RC22893 (W0309-028) September 5, 2003
Computer Science

IBM Research Report

A System and Multilevel Approach to Storage Performance Evaluation

Kirk Beaty, Norman Bobroff, Gautam Kar
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

A System and Multilevel Approach to Storage Performance Evaluation

(Kirk Beaty, Norman Bobroff, and Gautam Kar)
(IBM Research, Hawthorne, NY, 10532)

1.1 Introduction

Performance of magnetic disks has not increased as rapidly as other elements of a computer system such as processor, memory, and network speed. However, it remains difficult to identify when and how storage performance is affecting application performance. The low cost of memory and attention to caching strategies has helped compensate for the speed of the raw storage device. Applications are seldom instrumented to monitor the I/O or processor component of their work. Operating system instrumentation often provides data on other aspects of application progress such as user and kernel space CPU consumption and page faults. But it is difficult to isolate the contribution of storage to application performance using statistics reported by the operating system. We attempt to improve this situation by combining metrics taken at several layers of the operating system to quantify how storage performance is degrading an application. A central goal is to develop relations that quantify the impact of a storage device on applications and on the operating system. These relations should produce figures of merit that can be used to identify and predict performance bottlenecks. Ideally these are dimensionless and scalable so that they can be easily applied across heterogeneous systems. Additionally we would like to predict whether an increase in disk performance or change in configuration such as raid level would improve application response.

The types of data collected and analysis developed for this problem can also be applied to problems such as detecting storage contention by unrelated applications inadvertently sharing a common disk or data path such as fiber link to a storage area networked (SAN) device. A related problem is to define the baseline storage workload. Workload intensity and distribution among parameters such a reads, writes, sizes, sequential fraction, varies with time of day, week or month. Application related storage problems should be diagnosed within the context of the acceptable workload. For example, some applications have periods of operation during which extensive database updates occur. The performance of this I/O bound workload component is largely limited by storage. However, this does not necessarily mean it is worth a major investment in new storage. This decision is based on a combination of policy, economics and technical factors. The results here should help forecast the outcome of upgrading a storage device.

Our initial focus is to understand how storage performance is manifest at several layers of the operating system and application. There are two broad areas of this work. 1) Find the dependencies between system, application, and storage layers. 2) Develop techniques and figures of merit to quantify the performance related aspects of the dependencies. We want to do this in a minimally intrusive way, relying on existing or near horizon operating system instrumentation.

1.2 Scope of applicability

The study is relevant to applications and operating systems containing the common architectural layers outlined in Figure 1. These include the physical device, logical volume, logical file system, and application layers. The physical device layer refers to the performance by the software driver associated with a host bus adapter (HBA). The HBA may be accessing direct attached storage or a remote device such as a logical disk exported on a storage area network. The logical volume is a software element that represents block storage. The logical volume is mapped onto one or more physical disk partitions. The mapping may specify an access strategy such as stripping or a RAID level. The logical volume may also queue I/O requests and perform some optimization such as splitting or coalescing sequential requests. The file system is installed on a logical volume to impose directory and file structure on the block device. The file system typically provides or manages services such as a cache and read ahead strategy. Applications access storage through either the file system or the logical volume. Historically, databases have accessed the logical volume and provide their own cache and read-ahead strategies. Alternatively, operating systems such as AIX and Windows provide modes of file system access that bypass the file system cache. Thus, it is becoming more common for databases to use the file system thus simplifying management issues.

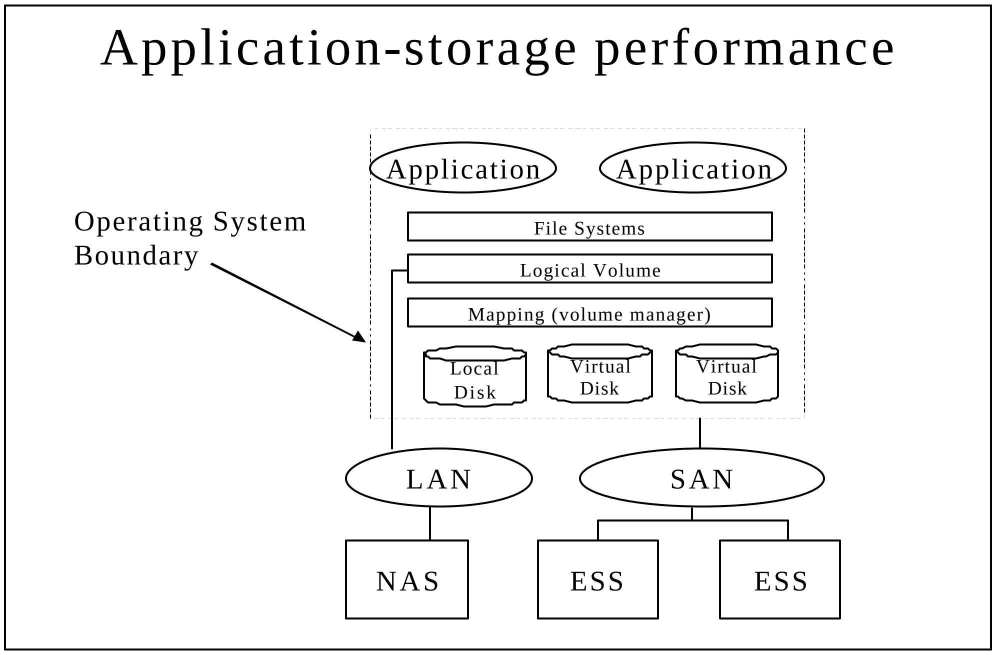


Figure 1 Layout of managed system with direct attached storage and storage area network volumes.

Applications can be distributed; SAN and NAS storage resources can be shared with many operating system images. We would like to extend the analysis to find bottlenecks and contention for storage resources across physical system boundaries.

1.3 Prior Work

Studies of hard disk performance measure properties of physical storage devices such as seek and transfer time. System oriented benchmarks report performance of a benchmark program executing at the logical file system level. Throughput and latency are typically measured while the benchmark varies parameters such as I/O size, concurrency, or fraction of sequential requests. This study extends this work by collecting data from mulitple levels of the system.

1.4 Experimental Setup

A key factor of the design of the experimental setup was flexibility. That is, not all was known up front (as is always the case) and therefore such design components such as the data collection methods and database schema needed to be extensible without requiring a major development effort to add new types of metrics and properties to the data gathered.

1.4.1 Design Overview

Configuration, dependency, and performance data from different levels of the system indicated below are stored in a central repository.

