A simplified approach to Windows disk tuning

Mark B. Friedman
Demand Technology
1020 Eighth Avenue South, Suite 6
Naples, FL USA 34102
markf@demandtech.com

The basic elements of disk hardware performance are common across all platforms. What complicates disk tuning today on every platform, including Windows, is virtualization technology that hides the true nature of the physical disk from the host operating system. This session discusses a simple procedure to measure the performance of the physical disk entity and establish reasonable service time expectations for any virtual or physical device. If actual measured performance exceeds these service expectations, you can then decide on an appropriate disk tuning strategy.

Introduction

The paper outlines a simplified approach to disk tuning in the Windows environment. Since essentially the same disk hardware is currently used across all the major enterprise computing platforms, the basic elements of disk hardware performance are common across those platforms. This certainly does simplify many aspects of disk configuration and tuning on the Windows platform. Meanwhile, the Windows operating system manages to inject some unique considerations into procedures designed to optimize disk performance, some of which are discussed here. These include the disk performance measurements that the OS provides and the disk device driver options that can influence performance. Outside these platform-specific considerations, disk device performance on Windows machines is virtually identical to most varieties of UNIX servers. Except for the differences in the linkage used to connect disk devices to the host machine — e.g., Fibre Channel vs. FiCON – disk device performance on Windows is also identical to the IBM mainframe z/OS platform.

What complicates disk tuning today on every platform, including Windows, is virtualization technology that hides the true nature of the physical disk from the host operating system. From inside Windows — which is where the measurements of disk performance originate — what appears as a physical disk is likely to be a virtual disk of some sort, especially in an enterprise storage environment. Virtual disks are entities that RAID controllers export. They impersonate the characteristics of a physical disk device. Virtualization engines — usually embedded inside the disk controller — mask the physical characteristics of the underlying disk hardware. Whether the physical disk the host sees is a single physical disk entity or a series of disks or disk segments organized into a RAID grouping and buffered using cache memory is transparent to the host software.

Traditional approach. The traditional approach to disk performance and tuning is to use the device's physical characteristics — seek time, rotational delay, data transfer speed and protocol delay — to create a set of reasonable service time expectations. Next, the device is treated as a single server in a simple M/M/1 queuing model to derive reasonable expectations of queue time under load. Then, the performance analyst compares the actual performance of the device to the modeled set of expectations. If actual performance levels greatly exceed these reasonable expected levels of performance, then there are grounds for pursuing various performance-oriented configuration and tuning strategies. If actual performance levels are roughly equal to reasonable expectations, there is little merit to pursuing any of the configuration and tuning options available, other than perhaps buying faster devices.

Applying this proven technique to today's virtualization engines involves creating a queuing model of the disk subsystem that is exporting these virtual disks, where each of the internal components of the disk subsystem is loaded based on the measured workload. Solving the queuing model will show which components of the disk subsystem saturate first under load and constrain performance. While undoubtedly that is an effective method, there are any number of complexities that arise in typical enterprise storage area networks that may make that approach quite difficult to put in practice, even for experienced disk performance analysts. In particular, it may become necessary to understand in some detail the underlying physical configuration of disks, links, and caches that comprise the virtual volume.

When poor disk performance makes it necessary to figure out what is going on behind the scenes with one of these virtual disks, the task can be quite daunting. One aspect of the challenge is that the architecture of the disk subsystem, or *storage processor*, that presents the device image to the host OS can be fairly complex, replete with internal processors, buses, and the physical disks themselves.[1] The virtual disk data that is recorded is usually mapped to two or more physical disks, or portions thereof, in an arrangement that conforms to one of the RAID levels. Performance considerations of the popular forms of RAID are also well-known.[2]

When the virtual disks the storage processor exports are distributed across multiple hosts, there may be other complications. These include the necessity to amalgamate disk performance measurement data from different and often incompatible sources. Many storage processors do generate statistics that can be used to illuminate many aspects of their performance, but it may be difficult to synchronize them to the different sources of host-based measurements.

Benchmarking. Without needing to know what is going on behind the scenes of the virtualized physical disk image, the performance analyst can fall back on using a simpler method that is no less empirical and robust. This alternative method avoids attempting to build an idealized model of the disk subsystem and gathering measurement data to load it appropriately. Instead, to establish reasonable service time expectations for the virtual device, you measure its performance directly. This empirical data on virtual disk performance expectations is made under a controlled load. This paper will suggest a compact set of specific benchmark measurements of disk performance that should be taken in a controlled environment that can be used to establish a set of reasonable device performance expectations.

Once you have established reasonable expectations for device service time in this simple and direct manner, you can compare actual service time of the disks that the host system is accessing using measurements that are easily obtained in the Windows environment. If your actual measured performance levels far exceed reasonable service expectations, you can then decide on an appropriate disk tuning strategy to improve the situation. This paper will also describe in general terms the most effective tuning strategies you can employ, along with guidelines that will assist you in determining which are likely to be most effective for your specific performance problem.

Characterizing disk performance

The time it takes to access data on a physical disk is the sum of the following mechanical components, information which can usually be acquired from the specifications provided by the disk manufacturer:

Disk service time = seek time + rotational delay + data transfer

Seek time is the time it takes to re-position the read/write actuator from the current track location to the specified track location. This is sometimes called a *motion seek* because it requires a mechanical motion to move the disk arm from one spot on the disk to another. Seek time is roughly a linear function of the distance between the current track and the destination track, if you allow for some additional delay to overcome inertia initially, reach maximum speed, and braking at the end of the mechanical operation. A *minimum seek* is the time it takes to move from one track to its neighbor. A *maximum seek* moves the arm from one end of the disk to the other. An *average seek* is based on moving the head from one random location on the disk to another random spot. Statistically, this amounts to a seek of 1/3 the maximum seek distance. A *zero seek* refers to an operation on the current track that requires no mechanical delay to re-position the read/write arm.

Rotational delay is the time it takes for the selected sector within the designated track to rotate under the read/write head before it can be accessed. This is often called the device *latency*. This delay is a function of the disk's rotation speed. If the platters spin continuously at 10,000 revolutions per minute (rpm), then a complete rotation of the disk takes about 6 milliseconds, which is the maximum rotational delay. The specific sector being accessed to begin the read or write operation can be located anywhere within the designated track, as the track spins relative to the read/write head. Consequently, an average rotational delay is 1/2 of a complete disk revolution.

Data transfer time is the time it takes to transfer data from the host to the disk on a write operation or from the disk to the host on read operation. The device's data transfer rate, normally specified in MB/sec, is the product of the track recording density times the rotational speed. While rotational speed is constant for all tracks on a platter, the recording density usually varies. Data tracks that reside on the outer surface of the platter ordinarily contain twice as many data sectors as inner tracks. As a result, data can be transferred to and from outside tracks at twice the data rate of an inner track. The size of the data block being transferred also figures into the data transfer time. For example, if the average block size is 12 KB and the track transfer rate is 60 MB/sec, then data transfer time is approximately 0.5 ms.

