Impact of Solid-State Disk on High-Transaction Rate Databases

February 2005

By Solid Data Systems, Inc.

SolidData®

Contents

Executive Summary

Overview

Solid State Disks (SSDs) offer Companies *lower* costs per transaction in high-transaction rate database systems. SSDs decrease the number of systems required and lower overall hardware, licensing and maintenance costs, allowing each Database Server to achieve many more transactions per second. The reduction in the number of servers and disks increases system reliability and decreases maintenance and administration. In emerging countries where wireless and prepaid applications are proliferating, SSDs have proven to be extremely reliable storage for the most critical data. When Databases are less than 100 gigabytes, the increased density and lower costs of semiconductor memory now make it cost effective to store the entire database on SSD.

This white paper explains why traditional high-transaction rate database environments suffer from low performance and high costs. It contrasts RAID versus SSD technology in terms of impact on database performance and cost, it provides guidelines for determining whether an environment will benefit from placing all or part of a database on SSD, and it provides a case study for SSD that has been deployed in a database application in the telecommunications industry.

The Problem

High-transaction rate databases are typically comprised of small records (i.e., 4 or 8 Kbytes) that are often accessed randomly. Because the records are brief when compared to the time required to reach the data location, mechanical disk drives are paced by their ability to locate and retrieve information on the disks (i.e., disk access time). Disk access time becomes the dominant reason for slow database application performance. The problem is compounded by ever-increasing disk drive capacities because as companies take advantage of larger drives, they are accessing fewer spindles for the same amount of data.

Companies with high-transaction rate database applications (for example, wireless and prepaid applications) require many database servers to achieve required transaction rates. Not only are low transaction rates a problem, the cost per transaction is quite high as companies apply more RAID systems and servers toward solving the problem. As the number of physical resources increases, maintenance costs go up and reliability comes down.

The Traditional Approach

There are two techniques that are used in RAID configurations to compensate for slow database performance: (1) adding more drives, and (2) implementing disk cache.

Using RAID to stripe data across many drives can increase the amount of data that is accessed, but it does not reduce the disk access time required for each request. Further, databases are often unable to take full advantage of many drives due to the constraints of transaction integrity.

Caching is a technology that, under certain conditions, can improve access time. However, for

cache to significantly improve access time, either the cache needs to be large enough to comprise most of the underlying data or the data accesses cannot be random.

Both of the traditional solutions – large numbers of drives in RAID configurations and disk cache – can cost companies significantly more in hardware and can limit the flexibility of the database. Further, there still will be patterns of access that will not benefit by either of these solutions and will significantly limit overall transaction rate.

The Solution

When an application includes a high-transaction rate database with small records that are accessed randomly, mechanical disk drives just can't deliver optimum performance. Companies need to change the paradigm of their environments.

A fundamental change occurs when SSD is introduced into a high-transaction rate database environment. Because solid-state disks have no moving parts, they have substantially faster access time – approximately 500 times faster than mechanical disk drives. By eliminating mechanical latencies, the limiting factor in performance is no longer the storage device.

When databases are small, companies can put entire databases on SSD. For larger databases, they can allocate smaller subsets (such as roll back segments, temporary sort areas, indices, and heavily accessed tables) to SSD. In either case, the result will be significantly improved performance and reliability, and a total cost savings per transaction that can exceed 50%.

Database Performance

Why Mechanical Disk Drives are Slow

RAID is a marvel of technical advancement and it is indeed a great value when measured in dollars per megabyte – particularly as drive capacities continue to grow. However, the mechanical nature of disk drives limits the number of random accesses that can be achieved per second, causing the drives in RAID configurations to become a limiting factor in database performance. In order to understand how databases running on RAID are limited by access time, it is important to analyze and understand the access characteristics of disk drives.

There are two components of disk access time: (1) the average time it takes the heads to move to the appropriate track on the drive (i.e., seek latency); and (2) the average time it takes for the location of data on the platter to rotate under the heads (i.e., rotational latency). Seek latency varies with the distance the heads must move. Rotational latency is fixed, and for a 15,000 rpm drive is two milliseconds (one half of the time it takes for a platter rotation). The sum of seek and rotational latency is the access time. For the fastest disk drives available today, this is about four milliseconds.