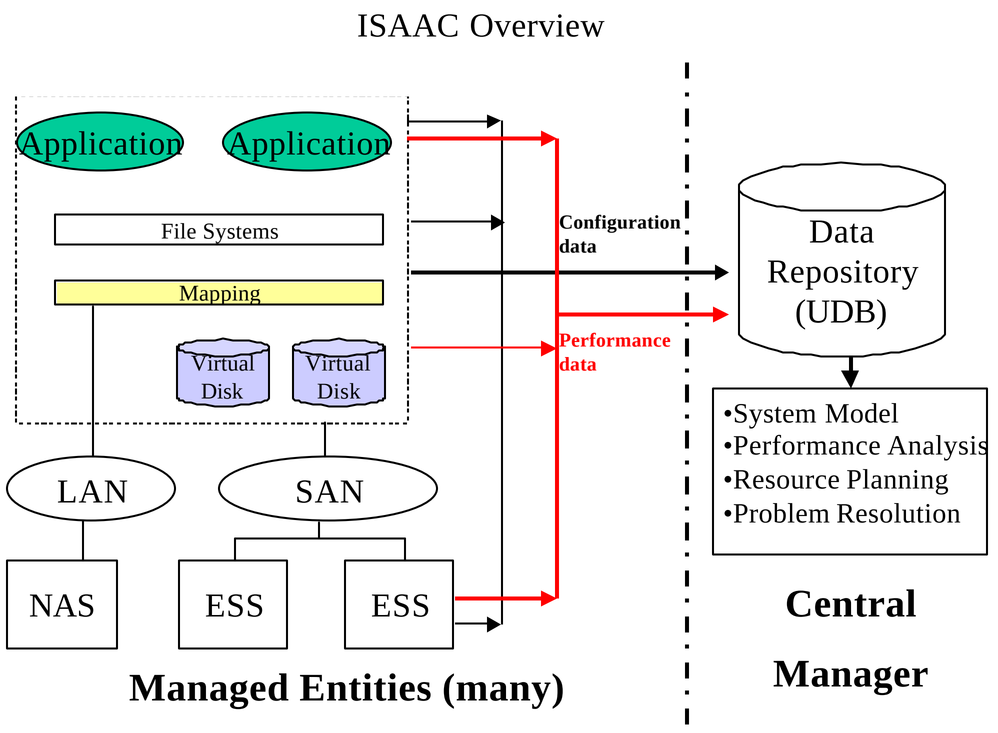
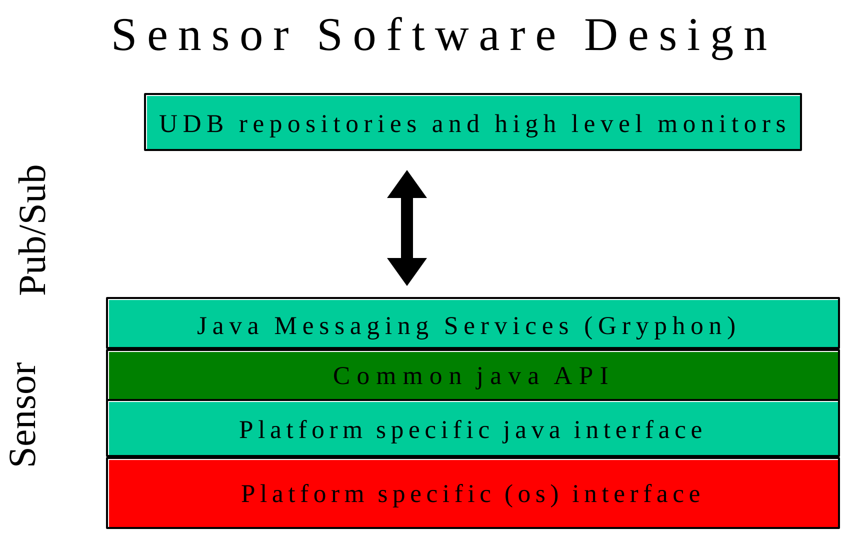


Figure 2 Overview of Data Collection

Performance data is collected from each managed object using a sensor. The relation between managed objects is discovered using a configuration sensor. Some detail of the sensor software we have for AIX is provided in Figure 2



Sensor Software Design
UDB repositories and high level monitors
Pub/Sub
Sensor
Java Messaging Services (Gryphon)
Common java API
Platform specific java interface
Platform specific (os) interface

Figure 3 Pub/Sub Model for Data Collection

Data flow in the system from the sensors to the central data repository is implemented on a pub/sub infrastructure using IBM's Gryphon implementation of Java Messaging Services (see Figure 3). The sensors publish on topics to which the repository subscribes. This provides content based rather than location based information flow. This feature facilitates the introduction of new high-level modules that selectively monitor data and makes a very modular communications architecture.

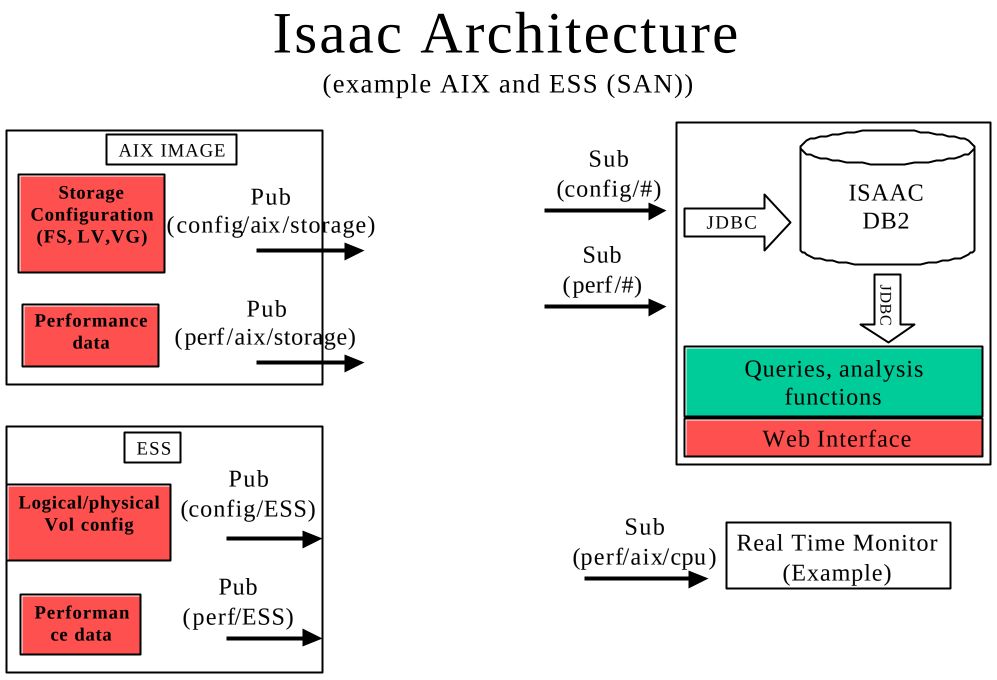


Figure 4 Data collection flow using Pub/Sub

1.4.2 <u>Operational Database Schema</u>

The database schema is also designed to be general, supporting flexibility in extensions. It is intended to be used as an operational database for real-time problem determination and additionally to serve as input to the Tivoli Enterprise Data Warehouse for storage of historical data for trends analysis.

