RC23812 (W0511-148) November 21, 2005
Computer Science

IBM Research Report

Host Metrics to Relate Application Performance to Storage Performance

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

Host Metrics to Relate Application Performance to Storage Performance

Norman Bobroff, Kirk A. Beaty, Andrzej Kochut
IBM T.J. Watson Research Center
{bobroff, kirkbeaty, akochut}@us.ibm.com

Abstract

The storage service time distribution at the host is introduced as a new metric for storage performance. Uses include problem determination and application performance diagnostics. This metric has considerable advantage over tradition storage system benchmarks, especially in shared storage environments. The utility and limitations of the IOWait operating statistic are also explored. A methodology to quantify the relation between application and storage system performance based on symmetric treatment of system resources is proposed.

1 Introduction

Performance of the I/O subsystem can be a key component of application and system performance. System administrators would like to know how application response time or throughput is affected by the storage subsystem. Can the storage subsystem be reconfigured to achieve better application performance? Another area of interest is how application performance scales with increasing load. Is the application scaling limited by processor utilization or storage system performance? Is it more effective to add processing power, or should expenditures be directed at the storage tier? It is difficult to answer these questions using benchmarks of storage components because of the variability and complexity of application access patterns to storage. Furthermore, applications and operating systems have adopted sophisticated strategies to minimize the effect of the large response time differential between memory caches and hard disks.

Migration from direct attached storage (DAS) devices to storage area networks (SAN) adds further complexity to understanding the relation between application and storage performance. A storage area network (SAN) environment introduces new degrees of freedom to the interaction between hosts and storage. SANs add resources such as caches, high end storage devices, and increase the concurrency of data access. These resources potentially improve overall performance over DAS. Read and write hits in a SAN cache provide an order of magnitude improvement in response time for individual storage requests that missed the local cache. But the overall benefit to an application of SAN cache hits depends on many factors. These include the application level of concurrency of I/O and CPU tasks, read-ahead and write-back strategies, and the relative amount of CPU and storage I/O activity. Additionally, SAN resources are shared with multiple hosts and response times can be degraded by contention for the fabric, storage caches, and disk controllers.

Traditionally, system administrators address these issues by reviewing and analyzing cumulative experience to generate 'best practices' for DAS or SAN device and fabric configuration, as well as data placement for an application such as a database or web server.

The approach taken in this paper is to consider host level metrics that contribute to understanding how application performance is affected by storage. A large number of application related performance metrics are available at the host. This work is restricted to metrics that they do not require application instrumentation and have not been widely explored in the literature. Two that in particular meet these criteria are;

- Service time distribution for storage access. This is measured at the Host Bus Adapter (HBA).
- IOWait, particularly how IOWait scales with application load.

Section 2 is a brief overview of the flow of storage requests from a host application to SAN storage provider. This helps in understanding some of data and analysis. Section 3 describes the host metrics studied in this paper and how they relate to application performance. Section 3 describes how each metric addresses some of the application and storage performance issues in the introductory paragraph of this section. The section explains how host level service time distributions are used to extract key performance features of the backend storage boxes. This is a general feature of host side measurements. By aggregating suitable host metrics it is often possible to obtain a quantitative analysis of the performance of SAN elements. As noted above, the converse is very difficult.

Section 4 explores a quantitative understanding of the relation between system throughput and response time when concurrent jobs use both the processor and storage resources. We explore whether these very different types of service centers (or resources) can be treated equivalently and symmetrically when quantifying the effect of their performance on overall system throughput and response time.

Disclaimer: This paper includes data from commercial host platforms, SAN fabric, and storage devices executing benchmark programs as well as production software. The results are not certified in any way and should not be used to infer capabilities of the platforms; they are for illustrative purposes only.

2 Flow of I/O Requests in a SAN

Figure 1 shows the key elements of a SAN that impact the performance of a SAN data request. An application I/O request proceeds either through the local file system or an application specific cache. The latter is common for commercial databases which allocate a configurable amount of system memory for a dedicated cache.