The file system allocation unit size normally determines the size of I/O requests. However, the Windows operating system transforms individual I/O requests into bulk requests under some circumstances. Demand page reads are sometimes grouped into bulk requests. In addition, write requests subject to the Lazy Write file cache flushes are also usually transformed into bulk requests. Bulk requests usually increase the average size of data transfers in blocks several times the file system allocation unit size.

Additional factors that significantly affect the speed of the device include the interface type (Fibre Channel, SCSI, ATA), interface speed, and the use of on-board cache, which is present on most disk devices built today.

Table 1 shows the performance characteristics of a typical server disk, based on the specifications published by the manufacturer.

Performance of virtual disks cannot be so readily characterized due to the use of disk arrays, RAID organization, and caching. Instead, the performance of virtual disks can be characterized by measuring the performance of the device on a range of workloads using an I/O load

Table 1. Disk hardware performance characteristics.

Performance Characteristic	Speed Rating
Spindle rotational speed	10,000 rpm
Average latency	2.99 ms
Seek time	
Average seek	5.0 ms
Minimum seek (track-to-track)	0.4 ms
Data transfer rate	
Inner track (minimum)	43 MB/sec
Outer track (maximum)	78 MB/sec

generator. For purposes of illustration, the Iometer (pronounced, "I ahm et er") disk benchmarking program is used here. Iometer is a disk benchmarking program originally developed at Intel for the Windows NT platform. It is currently available as an Open Source project, where it can be downloaded for free from a variety of Open Source web sites (see, for example, http://sourceforge.net/projects/iometer/). Iometer has a graphical console that allows you to set up and run benchmark disk I/O workloads. It can be installed and run locally. Or you can install the benchmark agents on remote machines and control them from the console. Using networking, you can drive an I/O load to a shared disk subsystem from multiple client machines, which is important when the capacity of the client machine is itself a constraint. In the examples illustrated, Iometer was run locally.

Determining what I/O workloads to execute that will calibrate the performance of the disk device fairly and accurately is the most important issue in designing the scope of your benchmark.

Synthetic workloads vs. actual or composite workloads

Pure synthetic workloads are simple, one dimensional tests, like a single run made using 100% reads of 4K blocks with random seeks. An actual or composite workload would be one based on measuring your production workloads — something you are going to have to do anyway when you compare the actual performance of the device to the expected performance levels that you derived from the benchmarking — and building a synthetic workload to match.

The approach discussed here advocates using a series of pure synthetic workloads that are used to characterize device performance across three vectors:

Reads vs. Writes. Some devices handle Reads and Writes differently. Physical devices may take slightly longer to write data than read it back. Writes can involve extra integrity checking that elongates the service time. Disk arrays configured as RAID 1 and 5 devices face a significant write performance penalty due to the necessity of having to maintain redundant data. [2]

Cacheability. Cache memory on board the device or the subsystem controller remains one of the most important performance options for disks. Access to data available in cache (a cache *hit*) is usually significantly faster than accessing the physical disk. On a read cache hit, the device (or subsystem) can return data without having to perform the physical operations otherwise necessary to access the media. On a *fast write* to cache, the device or subsystem will return successful completion status as soon as the data is safely stored in cache memory. Using *deferred write-back* caching, the write operation ultimately updates the disk, but not until some point later in time. Since the disk update operation is performed asynchronously, it is invisible to host-based measurement techniques. Instead, the host is only capable of measuring the duration of the fast write to cache.

Cache effectiveness varies as a function of the workload. The cacheability of the workload is better under some circumstances than others. It can be viewed as a continuum, as illustrated in Figure 1, that ranges from a random access pattern that yields very low Read cache hit ratios to highly localized access patterns that produce near 100% cached Read hits. Sequential access are viewed as a special case of a highly localized access pattern, with very similar, but not identical, performance characteristics. Caching controllers can normally detect Read sequential access patterns, leading to read-ahead and delete-behind actions that yield near 100% cache hit ratios with a very small cache footprint. RAID controllers that detect write sequential patterns can defer disk updates until an entire disk stripe and its associated redundant data can be written in an efficient bulk operation.

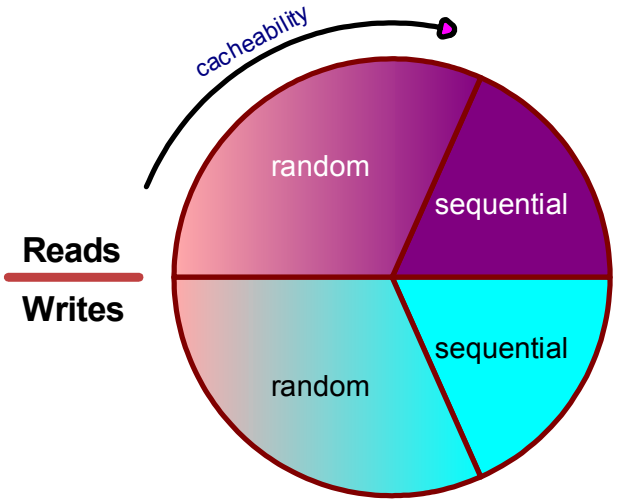


Figure 1. Characterizing I/O workloads

Block size. The data transfer component of disk service time is usually a linear function of the block size of the average transfer. However, block size can also interact with the RAID controller stripe set size. When disk striping is in effect, a Read or Write of a block of data that spans disk segments can be performed in parallel to multiple disks in the array.

A sample test regimen that will characterize disk performance along these three vectors might take the form illustrated in Table 2. Reads and Writes are separately categorized by sequential, random, and highly localized workloads. A range of exponentially increasing block sizes are tested.

TABLE 2. TESTING MATRIX FOR CHARACTERIZING VIRTUAL DISK PERFORMANCE

	Reads			Writes		
Block Size	Sequential	0% Hit Ratio	100% Hit Ratio	Sequential	0% Hit Ratio	100% Hit Ratio
4K	×	×	×	×	×	×
8K	×	×	×	×	×	×
16K	×	×	×	×	×	×
32K	×	×	×	×	×	×
64K	×	×	×	×	×	×
128K	×	×	×	×	×	×

Extraploating to production workloads. Measured results from this compact set of pure, synthetic workloads can then be used to extrapolate to the expected performance of the device on any actual disk workload. To perform this extrapolation, you must characterize the disk workload according to its Read/Write ratio, its cache hit ratios for Reads and for Writes, and the average block sizes of both Reads and Writes. If you are able to characterize your production I/O workloads along these three dimensions, you will be able to extrapolate from the benchmark results for the synthetic workloads to a set of reasonable performance expectations for actual workloads. I/O measurement data that is routinely available on the Windows platform can report accurately on the percentage of Reads, the percentage of Writes, and the average block sizes of Read and Write requests. If your caching controller is able to measure and report on its performance, you will have ready access to the cache hit ratio information that is required. If you do not have cache hit ratio information, you can usually work backwards from the weighted average service time equation mentioned below to derive reasonable cache hit ratio estimates.