Why Solid-State Disks are Fast

Solid-state disks have no moving parts and, as such, have substantially faster access times. ***SSD access times are presently about 10 microseconds – about 500 times faster than mechanical disks.***

By eliminating mechanical latencies, the limiting factor in performance is no longer the storage device. The limiting factor is now the ability of the CPU and the disk drivers to initiate enough requests. With a RAID configuration, accesses to disk have long delays between the initiation of an access and the servicing of that access. But with SSD, each access is serviced immediately as it occurs.

The Impact of File Size

Disk access time is negligible in some applications. Large files can offset the effects of access time as they are spiraled onto a drive in successive sectors on a track. As a rough metric, a 500 Kbyte file is about the size where files achieve a 50% utilization of the channel. At that size, the access time and transfer time are four milliseconds each.

Database records, however, are much smaller – typically 4 to 8 Kbytes. When disks seek randomly from record to record, with small records most of the time is spent seeking and only 1% to 2% of the time will be spent delivering the desired data (approximately 1 to 2 Mbytes per second). Figure 1 illustrates this point with a measurement of various block-size random reads and writes to a Seagate Model ST336753LC drive.

Block Size (in bytes)	Disk Accesses/Sec	Disk Data Rate Mbytes/sec
512	256	0.128
1024	248	0.248
2048	251	0.502
4096	246	0.984
8192	242	1.939
16384	230	3.686
32768	216	6.921
65536	198	12.732
131072	161	20.699

Figure 1

This figure illustrates that there is a performance issue with high transaction rate databases using mechanical disk drives for small, random records. As drives have grown in capacity to roughly 50 gigabytes per drive, less than .004% of the drive is accessible each second when randomly accessed in commonly used database sized records. It is worth considering just how much slower this is when compared to any other component of the database system. ***The server and storage channel are capable of roughly 100 times the disk data rate all the way to the connector at the disk drive.***

Methodologies for Compensating for Disk Access Time

There are two techniques that are used in RAID configurations to compensate for slow database performance: (1) adding more drives, and (2) implementing disk cache. While disk array vendors resort to complex disk cache algorithms and database designers go to great efforts to tune databases to get adequate performance, the underlying disk access time is never eliminated. The ever-increasing capacity of drives has exacerbated the problem, as often, fewer drives are used as they appear to meet capacity requirements. Typically, databases are extensively tuned to work well under one set of access patterns, then perform very poorly under different patterns. Batch loading jobs and recovery from logs are two examples where databases often under perform. The underlying disk access time is the reason.

Multiple Drives

RAID configurations rely on multiple drives. However, in all but the largest systems, they rarely use enough drives to significantly improve performance when accessed by databases. As drives have grown in size, as little as five drives often provide plenty of capacity for the database. As discussed earlier, RAID configurations with large drives work well for storage of large files, but for small block, randomly accessed data, many more drives are required – not for the capacity, but to provide increased data rates to the database.

If a single drive can deliver 2.0 Mbytes of 8K blocks per second, then two drives can deliver 4.0 Mbytes per second. This is accomplished by storing data on two drives. The application accesses the first drive, starts its four-millisecond wait and then spawns a second access to the second drive for a different 8K block, again waiting four milliseconds for the desired data to be read or written. As more drives are paralleled and accessed, the data rate increases. These are called concurrent accesses.

Note that the access time for any specific data has not improved, though. It still takes four milliseconds to access any data. Heads still need to move and platters still need to rotate under the heads. Therefore, back at the issuing application, there is still an average of four milliseconds between any specific request for data and the time that request is satisfied. Thus, each of these accesses remains open for four milliseconds. Four milliseconds is a very long time when measured in CPU cycles.

Therefore, while multiple drives can increase the amount of data accessed by increasing both data rate and read and write operations (IOPS), access time does not improve. As well, for an application to take advantage of multiple drives, the issuing application must have parallelism. In other words, the issuing application must have many independent, concurrent processes (or threads) that are independently spawned and are capable of accessing data. However, the requirements of transaction integrity in databases are often in conflict with broad, concurrent accesses. This conflict is at the core of database performance problems.

