Disk Subsystem Performance Evaluation: From Disk Drives to Storage Area Networks

Thomas M. Ruwart
University of Minnesota
Laboratory for Computational Science and Engineering
356 Physics
116 Church Street SE
Minneapolis, Minnesota 55455
tmr@tc.umn.edu
tel +1 612 626-8091
fax +1 612 626-0030

Abstract

Disk subsystems span the range of configuration complexity from single disk drives to large installations of disk arrays. They can be directly attached to individual computer systems or configured as larger, shared access Storage Area Networks (SANs). It is a significant task to evaluate the performance of these subsystems especially when considering the range of performance requirements of any particular installation and application. Storage subsystems can be designed to meet different performance criteria such as bandwidth, transactions per second, latency, capacity, connectivity, …etc. but the question of *how* the subsystem will perform depends on the software and hardware *layering* and the number of layers an I/O request must traverse in order to perform the actual operation. As an I/O request traverses more and more software and hardware layers, alignment and request size fragmentation can result in performance anomalies that can degrade the overall bandwidth and transaction rates. Layer traversal can have a significant negative impact on the observed performance of even the fastest hardware components. This paper walks through the Storage Subsystem Hierarchy, defining these layers, presents a method for testing in single and multiple computer environments, and demonstrates the significance of careful, in-depth evaluation of Storage Subsystem Performance.

1 Introduction

Disk subsystem manufacturers make many claims about the performance of their products. However, these performance claims cannot be taken out of context of the final implementation. Rather, it is necessary to evaluate the performance of disk subsystems within a configuration that is as close as possible to the actual configuration in which the subsystem will ultimately be employed. Such an evaluation requires a benchmark program that can closely mimic the access patterns of the intended applications and provide results that are *meaningful* and *reproducible*.

This paper presents examples of disk I/O performance anomalies and describes the cause of these problems as well as strategies to minimize their effects. The paper begins by describing the hardware and software components that an I/O request must traverse in order to move data between the computer system memory and the storage media. The Testing Philosophy and Methodology is then presented that describes how and why the individual components are evaluated as well as basic assumptions and tradeoffs that must be made in order to provide meaningful and reproducible results. The performance of the entire Storage Subsystem Hierarchy is then be evaluated under ideal conditions. This sets an upper bound for performance of the system as a whole. Knowing this upper performance limit, it is possible to address the *Impedance Matching Problem* which examines various performance anomalies and their causes. Example test results are given throughout the paper to illustrate relevant concepts.

2 The Storage Subsystem Hierarchy

The Storage Subsystem Hierarchy describes the levels of hardware and software that an I/O request must traverse in order to initiate and manage the movement of data between the application memory space and a storage device. The I/O request is initiated by the application when data movement is required either explicitly in the case of file operations or implicitly in the case of memory-mapped files for example. The request is initially processed by several layers of system software such as the file system manager, logical device drivers, and the low-level hardware device drivers. During this processing the application I/O request may be split into several inter-related "physical" I/O requests that are subsequently sent out to the appropriate storage devices to satisfy these requests. These physical I/O requests must pass through the Physical Connection Layer that makes the physical connection between the Host Bus Adapter on the computer system and the storage device. After arriving at the storage device, the I/O requests may be further processed and split into several more I/O requests to the actual storage "units" such as disk drives. Each Storage Unit processes its request and data is eventually transferred between the storage unit and the application memory space. The following sections present a more detailed description of each level in the hierarchy with respect to its function and performance implications.

2.1 Computer System

The Computer System is a critical piece of the Storage Subsystem Hierarchy in that it encapsulates all the software components and the necessary interface hardware to communicate with the Physical Connection layer (i.e. the Host Bus Adapter). The components within the Computer System include the processors, memory, and internal busses that connect the memory to the processors and to the Host Bus Adapters. The performance characteristics of each of these major components (i.e. the clock-speed, number of processors, processor architecture, memory bus bandwidth, …etc.) plays a significant role in the overall performance of the Storage Subsystem as will be demonstrated in a later section. However, given the fastest hardware available, the Storage Subsystem will only perform as well as the underlying software, starting with the Application Program

2.2 Application Program

The term "Application Program" as it is used here is any program running on the Host Computer System that requires data movement between the memory in the host computer and a Storage Unit. Application programs can be either typical User programs or can be parts of the Operating System on the host computer such as the paging subsystem. In any case, these programs have the ability to make I/O requests to any of the lower-level layers in the hierarchy if the Operating System provides an appropriate programming interface to do so. For example, the benchmark program used to gather statistical data for this study can access a storage unit through the file system manager, the logical volume device driver(s), or through the disk device drivers. In general, Applications that can manage and access the lower levels of the hierarchy achieve better performance than Applications that must traverse through the higher level layers such as the File System Manager.

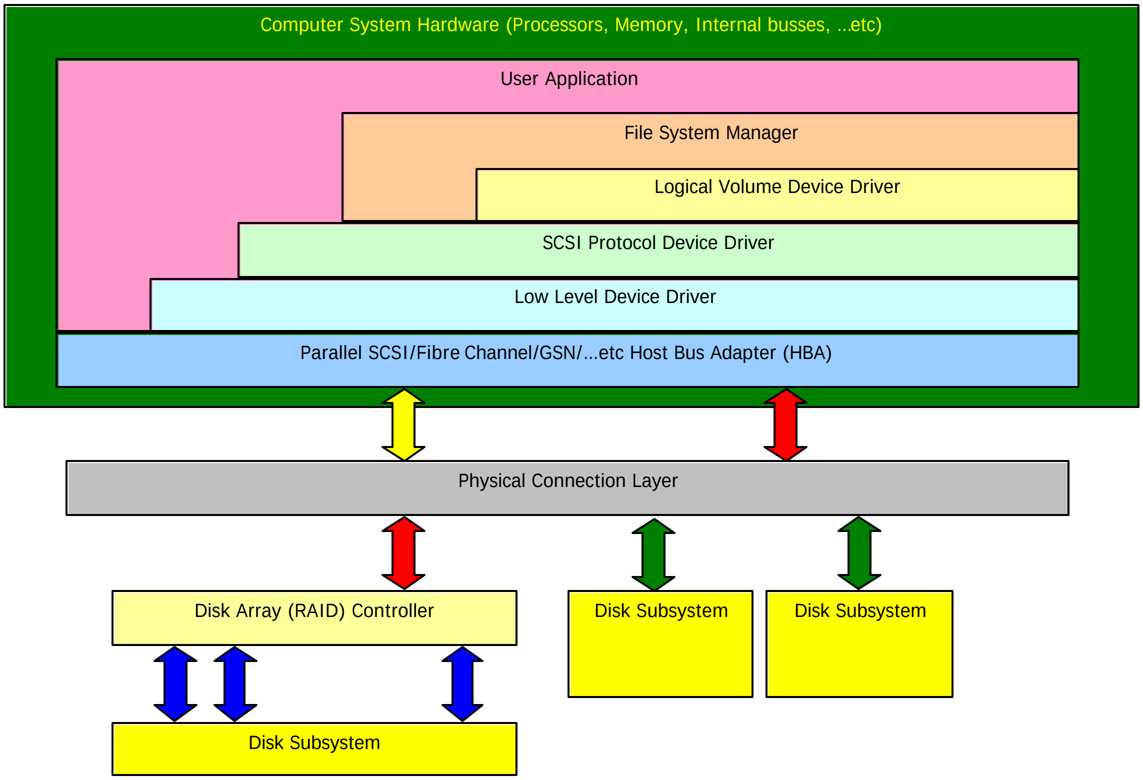

Computer System Hardware (Processors, Memory, Internal busses, ..etc)
User Application
File System Manager
Logical Volume Device Driver
SCSI Protocol Device Driver
Low Level Device Driver
Parallel SCSI/Fibre Channel/GSN/...etc Host Bus Adapter (HBA)
Physical Connection Layer
Disk Array (RAID) Controller
Disk Subsystem
Disk Subsystem
Disk Subsystem

Figure 1. The Storage Subsystem Hierarchy

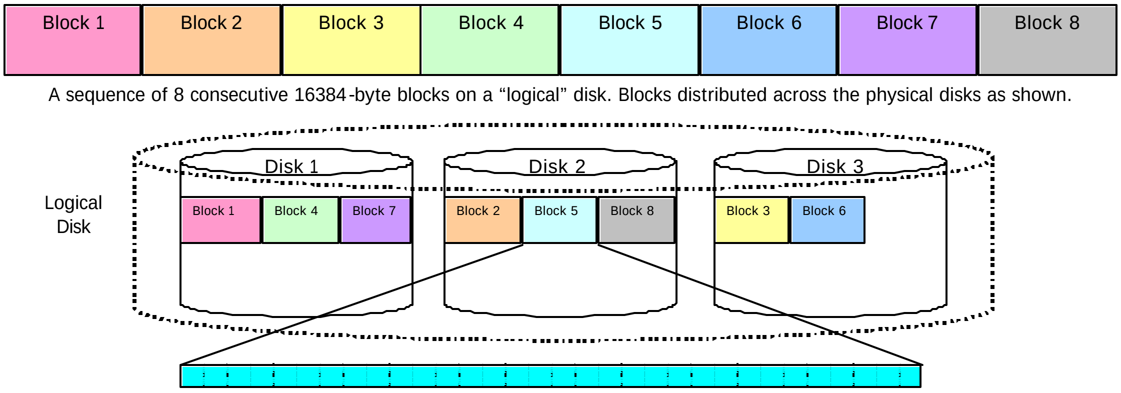
2.3 File System Manager

The File System Manager is mentioned here for completeness but it is not used in any of the testing performed for this paper with one exception. The File System Manager provides a level of abstraction for the Application Program in order to simplify the process of accessing data for the application programmer. Because of the amount of "indirect" I/O processing that can accompany a single Application I/O request (such as space allocation, inode lookups, …etc.), I/O performance testing "through" the File System Manager can become enormously complex and produce misleading results. Therefore, it is beyond the scope of this paper to include any testing through the File System Manager or to report the performance idiosyncrasies of the File System Manager itself. The I/O benchmark program used in this study always bypasses the File System Manager for data movement operations. The one exception to this occurs in the testing that was performed for multi-host access to a set of shared disks in a Windows NT environment. For these tests, a Shared File System was necessary in order to gain "shared" access to the disks from all the hosts involved in the tests. Unfortunately, this functionality is not yet easily available in the Device Drivers available under NT.

