hp storage

august 2002

technical white paper

disk array performance guidelines with application to the hp StorageWorks virtual array

executive summary

The original introduction of RAID storage array architecture provided a new approach to meet the ever-increasing online storage demands of enterprise computing environments. RAID arrays have provided upward scaling of storage capacity, availability and performance far beyond what was previously available in individual, large form factor disk drives. However, the new architecture brought with it a new operational model and new requirements for optimum configuration and usage of storage. As disk arrays first came into general use, many of the software environments and administrative practices were optimized for the old model. This has changed over time, but the situation persists to some degree even today. More recently, the addition of advanced virtualization features to disk array architecture has resulted in additional performance considerations. It can be very beneficial to consider the unique characteristics of disk arrays in general, and in many cases, the unique characteristics of particular disk array designs to realize the full performance potential that is available. This paper provides some background and guidelines regarding the configuration and usage of disk arrays for best performance and provides specific examples for the HP StorageWorks Virtual Array.

© Copyright Hewlett-Packard Company 2002

contents

executive summary 1
contents 2
disk array performance background 4
  why disk arrays? 4
  disk striping 4
    shallow striping 4
    deep striping 4
  caching 5
    write back caching 5
    write merging 5
    over-writes 5
    write caching and data integrity 5
    read pre-fetch 5
    read hits in a working set 5
    interaction with upstream caching 6
  redundancy 6
    performance impact of redundancy 6
    cost of redundancy 6
    RAID 1+0 6
    RAID 5DP 6
    other RAID 5 write processes 7
    AutoRAID addresses the performance impact of read/modify/write 7
performance guidelines 7
  array configuration 8
    performance guideline #1: RAID level 8
    performance guideline #2: disk configuration 8
    performance guideline #3: cache size 8
    performance guideline #4: cache operational parameters 9
    performance guideline #5: LUN configuration 10
    performance guideline #6: usage history 11
    performance guideline #7: I/O routing 12
  workload 12
    performance guideline #8: concurrency 12
    performance guideline #9: synchronous I/O 13
    performance guideline #10: I/O block size 13
    performance guideline #11: volume manager striping 14
    performance guideline #12: performance benchmarking 15
glossary of performance terms and concepts 15
  application 15
  workload/demand 15
  stream/process 15
  open loop workload/demand 15
  closed loop workload/demand 15
  workload intensity 16
  performance benchmark 16
  response time 16
  service time 16
  throughput 16
  MB/sec 16
  IO/sec (IOPs) 16
  TPS 16

© Copyright Hewlett-Packard Company 2002

sequential 17
multi-stream sequential 17
random 17
utilization 17
saturation 17
storage hierarchy 18
working set 18
thrashing 18
for more information 18

© Copyright Hewlett-Packard Company 2002

disk array performance background

A basic understanding of disk array architecture as it relates to performance will provide a basis for performance configuration guidelines. This is a brief treatment of the subject. A more complete description of RAID architecture is available in "The RAIDbook" published by the RAID Advisory Board: http://www.raid-advisory.com.

why disk arrays?

The capacity, availability and performance requirements of applications historically have exceeded the capabilities of individual disk drives. Coupled with commoditization of small form factor disk drives this has led to the wide spread use of disk arrays. The main values being added by disk arrays over the direct use of multiple disks by an application are manageability and modularity of function. For example, suppose an application has a requirement to store a single data file that is larger than any available disk drive. The application software could be modified so that it can split up the data file across multiple disk drives in an application specific way. With that approach, every application that had this requirement would need to be modified. Disk arrays provide a layer of virtualization above disks that presents composite capacity, availability and performance characteristics. With disk arrays, applications, system administrators and users need not be concerned with the details of combining multiple disk drives to form "virtual" disks.

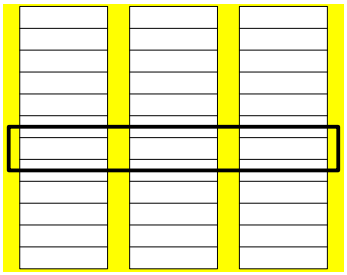
A disk array in the broad sense is any combination of multiple disk drives whose access and management is presented to higher levels in the system through some form of virtualization software. The virtualization software can take many forms including: file systems, volume managers, databases, storage device drivers, firmware on storage bus adapters and firmware in dedicated disk array controllers.