If the average block size of the actual request is known, you can extrapolate from these benchmark results using either the sequential or 100% cache hits columns because those results reflect operations that entail data transfer only. In the absence of disk striping, you can extrapolate using a linear regression model, solving for

$$y = mx + b$$

where **y** is the expected data transfer time; **b**, the y-intercept reflects protocol time, cache directory look-up, and other overhead delays common to all I/O operations; **x** is the block size; and **m** is the data transfer rate.

If the cache hit ratios of the actual workload are known, you can extrapolate service time for the actual workload using the weighted service time equation for cached disks:

$$w = \lambda * [p*(w_c) + (1-p)*(w_{disk})]$$

where **w** is the weighted average service time of the device, λ is the arrival rate, p is the probability of cached hit, w_c is the service time of cache hit, and w_{disk} is the service time to disk [3].

An example: measuring the performance of a single disk system.

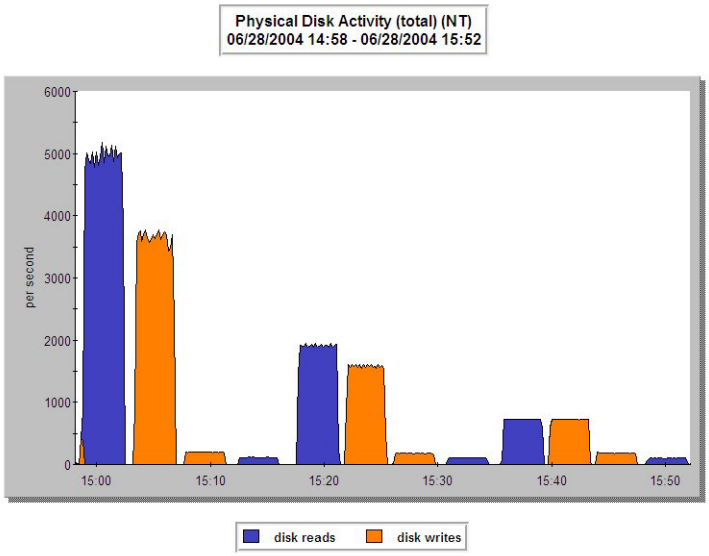
The easiest way to illustrate this approach is to apply it to the characterization of a simple, single physical disk locally attached to one of your servers. Because the single physical disk in this experiment is not accessed via a caching controller, the benchmarking regimen used to characterize its performance can be simplified. It will not be necessary to understand how well the device performs under a range of cacheability conditions. Nevertheless, the device contains an actuator-level buffer that can have a big impact on sequential processing. Consequently, it is important to test the device under conditions that show the influence of the on-board disk cache. On the assumption that physical disk data transfer rates are almost wholly a function of block size, the range of block sizes that need to be evaluated in this case can also be abbreviated. These considerations lead to the adoption of an abbreviated testing matrix that is illustrated in Table 3.

I/O rates for a single physical disk for these twelve test cases are shown in Figure 2. The data shown here is taken from the operating system's performance monitoring interface. (Iometer gathers disk performance statistics, too, which are substantially similar.) The timed trials illustrated here measured performance on tests that each lasted three

	Reads		Writes	
Block Size	Sequential	0% Hit Ratio	Sequential	0% Hit Ratio
4K	×	×	×	×
16K	×	×	×	×
64K	×	×	×	×

TABLE 3. ABREVIATED TESTING MATRIX FOR CHARACTERIZING PHYSICAL DISK PERFORMANCE

Figure 2. Disk throughput for twelve test cases.



minutes in duration, where each separate trial was also preceded by a 30-second "warm-up" period where no measurements were generated. The stability of these results suggest that the test period selected could well have been shorter without significantly biasing the results. In my experience, tests of one minute in duration are normally sufficient. Measurements using tests of shorter duration allow you to gather a full battery of test results for a physical or virtual device in one hour or less.

Sequential access

The measurement results show the significant difference in performance levels when sequential access is used. At the 4 KB block size, the physical device is capable of processing in excess of 5000 Read I/O requests per second. For 16 KB blocks, the sequential read rate declines to approximately 1900 I/Os per second. Using 64 KB, the I/O rate falls to 725 per second.

Figure 3 illustrates the relationship between block size and I/O rates for both sequential Read and Write requests. Figure 3 suggests the underlying relationship is non-linear, a hypothesis that can be explored further by obtaining results for intermediate block sizes. Using 4K blocks, the physical disk can process about 30% more Reads than Writes. As the block size increases, the Read and Write sequential I/O rates tend to converge. At 64KB, there is little difference in the Read and Write I/O rates. This strongly suggests the influence of another factor limiting the I/O throughput for large block operations, namely, the capacity of the link, which apparently saturates at higher block sizes.

By assuming that the device is driven to 100% utilization by the benchmark I/O driver, you can calculate the average service time of the requests by applying the Utilization Law. This calculation is illustrated in Figure 4. This chart shows 4KB sequential Read requests being handled in 0.2 milliseconds, on average, with Write requests taking just slightly longer. Accurately measuring disk services time of that magnitude used to be a problem in Windows NT versions 3 and 4 due to the granularity of the system clock. This granularity issue was addressed in Windows 2000 and above with the introduction of a new high precision clock API that times disk I/O events using the full timer granularity provided by the hardware clock.

It should be evident from these results on the sequential tests that performance of the native device is about as good as it gets. The device's built-in caching algorithms deliver a near 100% cache hit rate. Results with controller-resident cache mediating access to the physical disk should be equivalent — they can hardly ever get much better, except for the case where the controller front-end bandwidth exceeds the device interface bandwidth.

You can project the service time for a cache hit as a function of block size on the assumption that the 4K and 16K cases represent two measurement points on a linear

