: Breakdown of UDF Execution Times Varying d. n=1.5M.

d	Read Dataset (HD/SSD) (sec)	Generate Lattice (HD/SSD) (sec)	Write to Disk (HD/SSD) (sec)
2	0.4 / 0.4	0.2 / 0.2	0.6 / 0.6
4	0.7 / 0.7	0.5 / 0.4	1.2 / 1.1
6	5.4 / 5.2	4.9 / 4.6	10.3 / 9.8
8	21.8 / 19.6	19.9 / 21.6	41.7 / 41.2
9	35.9 / 32.1	47.5 / 51.4	83.4 / 83.5

is unable to use its fast read speed to gain an advantage on the HD.

4.2.3 *Merging Aggregations with UNION*

In this section, we are performed a similar set of experiments as in Section 4.2.1. The only difference is that we applied our optimization of using UNION ALL to merge all the small Insert/aggregation statements into one large Insert statement with many aggregations. Figure 1(1) and Figure 1(2) show graphs of the results as we varied both d and n, respectively. We can see that while the SSD and HDD both benefited from this optimization, the SSD was more affected. The reasoning behind this separation lies in the faster access speed of a solid state drive as opposed to a hard drive. By combining all inserts into one, we are lessening the effect of SSDŠs low write speed while enhancing the effects of its faster read speed.

4.2.4 *Cube generation with UDFs*

In this section of the experiments, we show the execution times of processing OLAP queries using UDFs. Figure 2 shows these execution times when varying d and n. The results that we obtained are as expected since the difference between the SSD and HDD are fairly parallel for increasing number of tuples, but their paths cross for increasing dimensionality.

In this section of the experiments, we show the execution times of processing OLAP queries using UDFs. Table 4 shows these execution times when varying d and n. The results that we obtained are as expected since the difference between the SSD and HDD are fairly parallel for increasing number of tuples, but their paths cross for increasing dimensionality. The increasing n does not affect the dimensionality and, thus, does not affect the size of the lattice stored in memory. We can see that for low dimensionalities, the SSD gradually becomes faster than HDD because at such dimensionality, the memory usage and times are not as important as the time to scan the entire table. However, as the dimensionality increases, the time to create and update the lattice begins to become a larger percentage of the overall execution time. As a result, the table scan is not as important, and the two storage drives are virtually parallel in their performance.

Table 6: Times for Slicing from Lattice. d=9. Total number of groups=860K

Query	# Groups	% All	HDD	SSD	% Diff
Q1	154668	18.0%	11.6	4.5	61.2%
Q2	752	0.1%	3.1	0.6	80.6%
Q3	476	0.1%	3.1	0.6	80.6%
Q4	16	0.0%	3.0	0.6	80.0%

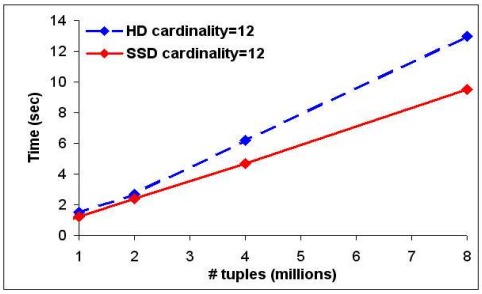


Figure 3: Performance of Pivoting Varying n.

For the increasing dimensionality experiments, Table 5, the trend was less stable. We found that the overall execution times for the SSD and HDD are quite similar to one another. For low dimensionality, the time to generate the lattice overshadows the time to write the results to disk. Due to this situation, the SSD, with its faster read speed, enjoys slightly better performance than the HD, with its slightly faster write speed. However, as dimensionality increases, more and more data are written to disk, causing the write speed to gradually become more important than the read speed.

4.3 OLAP Cube Exploration

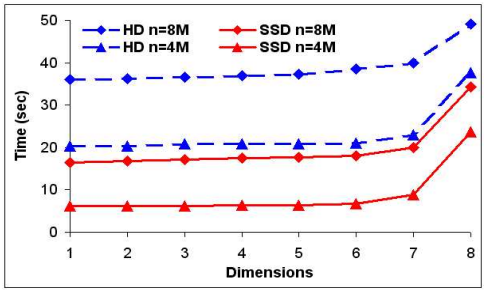
The second portion of OLAP processing involves the exploration and transformation of the computed OLAP cube. In the following two sections, we will compare the performance of solid state drives to hard drives in terms of slicing and dicing and the pivoting of the final results. Note that these experiments assume that the OLAP cube has already been computed and stored in a large result table.

