

# Using I/O Signature Analysis to Improve Database Performance

February 2009

By Solid Data Systems, Inc.

## Contents:

Executive Summary	1 - 2
I/O Signature Analysis	3 - 4
I/O Signatures That Benefit from SSDs	4 - 6
Summary	6 - 7



## EXECUTIVE SUMMARY

### Overview

When databases are performance-limited the dilemma often arises as to whether upgrading the server or the storage will provide the bigger benefit. This white paper introduces the concept of measuring an application's I/O Signature with existing UNIX or MS tools to easily determine the answer. Additionally, the option of utilizing solid state disk (SSD) when certain I/O Signatures predominate is examined. Oftentimes a small amount of SSD will yield the best performance value under certain conditions, whereas in other conditions placing the entire working set on SSD or an array of SSDs is optimal.

The tools and techniques in this white paper do not require a deep understanding of the application, and have been used for several years to identify deployments that will benefit from SSDs from those that can be improved through traditional server or storage upgrades.

### Understanding CPU Utilization and Latency

Servers can be viewed as boxes of CPU cycles. These cycles occur, X number per second, whether they are used or not. Efficient utilization of CPU means taking advantage of as many of these cycles as possible to do useful work. Modern servers have multiple high speed CPUs, resulting in large numbers of cycles. The tools discussed in this paper break these CPU cycles into four categories, each indicating a percentage of the total CPU cycles as shown in Figure 1 below. One of the categories is Percentage Idle. This is the time CPUs are asleep without an outstanding I/O or anything else to do. The second, Percentage User, is the percentage of CPU cycles used to run the application. Percentage System is the third component and is the percentage of the CPU cycles used to run system (kernel). The last is Percentage I/O Wait and indicates the percentage of idle CPU cycles wasted waiting for I/Os.

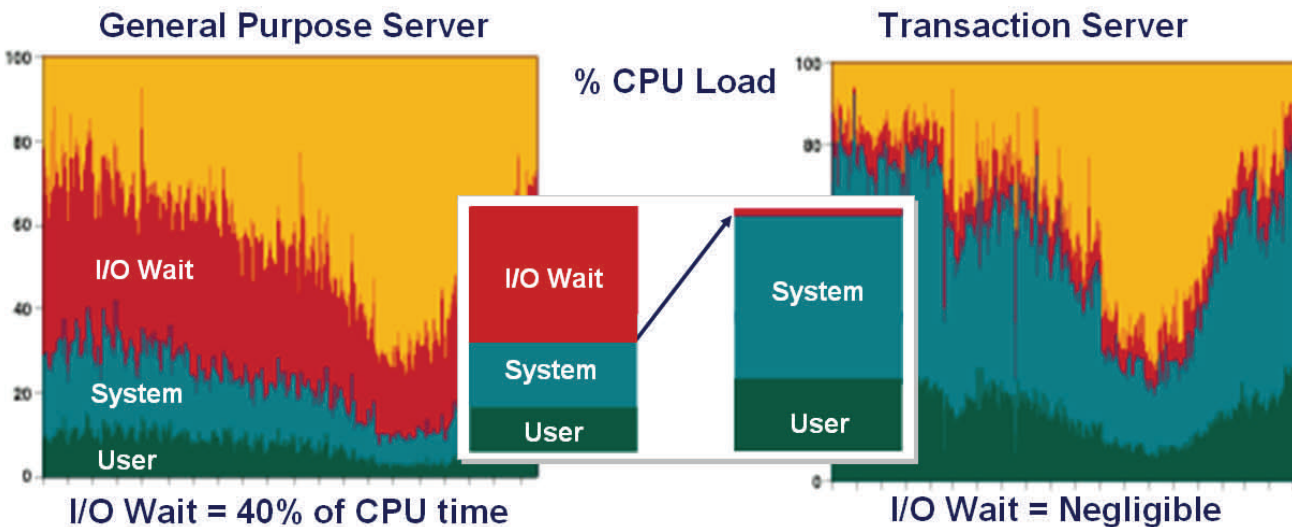


Figure 1

Disks can be viewed as very high data rate devices that are very slow to get to the locations where data is stored. Hence, if the disk read/write heads are over the desired data locations, then data can be written or read at high speeds; however, if heads have to move, then a very long delay (especially when measured in CPU cycles) will ensue before this high speed data transfer can occur. This delay is called latency.