The database schema is general, supporting the notion of 'managed systems', each of which contains a set of 'managed elements' and their relationships. A managed object has two types of attributes, properties and metrics. The former are string values and the latter numeric values. The actual type of any object is a property attribute, and the space of attributes is not restricted by the type attributes. Thus, managed objects are treated uniformly in the schema, i.e. not distinguished by their instance type. This schema permits new managed objects and object properties to be added at any time.

The database schema is also designed to handle a large number of performance observations acquired by periodic updates by the performance sensors. This detailed observation data is aggregated into summary database tables that store in condensed form the long-term history of the sensor metrics.

Database Repository Schema

MS – Managed Systems

- **ME – Managed Elements**
  - **Properties**
  - **Metrics**
  - **Metric Trends**
- **ME_DEP – Dependencies**
  - **Dependency Properties**
  - **Dependency Metrics**
  - **Dependency Metric Trends**

Extensible Architecture

- **Multi-platform support**
- **Easy to add properties/metrics**
- **Complex dependency trees handled**

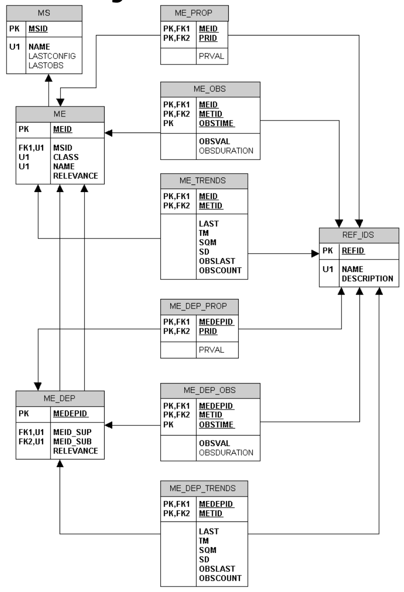


Figure 5. Database Schema

1.4.3 Test SAN Environment

We presently have a SAN environment to evaluate performance models. Figure The AIX host is a dual processor PowerPC3 architecture operating at 375MHz. There are two direct attached SCSI drives and several remote volumes mounted from a SAN. The SAN consists of a fiber channel switch and an Enterprise Storage Server (ESS). The ESS is fully configured with 96 disks and 32GB of nonvolatile cache. The lodestone storage consists of Fast-T disks. All devices are connected to a McData FICON switch through a single 1Gb fiber channel.

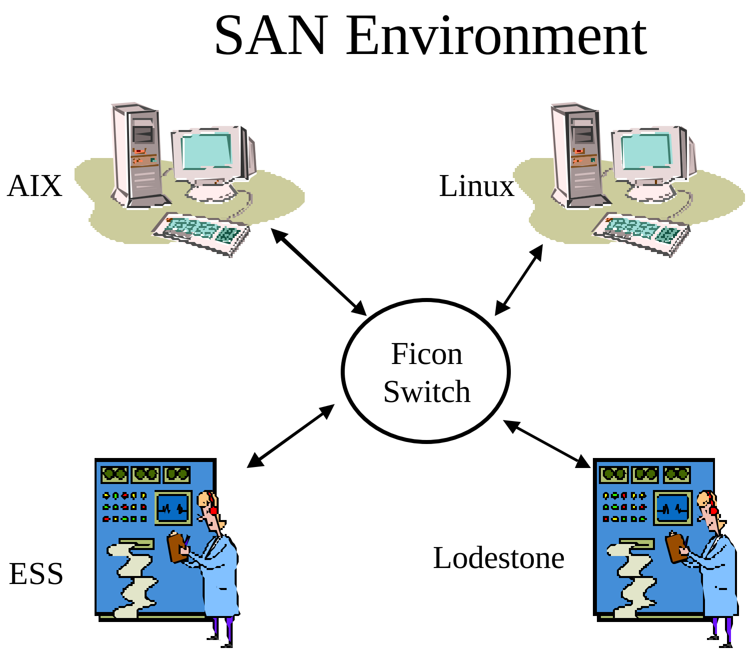


Figure 6. Present SAN configuration.

1.4.4 DBench

We use the DBench benchmark to apply load to the I/O system. DBench is the disk component of the NetBench performance suite. It is distributed as source code and compiled for AIX. A DBench process creates a directory sub tree populated with a large number of files. A script file defines a large number of file system operations that are executed against the directory and files. The profile of operations is provided in table Table 1. Execution is synchronous within a process, each I/O completes prior to reading and executing the subsequent line of the script. To increase load and generate concurrent operations DBench can create multiple processes using fork. The number of processes is specified on the command line.

Table 1. Profile of DBench file operations to c library

operation	Count	Time(s)
unlink	2095	1.413
rmdir	11	0.008
create	10999	2.067
read (total)	17881	7.200
read(<4KB)	14423	3.775
read(> 4KB)	3458	3.424
write	7475	9.450
write(<4KB)	4756	3.819
write(>4KB)	2719	5.630

close	8356	0.136
rename	435	0.287
qpathinfo	7372	0.197
qfileinfo	1849	0.028
findfirst	3573	0.264

We modified the DBench code to add a direct I/O option and improve the recording of performance data. The direct I/O option bypasses the cache in the logical file system (LFS) so that all reads and writes go to the disk. This makes the benchmark apply completely to the I/O system. We also improved the recording of performance information. DBench only provides a read and write throughput at the end of execution. Monitors of read and write times for small (<4K) and large (>4K) accesses were added to the base code. Reporting was also expanded to include summary statistics on all file activity listed in Table 1.

1.5 Results

This report discusses our observations and conclusions based on data from a single server ATango. The ATango machine has two PowerPC3 processing units operating at 375MHz. There are two direct attached SCSI drives and several remote volumes mounted from a SAN. The SAN consists of a fiber channel switch and an Enterprise Storage Server (ESS). The ESS and host each have a single connection to the switch. Remote volumes are created from a RAID 5 disk group on the ESS. The ESS has 32GB of nonvolatile cache. Since DBench accesses about 363MB of data it is assumed that all accesses are through the cache (We hope to verify this using StorWatch Expert).

1.5.1 Storage Response Time Analysis

Storage performance tests are conducted by executing DBench on ATANGO using direct I/O (bypassing the file system cache). Raw data for several of the collected statistics is indicated in Figure 7. The horizontal axis is time of day. The number of concurrent DBench processes was incremented with time and the arrows mark the start of execution for the given number of concurrent processes. Each data point represents data collected in a 10 second interval.