2.4 Logical Volume Device Drivers

The Logical Volume Device Drivers provide a mechanism to easily group storage devices into a single "logical" device in order to increase storage capacity, performance, and/or to simplify the manageability of large numbers of storage devices. The Logical Device Driver presents a single device *object* to the Application. The Logical Device Driver is then responsible for taking a single I/O request from the Application (or the File System Manager) and mapping this request onto the lower level storage devices, which may be either actual storage devices or other logical volumes.

There are many ways to configure a logical volume that consists of multiple underlying storage devices. One common configuration is to stripe *across* (also known as striping *wide*) all the storage devices in an effort to increase available bandwidth or throughput (operations per second). In a wide-striped logical volume, data is laid out on the disk in "stripe units". A stripe unit is the amount of sequential data that is transferred to/from a single storage device within the logical volume before moving to the next storage device in the volume. The stripe unit can be any number of bytes from a single 512-byte sector to several megabytes but is generally a constant within a logical volume (Figure 2).



A sequence of 8 consecutive 16384-byte blocks on a "logical" disk. Blocks distributed across the physical disks as shown.

A single 16384-byte "block" consists of 32 consecutive disk sectors, 512-bytes per sector.

Figure 2. The layout of a Logical Volume.

2.5 I/O Protocol Device Driver

The I/O Protocol Device Driver is responsible for translating the I/O request from the upper level device drivers into a form that fits the I/O protocol used to communicate the request to the underlying storage devices. In general, an internal I/O request consisting of a command (read or write), a data buffer address, and a data transfer length is converted into a SCSI command and will convey this information to the host bus adapter via the low-level device driver and the disk devices.

2.6 Low-Level Device Driver

This device driver takes the high-level information (i.e. the SCSI command) from the I/O Protocol Device Driver and interfaces directly with the host-bus adapter that will perform the actual data transfer between the storage device and application memory. For example, given a PCI-to-Fibre Channel Host Bus Adapter, this device driver will set up the host bus adapter with the address of the SCSI command buffer, the application data buffer, and the target device and then tell the host bus adapter to begin the operation. The host bus adapter will transfer the SCSI command buffer to the intended target device. The target device at some later point will request a data transfer operation that will be managed in part by the host bus adapter. At the end of the entire operation, an interrupt is generated to notify the Low-Level Device Driver of the completion status the I/O operation. Under normal circumstances, the Low-Level Device Driver then propagates the completion status to the upper-level drivers, eventually reaching the User Application Program.

2.7 Physical Connection Layer

This layer defines the hardware that physically attaches the host-bus adapter to the storage device. These connections can be as simple as a single 3-foot cable or as elaborate as a multi-stage communication fabric spanning many miles. In general, there are two types of physical connections: Parallel-busses or Channels and Serial Interfaces. Parallel busses include SCSI and IDE, Channels include IBM 370-type Block-Multiplexer Channels. Bus-type interfaces are most commonly used inside personal computers and other systems for system disks and other peripheral devices that do not require a great deal of performance (i.e. greater than 100 MB/sec). SCSI busses are also used for larger storage configurations that extend outside the physical boundaries of a computer cabinet. However, due to the nature of the Parallel SCSI bus architecture, the length of SCSI busses is severely restricted when compared to that of Serial Interfaces.

The most common Serial Interfaces include Fibre Channel, USB, and FireWire, to name just a few. For disk storage, Fibre Channel is currently the most prominent Serial Interface. Serial Interfaces have distinct advantages over the more traditional Parallel Bus architectures in the number of different connection *topologies* that are possible. These topologies include Point-to-Point, Loop, and Switched Network (Figure 3). Furthermore, the distance limitations of Serial Interfaces tend to be significantly longer than Parallel Busses making it easier to implement in physically large or extended configurations.

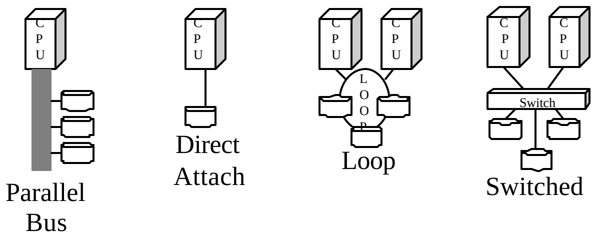


Figure 3. Storage Area Network Topologies.

Point-to-Point connections dedicate a single host connection to a single storage device. This is not the most efficient use of a host connection but does guarantee access to the device via that connection.

The Loop topology, also know as an Arbitrated Loop in Fibre Channel terms, behaves more like a traditional Parallel Bus. However, a Fibre Channel Arbitrated Loop, for example, can more easily accommodate multiple host computers as well as a larger number of storage devices. There are practical limitations on the number of devices and the overall length of a Loop that can be overcome using a Switched Network topology.

The Switched Network topology allows for any number of options in physically connecting the storage devices to the host computers. It is the most flexible in terms of allowing for multiple access paths to a single storage device, multiple host shared access, fault tolerance, and performance. However, this flexibility also means increased complexity in managing all the nodes connected to the SAN, whether they are host computers or disk devices. These multi-host, multi-device configurations are commonly referred to as Storage Area Networks or SANs.

2.8 Storage Device and Storage Units

The last two layers in the hierarchy are the Storage Devices and Storage Units. The distinction is that a Storage Device is made up of one or more Storage Units but can "appear" to be a single device. The example is that of a Disk Array which is a Storage *Device* that contains several individual Storage *Units* (disk drives) but can appear to the system as a single, very large, disk drive. In the case of a disk array, the I/O request is received from the host bus adapter and is divided up into one or more I/O requests to the underlying disk drives. Storage *Units* are individually addressable storage devices that cannot be further subdivided into smaller physical units. The principle example of this is a Disk Drive.

3 Performance Implications and the *Impedance Matching Problem*

Each layer of software and/or hardware between the Application and the Storage Device adds *overhead* and other anomalies that can result in highly irregular performance as viewed by the Application. Overhead is essentially the amount of time it takes for the I/O request to traverse the specific layer. The source of overhead in each layer is specific to a layer and is not necessarily constant within a layer.

An example of this is the overhead induced by the Physical Connection layer. A physical connection consisting of a short cable introduces virtually no overhead since the propagation of a signal at the speed of light over a 3-foot distance is not significant. On the other hand, propagation of a similar signal traversing a 20-mile storage area network through multiple switching units will introduce noticeable overhead [2].

An interesting artifact resulting from the *interaction* of the components in the Storage Subsystem Hierarchy is analogous to the Impedance Matching problem in electrical signal on wires. The term "Impedance Matching" is used as an analogy to what happens when there is a mismatch of operational characteristics between two interacting objects. In an electrical circuit, an impedance mismatch has an effect on the "performance" of the circuit in terms of its gain or amplitude at particular frequencies. In the Storage Subsystem Hierarchy, an "impedance mismatch" has more to do with things like I/O request size and alignment mismatches that have an effect on the performance (bandwidth or transaction rate) of the storage subsystem as viewed by the application. The *effects* of these mismatches can be viewed from several different perspectives including the Application perspective, the Disk Device perspective, and the System perspective. The effects of this phenomenon are presented in the sections that follow.

However, it is first necessary to describe exactly how these effects are identified and analyzed.

4 Testing Philosophy and Methods

When approaching the problem of evaluating a storage subsystem it is important to know and understand the operational boundaries of the applications using the storage subsystem. Performance tests are often run on equipment and results are generated or provided that have no real connection to the actual "application" that will be using the storage subsystem. The evaluation of a storage subsystem is intended to answer the basic question of how well applications perform on a storage subsystem in a given configuration.

There are many approaches to answering this question. One way is to run the application on a specific configuration of the equipment under evaluation. The configuration can be "tuned" until the "performance" is optimized for a specific application. However, this is not always easy to do nor is it an accurate method of testing performance if the behavior of the application is not well understood under all circumstances. Furthermore, the results obtained by testing a single application or a small set of applications may not extend beyond those applications to other applications or even to the same application as it (the application) scales in size, complexity, …etc.

The evaluation method advocated by this paper is based on the idea of testing the individual components of a Storage Subsystem followed by testing various "configurations" of these components. It is essential to first understand the performance characteristics of the individual hardware and software components of the entire storage subsystem before the combined performance of the overall subsystem can be accurately assessed. Successive layers/components of the Storage Subsystem Hierarchy are then added to the evaluation testing and the effects of each addition are recorded.

Each successive layer of the Storage Subsystem Hierarchy adds functionality and/or performance to the application. However, a side effect of each successive layer is to add overhead to each I/O request as well as a significant amount of complexity to the evaluation process. The increase in complexity results from the fact that each successive layer adds new independent variables to the performance tuning equation. As a result, this complexity grows exponentially with each successive layer. Understanding the effects on the performance of each of these variables as well as how the variables interact is the goal of the evaluation process. With this information, it becomes easier to identify the cause of performance problems and to compensate by adjusting these and other related variables.