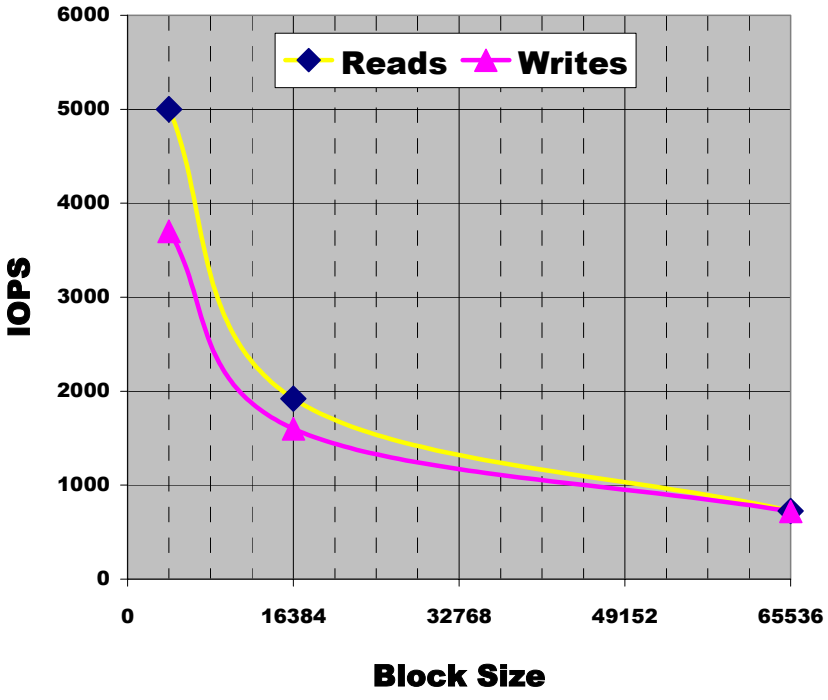
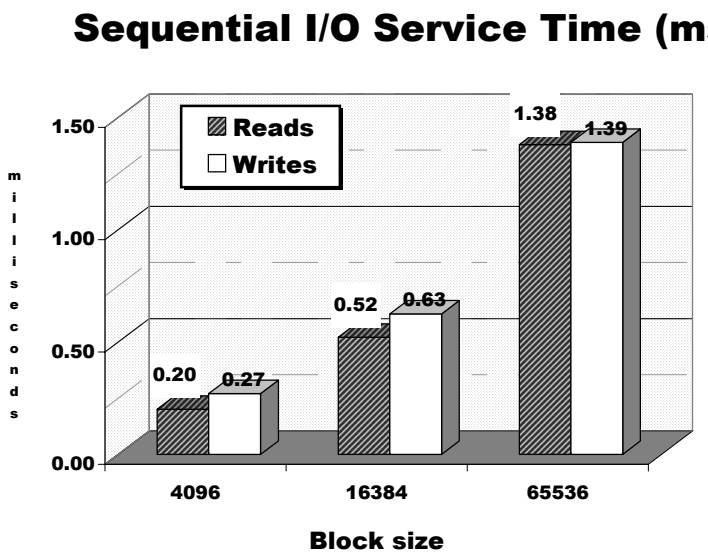


Figure 3. Maximum disk I/O throughput as a function of block size.

FIGURE 4. DISK SERVICE TIMES FOR SEQUENTIAL OPERATIONS.



model. Since the 64K trial shows evidence of saturation at the link, it is best to exclude those results from the linear model. This extrapolation is illustrated in Figure 5 using MS Excel's trendline and forecast functions:

Excel calculates the linear regression models for the straight lines that can be drawn between these two sets of two measurement points. Then the Chart trendline function can be used to forecast the linear model in both directions — forward to 64 KB and backward to 0. Based on linear projection, 32 KB Read and Write cache hits would complete in just under or just over 1 ms., respectively. In the linear equations shown, the y-intercept values can be thought of as *protocol time*, plus any other overhead components that are required to initiate and complete each I/O operation. The 64K Read actual point is also illustrated to emphasize that this calculation is a projection that will likely hold, until some other bottleneck in the path to the device becomes prominent.

Random disk I/O

The Random disk I/O results presented here represent a "best case" set of service time expectations for the device in at least two respects:

- The average seek distance is foreshortened because the Iometer test file is not allocated across the full device.

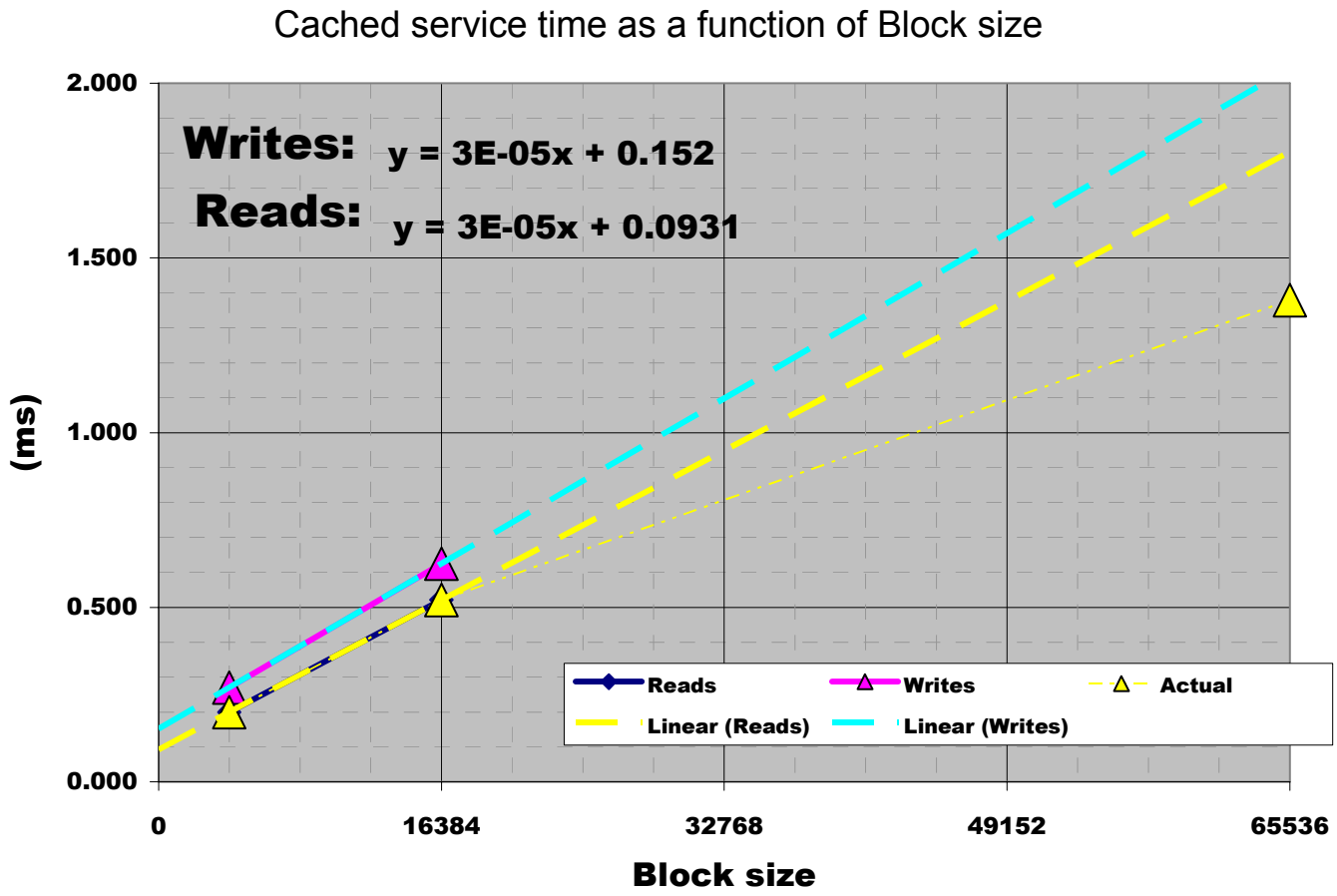
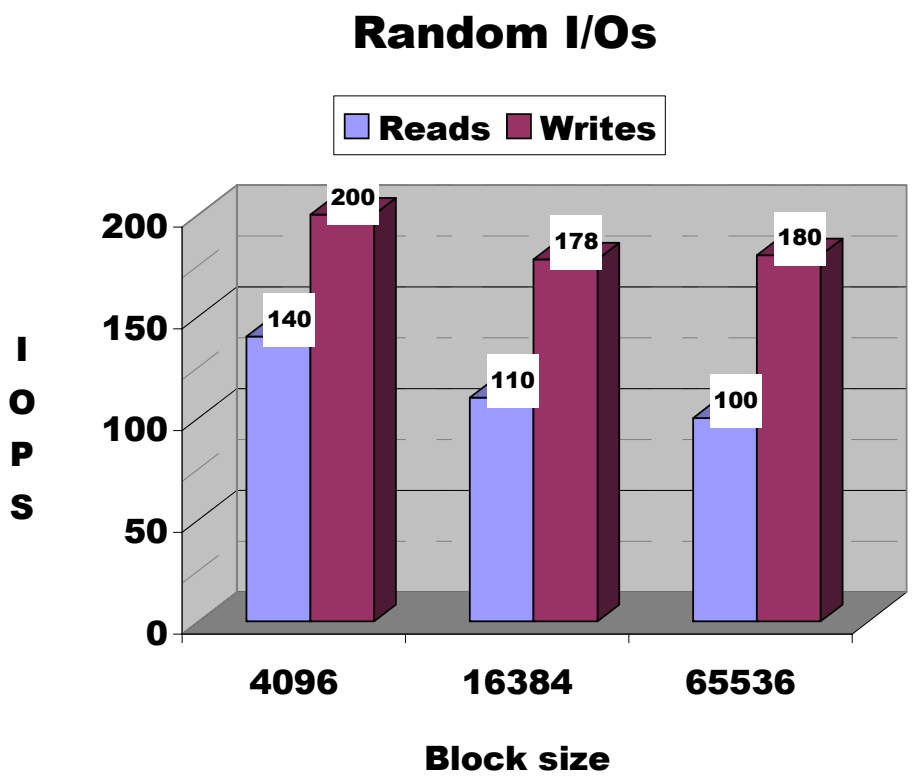