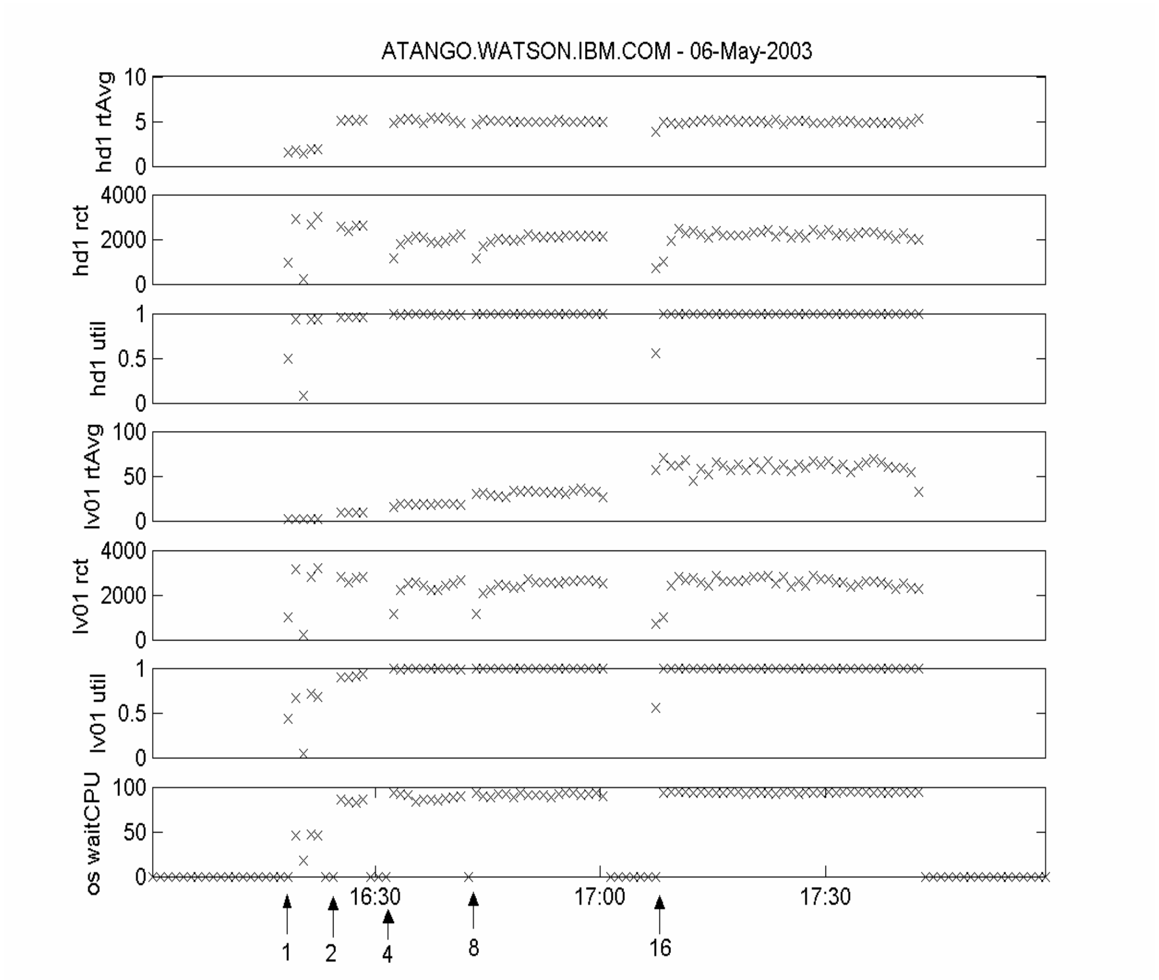


Figure 7. Raw data from a direct attached SCSI disk. The arrows indicate the number of concurrent DBench processes.

The data of Figure 7 are obtained with DBench using a direct attached SCSI disk. Figure 8 shows the comparable data for the ESS disk.