For example, the evaluation process would start by characterizing the performance of a single disk drive. Multiple disk drives can then be added to the same I/O Channel in order to test the performance limits of the host adapter. Several host adapters can then be added, each with a sufficient compliment of disk devices so as to saturate the system bus that connects the host adapters to the memory subsystem of the computer or to saturate the memory bus itself. In either case, the performance of the computer system internal data bus architecture is characterized. The disk drives can also be integrated with a disk array (RAID) controller and the performance of the RAID controller can be characterized as well.

5 The I/O Spectrum and Performance Metrics

There are several metrics used to gauge the performance of a disk subsystem. The *I/O Spectrum* is a concept that divides the performance metrics into two basic types. At one end of the spectrum is *bandwidth* and the other end is *transactions per second* or *IOPs* (I/O Operations per second). Bandwidth is simply the maximum number of bytes transferred per second. This is generally characterized by relatively few, very large data transfer requests per second. IOPs is a measure of how many relatively small data transfers can be processed by the disk subsystem per second. In general, as the size of the requested data increases, the performance moves from *IOPs* to *Bandwidth* along the I/O Spectrum (Figure 4.)

Transactions per second
Bandwidth

Figure 4. The I/O Spectrum.

Related to the IOPs metric are two other metrics worth mentioning: *Response Time* and *Jitter*. Response Time is simply the time it takes to get a piece of data once the request has been issued. It is important to note that Response Time is not simply 1/(IOPs). For example, if a Storage Device has an IOP rating of 2,000 I/O operations per second, this means that the storage controller can accept 2,000 I/O requests every second and that it can simultaneously deliver 2,000 responses per second. However, once an I/O request is received by the Storage Device the associated response may be the next response out, or it may be the 100th response out, or it may be the 6,000th response out. The associated Response Times for each of these possibilities is 1/2000th of a second, 100/2000? 1/20th of a second, or 6000/2000? 3 seconds.

Jitter is closely related to the Response Time metric. It is a measure of how much the Response Time *changes* over time. For example, given 1000 I/O requests that each have a required response time of 1/30th of a second, jitter measures how many of the 1000 requests failed to meet the response time criteria. Jitter is important in real-time applications that require Response Times to be *consistent*. Such an application is streaming video where the individual video frames must appear before or at the correct time, every 1/30th of a second for example, or the frame is dropped from being displayed.

6 The I/O Benchmark Program

As previously mentioned there are many aspects of performance that are of interest and there are many ways to gather performance data display the information in an easy to understand format. Simply stating that a disk drive can deliver 24MB/sec or 1500 transactions per second does not convey nearly enough information. Rather, the performance of a disk drive as a function of some other variable such as request size or media position is more informative. Furthermore, being able to capture and display this information in a time-correlated manner is useful in understanding the interaction of multiple components within a Storage Subssytem. This is especially important in a shared-access environment where a single computer system does not have the ability to control access to a storage subsystem.

The I/O benchmark program used to gather this information must have several qualities:

- Highly configurable
- Generate "reproducible" results
- Generate "reproducible" usage scenarios
- Very fine degree of tunability

There are many I/O benchmarking programs readily available such as BONNIE, IOZONE, DiskPerf, IOMeter, …etc. These programs all address different aspects of I/O performance and were not sufficiently focused on the fine details of I/O behavior to be used for this study. Therefore, the benchmark program used to generate the results in this paper has been specifically developed over the past several years at the University of Minnesota and contains features necessary to satisfy the criteria mentioned above. This program is called *xdd* and is available from the web site listed in the title of this paper. Xdd is used to measure many of the disk device performance characteristics as well as helping to identify many of the performance anomalies that appear in more complex configurations.

7 Testing Framework

Testing in a multiple-host environment required the creation of a framework to coordinate testing on multiple systems concurrently[4]. The two basic functions of this framework are:

- Accounting for the existence of multiple clocks
- Coordinating the initiation of tests to run concurrently on multiple hosts

Xdd makes use of precise time stamps to quantify and report storage performance characteristics. Each host accessing the shared storage has its own internal sense of time, and without a common reference clock it is impossible to interpret the relationship between tests run on separate hosts. Thus a consistent time base is needed in order to correlate test results generated by separate systems.

7.1 The Reference Clock

Each of the systems used for testing has a clock register that updates at a high frequency, allowing for very precise measurement of elapsed time. The resolution of this clock varies for different systems (ranging from 2 to 80 nanoseconds per tick or so), so clock values are converted to a common time unit (picoseconds) for the purpose of synchronization.

A very simple clock model to establish a common time base. It is assumed that all clocks run at the same, constant rate. Therefore it can be assumed that conversion from a given machine's "local time" to the common "global time" involves only the addition of a constant to the local clock's value. With this simplified model it is only necessary to determine the value of the constant difference between pairs of clocks.

One machine is designated to keep the global sense of time. That machine provides a service with which others communicate to determine the offsets of their own clock from the global time. Each client initiates a request to the server to get the current global time. The difference between the time value returned and the client's local time is recorded as the basis for the offset. This offset is further adjusted to compensate for the propagation delay required to carry the time request and its response over the communication medium. This propagation delay bounds the error in the difference between the estimated and the actual offset between the two clocks. This request/response protocol is repeated a number of times, and use the offset associated the minimum propagation delay as the final offset value.

7.2 Coordination of Concurrent Tests

With a common time base defined, it is possible to coordinate the initiation of tests on different host systems. Xdd is able to determine the time offset for the machine under test. The program is provided an indication of a (global) time at which all tests are to begin. This global time is converted to a localized start time using the offset value. Xdd then polls the high-resolution clock repeatedly until the start time has arrived. At that point test execution begins. Test results generated by individual hosts are buffered during test execution, and saved to disk for later analysis.

8 Disk Device Basics

In order to understand some of these performance anomalies, a short course in disk devices is necessary. It is assumed that the reader has a basic understanding of how Disk Drives are put together in the sense that they contain platters, heads, cylinders, sectors, and lots of 1’s and 0’s. However, it is worth describing some of the more subtle design concepts of a disk drive that have an impact on the performance. These concepts include Zoned Bit Recording, Caching, Rotational Latency, Seek Time, the On-board Disk Processor Overhead, Command Queues, and Disk Arrays.

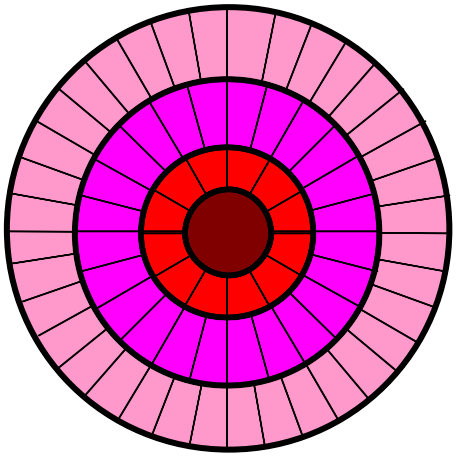
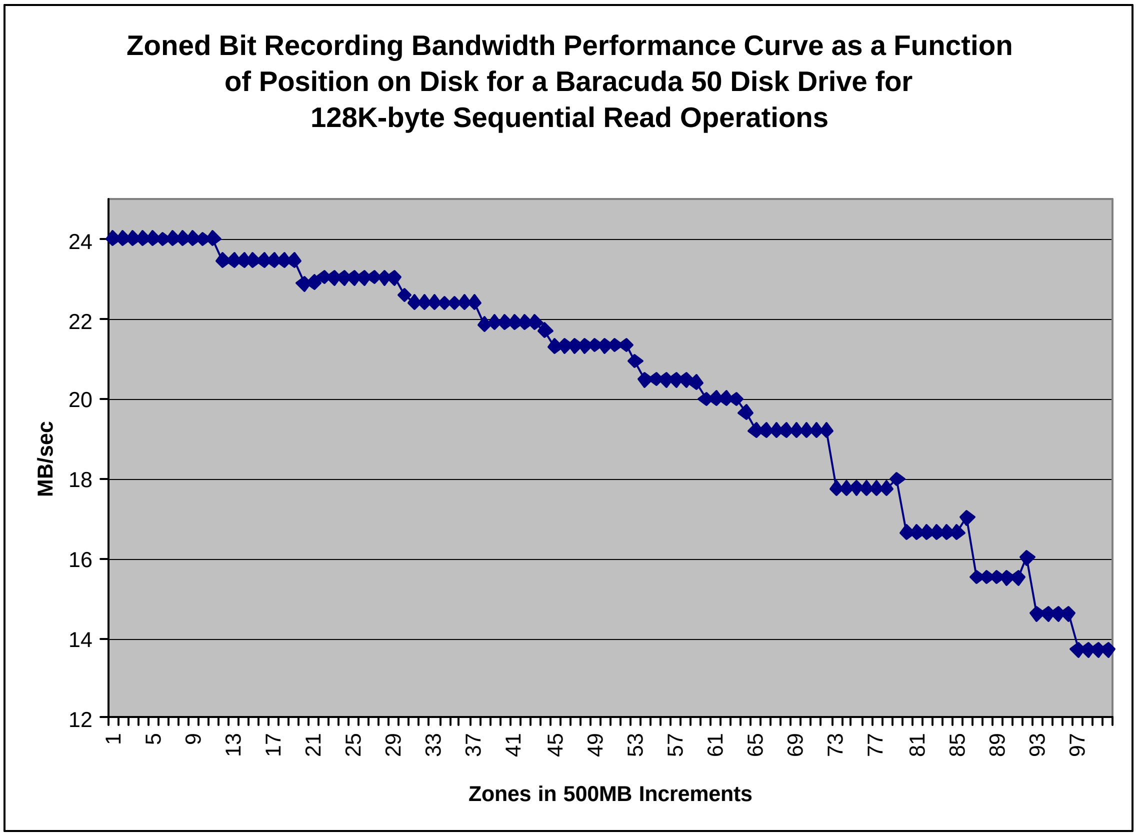


Figure 5. Zoned Bit Recording. Note how the outer band has more sectors than the inner bands.