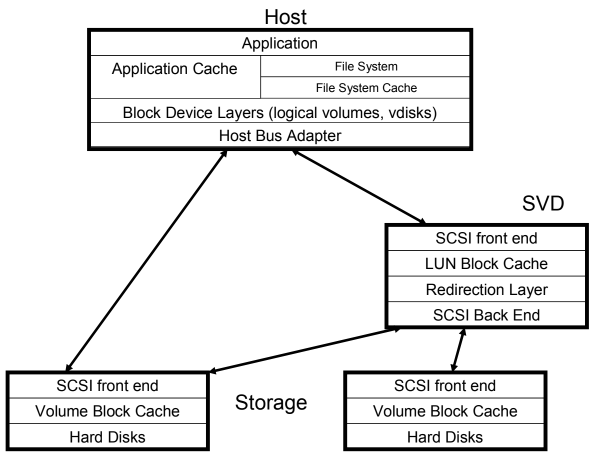

Figure 1. Overview of storage request flows in a SAN

When the request misses the local cache it is queued for handling at the HBA by the block device management code in the OS. The request may be subdivided or coalesced with adjacent blocks during this process. The HBA is then responsible for initiating the request to the appropriate SAN device for servicing. From a performance perspective the requests sent to the HBA and SAN represent the cache miss penalty associated with the application.

The host volume may be directly attached to the backend storage host in the logical sense (i.e. no intermediary SAN virtualization device as the left arrow in the figure), or remapped through a storage virtualization device (SVD). In the former case the request is served out of the storage device cache or the backend RAID array for a miss. In the latter case of a SAN virtualized volume the request is served in the SVD cache or redirected to a backend volume.

3 Metrics for quantifying application and system dependence on storage

In practice, it is preferable to find metrics that do not require application instrumentation or reporting. These can be correlated to application specific metrics in offline experiments to demonstrate their usefulness. We define several metrics that can be obtained from the OS and correlated with application response times.

Section 3.1 shows that the service time distribution is much more valuable than the summary response time statistics reported by operating systems. Efficient mechanisms to collect and store the distribution are the subject of Section 4.

The OS statistic IOWait is a potentially valuable metric quantifying the relation between application and storage performance. It is a widely applied for troubleshooting system performance, typically as a rule of thumb or best practice indication that storage is a bottleneck if, for example, it exceeds a threshold such as 30%. However, it is difficult to interpret the relation between this metric and application performance when there is significant concurrency of software tasks. Tasks that use the CPU while others are waiting on IO reduce the system IOWait. This masks the effect of IO on overall total response time. Section 3.2 develops a methodology based on IOWait as a function of concurrency that enhances the value of IOWait as a performance statistic.

Subsection provides some new perspectives on more familiar metrics. In particular, the significance of concurrency in servicing storage requests, especially for a SAN.

3.1 Service Time Distribution at Host Bus Adapter

Most operating systems report average, min, max, and standard deviation of service time for reads and writes at each drive. These summary statistics of the frequency distribution provide a general indication of storage performance. However, they are of limited value as indications of a storage performance problem, or finding a root cause, or suggesting a solution. The average read response time is sensitive to the cache hit ratio (CHR) in the SAN. However, temporal variation of the CHR in the SAN is expected in normal operating conditions because of changing workload from the host and its application. Furthermore, SAN caches are shared among hosts leading to variability in the CHR seen at a particular host. For these reasons, the traditional drive response time statistics are not necessarily useful in diagnosing a storage performance problem with remote storage.

Storage I/O requests are queued for service at the HBA. In the case of multiple HBAs, multi-path driver technology is used to load-balance the requests among the HBAs. The service time at the HBA is the time interval from when a request is initiated in the SAN until an acknowledgement of request completion, and data in the case of reads, is received back at the HBA. The service time distribution is the frequency distribution of these response times. Figure 2 shows the read response time distribution for four host volumes mapped to three exported SAN storage volumes, and a local IDE drive. The horizontal axis is the response time seen at the host. Data are for 4K reads requests on a 2Gb SAN fabric so the transfer time is small (a few microseconds) compared to the latency.

