The Constraints of Magnetic versus Flash Disk Capabilities in Big Data Analysis

Nathan Regola, David A. Cieslak and Nitesh V. Chawla
Interdisciplinary Center for Network Science and Applications (iCeNSA)
and
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana
{nregola,dcieslak,nchawla}@nd.edu

ABSTRACT

Solid state disks (or flash disks) are decreasing in cost per gigabyte and are being incorporated into many appliances, such as the Oracle Database Appliance [8]. Databases–and more specifically data warehouses–are often utilized to support large scale data analysis and decision support systems. Decision makers prefer information in real time. Traditional storage systems that are based on magnetic disks achieve high performance by utilizing many disks for parallel operations in RAID arrays. However, this performance is only possible if requests represent a reasonable fraction of the RAID stripe size, or I/O transactions will suffer from high overhead. Solid state disks have the potential to increase the speed of data retrieval for mission critical workloads that require real time applications, such as analytic dashboards. However, solid state disks behave differently than magnetic hard disks due to the limitations of rewriting NAND flash based blocks. Therefore, this work presents benchmark results for a modern relational database that stores data on solid state disks, and contrasts this performance to a ten disk RAID 10 array, a traditional storage design for high performance database data blocks. The preliminary results show that a single solid state disk is able to outperform the array for queries summarizing a data set for a variety of OLAP cube dimensions. Future work will explore the low level database performance in more detail.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Performance attributes*

General Terms

Measurement, Performance

Keywords

Solid State Disk, Query Performance

1. INTRODUCTION

Decision support systems have the ability to revolutionize business operations as an increasing volume of data is collected and analyzed. Organizations are increasingly building large data warehouses. Often, a major application for data warehouses is a dashboard that provides key summary metrics to decision makers. For example, the publicly available data set that we utilized would allow a consumer or government official to determine the number of aircraft delays at major airports by airline or airport over a given time period. Based on our experience with analytics, it is assumed that the dashboard should return up-to-date data and calculate the metrics in near real time.

For most large data warehouses, it is also assumed that the amount of memory is smaller than the data. It is also likely that all data warehouse indexes will not fit in memory for a multi-terabyte data warehouse. Therefore, frequent access to the storage system will be necessary to retrieve the data that will be utilized to build a dashboard view. Typically, relational databases with indexes configured for data implies that there will be random I/O as the database accesses the indexes and then accesses the relevant database blocks referenced in the index (unless all relevant attributes are contained in the index). For a storage system, these patterns will likely translate to random accesses (unless the data is stored in sorted order and the data is only accessed in this specific order).

Traditional storage systems that are based on magnetic disks have very high random access times. High random access times are problematic because database queries require the database engine to access data on multiple tracks. While database indexes prevent the database engine from reading the entire database table, the database engine still must access pieces of data that are randomly spread across the magnetic disks (unless the data is sorted in the order that it is read sequentially). Solid state disks are promising replacements for magnetic disks in large databases since they require less time to perform random reads and random writes. However, the algorithms that are implemented in solid state disk firmware are significantly more complex (from a predictable performance perspective) than magnetic disk firmware. Solid state disk firmware is complex because each block of a solid state disk has a limited lifetime (it can only be erased and rewritten a finite number of times). Additionally, the maximum throughput is obtained by reading data in parallel from multiple flash dies or planes. The goal of the flash firmware is to optimize this trade-off. Each manufacturer is obviously free to optimize their solid state disk products according to their own requirements. For example, each flash disk contains several dies containing the flash packages, logic for the host interface, and a controller. The controller generally contains several items–RAM (for buffers), a processor, and logic to mediate access to the communication buses (which connect to the flash packages). The specific details of how to build the flash disk including how many dies to utilize, their sizes, and the bus architecture to connect to the dies could differ by product. Additionally, the algorithmic trade-offs implemented in the software can be manufacturer and product dependent. Consumers have a wide variety of devices to choose from. However, their performance is dependent on the workload (due to the complexity of the solid state disks, which is discussed later). Obviously, benchmarking many drives with many workloads is expensive from a time and material standpoint. Additionally, the performance of the drives can change over time, as the solid state drive begins to wear (blocks become unusable after their rewrite count is exceeded). This paper utilizes a solid state drive to demonstrate a benchmarking process that we believe will be useful to readers to determine the suitability of a given solid state drive for an analytic dashboard workload. Broadly speaking, this work demonstrates the feasibility of solid state drives to improve the response time of analytic dashboards. The paper presents benchmark results from a solid state drive and a traditional RAID 10 array and demonstrates that the flash disk is able to outperform a ten disk RAID 10 array of modern 7200 RPM enterprise SATA disks for a variety of OLAP data retrieval tasks.