FIGURE 5. LINEAR ESTIMATES OF DATA TRANSFER TIME AS A FUNCTION OF BLOCK SIZE.

FIGURE 6. RANDOM IDISK /O THROUGHPUT



- An attempt was made to reflect load dependent behavior that improves average device service time in the presence of queued I/O requests to the disk.

Both of these considerations reflect an attempt to construct a synthetic workload that is a closer approximation of real world conditions where allocations frequently do not span the entire disk, the actual workload is a mixture of random and zero seeks, and the load dependent behavior of current disk technology is quite significant during periods of severe disk contention. These considerations are discussed in more detail below.

The Random disk Read and Write trials were executed against a file called iobw.tst that Iometer allocated on the C drive, as specified in the Iometer Disk Targets panel. In this instance, a 1,000,000 sector file was specified. At 512 bytes per sector, the target file was 500 MB in size. Iometer then executes random seeks within this file extent. On this test system with a 72 GB hard drive that was less than 15% full, the iobw.tst file is expected to be allocated with little or no fragmentation across contiguous physical disk sectors. The random seeks that Iometer generates are limited to seeks within the scope of the iobw.tst test file. Consequently, random seek performance in this test is subject to shorter average seeks than can be expected if the entire disk volume were full.

The Random disk tests also specified that up to four I/O requests to the device could be issued concurrently. This emulates SCSI command tag queuing to a queue depth of 4. When it is enabled, SCSI command tag queuing can significantly improve the device's performance under peak loads, as demonstrated in [4]. When SCSI command tag queuing is enabled, the device is free to select the queued request that can be serviced fastest next, which results in the device minimizing seek and rotational delays by selecting that queued request that is closest to the current arm position. When SCSI command tag queuing is enabled, the disk is best modeled as a *load dependent server* [5], where its average service time will decrease under higher loads. Limiting the disk queue depth to 4 imitates a heavily loaded device during a peak period when SCSI command tag queuing is beneficial, without so overloading the device that it becomes a performance bottleneck.

The Random I/O throughput results that were measured are illustrated in Figure 6. Random writes uniformly outperform random reads because the disk actuator buffer is more effective for write operations. Write data is passed to the disk buffer at full interface speed, which exceeds the data rates the physical media supports. Random write I/O throughput also appears to be relatively insensitive to changes in block size in this series of tests. Uniform results for random Writes of varying block sizes can be expected so long as the size of the block being written does not exceed the size of a disk track. On the other hand, random Read average I/O service time increases 40% from 7.1 ms. for 4K blocks to about 10 ms. for 64K blocks.

Device service time expectations summary

Table 4 summarizes the device service time expectations for this series of twelve tests.

TABLE 4. PHYSICAL DISK I/O SERVICE TIMES

Block Size	Reads		Writes	
	Sequential	0% Hit Ratio	Sequential	0% Hit Ratio
4K	0.20	7.14	0.27	5.00
16K	0.51	9.09	0.63	5.62
64K	1.38	10.00	1.39	5.56

Queue time expectations

A *queue time* expectation should be added to the service time to calculate an expected device *response time*. After measuring the device service time on a range of workloads, you can establish reasonable queue time expectations using simple queuing models. Figure 7 illustrates modeling device performance using a simple M/M/1 queue, where queue time, W_q, can be estimated as a function of the service time, W_s, and the device utilization, u, according to the following formula [6]:

$$W_q = (W^s*u)/(1-u)$$

For the sake of a more simplified presentation, calculations for only selected device service time values from Table 3 are displayed here.

Comparing actual measurements to expected results

Once you are able to establish a baseline set of reasonable performance expectations for your disks, you are ready to compare actual measurements to these expected results. If actual measurements do not depart very much from expected values, there is very little reason to believe that disk configuration and tuning strategies will bring much benefit. If the gap between measured performance and expected performance is substantial, then understanding the nature of the gap will help you to select an appropriate tuning strategy. It is important to understand whether this gap is due to disk service time that is significantly worse than expected or if it is due to significant queuing, either of which can be improved through tuning.