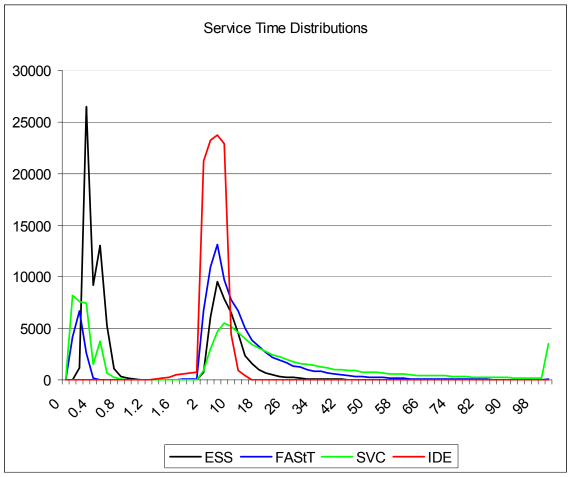

Figure 2. Service time distribution for several SAN devices.

The distributions are typically bimodal. The concentration of events with response times less than approximately 2ms corresponds to cache hits. The cache hit peak exhibits fine structure. This is caused by the presence of multiple caches in a SAN. The curve in Figure 2 labeled SVC is data from a backend RAID storage controller (Engenio FAStT 700) accessed through an intermediary SVD (see Figure 1). The SVC has a large cache that produces the fastest response times. Requests that miss the SVD cache proceed via the SAN fabric to the FAStT. A fraction of these requests are serviced from the controller cache. This leads to a second cache hit peak of the figure.

The peak at longer response times is the result of physical disk accesses. Although most accesses are serviced in a well defined time, an extended tail is present for several devices. The tail is indicative of the request scheduling and reordering algorithms in the SVD and backend storage controllers.

Inclusion of the originating process ID in the IO event timing information allows the distribution of Figure 2 to be aggregated in a number of useful ways. For example by: reads, writes, application process, or class of process when workload management software is available. Instrumentation issues are described in section 4.

The service time distribution is a valuable source of metrics that quantify how SAN performance is reflected at the host. One of the most important is the dynamic measure of the effective CHR seen at the host. The CHR is the ratio of events under the two peaks in the service time distribution. For example, this allows dynamic measurement of the CHR at many levels of aggregation.

There are performance differences between local and SAN cache hits. A cache hit in the SAN incurs significant overhead relative to a hit in the local host. This is caused by the fabric protocol time and is about 0.1ms for a

2Gb/s fibre channel SAN using the SCSI fibre channel protocol (FCP). This exceeds a local host cache response time by about a factor of 100. However, 0.1ms still represents a typical gain of 10 to 100 over a physical disk access. (This is one reason why host summary stats are not illuminating for SAN storage. The average response time can vary greatly with a small change in hit ratio of remote caches.) Another distinction is that a local cache hit does not cause the requesting thread to yield the processor, while a remote request does. *One might expect yielding to limit the performance benefit of an IO bound process competing with a CPU intensive process on a single processor machine*. However, we found that this is not the case. Dispatchers and schedulers for major operating systems prevent IO thread starvation by providing a boost to the process waiting on IO (Linux, Windows, AIX, zOS).

The effective SAN CHR can be used in several ways to understand and optimize application performance. The SAN CHR is especially important from a performance perspective because it reduces the miss penalty of the local host cache. A history of SAN CHR can be co-analyzed with similar logs of host or application utilization and performance. This data establishes baseline performance, and measures correlations between the application and CHR metrics. The general discussion of the relation between application performance and the metrics such as SAN CHR is delayed until section 3.2 because the IOWait metric plays a significant role in the analysis.

For applications that do benefit from increased cache, many practical considerations enter in to the decision of whether to place the cache on the hosts or in the SAN. These include whether the host OS has memory limits, and the cost/gain of adding cache to many hosts against a centralized SAN cache. When a SAN cache offers the best solution, the dynamic SAN CHR can be used to optimize this cache. Although not available in current SAN protocols, many methods of dynamically passing information from hosts to SAN cache have been proposed (e.g. [1-3]). Host based dynamic CHR measurements offer an intelligent basis for driving optimizations such as hints to SAN caches to dynamically prioritize or partition cache space for certain hosts or applications.

We briefly comment on other uses of the service time distribution. The position and width of the peaks can be combined with the historic minimum to infer the amount and root cause of SAN congestion. A broadening and shift of the cache hit peak indicates congestion in the fabric or at the cache itself. A similar change in the physical disk peak indicates contention at the disk controller in the backend storage device. This contention analysis is presented in detail elsewhere [4].