2. BACKGROUND

The basic concept of a magnetic hard disk is likely familiar to every computer science student–the disk consists of at least one platter, a motor, and a movable disk arm that is capable of accessing any "track" of the disk by repositioning itself. As the disk platter(s) rotate, the disk arm can read and write data from the sectors on a given track. The seek time is the amount of time that is necessary for the disk arm to move from track to track (moves to close tracks would be faster than moves to distant tracks).

Solid state disks are composed of flash packages. A flash package contains at least one chip known as a die, but often contains multiple dies. Dies contain multiple blocks, organized as *planes*. Blocks are composed of pages. Pages are the smallest unit of storage that solid state disks algorithms operate on.

Solid state disks have several properties that increase the complexity of their firmware and explain why solid state disk performance is workload dependent. First, writes can only occur to fixed size blocks (composed internally of pages) and rewriting is not possible at the page level. If the operating system wants to modify a 4K file system block, which we will assume for this example maps to a single page, the flash disk must rewrite the entire flash disk block. Secondly, each page has a limited rewrite lifetime. Thirdly, high performance (where high performance is defined to be higher throughput than a magnetic disk) is typically only achieved when multiple pages are accessed in parallel. Therefore, the algorithms must attempt to place blocks sequentially when writing across multiple planes and/or dies if the data is to be later read in parallel.

The algorithms in a solid state disk must track the free blocks (and pages within those blocks) to determine where new writes can occur to optimize the performance of future parallel reads and limit the affects of wear-leveling. The architecture of the solid state disk has a direct impact on the performance of a database system that is built upon the disks.

3. METHOD

Based on the knowledge that solid state disks and traditional hard disks perform differently, this work explores how an OLAP workload performs on both types of storage systems. First, the file system level performance of each system was determined to establish the maximum possible throughput. Then, we utilized database level queries to determine how this workload performed on each storage system.

3.1 Experiment Setup

The experiments were performed on a custom built Supermicro server in a chassis that supports twenty four 3.5" disks. The system was designed to support a large data warehouse and attempted to maximize I/O performance where feasible. For example, the system has two storage controllers, each capable of using eight lanes of PCI-Express bandwidth to the chipset. The motherboard has two Intel 5520 chipsets, each capable of supporting thirty two PCI-Express lanes. The system supports a combined sixty four PCI-Express lanes to the chipsets and the CPUs. Two controllers are necessary, because each disk is capable of connecting at 2.4Gb/s or 300MB/s after overhead from 8b/10b encoding is considered (on a 3Gb/s link). Assuming that the system is eventually converted to be fully populated by solid state disks, the maximum transfer rate for twenty four disks, 7200MB/s, would overwhelm a single PCI-Express 2.0 x8 connection (4000MB/s), assuming a unidirectional transfer of data to or from the disks.

The storage systems that were benchmarked consist of a single Crucial M4 256GB solid state drive (firmware version 0309) for solid state tests and a RAID 10 array composed of ten disks to represent a traditional array of modern SATA hard drives. RAID 10 is the fastest type of RAID (including nested RAID levels). Redhat Enterprise Linux 6.2 (x86_64) kernel version 2.6.32-220.4.2.el6.x86_64 was used since it supports the "discard" option for the ext4 filesystem to inform solid state disks of freed blocks.