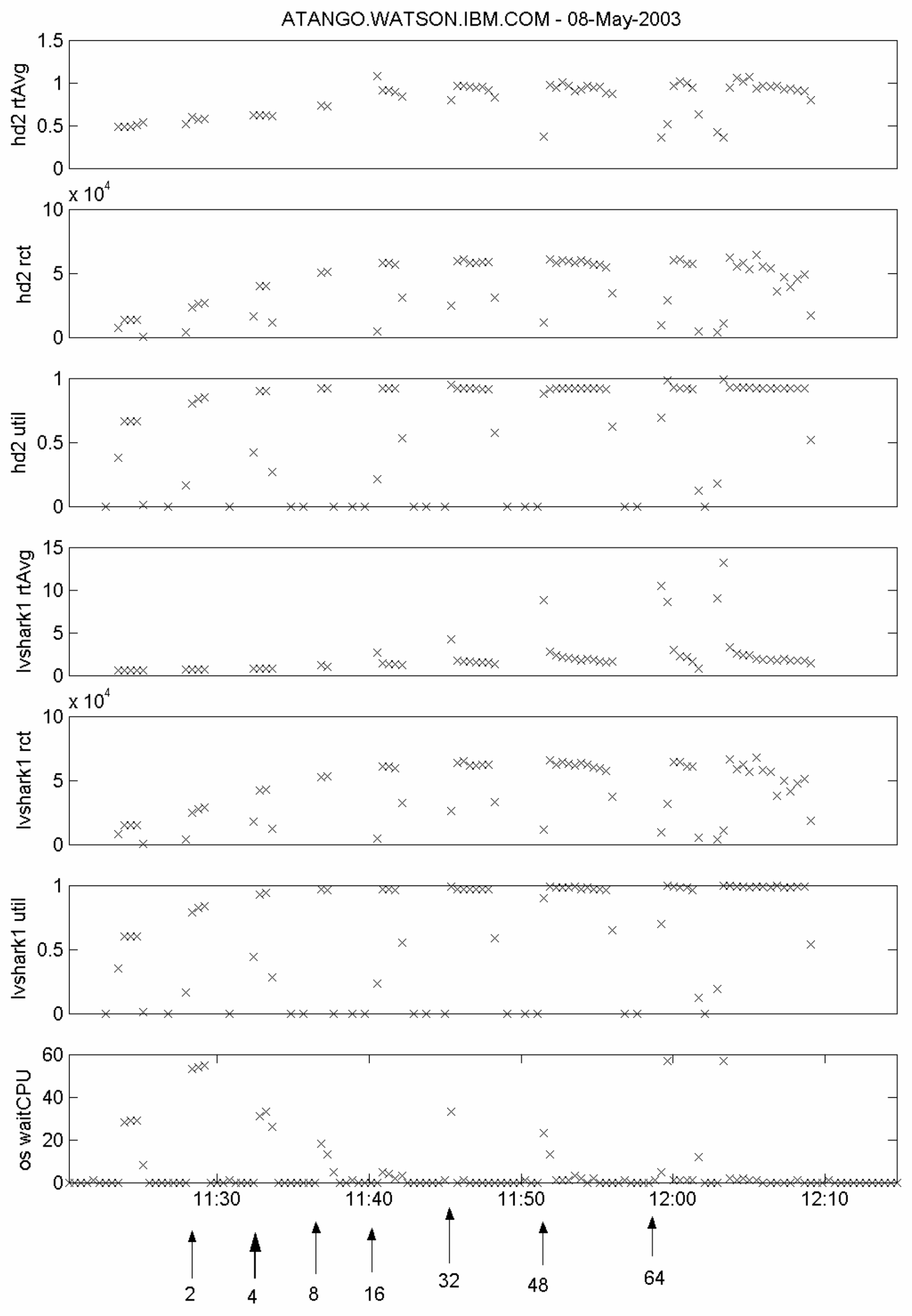


Figure 8. Raw data on an ESS volume. The arrows indicate the number of concurrent DBench processes.

In each figure the top three data sets are the read time average, number of reads, and utilization in each 10s sample for the physical disk. (For the ESS data this means the data acquired by the device driver in the operating system. See Figure 1 for further explanation.) The subsequent 3 plots show the corresponding data for the logical volume. The bottom chart is the percent of I/O wait reported by the vmstat call on AIX. Wait CPU is the fraction of time the processor is idle but there are outstanding I/O requests. For a multiprocessor system the reported wait CPU is normalized by the number of processors.

Some interesting features are apparent in the raw data. For both direct attached and ESS disks the reported hard disk utilization increases dramatically (to over 95%) prior to appreciable degradation in throughput for the entire test. This is not to say that utilization is unrelated to response. The dependence of response time on utilization is characteristic of a simple queuing model. However there is considerable scatter in the data. This is discussed below in the context of Figure 9. Furthermore, it is difficult to obtain a sufficiently accurate measure of utilization to use a threshold without many false positives. It is a universal property of queuing models that latency is a more sensitive function of load than utilization. Utilization is proportional to load, while latency is inversely related to 1-u.

The response time reported by the disk driver is provided by the upper trace in each figure. For direct attached storage it degrades quickly with load, by about a factor of three. This degradation factor coincides with the maximum queue length of three reported by the AIX 'lsattr' command. This is a fixed length queue that cannot be increased. The ESS performance scales much better with load. A queue length of 32 is reported for this device driver. Using a single machine to drive load we were not able to significantly load this queue. The response times of the top trace in Figure 8 suggests a peak slowdown of only about two to three. However, there are difficulties in using physical disk driver response times as an indicator of performance. A dimensionless number is preferred to an absolute criterion such as a certain number of milliseconds. This issue is resolved by normalizing response time to the best-observed historical performance interval having some minimum number of disk I/O events. This is the definition of expansion factor for a performance measurement. The second problem is that hard disk response time does not clearly reflect application performance. This is clear in the data for the direct attached device of Figure 7. The hard disk response time reaches a maximum at a lower load than the response time at the logical volume. This behavior is apparent by comparing the first and fourth trace from the top of the figure.

The logical volume layer provides a closer approximation to the response time an application would see. The read response at the LV plateaus with about 8 driver processes. We are investigating whether this result is caused by a limit on the number of I/O's queued, or the number of physical buffers available for the device driver. The application response time continues to increase linearly with load. As at the physical device layer, utilization does not appear to by a reliable metric for generating alerts on poor I/O response. The expansion factor at the logical volume can be used as a metric for detecting a poor performing disk. Relying on the analogy with a simple queue one might expect an expansion factor of three or four is reasonably near the 'knee' of the M/M/1 queue approximation to be a good indication. In fact, where reported by the operating system, the current queue length should be used.

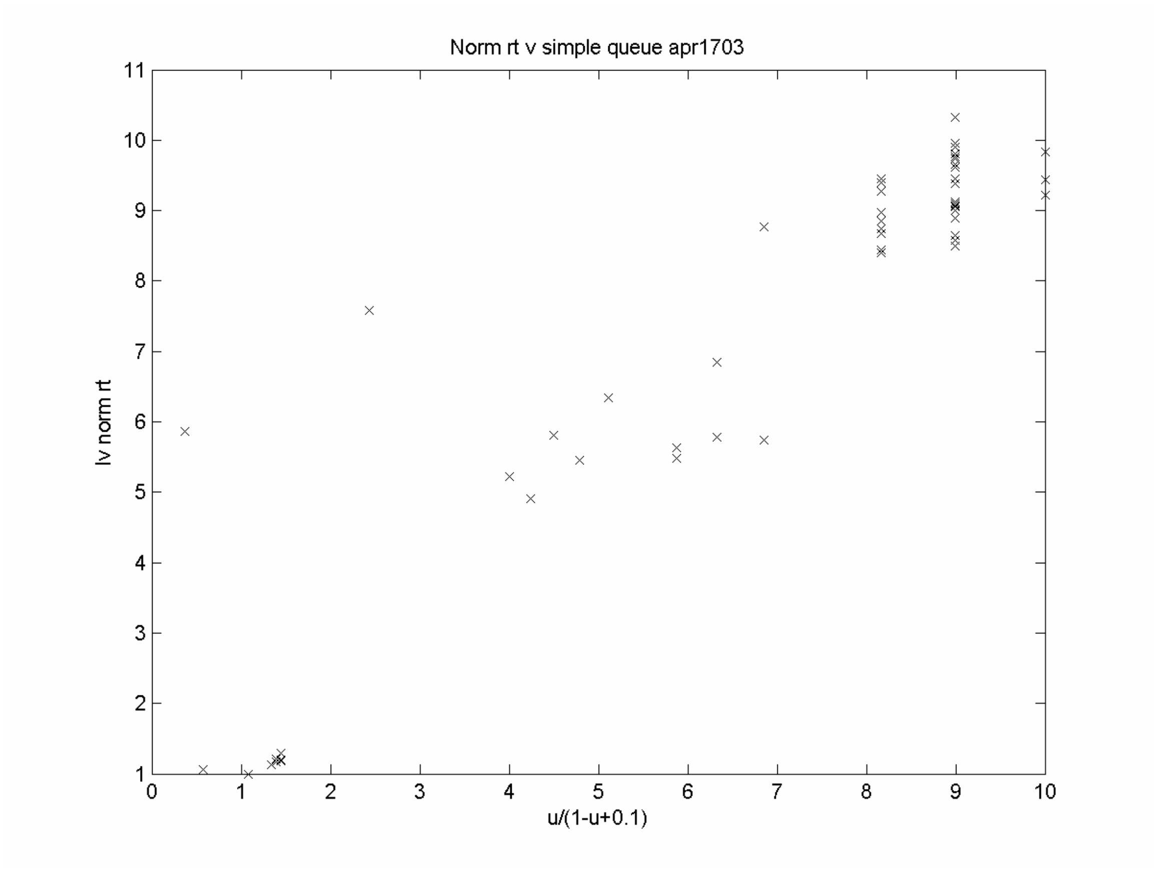


Figure 9. Expansion factor at the logical volume as a function of the prediction of the M/M/1 queue. U is the utilization, and the 0.1 is an empirical factor to account for the fact that we don't hit 100% utilization