The introduction commented that acquisition of key storage performance metrics at the host provides a more general solution than obtaining the statistic from SAN elements. The dynamic host CHR metric is an example. For a host volume mapped directly to an exported RAID device, the statistic can be obtained by direct query of the cache controller on the backend storage server. However, introduction of client based logical mapping such as striping or SAN based virtualization devices increases the difficulty of a non host solution. A performance manager would have to maintain explicit knowledge of the mapping, and perform and aggregate queries of the relevant cache controllers.

3.2 IOWait

System IOWait is the fraction of time the processor is idle and at least one thread is blocked on storage I/O. Figure 3 illustrates IOWait for a uniprocessor executing two threads. The vertical axis indicates the state of each thread as time increases along the horizontal axis. Each thread from a pool services a request from an input queue. The request represents a fixed amount of work, for example, building a web page or accessing an email. Each request requires the CPU, a storage system access, and concludes on the CPU. The vertical arrows indicate the thread finished the request and waits for the next piece of work to enter the system. IOWait is accumulated when neither thread is executing and IO is pending. On a multi-processor the OS reports the average over all processors.

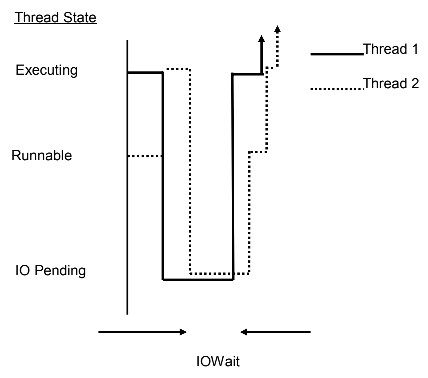

Figure 3.Thread state

IOWait is a function of many factors. A change in the frequency or service time of storage access reduces IOWait. This can be achieved by many well known optimizations such as increasing local or remote cache, or changing the storage configuration or RAID level. Concurrency of access to the storage system is an important factor in IOWait. Performance degrades when thread 2's IO waits for thread 1's IO to complete. This is especially important for a SAN both at the host, and at the storage controller where requests from multiple hosts may be forced to queue.

IOWait is also reduced by increasing the concurrency of workload. Servicing more requests in parallel allows the CPU to be active while other requests are waiting for IO service.

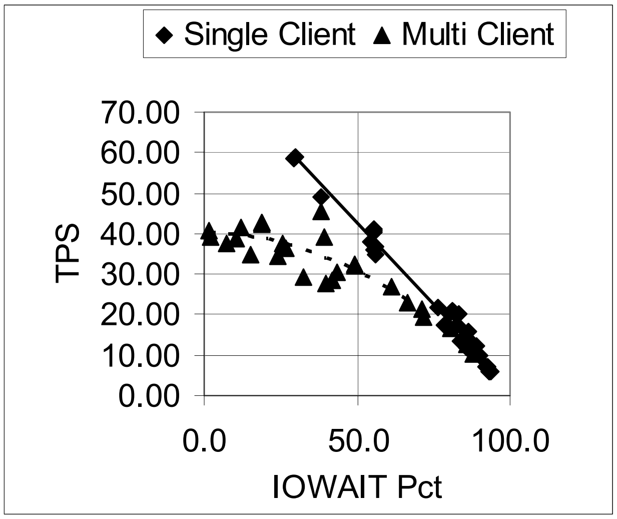

Figure 4. Throughput of TPC-C benchmark

Figure 4 shows the throughput of the TPC-C benchmark as a function of decreasing IOWait. IOWait was varied by using faster storage and increasing concurrency (multiple TPC clients). In both the single and multi-client data sets storage performance is modified by cache optimizations. As expected, the single client performance scales linearly with IOWait as processor utilization increases, with IOWait tending to zero with very fast (100% cache hit) storage.

Concurrency reduces IOWait as in Figure 3. But throughput reaches a plateau because the CPU overhead per transaction increases with the number of concurrent requests managed by the TPC-C and OS code. Concurrency has exposed the nonlinear scaling of the TPC application. In this case, storage performance degrades the response time of requests, but the throughput is CPU limited. It is not possible to quantify the relation between response time and storage performance from the data of the figure, illustrating a weakness of the IOWait statistic.