Latency is defined by Webopedia as follows:

*In general, the period of time that one component in a system is spinning its wheels waiting for another component. Latency, therefore, is wasted time. For example, in accessing data on a disk, latency is defined as the time it takes to position the proper sector under the read/write head.*

When disks are streaming, that is, performing large sequential read or write operations, disks can provide a lot of data per unit of time. However, when disks are seeking a large percentage of the time, they deliver very little data per second and most of the time is wasted. If a disk is randomly seeking for each access, it will be limited to about 2 megabytes per second of transferred data when the block size is 8K Bytes, a typical database block size. This indicates that the high speed bus at the back of the disk drive is only transferring data about 1% of the time. The common solution to this is to arrange a large number of disks accessed in parallel; even so, one can see that a very large number of disks are required to achieve a significant percentage of the available I/O bandwidth. An issue which is often overlooked is that the application and database must also be able to operate in parallel. That is, they must be able to issue an equivalently large number of concurrent I/Os to simultaneously access this large number of drives; otherwise, the paralleling of all these drives will not improve the situation. Databases and applications are frequently limited in this area.

Oracle<sup>1</sup> 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.*

Modern databases, servers, and mass storage devices have developed elaborate and complex caching technologies to assist with this problem by keeping heavily used data in various local semiconductor memories, thus avoiding disk latency altogether. However, in critical applications where loss of data cannot be tolerated, this data must also be written to disk to avoid catastrophic data loss. In addition, for caches to improve performance by a factor of ten over disks alone, hit rates have to be very high – greater than 90%. Sometimes this occurs, oftentimes not. Clustering servers adds yet another layer of complexity as data in cache typically is replicated across multiple servers and needs to be updated across the cluster.

What if there was a way to ensure the data required was always resident in “memory,” and at the same time provide you with increased reliability and performance predictability? How much would that benefit your applications?

In the final analysis, systems have become so complex that it is essentially impossible to accurately predict the performance of a large system under varying real world data loads or estimate where or why performance might be limited. Testing with smaller datasets often keeps a disproportionate amount of data in cache, thus masking full load performance issues. Fortunately, there is another practical approach that allows real-time analysis of a

<sup>1</sup>Ken Jacobs, Vice President at Oracle, states in his paper entitled "Transaction Control and Oracle."

production server's CPU cycle utilization and indicates which steps to take to improve performance, regardless of the inscrutability of the application.

## I/O SIGNATURE ANALYSIS

### Analysis Tools

Gathering the data to analyze the I/O Signature of an application is best achieved by using resident server tools. In the UNIX operating system this means the system activity reporter (SAR), or iostat. In Windows perfmon is widely used to gather the data. Recognize that these tools need to be run in a way that does not use too much of the server's resources. This is especially true if the server is heavily loaded. Typically, SAR and iostat use very little CPU; perfmon uses somewhat more and may not function well if the server is overloaded.

You will want to run these tools during a period when server performance is slow. It is difficult to project full load issues by analyzing the server when lightly loaded. With long billing batch jobs, for example, there may be periods when the server is operating efficiently and other intervals where performance slows. Therefore, these tools must be run at intervals throughout the batch job process.

### Analyzing the I/O Signature

Initially gather the following statistics:

UNIX/LINUX	MS Windows
<u>CPU Stats</u>	<u>CPU Stats</u>
% User	% User
% System	% Privileged
% Idle	% Idle
% I/O Wait	Not Directly Applicable
<u>Disk Stats</u>	<u>Disk Stats</u>
Queue Length	Average Disk Queue Length
% Busy	% Disk Time
Average Service Time	Average Disk Seconds/Transfer
Average I/O Size	Average Disk Bytes/Transfer
Reads/Sec	Disk Reads/Sec
Writes/Sec	Disk Writes/Sec
KB Reads/Sec	Disk Read Bytes/Sec
KB Writes/Sec	Disk Write Bytes/Sec

The CPU statistics will first give the biggest clue as to whether the server or the storage is the limiting factor in performance. If the total of User and System percentage is greater than 80% and I/O wait is less than 10% dur-

ing the period of poor performance, then the system is CPU-limited. If the application can take advantage of multiple processors, this would be the first step towards improving performance. Other alternatives are faster processors or an application rewrite to try to reduce CPU utilization.