8.1 Zoned Bit Recording

The data transfer is the rate at which actual *user data* can be read from or written to the media. This transfer rate can vary in such a way that depends on the *physical location* on the disk media where the transfer is to take place. This is due to a recording technique called *Zoned Bit Recording (ZBR)* whereby more data is written on the outer tracks of a disk platter than on the inner tracks. This allows for more efficient utilization of the recording area and hence greater overall capacity. ZBR is used on most current generation disk drives. Given that a disk drive spins at a constant rate, 7200 RPM for example, the outer zones that contain more data will transfer data at a higher data rate than the inner zones that contain less data.

This is clearly demonstrated in Graph1 where the Effective Data Transfer Rate is plotted against the physical position on the platter. Each increment along the X-axis is a 500MB segment of the disk. As data is read from segments successively further into the disk, the data rate at which the data is transferred decreases. However, the decreases are not gradual but are distinct "steps" along the performance curve. Each of the horizontal plateaus is a physical zone on the disk. This graph shows that there are 14 distinct zones on this disk which matches the manufacturer's specification. It is interesting to note that the width of outer zones is larger than the width of the inner zones.

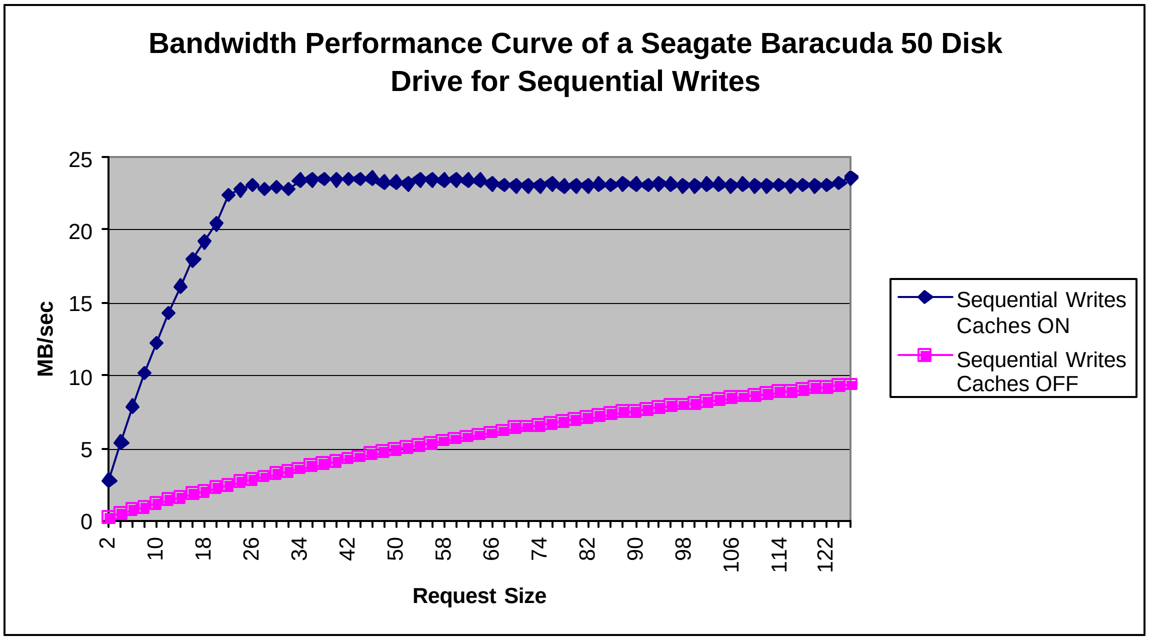


Graph 1.

8.2 Caching

When data is read off the disk media it is stored temporarily in a buffer *cache* before it is sent to the host bus adapter (controller). The data transfer rate from the cache onto the bus is normally done at the speed of the bus that is usually much faster than the transfer rate off the media. The cache can also be used during write operations to accept incoming data and "complete" the write operation before the data is actually written on the media. This can speed up the process of writing small amounts of data to a disk device by a factor of 10-100 since the requesting application does not need to incur the additional overhead of the seek operation and rotational delay associated with writing the data to the disk media.

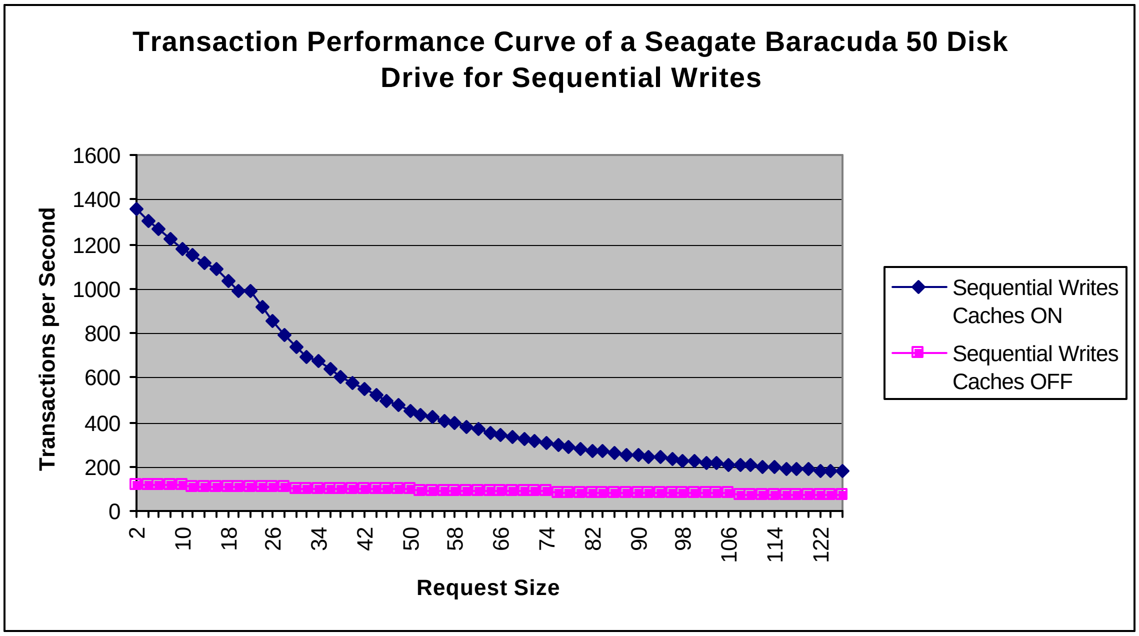
The size of the buffer cache also has an impact on the transfer rate performance of a disk drive. Consider a disk with no buffer cache. The data would then proceed directly from the bus to the media and would be limited to the data transfer rate to the media. If the buffer cache size was increased to one megabyte for example, then data transfers could proceed between the cache and the bus at bus speeds while simultaneously transferring data between the media and the cache at media speeds. Buffer caches can be very helpful when streaming data sequentially off the disk media. After data from a single read request is transferred into the buffer cache, the disk can perform a *read-ahead* operation and continue to transfer subsequent data into the buffer cache in anticipation of the next read request. Without a buffer cache and the read-ahead operation, the subsequent request would arrive and the disk would have to wait for an entire rotation of the disk before the data transfer could begin again.



Graph 2.

Graphs 2 and 3 demonstrate the effectiveness of a Write Cache for purely sequential write operations. The graph plots Bandwidth and Transaction performance as a function of request size. It is clear that the write cache significantly improves the performance of the disk for any size write operation. For small operations, in the 1024-byte per transaction range, the transaction rate is approximately 14 times higher when using the cache for write operations than having the cache disabled.

Graphs 4 and 5 further demonstrate the effects of caches on purely random transactions: reads and writes. These graphs show that random read operations do not benefit from the cache and closely track the performance of non-cached random write operations. However, the cache is still effective in improving the performance of small random write operations up to about 64Kbytes where the performance curve tracks the non-cached performance of both reads and writes.



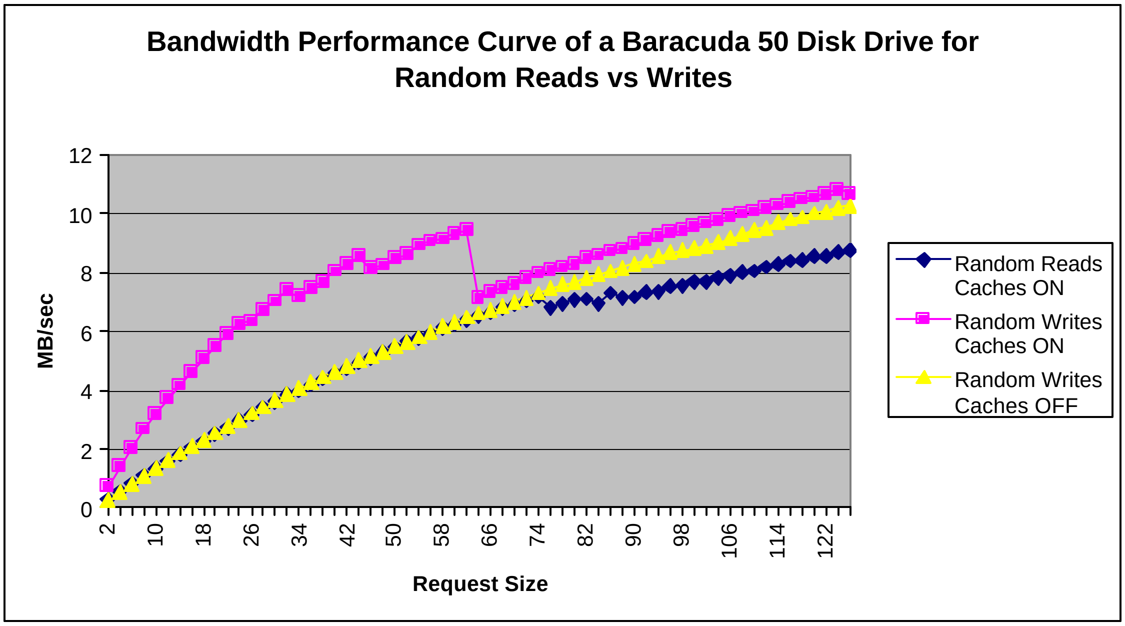
Graph 3.