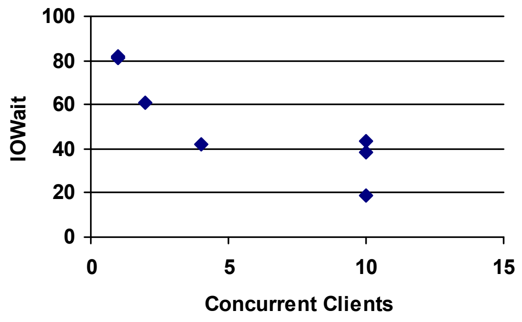

Figure 5. Scaling of IOWait with concurrency for TPC-C

Figure 5 isolates the effect of concurrency on IOWait. IOWait decreases as processes consume CPU while others are serviced in the IO system. At a concurrency of 4 clients, access to the storage system appears to become serialized. This prevents the system from becoming CPU limited.

In general, three types of scaling behavior of IOWait with concurrency (n) are possible, as indicated in Figure 6, which we now explain. The path taken by the system depends on the relative scaling of CPU(n) and IOWait(n). CPU(n) is the amount of processor time required to complete a unit of work when ‘n’ threads are concurrently servicing work. CPU(n) increases with n because both application and kernel operations require more cycles to manage data structures and scheduling. Similarly, IOWait(n) increases as requests queue awaiting access to the storage system.

In the storage limited pattern IOWait(n) is growing faster than CPU(n). Additional work uses some of the CPU cycles available during IOWait, but causes the IO delay for all jobs to increase. Thus IOWait reaches a point where it fails to decrease. The storage limited pattern occurs when the storage system cannot concurrently service the increased number of IO requests. Request queuing mitigates the potential gain in CPU utilization and application throughput. The bottleneck can occur as a software lock (e.g. database table), at the HBA or the backend storage controller. Commercial HBA cards are often specified as having the capability to handle 256 outstanding FCP requests. However, we have observed that the drivers saturate at 32 requests. HBA concurrency is improved by adding FCP adapter cards and a multi-path driver which supports dynamic load balancing between the cards. Queuing at the backend controller is reduced by distributing requests across controllers. Best practices emphasize the importance of data striping across large numbers of disks and controllers [5].

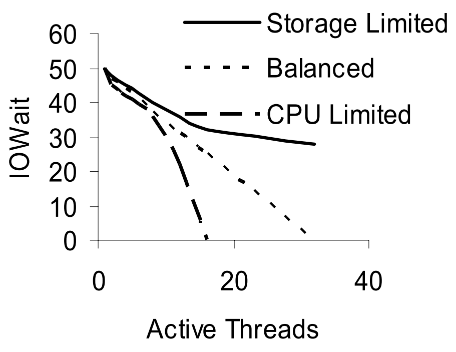

Figure 6. IOWait as function of concurrency

In the CPU limited condition, CPU(n) grows faster than IO(n). The extra CPU cycles quickly fill in the processor idle time during IOWait as concurrency is increased. CPU utilization approaches 100%. This situation typically occurs when CPU utilization and concurrency are high. The balanced state is a subset of the CPU limited condition in which CPU utilization is scaling linearly with increasing clients.

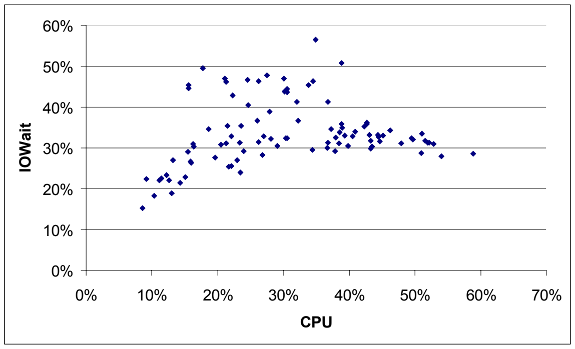

Figure 7. CPU and IOWait

Another perspective on the relation between IOWait, CPU, and concurrency is provided in Figure 7. The data are taken from a production Domino Version 6 mail server running on an AIX partition and present a scatter plot of IOWait and CPU samples taken over the prime shift. During this time the number of concurrent user sessions varies from approximately 180 to 1000. Figure 8 shows the CPU utilization for the data of Figure 7 as a function of the number of concurrent user sessions. User sessions have long term persistence and include user think time so they do not directly translate to number of concurrent requests active on the server. However, as indicated by the linearity of the CPU utilization data, the session count is a reasonable proxy for concurrent work load.