To extend this investigation, Figure 9 compares the expansion factor (queue wait time) at the logical volume with the prediction of the M/M/1 model. The horizontal axis is the queue length that M/M/a predicts based on the utilization of the logical volume. The assumption is that the average queue wait time is proportional to u/(1 – u + overhead). The overhead is an empirical number because we know the queue is not infinite when AIX reports 100% utilization. The vertical axis is the normalized read times. From properly analyzed historical data in our operational database it may be possible to apply a heuristic model and deduce a target expansion factor (or internal queue length) at which to report application performance on this logical volume may be degrading. That is, provide a more erudite threshold for basing alerts than can be achieved from the utilization.

1.5.2 Multi-Level Response Analysis

It is more interesting to explore how the data from multiple levels can be combined to improve the analysis. The expansion factor for the hard disk and logical volumes is broken out in upper two traces of Figure 10. The bottom trace shows the ratio of the expansion factor at the logical volume to the physical disk. When this ratio exceeds unity, delay at the physical disk driver is causing a backup to the logical volume. It is suggestive that when this ratio exceeds five, utilization and processor wait time are nearly 100%.

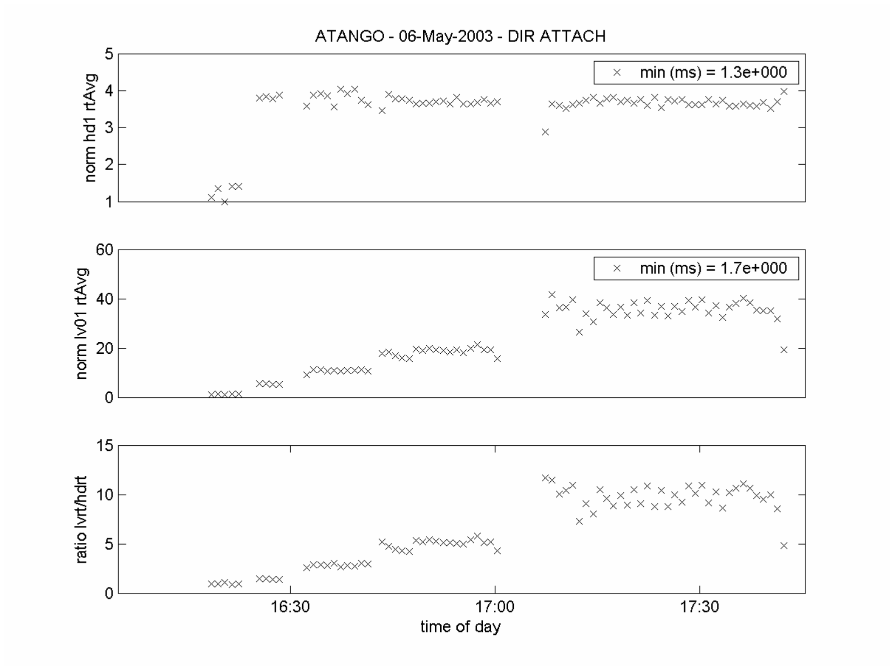


Figure 10. Expansion factors for direct attached storage and logical volume corresponding to Figure 7. The lower trace is the ratio of the expansion factors at the logical volume to the physical driver.

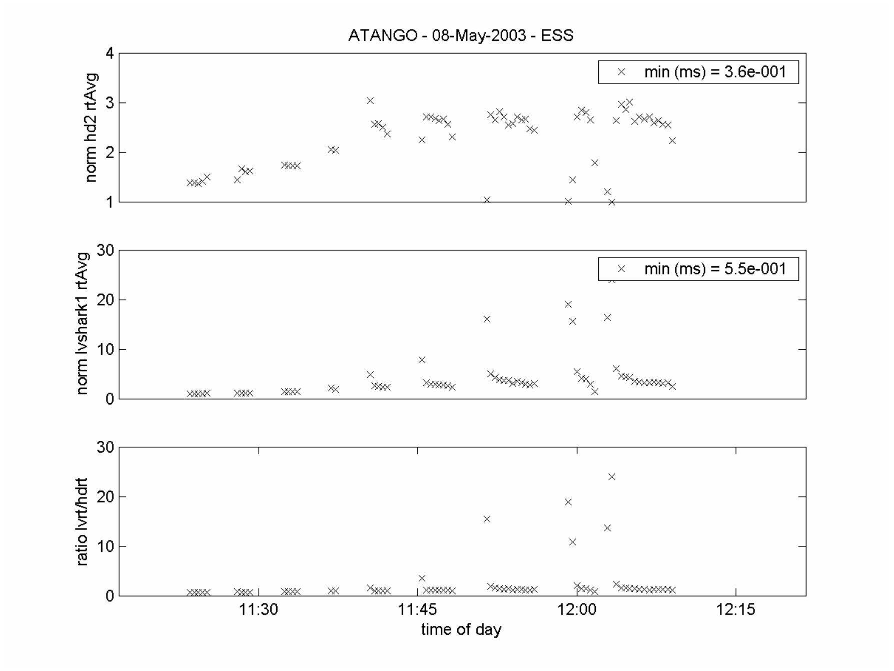


Figure 11. Expansion factors for direct attached storage and logical volume corresponding to Figure 8. The lower trace is the ratio of the expansion factors at the logical volume to the physical driver.