Oracle[1] says the following:

> As multiple users access the same data, there is always the possibility that one user's changes to a specific piece of data will be unwittingly overwritten by another user's changes. If this situation occurs, the accuracy of the information in the database is corrupted, which can render the data useless or, even worse, misleading. At the same time, the techniques used to prevent this type of loss can dramatically reduce the performance of an application system, as users wait for other users to complete their work before continuing. You cannot solve this type of performance problem by increasing the resources available to an application *because it's caused by the traffic visiting a piece of data*, not by any lack of horsepower in the system that's handling the data.
>
> Although concurrency issues are central to the success of applications, they are some of the most difficult problems to predict because they stem from such complex interactive situations. The difficulties posed by concurrent access continue to increase as the number of concurrent users increases. Even a robust debugging and testing environment may fail to detect problems created by concurrent access since these problems are created by large numbers of users who may not be available in a test environment. Concurrency problems can also pop up as user access patterns change throughout the life of an application.

[1]Ken Jacobs, Vice President at Oracle, puts it in his paper entitled “Transaction Control and Oracle”

Clearly, it takes a large number of parallel drives and a large number of concurrent accesses in order to achieve substantial utilization of the channel. If accesses are not concurrent, but occur serially, as is to some degree necessary with database operations, then the benefits of multiple drives are reduced.

The use of multiple drives also substantially increases the failure rate. The Mean Time Between Failure (MTBF) for thirty drives is thirty times worse than the MTBF of a single drive. Further, large numbers of disks, especially high performance drives, require substantial power and generate substantial heat, which, in turn, accelerates drive failures.

Caching

Caching is a technology that, ***under certain conditions***, can improve access time. With caching, recently accessed data is maintained in a memory area in the RAID system so that subsequent accesses to the data are filled from the memory area rather than from disk. The capacity of this memory area is some small percentage of the underlying disk storage. When the data accessed is in cache, the access time can be very fast because there is no mechanical delay. But when the data is not in cache, the access must go to the disks.

Not only is data stored in the cache, management tables also need to be maintained in cache. The tables keep track of things such as what data in cache correlates to what disk location, and whether data in cache has been superseded by more recent data. Further, as only a small percentage of the data on the disks can fit into the cache, an algorithm is used to select which data will be deleted from cache when new data is inserted. In order to do all this and still maintain a low access time, a high performance cache manager processor is needed to administer the cache, requiring a considerable amount of memory and other resources.

Reads From Cache

When data is read from cache, this is referred to as a “cache hit.” When data is not in cache, then this is referred to as a “cache miss.” Unfortunately, misses are costly, as each miss requires the full access time of a disk before the read is completed. The following chart shows the improvement in average access time as a function of cache hit rate.

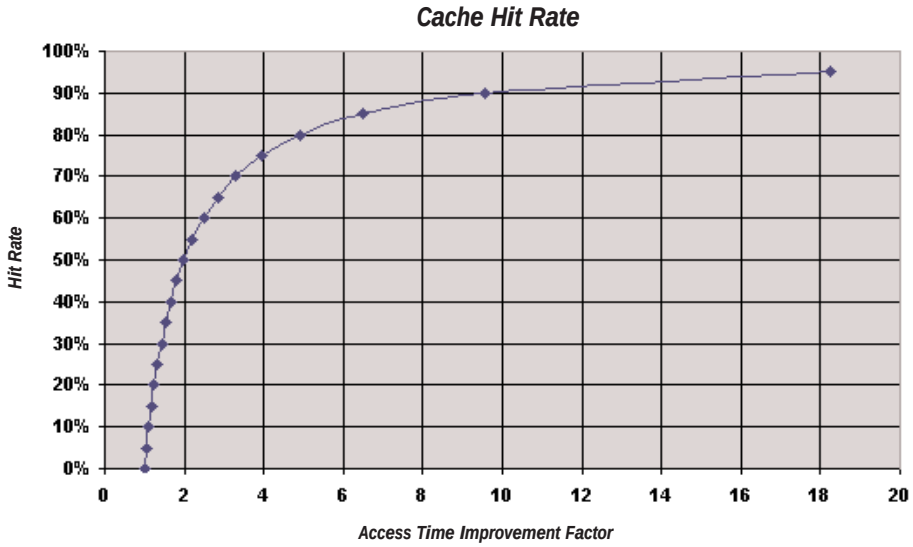


Figure 2

Note that a cache hit rate of 90% is required to cut average access time by a factor of 10. Note as well that if half the accesses are serviced through cache hits, a 50% hit ratio, the overall average access time has only been improved by a factor of two.