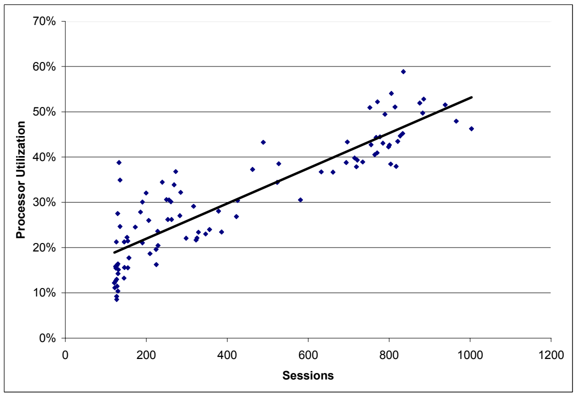

Figure 8. CPU utilization

The storage system distributes mail files across 72 hard disks attached by a serial SSA bus in order to distribute load

This system demonstrates an interesting behavior of IOWait with the number of concurrent session. After an initial increase, IOWait plateaus at about 30%, even as average CPU utilization increases. The plateau occurs because the Domino server has an admission policy that limits the amount of work in the system to N jobs. As the sessions increase, requests are queued 'externally' i.e. not admitted to the processor center when they exceed N. As the sessions increase more overall time is spent handling jobs so CPU increases at the expense of idle time. However, because the number being concurrently serviced in the processor or storage can never exceed N the IOWait doesn't decrease. This case illustrates the difficulty of interpreting the IOWait statistic.

It would be much more useful to have a metric that directly agrees with our intuition of what storage utilization would be. This metric would allow a fully symmetric treatment of the processor and storage resources. This subject is pursued in section 4.

3.3 Concurrent I/Os

Most adapters report utilization, commonly defined as the fraction of elapsed time that an I/O request is pending on the adapter. This definition is usually deficient because adapters and protocols are often capable of supporting multiple concurrent requests. A high reported utilization does not mean the adapter or SAN channel is a bottleneck. Thus, it is useful to measure the distribution of outstanding requests at the adapter. For SCSI over FCP we have observed that current cards (e.g. QLogic 2300) support up to 32 outstanding requests. When the average number of outstanding requests is at the HBA limit the host can benefit from an additional adapter and parallel connection to the SAN. A significant benefit of SAN storage over direct attached is the concurrency of request service.

4 Relating storage performance to application performance

The primary metrics at the application level are throughput and response time. These depend on the programming model, processor, and storage performance. Some of the central questions we would like to address from system metrics are:

- How do application performance metrics respond to changes in processing power, storage response time, or the programming model?
- If application demand grows how will response time scale and what is the bottleneck resource?
- When storage is the bottleneck, what area of the system or storage infrastructure can be improved to provide the greatest application level benefit?
- What is the most cost effective way to improve application performance metrics?

Furthermore, we would like to express the answers in terms of metrics and analysis that is familiar and readily comprehended by system administrators. In general, these questions cannot be answered without detailed knowledge of the application resource demand and possibly the introduction of new OS level metrics.

We now explore the impediments to a general solution to gain insight on the metrics and analysis required to provide direct answers.

One difficulty is the asymmetry in the typical treatment of CPU and storage utilization. CPU utilization is the fraction of time the processor is occupied. This statistic is generally accepted considered to be a linear measure of the amount of work done in the CPU. The comparable definition (the fraction of time that one or more requests are outstanding at a disk controller) does not apply to a storage system. A disk has many optimizations such as cache, read-ahead, and request reordering. The effect of these on service time for different request workloads (defined by locality of reference) is highly variable. Thus storage utilization should be treated within the context of a flow dependent service center i.e. the throughput is a function of the number of jobs in the service center [6].

In fact, closer examination of the processor utilization metric shows similar difficulties. CPU utilization does not always represent a proportional measure of progress. The processor is subject to workload dependent variables such as efficiency of hyper-threading architectures, memory stalls, and load dependent cache effects, especially in a multiprocessor environment.