4.3.1 *Slicing and Dicing*

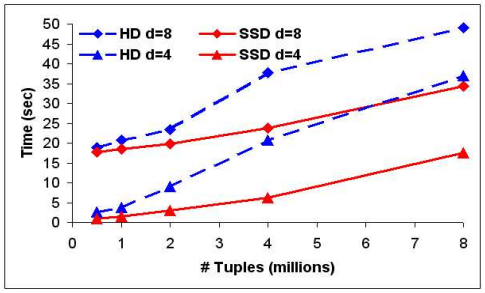
Slicing and dicing involves the retrieval of different sets of dimensions from the OLAP cube. Table 6 shows that solid state drives are much faster than hard drives for the retrieval of slices of the data cube. The reasoning behind these results lies in the face that the SSD holds a large advantage over the HDD when it comes to read speed. Since slicing or dicing can be simplified to a single or multiple Select statements that extract previously calculated values, access speed is more important than write speed. Even if writing to disk is involved, the large advantage in reading time helps the solid state drive consistently stay ahead of the hard drive.

4.3.2 *Pivoting*

Another intricate part of OLAP operations is the pivoting command. With this, the user is able to transform a table by making rows into columns and vice versa. This technique

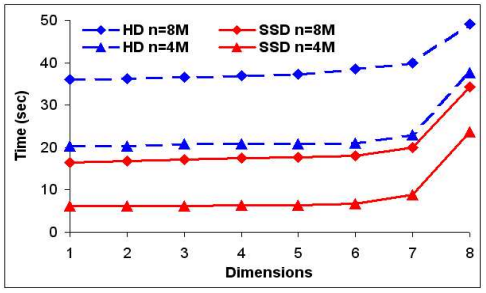


(1) Cube Generation Using UNION Varying d.

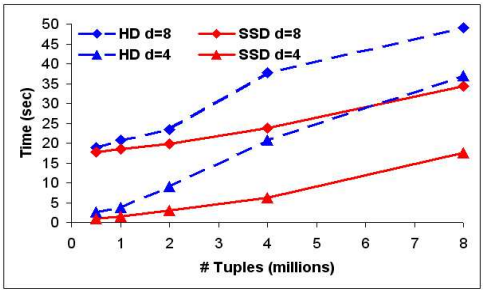


(2) Cube Generation Using UNION Varying n.

Figure 1: Performance of OLAP with UNION ALL.



(1) Cube Generation Using UDFs Varying d.



(2) Cube Generation Using UDFs Varying n.

Figure 2: Performance of OLAP using UDFs.

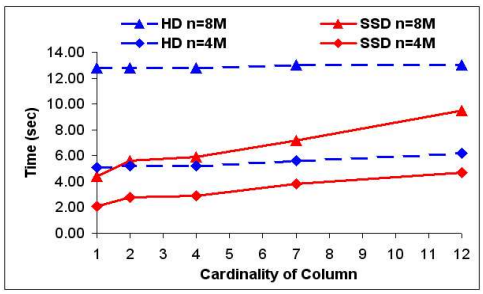


Figure 4: Performance of Pivoting Varying Cardinality.

is somewhat in the middle of retrieval and full cube computation when it comes to performance. A pivot does not lean heavily towards writes or reads. Instead, pivot requires a combination of the two in order to obtain the results. We conducted experiments that varied both in the n and in the cardinality of the column to be pivoted. Figure 3 and Figure 4 shows that solid state drives performed better than hard drives in both situations. These results can be explained by the fact that the speed of pivoting is based more on data access speed and memory speed than writing, which occurs only at the end. Since memory speed is the same for both SSD and HD, the main difference falls on access speed, which is dominated by the solid state drive.

4.4 Discussion

The purpose of this paper is to answer the question of whether it is beneficial to perform OLAP processing on a solid state drive. Table 7 provides a summary of some of the experiments from this paper. If we only observe the performance aspects of the comparison, then it is definitely worth it to switch out traditional hard drives for solid state drives with regard to the faster OLAP query execution. Our experiments have shown that for a majority of the OLAP operations, the SSD performed far better than the HD, especially when optimization is applied. Additionally, even though SSDs have a limited number of writes, they are still far more stable than traditional hard drives. If a SSD is dropped onto the floor, the user can be sure that no parts had been jolted around.

On the other hand, when we consider the costs of purchasing a solid state drive as opposed to a magnetic hard drive, there is a large different. The average SSD often costs nearly ten times more than a similar HD. The question becomes one of whether the improvements in performance can offset the additional costs of switching to solid state drives. While the answer is quite hazy and depends largely on the financial status of the organization, the future should bring a more concrete answer. As we observe the trends for SSDs, we can see that, in the past few years, these storage drives have been rapidly dropping in price. As with many other technological devices, the longer we wait, the less expensive the device. Additionally, the read and write speeds of solid state drives are continuing to increase at an impressive rate. As a result, it is not far fetched to believe that within a few years, solid state drive technology will have developed to a point where they can outperform hard drives on both read and write speeds. If we couple this with lowering costs, then in the near future, we know that the benefits of the solid

state drive might overcome the shrinking cost difference between these two types of storage devices.