Thus, for cache to significantly improve access time, either the cache needs to be large (i.e., it must contain a significant fraction of the underlying disk data), or the data accesses *cannot be random*. This means that there must be some intrinsic characteristic of the pattern of the accesses such that once an access has occurred, there is a very high probability that the data will still be in cache when the next accesses for that data occur.

Often, this intrinsic characteristic exists, for example, a customer record may be accessed and then quickly thereafter modification may be made to the record. Note, though, that if on average, only two accesses are made to this record, the cache-hit rate is only 50% and the cache has brought only limited benefit. ***Thus, the characteristics of the access pattern largely determine whether disk cache is of benefit.***

Surprisingly, the case where the amount of cache is a significant fraction of the underlying data on disk often does occur. Unfortunately, it is typically during the development phase of the database when it is common to use small test databases. Because all or nearly all of the underlying data remains in cache during development, hit rates approach 100%. The small development database is not a true example of the real world environment, so the performance limitations of the disk accesses are undetected until the entire database is moved into production.

Writes to Cache

Writes to cache are performed in two ways, write-through and write-back. Write-through cache returns a write-complete status to the initiating process when the data is written through the cache to the disk platter. Write-through cache means that the full disk access time will occur, so the cache

supplies no performance advantage on writes, but does have the side benefit of putting data into cache for subsequent read accesses.

Write-back cache means that the write complete status is returned to the initiating process when the data is initially written into the cache, and the cache manager processor takes responsibility for getting the data written from the cache to the disk platters. Thus, write-back cache becomes a buffer for write data. Clearly, write-back mode can have a very low access time, but it does increase the risk that data could be lost due to a disk system failure. In addition, if enough writes are written into the cache quickly enough, the cache will fill up, at which point subsequent writes will only occur as room becomes available. This means additional writes will occur at the underlying disk access time, the rate at which data can be removed from the cache.

Note as well that if large amounts of extraneous data (e.g., a large file) are read from or written to the disk subsystem, other pertinent data, which might have supplied cached hits, is supplanted in the cache. This is called cache flushing and means that cache hits will go to zero until the pertinent data repopulates the cache. This indicates that RAID arrays for databases should not be shared with other applications.

Transaction Rate Comparison

Below are the transaction rates of a modern solid-state disk and, for comparison, the previously discussed mechanical disk drives.

Block Size (in bytes)	*SSD Accesses Per Second*	*Disk Accesses Per Second*	*SSD Data Rate Mbytes Per Sec*	*Disk Data Rate Mbytes Per Sec*
512	12903	256	6.5	0.1
1024	12805	248	12.8	0.2
2048	12579	251	25.2	0.5
4096	12190	246	48.8	1.0
8192	11175	242	89.4	1.9
16384	8878	230	142.0	3.7
32768	5290	216	169.3	6.9
65536	2856	198	182.8	12.7
131072	1479	161	189.3	20.7

Figure 3

Clearly, disk access time is the dominant delay and is orders of magnitude longer than that of the rest of the CPU and storage channel. Note as well, that as accesses get larger, rotating disks work better. For this reason, operating systems bring additional blocks into server memory independent of whether the application asked for them or not. The theory is that since such a long time was spent getting to the location, only a little more time will be spent to bring in substantially more data. This is called read ahead and is a form of dumb caching. Again, this often provides a benefit with large files, but is not likely to provide a benefit if data is truly random.

Benefits and Costs of Solid-State Disk Technology

The Paradigm Shift with SSD

A fundamental change occurs with SSD. By eliminating mechanical latencies, the limiting factor in performance is no longer the storage device. It is now the ability of the CPU and the disk drivers to initiate enough requests. With a RAID system, accesses to disk had long delays between the initiation of an access and the servicing of that access. But with SSD, each access is serviced immediately as it occurs.