The dataset utilized for this work is a publicly available dataset from the Bureau of Transportation Statistics at the U.S. Department of Transportation, named "On-Time Performance" [2]. This dataset consists of records of each scheduled flight for seventeen United States based airlines spanning approximately twenty years. The data contains the origin airport and destination airport of each flight, along with the length of flight, length of delay, type of delay (if delayed) and various other fields surrounding on time arrivals of scheduled air service. Scripts to download the data, create a database table, and create indexes are available at our website[1]. We utilized a ten year period of data, repre-

[1]`http://www.nd.edu/~dial/data/ontime.html`

senting approximately 50 GB, for testing. A hypothetical use case of such data may be an airport manager monitoring the performance of flights originating from their airport, and comparing the results to airports of a similar passenger volume and city size.

3.2 I/O Performance

IOzone was utilized to determine the ideal block size of operations on the solid state disk (with the ext4 file system). The server was restarted between each IOzone [5] run, and the kernel was limited to 1536MB of RAM during testing to limit the affects of file system caching. With this information, it allows us to determine the ideal block size for database I/O. Based on the random read performance, it appears that we must attempt to utilize a block size that is larger than 4KB. If the database block size is too large, and spans too many rows, we will retrieve rows that may not be needed (if queries do not access all rows). The random write throughput of a 32KB block is close to the maximum write throughput. Database block sizes were tested at 8 KB and 32 KB, and the results were quite similar. The standard database block size is 8 KB, and so this was ultimately selected for database testing. The result of IOzone benchmarking are presented in Figure 1. The RAID 10 array is able to achieve impressive sequential read and write performance compared to one solid state disk. However, the random read and random write throughput of the RAID 10 array lags behind that of the solid state disk for almost all sizes of operations.

Item	Quan.
Supermicro X8DTH-6F Moteherboard	1
Intel X5650 2.66GHz CPU QPI=6.4GT/s	2
Adaptec 51245 SATA/SAS RAID Controller	2
Western Digital WD2003FYYS 2TB disk	12
Western Digital WD1003FBYX 1TB disk	12
Kingston 8GB 1333MHz DDR3 ECC Reg CL9	6
Crucial M4 CT256M4SSD2 Solid State Disk	1

Table 1: Hardware Specifications

File System	RAID Type	Size (usable)	Controller
/boot	10	100MB	1
/	10	1TB	1
/archive	10	1894GB	1
/home	6	4TB	1
swap	6	96GB	1
/data	10	9TB	2
/ssd	N/A	256GB	3Gbps

Table 2: File system layout

3.3 Database Performance

Dashboard applications are often visualization tools intended to view and interact with a large volume of data from data warehouses. The number of dimensions in an OLAP cube is one metric of describing the complexity of an OLAP dataset. To facilitate testing of OLAP data retrieval tasks, the latest version of a relational database was utilized for benchmarking. The benchmarking occurred in a separate database instance that was setup for the test data (other database instances were not running during testing). A separate tablespace was created for the test schema, and the only data contained in this tablespace was data for the test table in the test schema. Between each test of table loading, the tablespace was "dropped" from the database and recreated from a script and the database was restarted.

The prototype dashboard utilized a series of queries that would be appropriate for the dataset utilized (i.e. determining the number of delays between each city pair in the database). Sixty-nine possible queries that extract various cubes of data were created and executed against the database in order to benchmark the database performance when data was stored on a solid state disk and a traditional disk array. The queries were constructed programmatically. Additionally, indexes for each query were constructed so that each query had an index available that covered its specific set of attributes. For example, if a few basic indexes were constructed to cover the first dimension attributes these indexes would optimize the performance of the first dimension queries. While indexing the first dimension would aid the performance of the second dimension queries, it would do so disproportionately since 100% of the first dimension's attributes would be indexed, but only a fraction of the second dimension's attributes would be indexed. Therefore, to show results that are consistent across dimensions in terms of "benefit from indexes", we elected to create indexes for each query. In a production environment, based on the frequency of use of various indexes, one may decide to forgo the significant space utilization required to support such an indexing scheme.