> **Application note:** There is great advantage to locating the disk array software on a dedicated disk array controller as in the implementation of HP's StorageWorks Virtual Array storage products because it can be tightly coupled to the array hardware and specifically optimized for control of a storage array.

disk striping

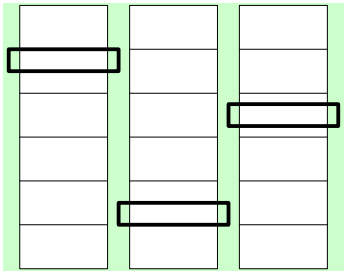
One of the simplest ways to form a disk array is to use a striping algorithm to spread data across multiple disks. This provides higher capacity since the capacity of all the disks are added together. It has the potential to provide higher performance because all the disks can possibly be used in parallel to service the workload. The performance benefit is realized only if the combination of the workload and the array management algorithms actually result in parallel disk operation. A number of methods are available to generate parallel disk operation.

shallow striping



With shallow striping, the size of the block of data placed on each disk in a stripe (the stripe block) is small relative to the read/write request size prevalent in the workload. A single read/write request from the workload corresponds to a sequential set of stripe blocks in the striping sequence, each from a different disk in the stripe. This has the effect of multiplying the data transfer rate by the number of disks on average used to service a request. This mode of operation is usually associated with the RAID level 3 and with applications that have a high sequential MB/sec demand although it is not a formal part of the RAID level 3 definition. This kind of striping is not implemented by HP StorageWorks Virtual Array products but is described here for completeness.

deep striping



With deep striping, the size of the block of data placed on each disk in a stripe (the stripe block) is large relative to the read/write request size prevalent in the workload. Most read/write requests are contained within a single stripe block on a single disk in the stripe. This does not effectively generate parallelism in disk operation unless there is concurrency in the workload as it appears at the disks. The concurrency can be inherent in the workload of the application due to multiple processes in the application or can be generated by data caching and management algorithms in the disk array. With the proper level of concurrency in the disk workload, deep striping has the effect of multiplying the I/O rate by the number of disks in the array. This mode of operation is usually associated with the RAID levels 0, 1+0, 4 and 5 and with applications that have a high random IO/sec demand although it is not a formal part of the definitions of those RAID levels.

> **Application note:** The HP StorageWorks Virtual Array products implement deep striping as RAID 1+0 and RAID 5DP.

© Copyright Hewlett-Packard Company 2002

Application note: A closed loop, single process application does not generate in itself a workload that contains concurrency. Some applications are of this nature by design because that is the true nature of the workflow process implied by the application. Others are this way because storage performance implications have not been considered to any large degree in the application software design. Perhaps of less importance but worth noting is that some performance benchmarks also behave this way by default or by configuration. This type of application will not realize the performance benefit of deep striping unless some other means is used to create concurrency.

caching

Read and write caching is one means often implemented in disk arrays to enhance performance. It can enhance performance in a number of ways.

Application note: The HP StorageWorks Virtual Array products implement read and write caching.

write back caching

One way that write caching can enhance performance is to create additional concurrency in the disk write workload not present in the application workload through write back caching. Concurrency is created because the application does not wait for a write operation to be completed by a disk. Rather, the write is reported as complete to the application immediately after the write data has been received and stored in the write cache of the storage device but before it is actually written to disk. This releases the application to continue and create another write request much faster than if the application had waited for completion of the disk write. In this way, an application can generate writes much faster than they can be completed by the disks and so they can be serviced in parallel by multiple disks. If the write cache becomes full, the application is forced to wait for the completion of disk writes as if a write cache were not available. However, in this case the write throughput has been increased because of parallel disk operation and other write cache optimization methods. The write cache acts like a buffer between the application and the disks to absorb temporary bursts of high write demand.

write merging

Another way that write caching can improve performance is by merging sequential writes into a single, larger write request to disk. While it would seem that this would reduce write performance because it reduces concurrency, there are actually two performance benefits. One performance benefit occurs if the merged writes would have all addressed the same stripe block on the same disk anyway. In this case, concurrency is not reduced by merging the writes and only one instance of overhead occurs instead of one for each unmerged write. Another performance benefit of sequential write merging is to reduce the number of parity updates that are needed to write the data. Disk arrays (except for RAID level 0) store both application data and redundant data (see "redundancy" section below). If writes to multiple stripe blocks in a stripe can be processed in one operation, the parity for the stripe need only be updated once instead of once for every stripe block written. If a long series of sequential writes can be merged such that the merged write covers an entire stripe of a deeply striped array, the write data throughput (MB/sec) of the array can approach that of an array with shallow striping.