This changes the paradigm of the application and database. Now, there is less benefit to parallelism because a single SSD can handle all the transactions a server can request as they occur. The result is that concurrent access is no longer as vital to the performance of the application. Processes do not remain open for extended periods as data is now retrieved or written as fast as the server can communicate with the storage channel, so serialized transactions are no more costly than concurrent ones and random operations take no longer than sequential ones.

This manifests itself in systems that are not access pattern sensitive, as is disk cache. Regardless of the access pattern of the data, an SSD based system can keep the storage channel operating at a substantial percentage of its available bandwidth. This is now limited by the server and the channel, rather than the storage device.

Again, this is not strictly an issue of IOPS or data rate. These, theoretically, can be achieved with many drives. Instead, it is an issue of latency – the time from the initiation of an access to the time of its completion – that fundamentally differentiates SSD from RAID, and consequently benefits database applications.

Cost of Lost Data

Potentially, the most significant cost in a database application is the cost of lost data. Fortunately, the statistical chance of losing data when using SSD is negligible. ***The Mean Time Between Failure (MTBF) of SSD is greater than two million hours. This translates to a data loss once every 228 years per SSD. If used in redundant pairs, the possibility of losing data is vanishingly small – once every 310,000 years.*** Telecommunications providers are using SSD in emerging countries because, unlike rotating disk, SSDs have no scheduled maintenance and do not require operator intervention due to power failures. As a case study example, in a carefully monitored 24X7 telecommunications site with more than two hundred SSDs and four million operation-hours, no data has ever been lost.

Cost Per Megabyte Versus Cost per Transaction

There is no debate that RAID is a great value when measured in dollars per megabyte. However, in order to provide adequate performance in database applications, additional disks and a great amount of disk cache are required. This significantly increases the cost of RAID as the cost per transaction (i.e., the cost of the storage and CPU) becomes very high. Therefore, applying a "dollars per megabyte" metric is inappropriate, especially when databases are small. (It is also

important to note that even when multiple disks and disk cache are added, databases are still limited in performance due to disk access time.)

High transaction rate databases, such as those used in telecommunications applications, need the lowest possible cost per transaction. In other words, they need very high levels of performance and high levels of reliability at the lowest overall cost.

Cost Analysis of RAID Versus SSD

In the cost analysis shown below, one server with SSD was compared to three servers, each with their own RAID arrays. Both configurations achieved approximately the same transaction rate, though the configuration without SSD required significantly more disk drives to achieve the performance than were actually required for capacity. The same level of performance was achieved with a single server and 80 gigabytes of mirrored SSDs.

Solid-State Disk Cost Analysis		
Cost (in $1,000s)	***1 Server with SSD***	***3 Servers without SSD***
Servers - $70k/ea.	$70	$210
Disk Array - $15k/ea.	N/A	$45
2-SSDs per server - $40k/ea.	$80	N/A
Total Hardware	$150	$255
Hardware Savings	***41%***	
Service/Support 3yrs	$50	$115
Administration 3 yrs	$38	$113
Total S/A	$88	$228
Service, Support & Admin Savings	***61%***	
Total Cost of Ownership	***$238***	***$483***
Total Savings cost per transaction	***50%***	

Figure 4

In this example the SSDs were 25 times the cost of the RAID arrays when measured in dollars per megabyte, but when the total system was taken into consideration, a 41% savings in hardware was achieved with SSDs. Further, there was a 61% savings in administration, service, and support costs. ***Overall, the cost per transaction of the SSD based system was half that of the RAID based system.***

What is not shown in this cost analysis are the financial benefits of faster time to market and the improved flexibility of systems designed with SSD. Regardless of the pattern, data will always be accessed as fast as the server can operate. Thus, when future queries that were potentially unanticipated during the database design are needed, they will not be limited by disk access time.

Implementation of Solid-State Disk

Large Databases

When databases are large (i.e., greater than 100 gigabytes), using a combination of SSD and RAID is a cost effective solution. This is especially true when the online performance of the database is adequate, but batch updating of the database exceeds the available time window or queries take too long to complete.

There are smaller subsets of the database – such as transaction roll back segments, temporary sort areas, indices, and heavily accessed tables – that can selectively be stored on SSD. Often, the result is not only substantially shorter batch jobs, but also faster queries, backup, and recovery as well.