The queries were executed and the time was logged on a per query basis, and the results presented. The server's RAM was artificially limited through the Linux kernel at boot time to limit the effects of file system caching[2]. The database engine RAM consumption was fixed at approximately 980MB (out of 1536 MB total available to the kernel during testing).

4. RESULTS

The first step in establishing the performance capability of the flash based database system is to estimate the performance of the storage system to aid the selection of database block size and ideal sizes of file system operations. IOzone was utilized to determine the optimal block size for various file system operations. The results are presented in Figure 1. The 9TB ten disk RAID 10 array is able to achieve impressive sequential speeds for read (r) and write (w) at approximately 960MB/s and 540MB/s, respectively for operations over 4KB/s. However, the random read (rr) and random write (rw) performance is significantly lower, especially for operations under 8MB and 256KB, respectively. Comparing these results with the solid state disk, it is noteworthy that the solid state disk is able to achieve near peak random write performance (for our 3Gbps interface) at approximately 4KB and its random read performance is over 188MB/s at approximately 4KB operations, contrasted with 0.79MB/s for the RAID 10 array. Based on these results, the solid state disk should support faster queries when utilizing a subset of the data (as we would expect the workload to be partially composed of random reads when accessing

[2]The server has 48GB of RAM, as noted in Table 1, and therefore would be capable of caching over 95% of the database in memory if the operating system's RAM was not limited.