8.3 Rotational Latency and Seek Time

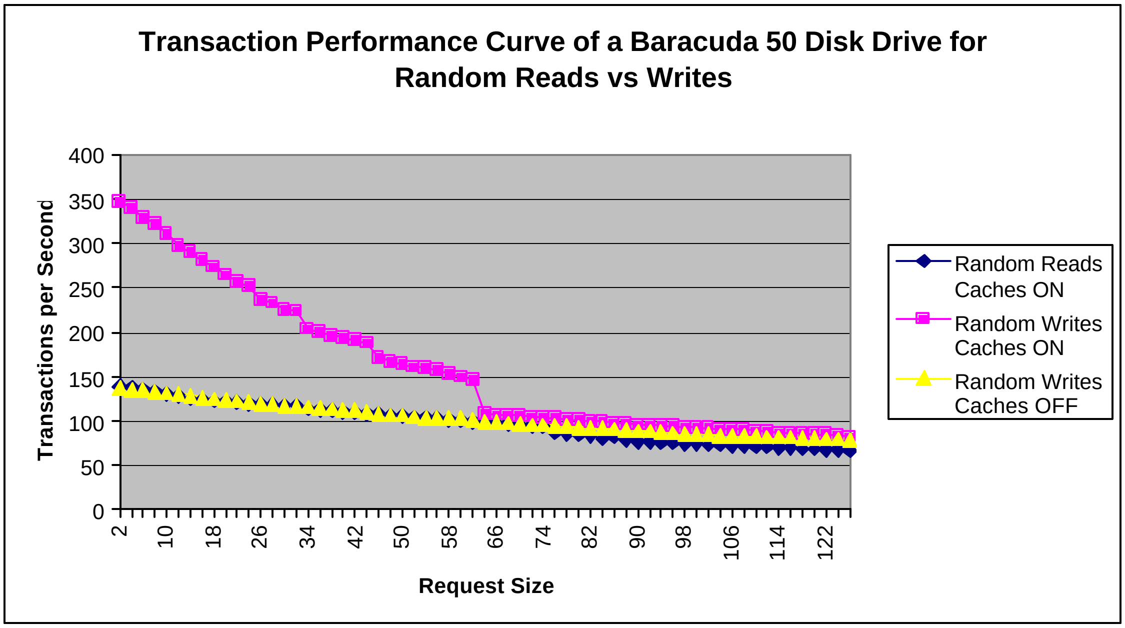
The Rotational Latency is the time that it takes for the disk platter to rotate such that the requested sector is directly under the read/write head. The rotational latency is simply 1 divided by the rotational speed of the disk. Rotational rates for the most common disk drives are 5400, 7200, and 10,000 revolutions per minute. This translates to rotation times of 11.1ms, 8.3ms, and 6ms respectively.

The seek time is the time it takes to position the read/write head over the correct cylinder on the platter. This time can vary by a factor of 10-20 from a single track-to-track seek to a full drive seek (from cylinder 0 to the last cylinder on the disk). Typically seek times range from slightly less than 1 millisecond to about 20 milliseconds for a full seek. Seek operations for write operations take longer than those for read operations because write operations need to seek to the required cylinder and be in perfect alignment before starting the write operation. Read operations however, can start reading before the head completely settles.

Graphs 4 and 5 show the effects of Rotational Latency and Seek Time on read and write performance when the I/O operations are randomly distributed over the disk. Graph 4 plots Bandwidth as a function of Request Size and also shows the effectiveness of the Write Cache on Random Write operations. Graph 5 plots IOPs as a function of request size for the same access pattern. It is clear that for this particular model of disk drive, the write cache does not have any impact on performance for request sizes beyond 64Kbytes.



Graph 4.



Graph 5.

8.4 On-Board Disk Processor Overhead

The on-board disk processor overhead is the amount of time it takes the disk drive to set up a data transfer not including the seek time and data transfer time. This becomes critical for small data transfers. As the data transfer size becomes smaller, the ratio of the actual time to transfer the data to the time to set up the transfer command gets smaller. On disk drives this is only a problem on for request sizes less than 8192 bytes. On disk arrays however, the processor overhead is significant for data transfers as high as 512Kbytes.

8.5 Command Queues

Most all SCSI disk devices have on-board Command Queues that allow the disk device to queue I/O requests locally to reduce the dead-time between requests. The disk device controllers may also have the option to re-order requests in the queue. An example of this is *seek re-ordering*. As requests come into the disk device, the controller may choose to execute those requests that have data physically located near on another and postpone the execution of a request that requires a longer seek operation. This has two side effects. First, the number of transactions per second is maximized by this strategy. Secondly the order of the requests coming *in* is not necessarily the order in which the requests come *out* of the disk device (i.e. it is not a FIFO). Thus, the *response time* of any particular request is not guaranteed. It is possible, however, to disable command queues and/or alter the caching and seek algorithms on many disk devices in order to attain the desired behavior but it is important to note that use of the command queues can result in these performance anomalies.

8.6 Disk Arrays

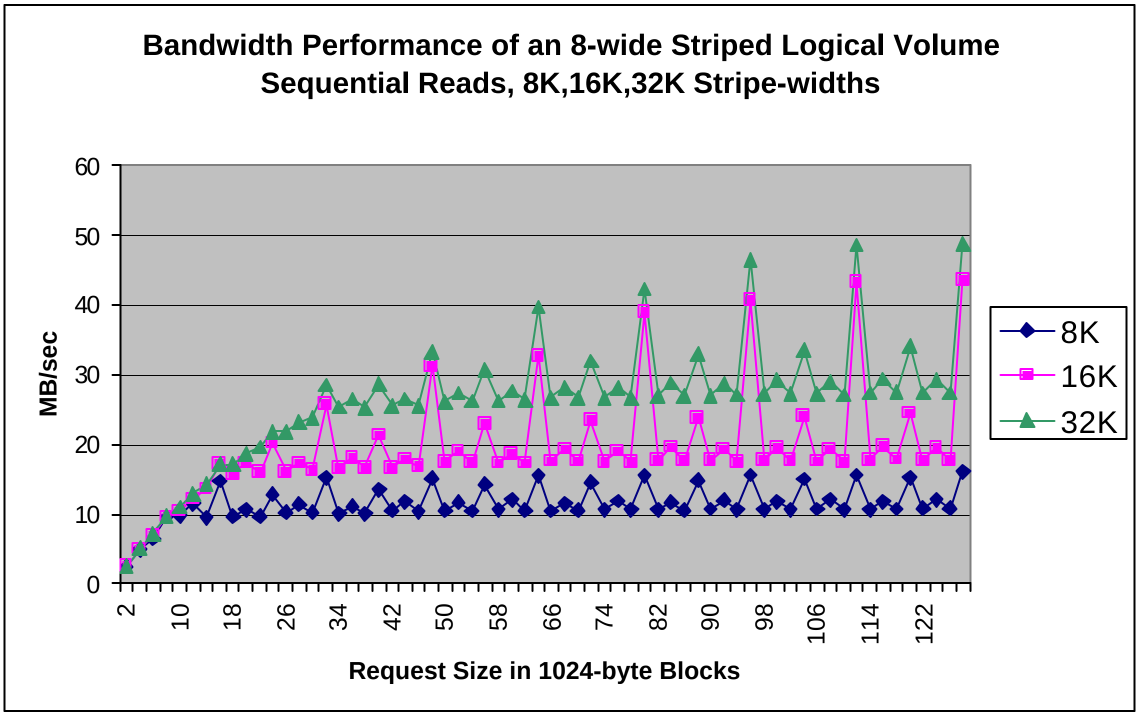
Disk arrays also know as a Redundant Array of Independent Disks (RAIDs) consist of a Disk Array Controller and several disk drives. There are several RAID levels of which two are of interest here: RAID level 3 and RAID level 5. In each of these RAID levels there are several data disks and a redundant or parity disk. RAID 3 uses a dedicated parity disk whereas RAID 5 distributes the parity data among all the disks in the array.

Another distinguishing factor between RAID 3 and 5 is that for each request that comes into a RAID 3, every disk in the array must accessed for each of these requests. This simplifies the internal architecture the RAID 3 and allows for maximum bandwidth. In a RAID 5 disk array however, the disk drives can be accessed individually which maximizes the IOPs performance but significantly complicates the internal architecture and configuration options.