Compatibility with Servers

Except for improved access time, SSDs appear to database servers as standard storage devices. Parsing selected database areas to SSD is no different than moving them to an additional RAID system. SSDs are appliances that are compatible with all servers, and in most environments they do not require specialized drivers or software.

Applications

When Does SSD Make Sense?

SSDs are not required for every database application. Low activity databases where batch jobs, backup, or recovery are not an issue are often well serviced by RAID systems. However, when a database has ongoing performance problems and constantly requires tuning, SSDs can provide immediate relief. Whenever high volume transactions are anticipated, SSDs will always provide additional performance and shorter design times.

SSDs are indicated when a database has one or more of the following characteristics:

- High Transaction Rate
- Batch Job, Fast Recovery, or Backup is a Critical Operational Component
- Database Has Already Been Heavily Tuned
- Undetermined Or Changing Usage Patterns
- Unacceptable Query Duration
- Queries Affect Other Database Functions
- High Peak Loads
- Short Time to Market
- Real-Time Databases
- 24 Hr On-Line Operation
- Limited Available Maintenance Windows

Tuning

When databases exhibit one of more of the above characteristics, the management decision is

whether to tune or add SSDs. Databases need tuning as part of the initial deployment and this initial tuning often provides easy and substantial performance gains. SSDs are indicated when subsequent tuning provides only marginal improvements and when performance unduly constrains database flexibility, such as restricting queries to off hours, or risks corruption due to reduced transaction integrity settings.

While the addition of SSDs offers immediate performance gains, there is often additional benefit from subsequent database tuning. When possible, minimization or elimination of read ahead caching by the operating system will free up additional channel time for accesses of desired data and can often provide even more SSD performance. Often raw accesses can provide impressive improvements by minimizing operating system overhead.

Case Study: Wireless Short Messaging Service

- Over 200% performance improvement
- Cost per message reduced by nearly two-thirds

Company

The client in this case study is an integrator and known market leader in the mobile wireless telecommunications space. The company is a diversified global provider of systems integration, turnkey solutions, and business process outsourcing, with nearly $2B in annual revenues and offices in over 30 countries. Other markets include energy, utilities, financial services, and transportation.

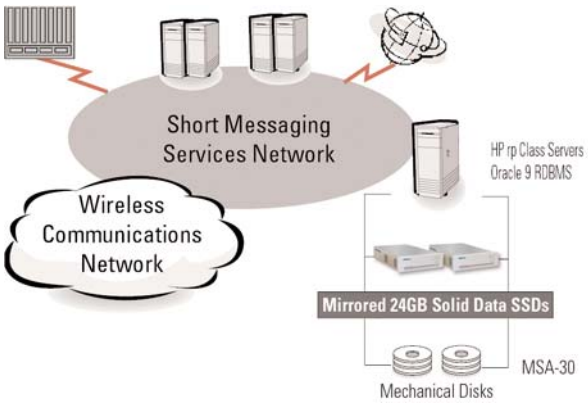
Mobile Wireless Networks Business Unit

Exceeding 100 billion messages per month, Short Messaging Service (SMS) traffic has grown explosively worldwide. The integrator is constantly challenged by its competition to maintain market leadership. Focus on Multimedia Messaging Service (MMS), SMS and Unified Messaging Service (UMS) solutions has enabled the integrator to provide best-of-breed next generation premium messaging services to a majority of operators worldwide.

Architectural Challenges

High performance is mandatory in the integrator's business for services such as billing, payments, mobile commerce, and customer care. The integrator's ability to keep market leadership in these services hinges on achieving the maximum performance from as few servers as possible, while maintaining an architecture that is highly scalable, reliable and maintainable.

Such performance requirements create heavy database transaction loads on the CPUs and extreme I/O loads on the underlying storage subsystems. During peak times, it was common to see high I/O wait states where the processors would sit idle because they were waiting on disk I/O to complete.



SMS Architecture

Solid-state disk produced dramatic results, increasing message capacity per day by over 200% and reducing cost per message by 65%. This improvement was achieved with an incremental investment equal to 15% of the base cost.