5. RELATED WORK

For this paper, we analyze the feasibility of performing OLAP operations using solid state drives. To the best of our knowledge, no one has conducted such an analysis. The authors of [7] consider flash drives to have a large potential to be used in database servers. Nevertheless, recent work regarding solid state technology and the DBMS has focused on altering the basic architecture or algorithms of the database. A 3-level memory system is proposed using flash as the middle layer in [3]. The authors in [10] propose a new join algorithm to be used in a PAX-based system. Similarly, [7] proposes changing the current database algorithms to better utilize solid state drives. Also, in [6], the author looks into the possibility of using SSD for enterprise DBMSs. However, the authors only looked at how the SSD would assist in low-level tasks, such as transaction logging, and not at how it would affect the actual SQL performance. In contract, our research is conducted on a commercial database and looks into utilizing tools that are already present, without changing any algorithms.

On the OLAP side, little research has been published regarding the improvement of performance when using solid state drives. The focus has been in developing new data structures or pushing more calculations into main memory. For example, the authors of [11] proposes the use of a condensed version of an OLAP cube that utilizes prefix and suffix redundancy reduction to generate smaller lattices. On the main memory side, our previous work in [1] uses user-defined functions to improve the performance of cube generation. While this paper has the same goals of improving performance, we analyze it from the point of altering hardware instead of algorithms.

6. CONCLUSIONS

In this paper, we showed that the SSD is able to perform at or above that of the HDD for complex OLAP operations such as cube generation, slicing, and pivoting. We showed that in cases where the number of reads and writes are similar to each other, the SSD is not able to outperform the HD by as much as other situations. However, we also showed that there are methods by which one can tip the balance in favor of SSD by minimizing small random writes and maximizing reads. Both storage devices performed equally well in terms of lattice generation without optimizations, but once the optimizations were applied, the SSD was much better than the HD. We also observed that solid state drives do not have a large impact on user-defined functions. However, an interesting situation was discovered in which the execution time of a fully optimized lattice generation performed in a SSD with SQL was actually faster than performing the same generation using a UDF. Finally, we provided a discussion on the benefits of using solid state drives for OLAP operations as opposed to traditional magnetic hard drives. We concluded that at the current time, the decision for which storage device is better becomes more of a business decision than a computer science decision.

For future work, we want to study other kinds of solid state drives. We want to conduct a more thorough analysis of storage strategies and compare their impact on performance. We also want to explore more SQL query optimizations that take advantage of how SSDs read and write data blocks. Finally, we want to repeat our experimental analysis on other DBMSs to verify our findings and understand general trends.

7. REFERENCES

[1] Z. Chen and C. Ordonez. Efficient OLAP with UDFs. In *Proc. ACM DOLAP Workshop*, pages 41–48, 2008.

[2] C. Garcia-Alvarado, Z. Chen, and C. Ordonez. OLAP with UDFs in digital libraries. In *Proc. ACM CIKM Conference*, pages 2073–2074, 2009.

[3] G. Graefe. The five-minute rule twenty years later, and how flash memory changes the rule. In *Proc. ACM DaMoN Workshop*, 2007.

[4] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-total. In *ICDE Conference*, pages 152–159, 1996.

[5] C. Hwang. Nanotechnology enables a new memory growth model. *Proceedings of the IEEE*, 91(11):1765–1771, 2003.

[6] S. Lee, B. Moon, C. Park, J. Kim, and S. Kim. A case for flash memory SSD in enterprise database applications. In *Proc. ACM SIGMOD Conference*, pages 1075–1086, 2008.

[7] S.W. Lee and B. Moon. Design of flash-based DBMS: an in-page logging approach. In *Proc. ACM SIGMOD Conference*, pages 55–66. ACM, 2007.

[8] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[9] C. Ordonez and Z. Chen. Evaluating statistical tests on OLAP cubes to compare degree of disease. *IEEE Transactions on Information Technology in Biomedicine (TITB)*, 13(5):756–765, 2009.

[10] M. Shah, S. Harizopoulos, J. Wiener, and G. Graefe. Fast scans and joins using flash drives. In *Proc. ACM DaMoN Workshop*, 2008.

[11] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: shrinking the petacube. In *ACM SIGMOD Conference*, pages 464–475, 2002.