over-writes

Write caching can improve performance if the workload has a write working set that fits in the write cache. A cache over-write is a write that overlaps another write already held pending in the write cache. The overlapped portion of the pending write is replaced by the new write in the cache. This eliminates the need to post the overlapped portion of the pending write to disk. This can improve performance by reducing utilization of the disks.

write caching and data integrity

The risk to data integrity posed by write caching is usually addressed in disk array implementations with batteries. This has no direct effect on performance characteristics but it does enable the many performance benefits of write caching like write back caching.

read pre-fetch

One way that read caching can enhance performance is read pre-fetch. In read pre-fetch the read cache detects a sequential pattern of read access and anticipates the application workload by pre-fetching the next set of sequential data into the read cache before it has been requested by the application. Then when the pre-fetched data is requested, it is available immediately from the read cache rather than reading from disk. The pre-fetch can occur concurrently with reads that have already been requested creating concurrency in the disk read workload and resulting in parallel disk operation. Another name used to refer to read pre-fetch is "read ahead."

read hits in a working set

The read cache can also enhance performance if the workload has a read working set that fits in the read cache. In this case, there is a high probability that some requests will be satisfied from data that is already resident in the read cache from a previous request. This provides faster response time to the read request since it doesn't have to wait for the disk and

© Copyright Hewlett-Packard Company 2002

reduces disk utilization. However, this type of hit in the read cache is far less likely than a read pre-fetch hit because the read working set is often much larger than the read cache.

interaction with upstream caching

The cache in a disk array may not be the only cache in the data access path of a system configuration. For example, file systems and databases often implement their own data cache. This is referred to as an upstream cache. Upstream caches can heavily influence the application workload as it appears at the disk array. They often implement many of the same caching algorithms implemented in the disk array cache. In many cases, the behavior of multiple caches is complimentary. It is also possible that the behavior of multiple caches is contradictory. A prime example of contradictory cache behavior is a CPU resident read cache similar in size or larger than the read cache in a disk array. The CPU read cache would catch most the working set read hits so the disk array read cache would have very few working set read hits. On the other hand, read pre-fetch in a disk array read cache can act like an extension of read pre-fetch in a CPU read cache. It is important to consider how data caching outside the disk array will impact array performance.

> **Tip:** For a multi-server configuration in which data is not shared between servers, increasing server cache may provide greater value than increasing array controller cache. Increasing cache in a sequential workload may have minimal impact to performance. See cache size guideline section below.

redundancy

Disk arrays distribute application data across disks in a way that is transparent to the application. By design intent, the application does not have visibility to the details of the layout of the application data objects on the disks. As a result, for an array without redundancy, the loss of any one disk in the array could render useless the whole application data set stored since arbitrary portions of the application data would no longer be available. The availability of the application data set is dependent on the combined availability of all the disks in the array. Without redundancy, the combined availability is that of a single disk divided by the number of disks. Availability decreases as more disks are added to the array while capacity and performance grow. This is contrary to application needs that typically require capacity, performance and availability to all scale upward. Together, this results in a need for redundancy so that availability does not depend on the failure of a single disk.

performance impact of redundancy

Maintaining data redundancy in a disk array has an impact on the performance of writes but not on the performance of reads. Additional disk writes are needed so the data can be stored in a redundant form on multiple disks instead of just on a single disk. Then if a single disk fails, the data can be recovered from redundant data stored on the surviving disks. The additional disk writes needed to maintain redundancy in some cases can proceed in parallel on multiple disks so there is not a significant impact on the response time of the write operation. Even so, the need to write to multiple disks does increase the utilization of the set of disks over the non-redundant case so there is a performance impact. In other cases, the process required to maintain data redundancy results in some serialization of disk operations. In these cases, the performance impact includes both response time and disk utilization effects.

cost of redundancy

In addition to the performance impact of redundancy, there is also a cost to yield the same virtual capacity as a non-redundant array. Additional disks are needed to store the redundant data.

RAID 1+0

In the HP StorageWorks Virtual Array implementation of RAID 1+0, the data is both striped and mirrored on the disks. Each write request issued from the write cache to the disks actually results in write operations to both the primary disks and the mirror disks. The primary and mirror write operations proceed in parallel so the performance impact of redundancy is restricted to increased disk utilization. This results in an approximate factor of two reduction in the IO/sec performance that could have otherwise been obtained from the disks without redundancy since twice the number of disk operations are required (N/2 where N = # of disks). For example, an array of 30 disks, each of which is capable of 100 IO/sec at a given response time would provide about 1500 IO/sec of write performance (30 x 100 / 2) in RAID 1+0 mode. Since the read and write IO/sec throughput of disks are approximately the same, a simplifying rule is that the IO/sec write throughput of RAID 1+0 is expected to be about 1/2 the IO/sec read throughput. Two full copies of all data are maintained in RAID 1+0 making the cost approximately a factor of two higher than a non-redundant array (2 x N where N = # of disks = required virtual capacity / capacity of a single disk).