As the entire database for this SMS application was relatively small (less than 50GB), the integrator tried a very common method of tuning database I/O performance. The company striped the entire database across an array of multiple physical disks. This only slightly improved performance, but it created a significant amount of I/O wait and did not address disk access latency.

Solution: Solid Data Solid-State Disks

The integrator placed the entire database on mirrored pairs of Solid Data SD3000 Fibre Channel Solid-State Disks. Benefits of the SSD integration included:

- CPUs are able to do more work because the CPU "I/O Wait" state which was caused by excessively high disk I/O was eliminated
- Each server is more scalable, allowing for future expansion without requiring additional hardware
- The MTBF is significantly increased with a more reliable storage infrastructure, which saves costs in reduced maintenance and downtime
- Maximum performance was achieved because all data is now accessed at memory speeds
- The total cost of ownership was reduced as less hardware is required to achieve the desired workload

Eliminating the I/O wait also improved other critical operation, administration, and maintenance functions. ***For example, the SSDs reduced the normal purge cycle of database records from three hours to 45 minutes.*** Bill generation from SMS records requires data persistence, protection, and operational reliability. Here, Solid Data SSDs offer speed approaching that of main memory, with the added advantage of data persistence and greater scalability.

Ultimately, by placing the entire database on mirrored pairs of Solid Data SD3000 Fibre Channel Solid-State Disks (SSD), The integrator's SMS system configuration now delivers extremely high performance, scalability and availability.

	Before (Original System)	***After (System with Solid-State Disks)***
RDBMS	Oracle9	Oracle9
Configuration	HP rp Class Servers HPUX 11 HP RAID Storage Database elements striped across the RAID	HP rp Class Servers HPUX 11 Mirrored Solid Data Systems Model SD3000 SSDs: · 24 GB capacity each · Fibre Channel · Entire database contained on SSD
Application Performance	Daily Message Throughput: 2,500,000 Messages per day	Daily Message Throughput: 7,000,000 Messages per day

Result

Solid-State disks clearly deliver a compelling payoff. With only a nominal incremental investment, ***the integrator achieved more than a 200% performance improvement and reduced per-message costs by nearly two-thirds, from 5.1 cents to 1.8 cents.***

Summary

When an application includes a high-transaction rate database with small records that are accessed randomly, mechanical disk drives just can't deliver the required performance. Although RAID appears to be a great value when measured in dollars per megabyte, the mechanical nature of disk drives is a limiting factor, and it can result in many hidden costs for high-transaction rate databases. Attempts to increase the performance in these RAID environments (i.e., adding more drives and implementing disk cache) will result in extremely high transaction costs.

Because of the high-cost of RAID in high-transaction rate database environments, companies need to change the paradigm. A fundamental change occurs when SSD is introduced. ***Because solid-state disks have no moving parts, they have substantially faster access time – approximately 500 times faster than mechanical disk drives.***

And, not only is performance multiplied exponentially with SSD, MTBF is radically improved as well. The MTBF of SSD is greater than two million hours. ***This translates to a data loss once every 228 years per SSD. If used in redundant pairs, the possibility of losing data is vanishingly small - once every 310,000 years.***

In this white paper, an example was given where SSDs were 25 times the cost of the RAID arrays when measured in dollars per megabyte. But when the total system was taken into consideration, a 41% savings in hardware was achieved with SSDs. Further, there was a 61% savings in administration, service, and support costs. ***Overall, the cost per transaction of the SSD based system was half that of the RAID based system.***

When implemented in high-transaction rate database environments, SSD products from Solid Data Systems deliver:

- Maximum performance by accessing all data at memory speeds
- Lower transaction costs (often 50% lower or more)
- No I/O wait states, freeing up the CPU to do more work
- Increased MTBF, which saves costs in reduced maintenance and downtime
- Improved server scalability, often without additional hardware
- Lowest total cost of ownership

About Solid Data Systems

Founded in 1993, Solid Data is the leading global provider of solid-state disk systems. Solid Data holds core patents in solid-state disk (SSD) technology and developed the first non-volatile solid-state disks. Solid Data's durable and time-tested SSD technology lowers the cost and increases the uptime of transaction-oriented databases. Solid Data has over 3,700 systems installed worldwide, and is headquartered in Santa Clara, California.