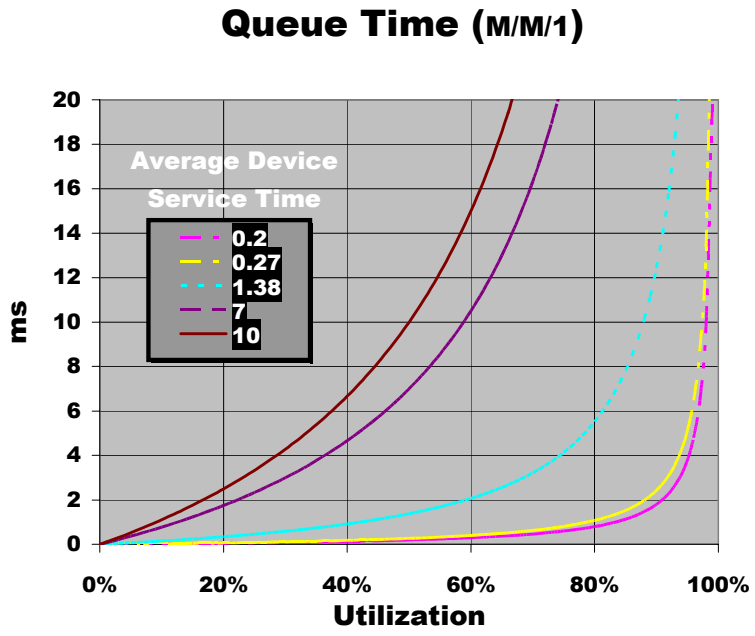
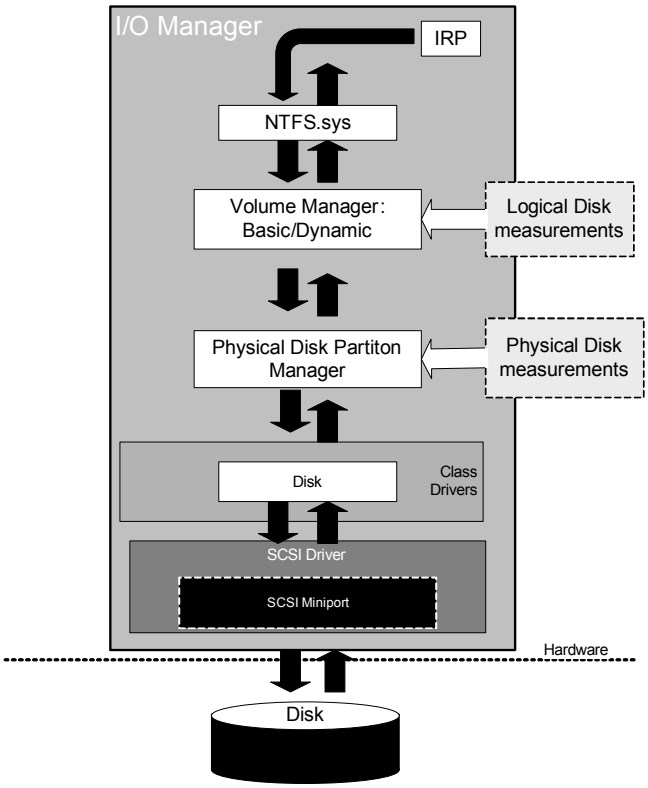


Figure 7. Estimated queue time (M/M/1).

Figure 8. The I/O Manager stack in Windows Server 2003, showing the layers where the disk performance measurements are take,.



Two measurement layers embedded into the I/O Manager device driver stack are the source for all disk performance statistics in Windows, as illusrtated in Figure 8. One layer provides measurements of logical disks, while the other is responsible for measuring physical disks. (Of course, as emphasized above, what the OS views as a physical disk entity, may be a virtual disk instead.) Currently, in Windows Server 2003, the measurement functions are incorporated directly into the appropriate I/O Manager functional layers, as illustrated in Figure 8.

The physical disk partition manager function is performed in Windows Server 2003 by partmgr.sys. The logical disk management functions are performed by dmio.sys. Both logical and physical disk measurements are enabled by default in Windows 2003 — indeed, there is no way to turn off these essential performance measurements in the latest version of the Windows OS.

Prior to Windows Server 2003, disk performance measurements were optional. Statistics were gathered by a separate measurement layer, diskperf.sys, that had to be enabled manually using the **diskperf** command. In Windows 2000, the Physical Disk measurements were enabled by default, while the Logical Disk

measurements had to be enabled manually using the **diskperf -yv** command. In earlier versions of Windows, both sets of disk measurements were disabled by default and had to be enabled manually. Many system administrators have leaped to the assumption that the measurements were made optional due to their impact on performance and would refuse to turn them on. The fact that the measurements are now standard features of the operating system should allay those unfounded fears once and for all.

A more persistent source of confusion in the Windows disk performance measurements arises due to a mixture of measured values that you can rely on and calculated values that are often misleading. The metrics that the disk performance measurement layer measures directly are cataloged in Table 5, which was extracted from the documentation provided in the platform SDK: The measurements taken supply throughput (Disk Read Bytes/sec, Disk Write Bytes/sec), device response time (Avg. Disk sec/Read, Avg. Disk sec/Write), the inverse of device utilization (% Idle Time), and arrival rates (Disk Reads/sec, Disk Writes/sec), broken down by Read and Write operations. In addition, the Current Disk Queue Length counter supplies an instantaneous measure of the number of outstanding requests, including both those requests in service at the device and those that are queued in the device driver stack.

All the remaining disk performance counters that are available in Windows are derived from these basic measurements. Disk Transfers/sec, for example, is the sum of Disk Reads/sec + Disk Writes/sec.

Several of the derived measurements are a source of much confusion. These include % Disk Read Time, % Disk Write Time, and % Disk Time, which are also inaccurately named. From their names alone, the % Disk Time counters suggest that they represent measures of disk utilization. The official Explain text that accompanies these counters inside the System Monitor applet reinforces the confusion: "% Disk Time is the percentage of elapsed time that the selected disk drive was busy *servicing* [emphasis added] read or write requests." Unfortunately, the calculation used to derive % Disk Time is based on neither service time nor device utilization. To make matters worse, the % Disk Time counter is the single counter in both the Physical and Logical Disk objects that is recommended for use when you first start up a System Monitor reporting session.

TABLE 5. THE METRICS THAT THE DISK PERFORMANCE MEASUREMENT LAYER MEASURES DIRECTLY

Metric	Description	Counter Name
BytesRead	Number of bytes read.	**Disk Read Bytes/sec**
BytesWritten	Number of bytes written	**Disk Write Bytes/sec**
ReadTime	Time it took to complete the read	**Avg. Disk sec/Read**
WriteTime	Time it took to complete the write	**Avg. Disk sec/Write**
IdleTime	Specifies the idle time	**% Idle Time**
ReadCount	Number of read operations	**Disk Reads/sec**
WriteCount	Number of write operations	**Disk Writes/sec**
QueueDepth	Depth of the queue	**Current Disk Queue Length**
SplitCount	Number of split I/Os. Usually an indicator of file fragmentation	**Split IO/sec**

Over a single measurement interval, % Disk Time is calculated as follows:

% Disk Time = MIN(100,(ReadTime+WriteTime)*100/Duration)

The sum of ReadTime + WriteTime represents the elapsed time of all completed I/O operations during the interval. Since these measurements include any queuing delays, for busy devices the sum of ReadTime + WriteTime can easily exceed the interval duration, which would lead to the System Monitor reporting values in excess of 100%. Reporting derived values of % Disk Time in excess of 100% must have troubled the developers of the initial production releases of Windows NT, so they capped the values this counter reports at 100%.