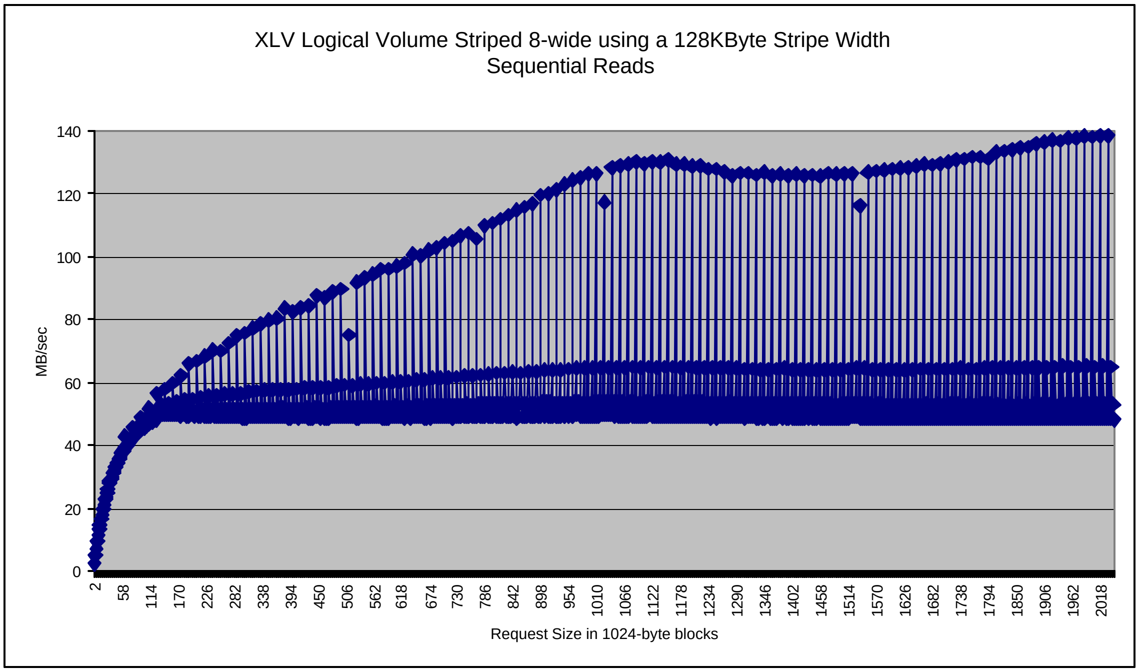
Other important factors in the bandwidth performance of a disk *array* are the internal striping factor and the mode in which it is running and. The internal striping factor is the number of bytes accessed on an individual disk within the array before proceeding to the next disk in the stripe group. Typically, on RAID 3 disk arrays this is 1 byte and is generally not configurable. On RAID 5 disk arrays the striping factor can range from 512 bytes to 64 Kbytes or more. Small striping factors in RAID 5 disk arrays lead to good Transaction performance but relatively poor Bandwidth performance. Conversely, large striping factors in RAID 5 disk arrays lead to poor Transaction performance but good Bandwidth performance.

8.7 Logical Volumes

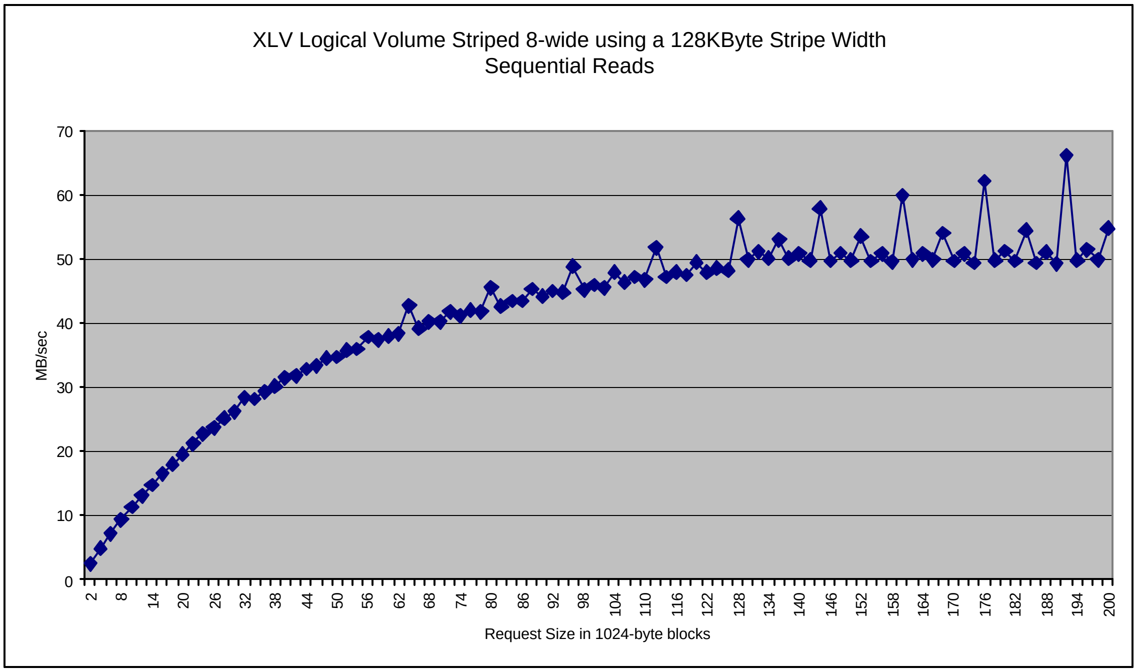
Even though Logical Volumes allow for scalable performance, there are performance anomalies that occur within a Logical Volume that are not entirely obvious. These anomalies manifest themselves as dramatic shifts in performance that are triggered simply by a change in the amount of requested data or from the alignment of the data on the logical volume. The following graphs, 6-10, show a variety of these performance anomalies that are a direct example of the Impedance Mismatch problem. Each of these volumes was created using the Silicon Graphics XLV Logical Volume Manager and measurements were taken from an SGI ONYX2 computer system with eight processors and a Dual Channel Prisa PCI64 Fibre Channel Host Bus Adapter.



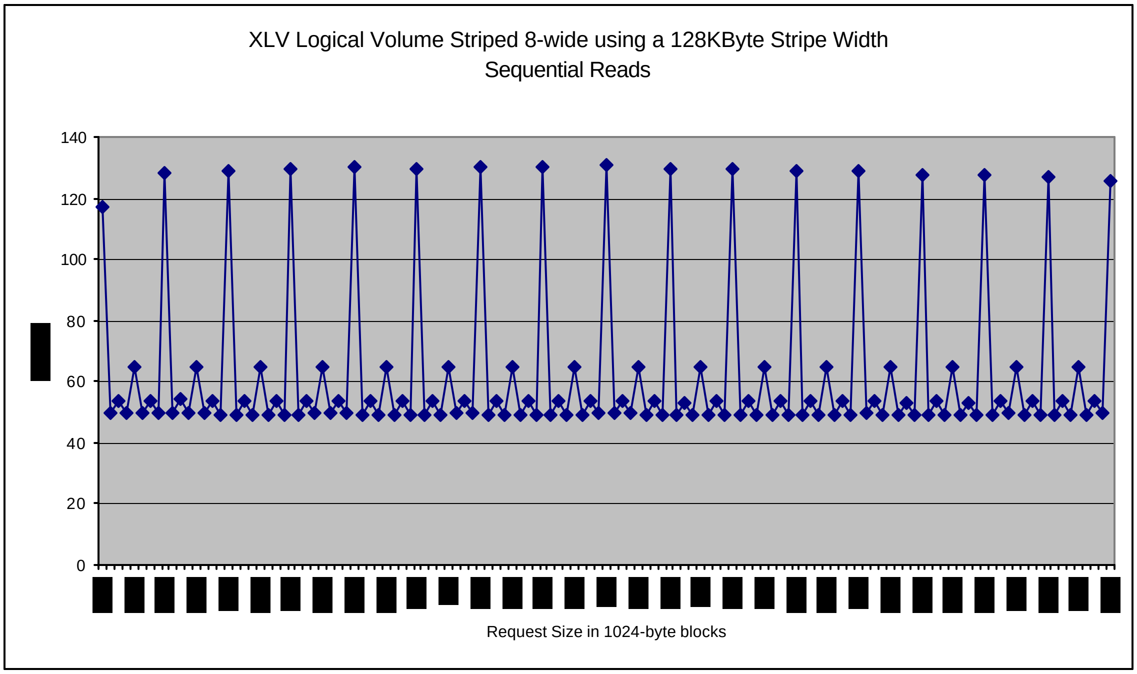
Graph 6.



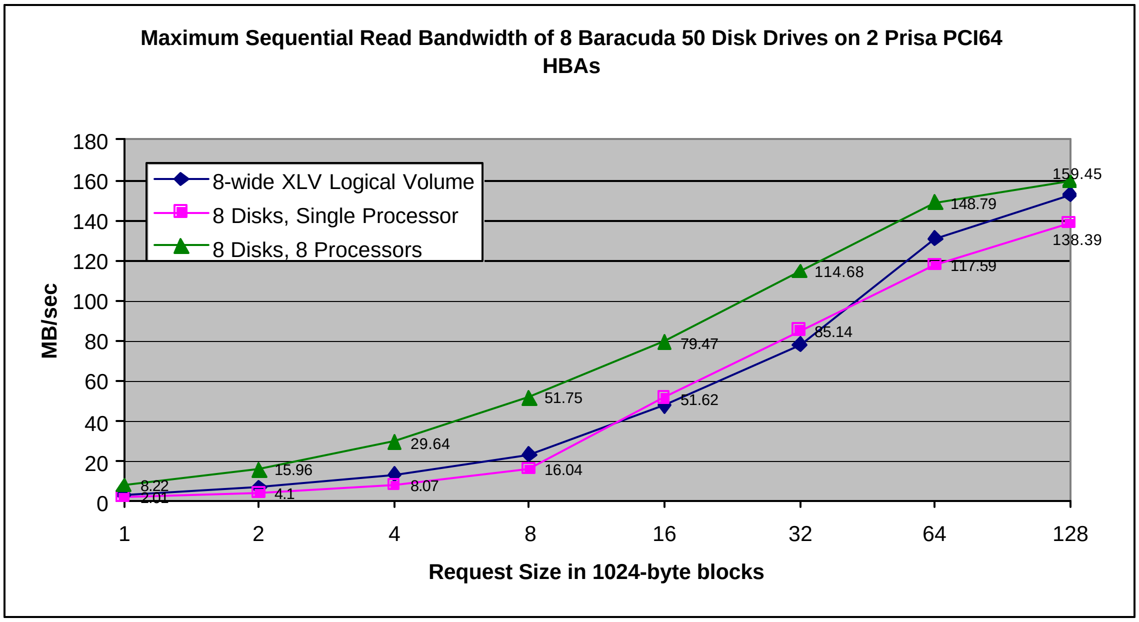
Graph 7.