If there is a substantial amount, greater than 15%, of I/O Wait (or in Windows a queue length is high and idle is high), then further scrutiny is necessary. Examine the disk statistics. Are any of the disks more than 80% busy? If so, examine the disk queues, service times and average I/O size for those drives. If the average I/O size is approximately a database block of 2K to 16K Bytes, then service times in the area of 10ms indicate that this disk is servicing largely random I/Os. This I/O Signature indicates the disk is providing little data and may be limiting performance.

If these disks have a queue of 2 or more (per physical disk), this indicates that the application has issued I/Os that are waiting to gain access to the drive which indicates a potential I/O bottleneck. One should always reference the I/O latency at this point via the Service Time or Disk Seconds/Transfer counter especially if large RAID arrays are used where the number of physical disks that make up logical volumes cannot be determined easily. When this is the case, a determination has to be made as to the drive's contents. Because excessive paging can make substantial use of a hard disk, it is possible that a memory shortage is causing this disk activity; if the disk's contents include a page file it is likely the server needs more memory. If there is no page file, it may be that a single hot table or a highly used database element such as a temporary sort area may be the culprit. It may also be that multiple hot elements may have been placed on the same drive. Initial improvements can be as simple as moving one of the elements to a different drive.

## I/O SIGNATURES THAT BENEFIT FROM SSDs

### When Traditional Upgrades Aren't Good Enough

Spreading database elements over multiple disks assumes the application can issue concurrent I/Os to the element. Sometimes this will not be the case due to synchronous I/O being utilized or because the application itself needs to get data from the drive before it can determine the location of the next I/O. In this case queues will be less than 1; however, if the service times are long, these drives are still limiting performance. This is typical of a single-threaded process, and attempts to use parallel drives or multiple processors will have little impact. The I/O Signature of this type of process is low CPU utilization (as identified by both User and System percentages) during the period while this drive is busy.

In this case and in the case where merely spreading the data over more drives does not bring enough overall improvement, SSDs can be a remarkable spot fix. Because they have no heads to move, random I/Os that take 10s of milliseconds with standard hard disk arrays will be serviced in 10s of microseconds with SSDs. If a few elements of the database are gating overall performance, SSDs can often provide a big performance gain.

There is also a flipside to this case where large applications can issue concurrent I/Os in widely multi-threaded processes. In these cases, the database could have many data elements gating overall performance, each requiring a low latency, persistent drive to service the onslaught of I/O requests. Queues on both the CPU and disks can start climbing quickly in this situation if the appropriate best practices are not implemented to take full advantage of these multiple threads. An SSD array is often the optimal solution for this type of environment where multiple threads require a high amount of both bandwidth and transactional performance. In either case, one must understand the performance capabilities and Best Practices of configuring the underlying storage

subsystems, whether they consist of SSDs or HDDs.

The following table compares an HDD drive and an SSD running on UNIX using IOTest which issues random I/Os at various block sizes. Note that at an 8K block size the SSD is capable of nearly 50 times the data transfer of the rotating drive.

Block Size	Access Rate – I/Os per Second		Data Rate – MBs per Second	
	SSD	HDD	SSD	HDD
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
8192	11175	242	89.4	1.9
16384	8878	230	142	3.7
32768	5290	216	169.3	6.9
65536	2856	198	182.8	12.7
131072	1479	161	189.3	20.7

**Table 1**

Due to their ability to access data a memory speeds, SSDs free up the percentage of the CPU cycles waiting on I/O for improved server utilization, often preventing the unnecessary server upgrade. Furthermore, if the server already has a significant percentage of its CPU cycles idle waiting for I/O, then more CPU cycles will be largely wasted. In this case, it is the storage and not the server that is limiting performance.