Windows NT version 4.0 subsequently introduced a set of similar Avg. Disk Queue Length counters based on the formula:

% Avg. Disk Queue Length = (ReadTime+WriteTime)/Duration)

which effectively removed the artificial capping. This formula is, of course, a variation of Little's Law and is described appropriately in the accompanying Explain text. Nevertheless, the % Disk Time counters were retained, which was probably a mistake.

Beginning in Windows 2000, the disk performance layer measures disk utilization, albeit indirectly by calculating % Idle Time. Using % Idle Time, it is possible to derive disk utilization, then apply the Utilization Law to calculate disk service time, and, finally, separate device service time from the measured value of disk response time to calculate disk queue time. The straightforward calculations to produce these performance statistics is illustrated below, which shows the fields that are directly measured in italics to distinguish them from the derived values you need to calculate:

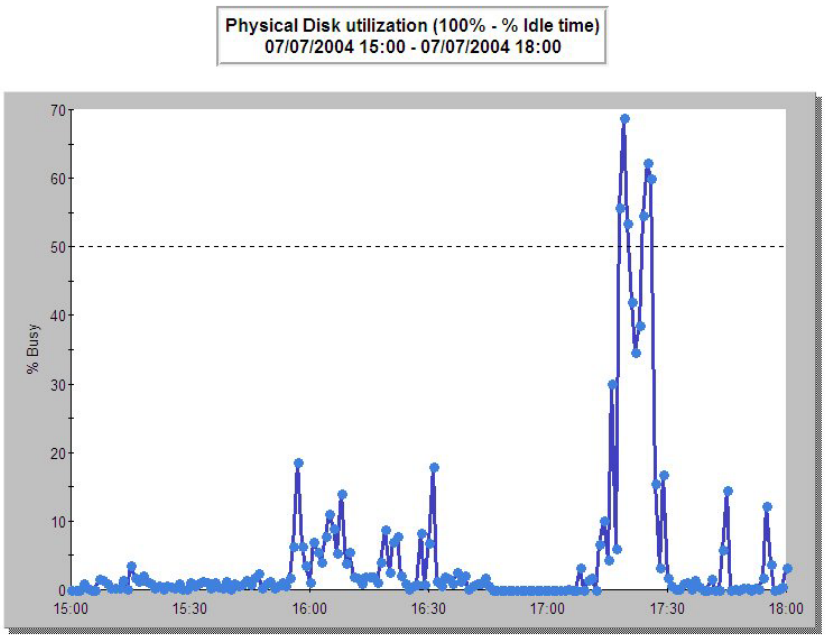
Disk utilization = 100 - *% Idle Time*

Disk service time = Disk utilization + *Avg. Disk transfers/sec*

Disk queue time = *Avg. Disk secs/Transfer* - disk service time

Decomposing the disk response time measurements that Windows provides into service time and queue time allows you to determine if the disk is performing poorly or if the disk is overloaded. Of course, the actual service

Figure 9. Disk utilization is derived from % Idle Time.



time reported should be compared against the reasonable service expectations that you have derived empirically to see if there is a disk service time gap of a magnitude large enough to be of concern.

Figure 9 illustrates the first of these calculations, showing disk utilization over a three-hour period for the same physical disk that was used in the earlier benchmarks. The disk is lightly loaded for the most part, but there is a period of heavy I/O activity shortly after 17:00 hours where disk utilization rises above 50% and response time starts to deteriorate.

Figure 10 completes the disk performance picture by showing disk service time and queue time over the same three hour period.

Figure 9 and 10 clearly show that at low I/O rates disk response time is consistently under 3-5 ms., with very little queuing delay. Compared to the disk service time expectations for this disk that we examined earlier, these service times are well within an expected range of values for a disk I/O workload with up to 50% sequential access. However, as the disk gets consistently busier, there is a greater chance for queuing delays to occur. During peak loads, queue time delays are larger than device service times, with several intervals showing disk response times in excess of 15 ms., double the reasonable performance expectations for the disk that were established experimentally.

Disk performance improvement strategies

Adopting an effective disk tuning strategy begins with understanding where your disk bottleneck is. As discussed in the previous section, "Comparing actual measurements to expected results," you need to find out if the disk is performing poorly (if service time > expectations) or if the disk is overloaded (if queue time > service time). Some configuration and tuning strategies address the problem of poor disk service time. Improving disk service time reduces the load on the disk, so indirectly these strategies also reduce queuing delays. Other strategies are primarily aimed at workload balancing, which has very little to do with improving disk service time, but directly reduces queuing delays. If disk service time is poor, a strategy that attacks queuing delays will yield little improvement. If disk queue time delay is a major factor, a strategy aimed at balancing the disk

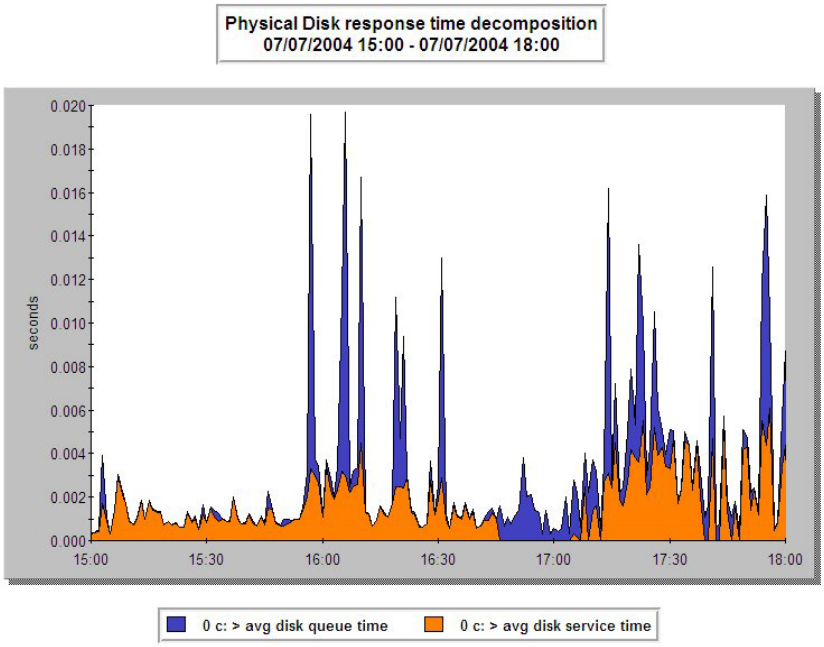


Figure 10. Disk service time and queue time.

workload better across multiple disks is often the easiest and simplest path to improving performance.

This section briefly discusses the most important and most common configuration and tuning strategies that are employed on any platform, including Windows, to improve disk performance. For each strategy, we indicate whether it is primarily focused on improving disk service time or disk queue time.

The disk configuration options considered include:

a. Deploying faster disks,
b. Spreading the disk workload over more disks,
c. Using disk array technology and RAID to spread the disk workload over more disks, and
d. Implementing cached disk controllers.