The comparable ESS data is presented in Figure 11.The ESS results are interesting because of the effect the improved physical disk speed and scalability has at the logical volume and, by inference, application layer. ESS improves the minimum read time at the physical layer by slightly more than three. Maximum throughput increases by about the same ratio. As with a direct attached disk, the physical layer performance degrades until reaching a slowdown of about three. The limiting response time occurs around 16 concurrent processes rather than 4. However, the expansion factor at the logical volume is much lower for the ESS than for the direct attached disks. Although the physical disk driver reports saturation, the expansion factor at the logical volume is about three. An important observation is that a modest improvement in response at the physical disk reduced the expansion factor at the logical volume by about an order of magnitude at high load. We believe much of the improvement is attributable to the parallelism in the SAN ficon channel. For direct attached storage, all requests are queued at the disk. The SAN ficon connection allows up to 8 requests in parallel. At the ESS the requests are serviced from the cache. Thus most of the SAN response time is in the fiber connection. Therefore, the ability to issue parallel requests greatly increases the throughput while the latency is essentially constant. This hypothesis is being further investigated by adding StorWatch expert to the ESS. A further goal is to define a set of archetype classes of storage workload. For each class we can model the potential benefit of a given improvement in storage level performance. Based on the workload profile of an application the benefit of an improvement in storage can be determined. Related discussion is included in sections 1.5.4 and 1.5.5.

1.5.3 Contention and Hotspot Detection

These data suggest some approaches to disk contention and hotspot detection between applications running on different hosts that share common storage through a storage network. Contention can apply to any resource in the SAN, such as interconnects, switches, caching devices, or read disks. Contention is characterized by increased latency at low throughput. By this we mean that the latency exceeds that expected based on historical data relating throughput and latency. The key challenge is to how to aggregate and categorize historical performance data to quickly detect anomalous performance degradation.

1.5.4 Effect on Application and Processor Utilization

A decrease in storage performance as seen by the logical volume I/O performance does not mean that application is affected proportionally. Applications that have child processes or are multi-threaded may do asynchronous I/O or processor intensive work in parallel with I/O. They or the file system may do read-ahead to reduce the impact of storage performance. Operating systems do not measure I/O wait cycles on a per process basis. Some insight can be gained from the aggregate values reported for the system as a whole. It is difficult to assess the storage impact on an application that is storage intensive, but not I/O bound. And a figure of merit for storage should be combined with the wait I/O time data to provide an enhanced threshold to signal an alert. Figure 12shows the measures of processor utilization for the DBench data with ESS of Figure 8.

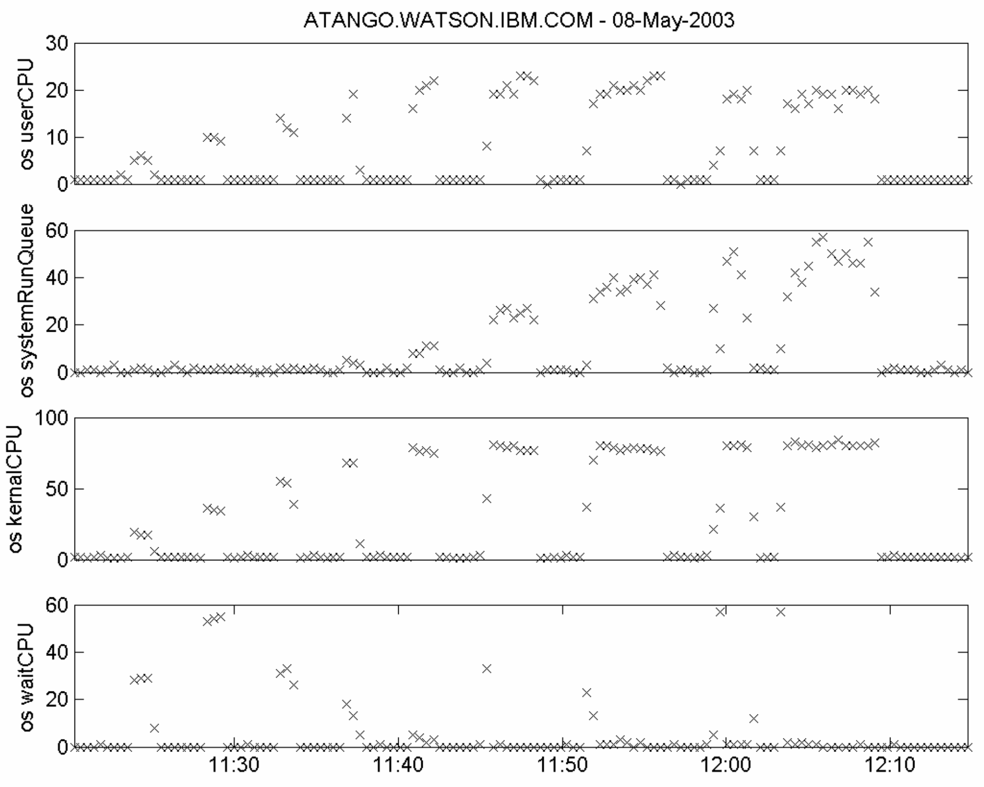


Figure 12. OS processor state for the ESS storage

To understand the wait CPU trace (Bottom trace in the figure) the first activity at about 11:20 is a single DBench process. The wait CPU is about 30%. Since this is a dual processor machine that suggests that on average a DBench processes is about 60% I/O and 40% processor on this machine. Two concurrent processes cause the wait I/O to go to 60% as each CPU is executing a DBench process. As additional processes are added, parallel CPU and I/O activity occur and CPU wait begins to decrease. User and kernel CPU activity increase, indicating significant processor work executing in parallel with the I/O. In fact, the system becomes processor bound at about 16 processes. The reason for this change compared to same process using direct attached storage is that the proportion of processor to I/O time is increased and storage is no longer a bottleneck. Furthermore, the system is better optimized since both I/O and processor subsystems are fully utilized. It would clearly be useful to identify systems and classes of application workload that would benefit from faster storage.

1.5.5 Workload Profiling

To quantify the value of improved storage on systems and applications it is important to identify the I/O and system state. I/O state is parameterized by many variables such as I/O rate, write/read ratio, request size, and sequential fraction. System state is concerned with the processor utilization on both an application and aggregate scale. The time dimension is also important. If a system is I/O bound for a short period during the day the value in upgrading the storage system may be minimal. Also, if I/O bound intervals overlap consistently by time of day for several applications it is may be possible to improve performance by rescheduling rather than hardware upgrades. To identify these states we can plot the performance data in the multidimensional workload space and attempt to find patterns of activity where storage is a factor. A simple example from the data collected in this paper is shown in Figure 13.