RAID 5DP

In the HP StorageWorks Virtual Array implementation of RAID 5DP, the data is striped and two blocks of redundant data (parity blocks) are maintained with each stripe. A write request issued from the write cache that addresses a portion of a single stripe block and is not merged with any other write request is handled by the traditional RAID 5 read/modify/write process. The data from the data disk and each of the two parity disks is read into a buffer, the old data is subtracted from

© Copyright Hewlett-Packard Company 2002

the two parity blocks, the new data is added to the parity blocks then the data and parity is written back to the disks. The read/modify/write sequence is serial on each of the disks so there is a response time impact to maintain redundancy. However, with write back caching in the array write cache, the response time impact is not visible to the application unless the array write cache is full because demand is approaching the saturation level. The read/modify/write sequence proceeds in parallel on each of the three disks so there is also a disk utilization impact to maintain redundancy.

The read/modify/write process in RAID 5DP results in an approximate factor of six reduction in the IO/sec performance that could have otherwise been obtained from the disks without redundancy since six times the number of disk operations are required (N/6 where N = # of disks). For example, an array of 30 disks, each of which is capable of 100 IO/sec at a given response time would provide about 500 IO/sec of write performance (30 x 100 / 6) in RAID 5 DP mode for a workload that results in mostly read/modify/write operations. Since the read and write IO/sec throughput of disks are approximately the same, a simplifying rule is that the IO/sec write throughput of RAID 5DP for this type or workload is expected to be about 1/6 the IO/sec read throughput. Two additional disk's worth of capacity are required by RAID 5DP to store the redundant data. The additional cost for redundancy in RAID 5DP is the incremental cost of the two extra disks (N + 2 where N = # of disks = required virtual capacity / capacity of a single disk).

RAID 5DP has much greater data availability than traditional RAID 5 because there are two parity disks per stripe instead of one. A RAID 5DP stripe can sustain the loss of two disks without loss of data availability. Traditional RAID 5 stripes are usually restricted to a maximum of four or five data disks to one parity disk (4+1 or 5+1) for availability reasons. The HP StorageWorks Virtual Array can have as many as 54 data disks (54+2) in a single stripe. The traditional RAID 5 would need as many as 14 parity disks (14 groups of 4+1 for a total of 56+14) to achieve the equivalent usable capacity. RAID 5DP has higher overall data storage efficiency and greater data availability.

other RAID 5 write processes

Read/modify/write is one write process that can be used in a RAID 5 implementation. There are additional write processes implemented in HP StorageWorks Virtual Array RAID 5DP and usually in traditional RAID 5 implementations that can be more efficient than read/modify/write. These processes involve the merging of multiple write requests into a single, larger operation that covers a larger portion of a stripe or even a whole stripe. The parity is updated once for the entire operation rather than once for each stripe block. In a traditional RAID 5, the ability to do this kind of merging is dependent on a sequential workload.

> **Application note:** The HP StorageWorks Virtual Array has additional flexibility to do partial and full stripe writes because AutoRAID technology allows blocks that are not sequential in the application data space to be merged into physically contiguous blocks in the RAID striping layout. This process is called "log structured writes."

AutoRAID addresses the performance impact of read/modify/write

Write merging is one way to avoid the performance impact of RAID 5 read/modify/write. However, some important classes of workload cannot be merged. An example would be a small, random write workload that is typical of an online transaction-processing (OLTP) database. Historically the performance impact of read/modify/write has been one of the major drawbacks to RAID technology.