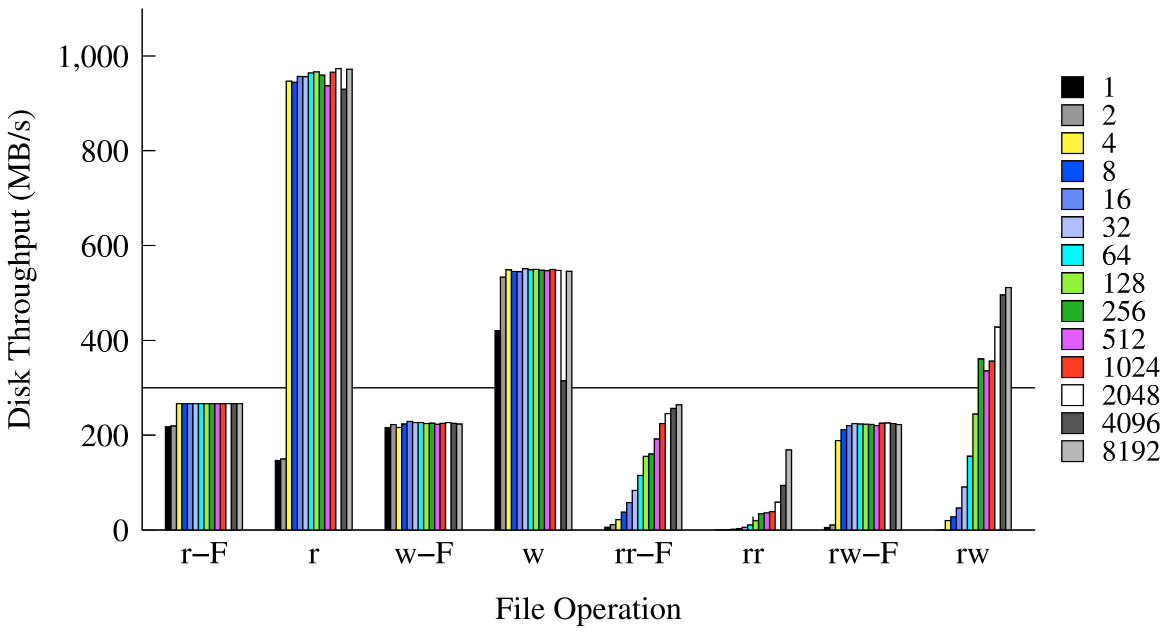


Figure 1: Disk throughput, in MB/s, of read, random read, write, and random write operations from IOzone. The legend denotes the block size of records in IOzone, in KB. The horizontal line at 300MB/s denotes the maximum possible transfer speed of the 3.0Gb/s interface for the solid state disk. Bars within each group are ordered by block size (from smallest to largest).

	0	1	2	3
Airline	48	3714.3	112469.6	513922.0
Dest	677	20260.5	205416.4	513922.0
Month	280	39301.3	372688.7	897187.0
Origin	664	20443.5	207013.0	513922.0
Year	24	4275.0	50112.7	130657.0

Table 3: The number of instances per dimension for diverted flights. For dimensions greater than 0, the average number of instances for that dimension is presented.

	0	1	2	3
Airline	18895	515278.3	6339232.2	20444383.5
Dest	113745	1674987.8	10014380.4	20444383.5
Month	84418	2941503.7	16888380.3	32454745.0
Origin	127367	1677018.3	9956825.0	20444383.5
Year	13091	525199.3	3845519.3	8434022.0

Table 5: The number of instances per dimension for delayed flights. For dimensions greater than 0, the average number of instances for that dimension is presented.

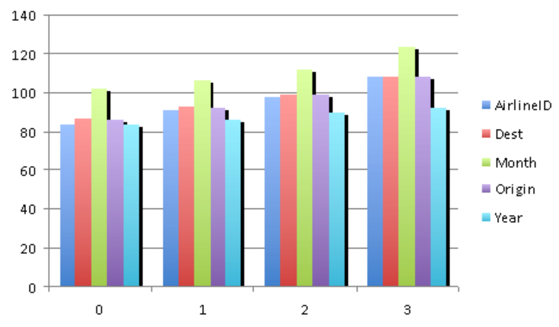
database indexes and a portion of the data, assuming the database engine does not decide to perform a full table scan).

	0	1	2	3
Airline	48	3841	141526.2	650070.0
Dest	687	23645.75	257684.0	650070.0
Month	280	47793.3	478212.7	1149864.0
Origin	684	23909.8	259708.8	650070.0
Year	24	4500.0	54923.0	150276.0

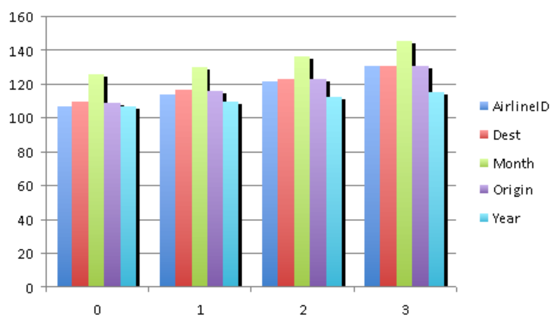
Table 4: The number of instances per dimension for canceled flights. For dimensions greater than 0, the average number of instances for that dimension is presented.

Figure 2 presents the results for our timing experiments. The enumeration of the series represents the average timing for the dimension in question across the listed number of additional dimensions. Thus, Series 0 represents the timing for each dimension independently (i.e. the bar for Airline represents the timing of Airline alone). Series 1 represents the average timing of the dimension in combination with all other dimensions (i.e. the bar for Airline is the average timing of Airline-Dest, Airline-Month, Airline-Orig, and Airline-Year). Series 2 is the average timing for each dimension in combination with two other dimensions (i.e. Airline-Dest-Month, Airline-Dest-Orig, Airline-Dest-Year, Airline-Month-Orig, Airline-Orig-Year). We hasten to note that since Month and Year contain redundant information, these experiments never consider their combination in conjunction.