Graph 8.



Graph 9.



Graph 10.

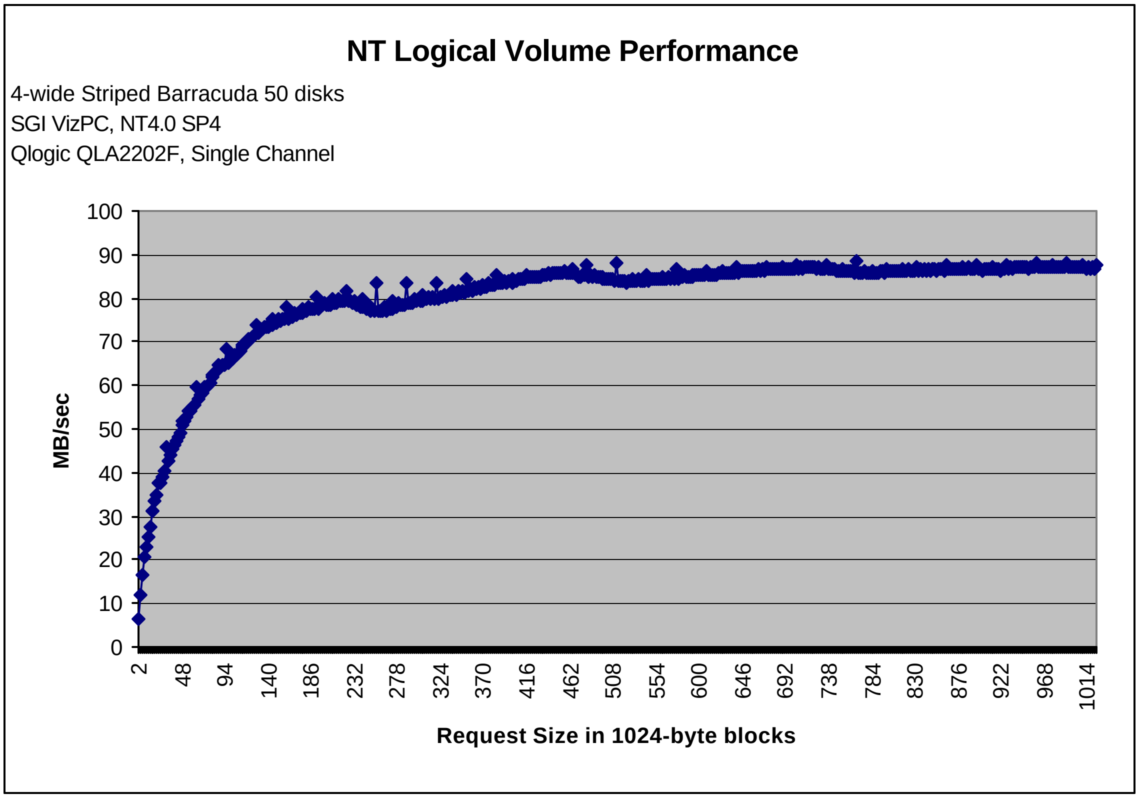
Graph 6 shows the performance of three 8-wide logical volumes with difference striping factors. This graph shows that the overall sequential read bandwidth increases as the stripe unit size increases but so does the *variability* in the bandwidth. For example, the Logical Volume using a 16Kbyte stripe unit can have its performance vary from 18MB/sec up to 44MB/sec simply by choosing a different request size (the number of bytes requested by the application on each read operation).

Graph 7 is a more dramatic view of the same phenomenon but this time on a logical volume using a 128Kbyte stripe size. The subsequent two graphs, 8 and 9, zoom in on the lower end and middle of the Request Size scale. Graph 8 shows a smooth ramp-up in performance as more data is requested. Graph 9 focuses on the dramatic performance difference of the different request sizes. The peaks in graph 9 occur at 16Kbyte intervals and fall on multiples of 16Kbytes. The valleys occur when the request size is not an even multiple of 16Kbytes. The important point of each of these graphs is do demonstrate the magnitude of this problem.

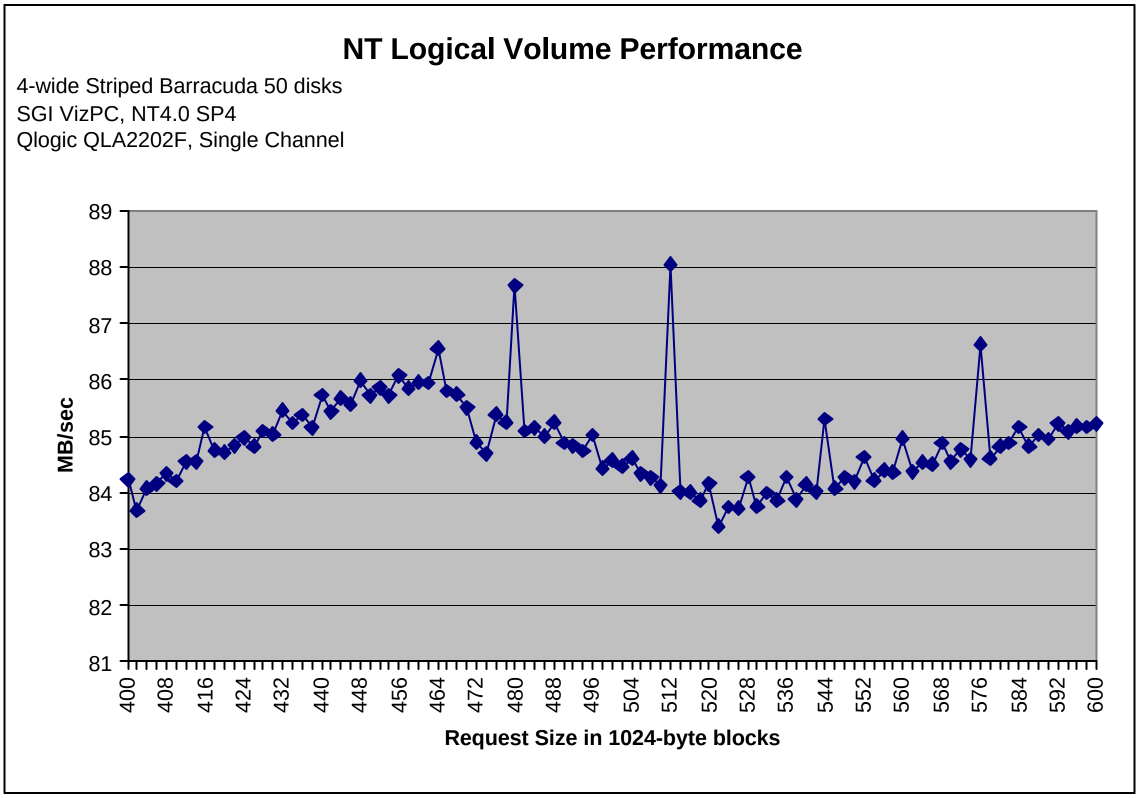
Graph 10 demonstrates another aspect of the Impedance Matching problem that has to do with processor allocation. On this graph the peak read bandwidth for an 8-wide logical volume is plotted against the peak performance of two groups of 8 xdd threads each running to a single disk. One of the 8-disk xdd thread groups is assigned to a single processor in the SGI ONYX2. The other thread group is distributed across all 8 processors in the ONYX2, one thread to each processor. It is clear that the distributed case performs significantly better than the logical volume and the single-processor case. The reason for this has to do with the fact that at lower request sizes more requests are processed per second. It turns out that a single processor gets overwhelmed with processing requests with between 6-8 of these particular disks each running at full speed. When the request processing is distributed across multiple processors, a higher overall performance rate is observed. Also, since the single-processor case closely follows the 8-wide XLV case, it can be concluded that the performance limitations of the XLV logical volume is due to a problem with having all the XLV request processing funneling through a single processor.

Not all Logical Volume software is created equal though. Graphs 11 and 12 show the performance curves for a 4-wide Windows NT Logical Volume striped set of disks. The performance of this logical volume does exhibit some performance variation but not nearly as dramatic as the variations seen in the XLV logical volume. Graph 12 focuses on a small part of Graph 11. This shows the variation to be about 4MB/sec as opposed to the 80MB/sec seen in Graph 9.

The conclusion here is that the Logical Volume performance variations shown in the past several graphs is a function of the Logical Volume software and associated implementation parameters. A more detailed analysis of these Logical Volume performance anomalies is presented detail in [2]. The purpose of these graphs is to demonstrate that things can go wrong and how they go wrong.



Graph 11.