> **Application note:** AutoRAID technology in the HP StorageWorks Virtual Array addresses this issue by providing the option to implement a storage hierarchy for writes using RAID 1+0 and RAID 5DP. RAID 1+0 has higher performance for writes (approximate N/2 compared to N/6) while RAID 5DP has lower cost (approximate N + 2 compared to 2 x N). AutoRAID is designed to maintain the write working set in the higher performance RAID 1+0 and the rest of the application data set in the lower cost RAID 5DP. The application receives the approximate write performance of RAID 1+0 and the approximate cost of RAID 5 DP as long as the write working set fits in the RAID 1+0 storage area. AutoRAID reserves 10% of the usable capacity for RAID 1+0. This is based on studies that have shown the write working set is typically less that 10% of an application data set. If multiple applications on the same or different servers use a single array, the 10% rule applies to the fraction of the array usable capacity dedicated to each application data set so that the total write working set for all applications is still 10% of the array usable capacity.
>
> AutoRAID implements policies whose purpose is to maintain the write working set in RAID 1+0 at all times. Performance metrics are available in the performance log to observe the operation of AutoRAID policies. The metrics can be used to determine how well AutoRAID is achieving this goal.

performance guidelines

Array performance guidelines can be separated into two broad categories. The first are those related to the configuration of the array itself. The second are related to the nature of the workload presented to the array.

© Copyright Hewlett-Packard Company 2002

array configuration

Array performance guidelines related to array configuration include the hardware configuration of the array - things like the number and speed of the disks, the amount of cache and the number of host and disk I/O channels for example. It also includes the software configuration settings of the array - things like RAID level, and caching parameters.

performance guideline #1: RAID level

The most basic configuration choice found in most arrays is the choice of RAID level. This selection will affect the cost of storage and many of the operational characteristics of an array, including performance. Typical choices available are RAID levels 0, 1, 1+0, 3 and 5. The performance capabilities of the different RAID levels implemented in an array are usually about the same for reads. RAID 3 is an exception because it usually uses shallow striping to optimize high bandwidth, single process sequential workloads. The impact of RAID level choice occurs on writes because extra work is required to maintain redundancy in the stored data. RAID 0 almost always offers the highest write performance because it doesn't maintain any redundancy. RAID 1 or 1+0 is usually the next best for small, random writes and RAID 5 usually has the lowest write performance for small random writes. A good implementation of RAID 5 can provide higher performance for large sequential writes than RAID 1+0 because there is less redundant data to be written and so the redundant data consumes less of the controller's data transfer bandwidth.

Application note: The HP StorageWorks Virtual Array offers RAID 1+0 and AutoRAID as the RAID level choices. AutoRAID implements a storage hierarchy using RAID 1+0 and RAID 5 DP. AutoRAID will use free space for performance by keeping as much data as it can in RAID 1+0. When the available capacity is fully utilized in AutoRAID, a minimum of 10% of the space is used for RAID 1+0 and the rest is used for RAID 5 DP.

Tip: RAID 1+0 will provide the most robust VA performance behavior for a wide variety of workloads but at a higher cost of storage. AutoRAID can provide the same performance level as RAID 1+0 but at a lower cost of storage when the write working set fits in the RAID 1+0 space. Studies of applications have shown that the write working fits in the RAID 1+0 space in most cases of normal application usage.

performance guideline #2: disk configuration

The performance capability of an array, in some cases, is determined to a large extent by the disk configuration - the number and rotational speed of the disks that comprise the array. Generally, the more disks in an array, the greater the performance that can potentially be achieved. However, for most array designs, the array controller will eventually become the main performance limitation as the number of disks continues to increase. At that point, adding more disks to the array will not result in any additional significant increases in performance. The point where the controller limit is reached is highly dependent on the characteristics of the workload being serviced and on the array controller design. Each array controller design will have a different set of workload performance profiles as the number and rotational speed of the disks is varied. This kind of information is sometimes made available and if so, can be consulted when making decisions about the disk configuration.

Current technology disk arrays usually offer disk drives in two spindle speeds: 10,000 RPM and 15,000 RPM. As long as controller limitations haven't been reached in the configuration, the 15,000 RPM disks will usually provide higher performance than the 10,000 RPM disks. The performance benefit of 15,000 RPM disks over 10,000 RPM disks can be as high as 15-20% depending on workload.

Application note: The HP StorageWorks Virtual Array offers disks in both 10,000 RPM and 15,000 RPM speeds.

Tip: For the HP StorageWorks Virtual Array model VA7400 with 15,000 RPM disks, the controller performance point of diminishing return is reached at approximately 45 disks for an 8k random workload and 25 disks for a 64k sequential workload. For the model VA7410, it is about 55 disks for the random workload and 45 disks for the sequential workload. Additional disks will still provide greater storage capacity but no additional performance capability. However, the performance capability of additional disks may not be needed because of limited performance demand from the application.

performance guideline #3: cache size

Virtually all current technology disk arrays provide for caching of both read and write data. The cache size can usually be configured within some range of minimum and maximum size. The maximum cache size varies widely between different array designs and can range from a maximum of a few giga-bytes for some designs up to a maximum of tens or hundreds of giga-bytes for other designs. The selection of cache size represents a trade off between cost and performance. Generally, a smaller cache will have less performance potential than a larger cache. However, that is not always true. Many times the cost of additional cache is relatively small compared to the overall cost of an array so rather than trying to optimize the cache size selection, the maximum size will be used to ensure the greatest performance benefit available from cache will be obtained.