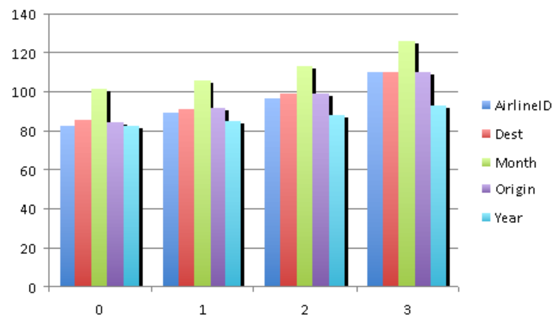
The same interpretation that is used to describe the dimensions in Figure 2 applies to Tables 3, 4 and 5. The number of instances in each dimension demonstrates the scale of results returned in each dimension for a given class of queries. For example, there are far more delays (see Figure 5) than cancellations (see Figure 4) or diversions (see Figure 3). This data, when coupled with the timing data, allows a trend to be observed. The smaller query classes, such as diversions and cancellations, only require, on average, 22% and 25.6%, respectively, more time to retrieve a three dimensional cube than a zero dimensional cube for the solid state disk. The RAID 10 array, which require more time to



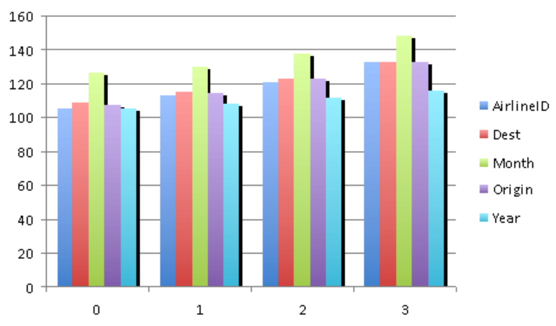
(a) Flight diversions - solid state disk



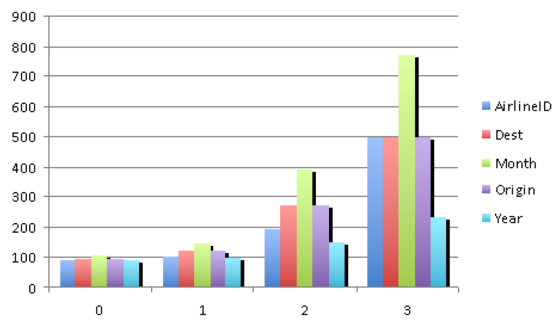
(b) Flight diversions - RAID 10 array



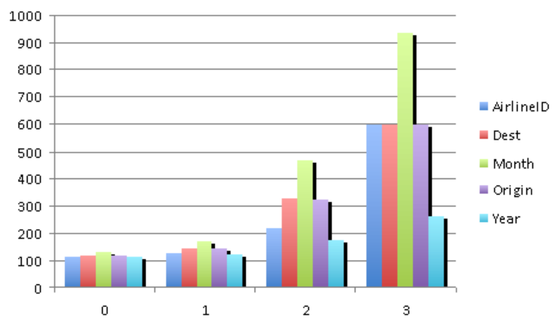
(c) Flight cancellations - solid state disk



(d) Flight cancellations - RAID 10 array



(e) Flight delays - solid state disk



(f) Flight delays - RAID 10 array

Figure 2: The average timing results, in seconds, for queries in various dimensions for flight diversions, cancellations, and delays. The results for solid state disks and the traditional RAID 10 array are presented on a single row for comparative purposes. The number on the x-axis for the groupings indicate the number of partner dimensions.

complete the zero dimensional queries, requires 17.4% and 19.6% more time to retreive the diversion and cancellaton three dimensional cube, respectively, over the zero dimensional cube. However, with a large data volume, the delay cube presents an interesting result. The time required for the three dimensional delay cube retrieval is 422.5% and 399.75% more for the solid state disk, and RAID 10 array, respectively. The singular solid state disk is still faster than the RAID 10 array for the large data retrieval. This suggests that moderate random read activity is involved since the flash disk has lower sequential read performance than the ten disk array and would not be able to outperform the ten disk array at a pure sequential read workload.