While the comparison so far has been around eliminating latency due to random I/O, Table 1 clarifies one other area where SSD can help. Note that for the largest block size, 131072 bytes, the SSD is faster than disk. This indicates that even as the block size gets very large, approaching sequential I/O, SSDs still maintain some performance advantage over rotating drives. With sequential I/O the performance of the rotating disk is not limited by latency, but rather by the platter streaming data rate. A faster data rate is the reason transaction logs or high speed queues benefit from SSD. Log block sizes vary depending on the database; however, they all write sequentially for performance reasons. Thus, transaction logs should be the only data on the drive. As stated by Microsoft<sup>2</sup>:

*Because all transactions are first written to the transaction logs, transaction logs should be on a storage device that has the lowest possible write latency.*

The I/O Signature of a log drive limiting performance is indicated by high levels of percent busy.

<sup>2</sup>Microsoft TechNet, <http://www.microsoft.com/technet/prodtechnol/exchange/guides/StoragePerformance/>

## When to Use SSDs

Over many years of deploying SSDs certain types of database applications emerge again and again as good candidates for SSD. These databases have one or more of the following characteristics:

- High Transaction Rate
- Batch Job, Fast Recovery, or Backup Is a Critical Operational Component
- Unacceptable Query Duration
- High Peak Loads
- Database Has Already Been Heavily Tuned
- Undetermined Or Changing Usage Patterns
- Queries Affect Other Database Functions

Applications that historically have benefited from SSD are:

- Billing Applications
- Wireless Applications
- Trade Clearing and Settlements
- Trend / Customer History Analysis
- Write-Intensive LDAP
- Metadata
- Health Informatics EMR
- HPC / Grid Computing
- On-Line (Web) Purchase Transactions
- Inventory Management
- Email (IMAP Directory)
- Fraud Detection

## SUMMARY

Because mass storage is viewed as a monolithic black box, the general perception is that SSDs and SSD arrays should be viewed the same way. In the case of small, high performance databases, placing the entire database on SSD does indeed work extremely well. This is a common approach in the telecommunications industry where transaction rates can exceed ten-thousand per second. The benefit is that regardless of changing access patterns or peak loads, SSDs will provide data as fast as the server can accept it, thus reducing the need to tune, stripe, or significantly modify applications or the database to get major performance increases. The downside to this technique is cost; as the dataset gets larger and larger the cost can become prohibitive. One needs to exercise caution when analyzing these costs, keeping in mind that the alternatives to SSD often involve the addition of several more servers, each with their own storage, licensing, administration and maintenance costs. Often a single monolithic database with SSD may have higher storage cost but substantially lower system cost overall.

Another view of SSD implementation in larger databases is to recognize there are subsets of the data (e.g., hot tables and indices), as well as subsets of internal database components such as temp space, queues and logs, with I/O Signatures that indicate they are limiting overall performance. Placing just this data and these components on SSDs can provide substantial database performance improvements at reasonable cost. This is typically a small percentage (2% – 5%) of the total storage, but yields substantial performance gains without the need to increase server compute power.

Gaining expertise in uncovering I/O Signatures will enable you to more rapidly and accurately determine how to optimally improve the performance of your database. For more information, go to [www.soliddata.com](http://www.soliddata.com).

### **About Solid Data Systems**

*Solid Data Systems, Inc., founded in 1993, is the leading global provider of DRAM-based enterprise and carrier-class solid-state disk storage systems. Solid Data SSD-based architectures increase peak transaction and application performance, use less power, require fewer servers and lower total cost. Solid Data has the largest install base of trusted Fiber channel- and SCSI-connected SSDs with proven MTBFs of up to 2 million hours. Available products include the SD2000, SD3000/SD3000X2, and StorageSPIRE - offering up to 1TB of high performance SSD storage in a single managed array. Solid Data is the only vendor to offer NEBS-certified SSD products for critical telecommunications applications. Solid Data customers are among the world's largest technology resellers, systems integrators, financial services companies, telecommunications providers and government agencies*