In addition, the following disk tuning strategies are briefly discussed:

a. Using memory-resident buffering to reduce physical disk I/Os
b. Eliminating disk hot spots by balancing the I/O workload across multiple disks, and
c. Defragmenting disks on a regular basis to increase the amount of sequential disk processing.

Table 6 summarizes the discussion that follows, identifying each strategy with the aspect of improved disk performance — service time or queue time — that it most effectively addresses.

The next section briefly considers these disk performance configuration and tuning options in more detail in the context of typical Windows Server machines.

TABLE 6. DISK TUNING STRATEGIES

Strategy		Performance Impact
Configuration	**Faster disks**	Directly improves disk service time. Indirectly reduces queuing.
	More disks	Directly reduces queuing.
	Disk arrays	Directly reduces queuing by balancing the load across multiple disks. RAID configurations suffer a Write performance penalty.
	Cached disks	Directly improves disk service time. Indirectly reduces queuing.
Tuning	**Host-based caching**	Reduces the overall physical disk load. Boosts the relative Write content of the physical disk workload. May increase disk service time.
	Load balancing	Directly reduces queuing.
	Defragmentation	Directly improves disk service time. Indirectly reduces queuing.

Performance-oriented Disk Configuration options

Faster disks. Disk manufacturers build a range of devices that span a wide spectrum of capacity, price, and performance. You may be able to upgrade to disk devices with improved seek time, faster rotational speeds, and higher throughput rates. Faster devices will improve disk service time directly. Assuming I/O rates remain constant, then device utilization is lowered and queuing delays are reduced. Note that the highest performing disks are also the most expensive. A device that improves disk service time by 20-30% may be 50% more expensive than standard models.

When the other strategies under consideration fail for one reason or another, buying faster disks will generally always produce *some* improvement. Of course, if you are already using the fastest disk devices available, you will have to resort to a different strategy.

More disks. When performance is an important consideration, you need to purchase higher cost, faster disks and more of them. If you are running with the fastest disks possible and you are driving utilization consistently above 50% busy, add more disks to the configuration and spread the I/O workload evenly across these additional disks. Spreading the load across more disks lowers the average disk utilization, which then indirectly leads to reduced queuing delays.

Disk arrays. The simplest way to spread the I/O workload across multiple disks is to install array controllers that automatically stripe data across multiple disks. If you are also interested in adding fault tolerance to large disk configurations, then you should be aware of the Write performance penalty associated with RAID technology. Unless you are reading and writing very large blocks, you should not expect that using disk arrays will improve device service time significantly. But by balancing the workload automatically across all the disks in the array, you eliminate disk hot spots and reduce overall queue time delays.

Cached disks. Cached disk controllers can often mask device latency and improve disk service time dramatically. Cache hits eliminate all mechanical device latency. Data can be transferred from the cache to the channel at full channel speeds, which is usually faster than data can be read or written to the disk media. For RAID controllers, consider battery-backed caches that are

especially effective in masking the device latency associated with RAID write operations.

Disk Tuning Strategies

Host-resident cache. The best way to improve the performance of applications constrained by disk I/O is to eliminate as many physical disk I/Os as possible using RAM-resident cache buffering. Replacing an I/O operation to disk — even one to the fastest disk drive around — with host memory access to disk data cached in RAM achieves more speed-up than any other tuning strategy considered here. Providing more host memory for larger application database and file caches should increase cache buffer hit rates and result in reduced physical disk activity.

Be aware that sometimes RAM is available, but Windows server application configuration parameters restrict the application from allocating additional database buffers.

The most effective memory-resident buffering succeeds in eliminating most disk Read activity, but no amount of host-resident buffering can eliminate all I/O to disk. Because file and database updates must be propagated to the physical disk eventually, Write activity as a percentage of overall disk activity tends to increase as host-memory caching gains in effectiveness. Effective caching also tends to increase the burstiness of the physical disk operations that remain due to the characteristic behavior of Lazy Write flushes.

Balance the disks. Disk storage capacity and disk back-up and restore are usually the primary considerations when you set up a new server, not disk performance. But most system administrators try to lay out their files initially by balancing the disk workload across available volumes. This is difficult to do well initially without the experience of knowing how these disks are accessed in a real production environment. Even if you have done a good job initially in balancing the I/O workload across the available disks, changes in the configuration and the workload are likely to create imbalances over time. Whenever one disk in the configuration becomes overloaded, its disk queue elongates and disk response time suffers. When this occurs, redistributing files and databases to better balance the I/O load will reduce disk queuing and improve response time.

Note: one of the easiest ways to ensure that the I/O load remains balanced across multiple disk drives is to use array controllers that automatically stripe data over multiple disks. Both RAID 0/1 and RAID 5 disk arrays spread I/O evenly across multiple disks.

Defragment disks to increase sequentiality. As file allocations start to fill up the file system, files are no longer stored on the disk in contiguous sectors. If the sectors allocated to the file are not contiguous, then attempts to read the file sequentially (from beginning to end) require more disk head movement. Instead of sequential operations that need no disk arm seeks between successive disk requests, numerous and long disk seeks back and forth across the disk are required to access successive fragments of the file. Under these circumstances, disk service time increases substantially above reasonable expected levels. Moreover, given how fast disk drives with built-in cache buffers can process sequential requests, as illustrated above, any increase in the proportion of sequential disk requests has the potential to speed up disk processing considerably. Long-running sequential processing workloads benefit the most from defragmentation. These include file transfers, disk-to-tape back-up, and other lengthy file copying requests.

Summary

This paper provides a framework that emphasizes a simple approach to disk performance and tuning on the Windows platform. In a storage networking environment what appears to the operating system as a physical disk entity is likely to be a much more complex object to analyze. This paper advocates a simple, direct measurement approach in order to establish a reasonable set of disk performance expectations for these devices. A full range of benchmark workloads that will characterize its performance in a comprehensive fashion can be executed against the physical disk entity in an about an hour or less. Performance expectations established through benchmarking can then be compared to actual disk performance statistics where the disk response times that Windows measures are decomposed into measures of service time and queue time. This response time decomposition is a crucial step. It allows you to determine if the disk is performing poorly or if the disk is overloaded, or both, which then allows you to choose an appropriate optimization strategy to correct the problem.

References.

[1] A. Brandwajn and D. Levy, "A study of cached RAID5 I/O," CMG *Proceedings*, Dec. 1994.

[2] M. Friedman, "RAID keeps on going and going…," IEEE *Spectrum*, April 1996.

[3] M. Friedman, "DASD access patterns," CMG *Proceedings,* Dec 1983.

[4] M. Friedman and O. Pentakalos, *Windows 2000 Performance Guide*, O'Reilly Associates, February, 2002.

[5] D. Menasce, V. Almeida, and L Dowdy, "*Performance by Design* : *computer capacity planning by example*, Prentice Hall PTR, 2004.

[6] A. Allen, *Probability Statistics and Queueing Theory with Computer Science Applications*, 2nd Edition, Academic Press, 1990.