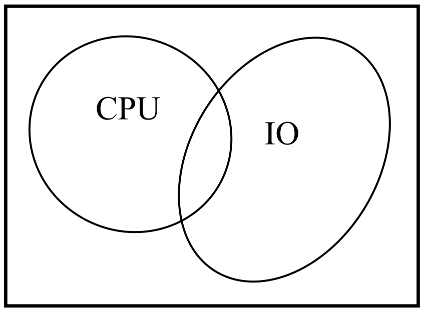

Figure 9. Symmetric view of CPU and IO utilization

A symmetric treatment of resource utilization is presented in Figure 9. The area of the box represents a utilization of unity. The size of each set represents the corresponding resource utilization of the CPU and IO subsystems. The shape is not important here. Joint utilization is proportional to the overlap, and IOWait is the fraction of storage utilization when CPU is idle. Figure 9 shows how the archetypical behaviors discussed in the context of Figure 6 are represented when resource utilizations are treated uniformly. The lines are the paths taken in utilization space as the load increases.

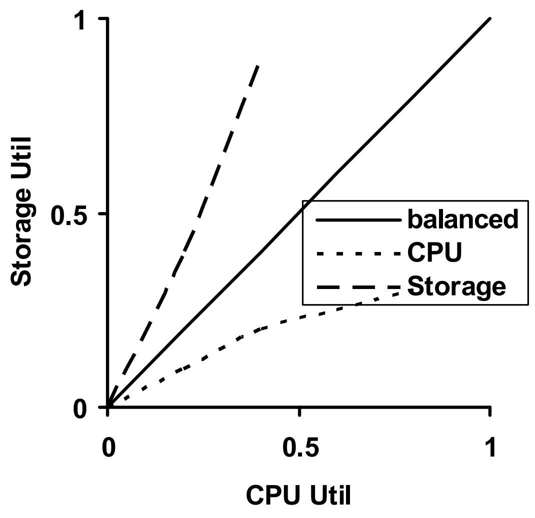

Figure 10. CPU and storage utilization archetype behavior with load.

Knowledge of the utilizations and how they scale with load allows determination of the limiting resource and the maximum achievable throughput. It also provides a means to quantify the value of performance improvements to either the processor or storage systems. Joint utilization is a useful indication of the concurrent resource access of the application programming model.

The other important application metric, response time, is not readily inferred from this data. It requires approximate knowledge of the periods of time spent at the resource

5 Conclusions

This work explores the merits of host based metrics for storage performance, and how they relate to application performance, with particular emphasis on a shared storage environment such as a SAN. The service time distribution at the host bus adapter provides a unique perspective on how storage accesses are perceived by an application program. It allows measurements of the dynamic cache hit ratio, as well as indicates where performance bottlenecks occur.

Accurately relating storage performance to quantitative application metrics such as throughput and response time is a difficult problem. The IOWait statistic is useful in certain circumstances as a measure of when storage utilization reaches 100%. This allows estimation of how the limiting throughput might change if the storage system is reconfigured. However, more information is required to predict how application throughput and service time respond to more general changes. The goal is a uniform methodology to measure utilization for all resources that the application consumes. This is a challenge because the maximum throughput of the storage system depends on workload, so new metrics are needed to represent this load dependent service center.

6 Acknowledgements

We want to thank Varada Manavalan and Brian Lynch for helping us to obtain data key to our experimentation from IBM GNA production servers.

References

[1] Lakshmi N. Bairavasundaram, Muthian Sivathanu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. X-RAY: A Non-Invasive Exclusive Caching Mechanism for RAIDs. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA '04)* , Munich, Germany, June 2004.

[2] Zhifeng Chen, Yuanyuan Zhou and Kai Li. Eviction-based Placement for Storage Caches. *The Proceedings of the USENIX Technical Conference (USENIX'03),* June 2003

[3] Theodore M. Wong, John Wilkes. My cache of yours? Making storage more exclusive. *USENIX Annual Technical Conference (USENIX 2002)*, pp 161-175 (Monterey, CA, June 2002).

[4] A. Kochut, N. Bobroff, K. Beaty, G. Kar. Management Issues in Storage Area Networks: Detection and Isolation of Performance Problems. In *proceedings NOMS 2004* VX, (Seoul)

[5] Bob Larson. Wide Thin Disk Striping. Sun blueprints online, October 2000, http://www.sun.com/blueprints

[6] Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevolk. Quantitative System Performance, Prentice-Hall 1984, ISBN 0-13-746975-6