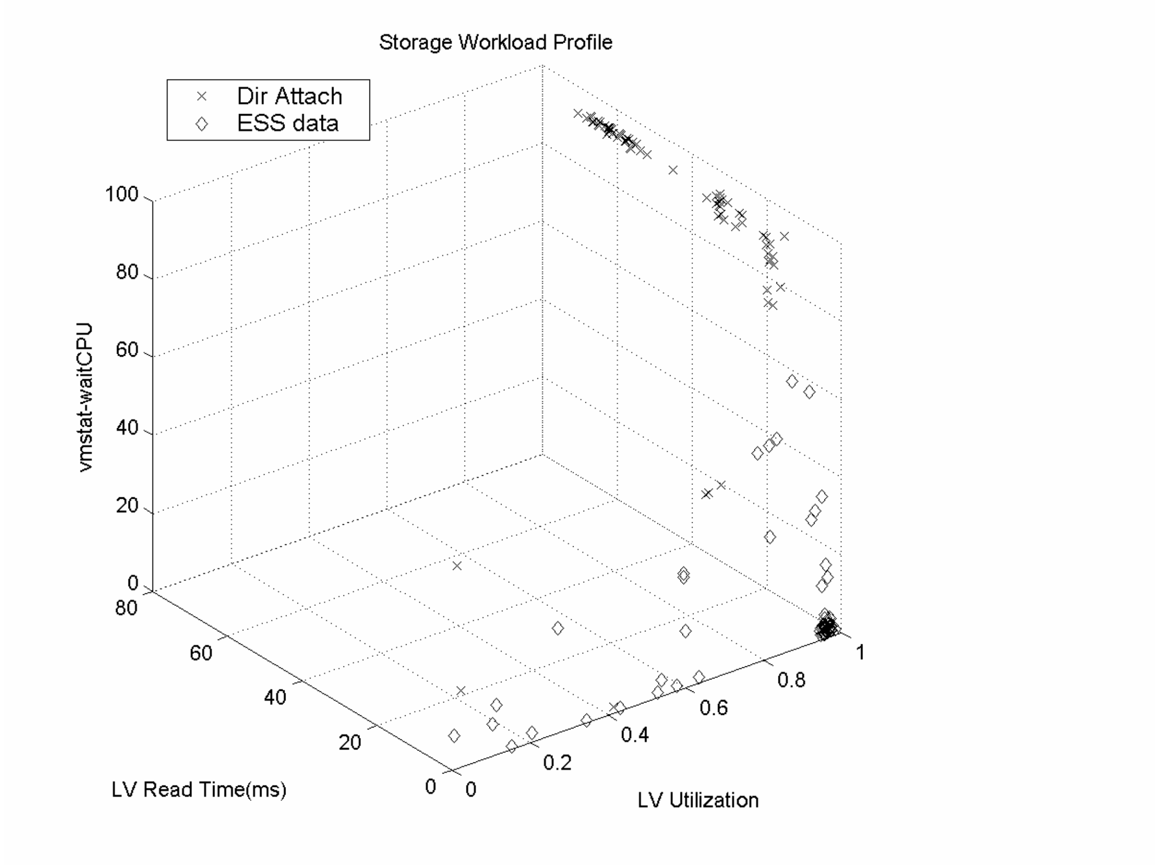


Figure 13. The workload profiles for the direct and ESS storage systems over a short period of the day are displayed in the scatter plot. The largest percentage of ESS accesses did not cause significant wait CPU time.

Each point is a sample of the workload space of the system. The data collection duration represents two periods of several hours. One sample period DBench was executed against direct attached storage ('x' markers), and the other against ESS ('diamond' markers). The density of data points shows a clear distinction in profile between the two periods of the day. The idea is to extend the analysis to include 24 hours of data and use analytic methods such as cluster detection to identify workload patterns by time of day or day of week. These patterns are combined with simple models and system data to quantify the potential benefit at the application or system level of improving storage performance. Such analysis may also be used to identify storage contention, or anomalous utilization of the system. Historical trending of the pattern data may also be useful in capacity planning or reconfiguration to optimize resource utilization.

1.6 Dependency

System dependency relations are important for identifying storage bottlenecks. Some of our conceptual work is outlined here. We have implemented system configuration mapping on Unix for the archetype operating and storage system layers discussed in this paper. When storage is network attached it is important to extend this map into the network.

We consider dependency relations between nodes in a graph as shown below. The nodes in the graph are classified into logical levels of an operating system and the dependencies are between elements at different levels. There are two perspectives for dependence relations between elements of a software or storage system. Structural dependency is how elements are interconnected in a static sense. The structural dependency graph is obtained from knowledge of system components. The dependencies are linear and the weights connecting nodes for the structural dependency graph are restricted to the binary set [0,1]. Dynamic or quantitative dependency is how data or control flows between elements as they are used. There are separate dynamic dependency graphs corresponding to each metrics accumulated at a node. For example, read counts, write counts, and read times produce different dependency strengths. Furthermore, the relations may not be linear, and we may be interested in relations between different metrics at each level such as response time at the LV as a function of read rates at the PV. The direction of flow (e.g. LV to PV or PV to LV) also produces different dependency relations.

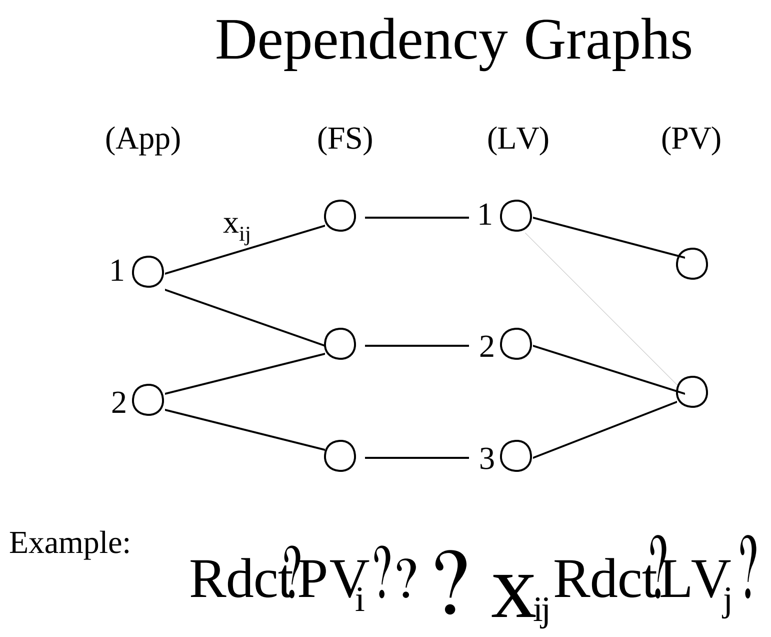


Figure 14. Example of a dependency graph for storage.

1.7 Conclusion

? It is erroneous to rely on disk utilization to indicate storage bottlenecks.

? The LV layer response combined with system wait I/O time may give a better figure of merit for determining application sensitivity to storage

? Standard deviation of response is useful measure of response bottlenecks. We observe good correlation at LV and PV layers between std and avg response time

References

1. Chen and Patterson, “Self-scaling Benchmark”, proceedings IEEE, 1991.