5. RELATED WORK

Related work consists of several areas such as database performance and solid state disk benchmarking for databases. Database performance has been studied since relational databases began to be used for mainstream business applications. For example, Christodoulakis' 1984 paper [3] discussed the assumptions that were made at the time to model database performance. Hsu et. al. [4] presents results of trace workloads from ten large global companies. They conclude that the dynamic nature of real world workloads is different from the static nature of TPC-C [9] benchmarks. They note that optimizing for TPC-C performance could have less than optimal performance on real world data sets since the TPC-C benchmarks do not reward dynamic algorithms in the database engine.

Solid state disks are revolutionary to the storage community since they can potentially improve the bandwidth from storage systems. Solid state disks improve the bandwidth from storage systems by reducing the time necessary for random reads and random writes. Random operations are important because they often make up a significant portion of modern workloads, such as database operations and have a direct impact on the scalability of such systems. Consequently, they have been studied by researchers for some time. Agrawal, et. al. [1] present an overview of the key properties of NAND solid state disk design and the tradeoffs that must be made by the solid state disk designers and incorporated into the algorithms. They also present a thorough overview of NAND disk design. Lee, et. al. [7] presented benchmark results for a commercial database. They benchmarked SQL transactions, utilized the TPC-B benchmark suite, studied MVCC rollback, and the affects of sorting and joins on temporary table spaces. A later paper by many of the same authors [6] updated their original study and includes a comparison of several types of solid state drives (two personal solid state disks and one enterprise disk). Their work demonstrates that a single SSD can outperform an eight drive RAID-0 array composed of 15K magnetic disks for an OLTP workload. Our paper explores the suitability of solid state disks for OLAP systems, and specifically, an application that relies on an OLAP–an analytic dashboard.

6. CONCLUSION

Solid state disks based upon NAND flash technology have unique constraints regarding write activity. However, they often exhibit high random read performance at a variety of block sizes. Traditional hard disks are able to achieve high sequential throughput. However, many database operations require random read and random write performance. Databases are increasingly utilized to support analytic dashboards and other data warehousing tasks where response time is a paramount concern. This paper presents results of IOzone file system benchmarks for a traditional RAID 10 array and a solid state disk. While the RAID 10 array exhibits impressive sequential throughput results, a single solid state disk can outperform this ten disk array in a variety of OLAP cube retrieval operations spanning simple cubes to more complex multi-dimensional cubes.

7. REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[2] Bureau of Transportation Statistics, Research and Innovative Technology Administration (RITA), U.S. Department of Transportation. Airline On-Time Performance Data.

[3] S. Christodoulakis. Implications of certain assumptions in database performance evauation. *ACM Trans. Database Syst.*, 9(2):163–186, June 1984.

[4] W. W. Hsu, A. J. Smith, and H. C. Young. I/o reference behavior of production database workloads and the tpc benchmarks – an analysis at the logical level. *ACM Trans. Database Syst.*, 26(1):96–143, March 2001.

[5] IOzone Filesystem Benchmark. http://www.iozone.org.

[6] S.-W. Lee, B. Moon, and C. Park. Advances in Flash Memory SSD Technology for Enterprise Database Applications. In *Proceedings of the 35th SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 863–870, New York, NY, USA, 2009. ACM.

[7] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A Case for Flash Memory SSD in Enterprise Database Applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1075–1086, New York, NY, USA, 2008. ACM.

[8] Oracle Corporation. Oracle Database Appliance.

[9] Transaction Processing Performance Council. TPC-C - Homepage, March 2012. Revision 3.3.2.