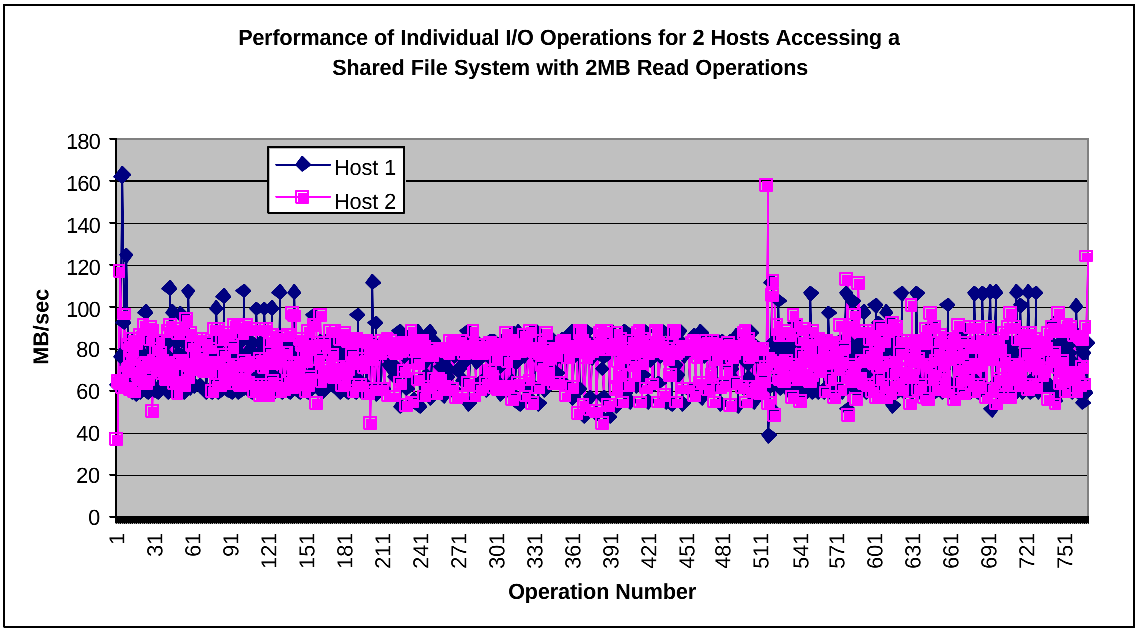


Graph 12.

8.8 Storage Area Network Performance Results

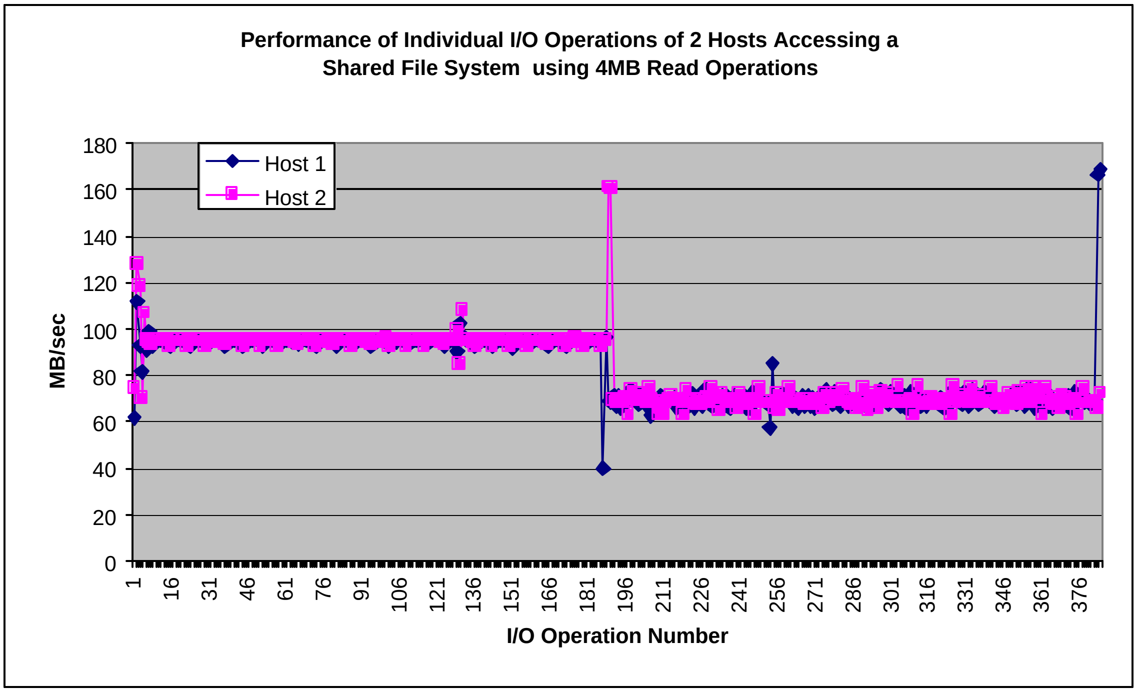
At this point the testing and evaluation process becomes more complex. When testing a storage subsystem on a single, isolated computer system, it is possible to correlate events (I/O requests, interrupts, data transfers, …etc.) in *time* at a very high resolution. This is possible because all the performance benchmark application runs on a single computer using a single "reference clock" where all events are based on that single reference clock. In a Storage Area Network however, it is necessary to run the benchmark application from multiple computers simultaneously, each accessing the same Storage Subsystem. Each computer system has its own reference clock from which events local to a computer system can be correlated. However, the notion of a "global" reference clock must be established in order to cross correlate events in time over all the systems. In other words, there must be a single reference clock on which to base all the events that occur on the Storage Area Network in order to understand the *interactions* between computer systems accessing a single Storage Subsystem. The generation of this global clock, discussed in section 7, is therefore critical to the evaluation testing process of these SAN configurations.

A simple test was run using two hosts accessing a single set of 16 disk drives configured as a single logical volume through a file system shared between two Windows NT PC computers. Each PC computer had two Fibre Channel connections to the logical volume. The first test consisted of sequentially reading a 1.6GB file using 2MB per request from 2 hosts reading the same file. The file was read three times with results reported for each



Graph 13.

pass. The net result showed each host was able to read the entire file at an aggregate rate of 73MB/sec. Graph 13 shows the instantaneous bandwidth performance of each I/O operation for both hosts. The graph is crowded but it does show that the performance limits of each host remained in a well-defined band from 50-90MB/sec/op.



Graph 14.

Graph 14 however shows the utility of displaying the time-stamped operation data. In this test the same file was being read by the same two hosts but with 4MB read requests. The first half of the I/O operations look very consistent. (There is a small “blip” at the 1/3rd and 2/3rd points on this graph that indicate when the second and third read passes started.) However, about half way through the second pass of reading the file there was an unusual drop in performance that is very evident in the graph. Both host computers saw the same performance decrease and at the same time. It is also apparent that neither host computer recovered from this performance problem. It is also not known what caused this anomaly but further analysis of the timestamp data may reveal an access pattern issue related to a caching idiosyncrasy of the disks.

9 Concluding Remarks

This paper shows that there is a large variation in performance for logical volumes caused by the Impedance Matching problem. This is primarily a result of having the I/O request traversing too many levels in the Storage Subsystem Hierarchy. The I/O request at each level can get resized and/or re-aligned in space and time. By the time the I/O request gets to the storage subsystem, it appears are many smaller requests distributed across many devices. Furthermore, what the application sent over as a “parallel” request can be broken up into a series of smaller, serialized requests to the storage subsystem. The results are demonstrated in a series of graphs that show what happens to the performance as seen by the application when a series of large requests are made to subsystems with different configurations.

These are just some of many examples of the manifestation of the Impedance Matching problem within a Storage Subsystem. Other Impedance Matching-like problems occur in the caches used on the disks arrays and disk drives with respect to their size and caching algorithms, multi-host Storage Area Networks, and the ever-changing bandwidths and latencies of the subsystem interfaces. These are all areas that are ripe for investigation given an adequate test and evaluation framework.

It was also demonstrated to some extent the value of having a testing framework with a highly resolved, global clock for the purpose of evaluating and analyzing the performance of a Shared Storage Subsystem in a Storage Area Network environment. This testing framework will become more critical as the systems become more complex and less predictable whereby more real-time empirically-based analysis will be required to resolved problems in large SAN configurations.

10 Future Work

Future and ongoing work includes but is not limited to:

- Integrating these techniques and testing framework with File System testing efforts
- Developing ways to collect subsequently study "real world" storage system activity data
- Improving and expanding the capabilities of the testing software to other operating environments
- Incorporating other storage devices such as tape drives into this testing framework

References

[1] This work was supported in part by the National Science Foundation, under the NSF Cooperative Agreement No. CI-9619019, and by the Department of Energy through the ASCI Data Visualization Corridor Program under contract #W-7405-ENG-48.

[2] Ruwart, Thomas M., “Performance Characteristics of Large and Long Fibre Channel Arbitrated Loops”, *Proceedings*, 16th IEEE Symposium on Mass Storage Systems / 7th NASA Goddard Conference on Mass Storage Systems and Technologies, March 1999, IEEE Computer Society Press

[3] Thomas M. Ruwart and Matthew T. O'Keefe, "Performance Characteristics of a 100 MB/sec Disk Array", Storage and Interfaces '94, San Jose, CA

[4] Alex Elder et al., "The InTENsity PowerWall: A Case Study for a Shared File System Testing Framework", *Proceedings*, 17th IEEE Symposium on Mass Storage Systems / 8th NASA Goddard Conference on Mass Storage Systems and Technologies, March 2000, IEEE Computer Society Press