© Copyright Hewlett-Packard Company 2002

An optimized cache size selection can be based on a few different criteria. The smaller cache sizes will usually operate as "speed matching" or "load averaging" buffers. In this mode the cache will absorb temporary bursts of I/O activity (either read pre-fetch or write back) with lower response times than could have been obtained directly from the disks. After a burst (for writes) or before an anticipated burst (for read pre-fetch) the cache will perform the actual I/O operations with the disks. Thus, the disk I/O activity implied by the burst is averaged over a longer time so the disks do not limit the performance during the burst. For speed matching operation, the optimum cache size will scale with the number of disks in the array.

As cache size increases, a point may be reached where an application working set of data will fit entirely into the cache. At that point, the application may experience a significant increase in performance because I/O's are being completed within the cache service time rather than a disk service time on a more continuous basis.

Estimating the cache sizes that are needed to provide effective speed matching or to contain an application working set can be very difficult to do for specific system configurations and applications. Because of this, as mentioned earlier, the largest cache size available will often be used.

> **Application note:** The HP StorageWorks Virtual Array offers a choice of a few different cache sizes per model. The available sizes fall into the speed matching range of sizes for most applications.

performance guideline #4: cache operational parameters

There is a wide variety of selectable cache parameters available in different array designs. Examples are: caching page size, cache algorithm modifiers and thresholds to control the filling and flushing of cache. Optimizing cache parameters of a particular array design for a particular workload environment can be a complex task. Often the characteristics of the workload are not completely known and they are dynamic. The exact effect of the cache parameters may not be fully documented and understood. Adjustments to cache parameters may interact with each other. All of these variables combined make it difficult to generalize about the optimal configuration of cache parameters. Each array design in a particular workload environment will be different.

> **Applincation note:** The HP StorageWorks Virtual Array has two caching parameters that can be configured to optimize performance. These are the resiliency mode and the read pre-fetch algorithm.
>
> The VA has always had a default read pre-fetch algorithm enabled in the read cache. Starting with firmware version HP15, a more effective read pre-fetch algorithm became available for use. It is disabled by default in VA7100 and VA7400 and can be enabled with the vfp command "cfg -p on". It is enabled by default in VA7410. The new read pre-fetch algorithm can provide significant performance improvements for certain sequential read workloads and doesn't have detrimental effects on other workloads so it is recommended that it be enabled at all times.
>
> One of four resiliency modes can be selected: High Performance, Normal, Restricted Normal and Secure.
>
> Each mode offers a different characteristic in the performance vs. data integrity continuum. It is important to understand all aspects of these modes, and select the mode that best meets the system and applications requirements in both dimensions. Mode operation selection is dynamic; it is easy to (and sometimes advantageous) to switch modes during normal operation.
>
> high performance mode
> In High Performance mode, the controller will acknowledge the write completion to the host upon receiving the data in to the write cache. This is true even if the host I/O specifically indicates that the I/O should wait until the data is written to the disk (see the discussion on SCSI Force Unit Access in the Normal mode section immediately below). High performance mode also reduces the rate of write operations associated with journaling the mapping meta-data. Journaling meta-data are writes generated by the controller to be used for recovery after a catastrophic NVRAM failure. These are analogous to the journaling a database performs to allow a consistent recovery after a system failure. Although the performance impact of the meta-data journaling varies throughout the life of the array and by workload to the array, a newly formatted array may use 2-5% of its performance.
>
> > **Tip:** When using firmware versions prior to HP14 with workloads containing numerous FUA tags, High Performance mode can be valuable.
>
> However, two aspects of High Performance mode are important to understand. First, the availability impact; High Performance mode prohibits the journaling of map changes on a timed basis. Map journals will still be written to the disks, but only after the journal buffer is full. The result is that a catastrophic NVRAM failure could lose a significant amount of data – the contents of the write cache plus all the data controlled by the meta-data still in the journal buffer. The second aspect of High Performance mode is write cache management. High Performance Mode will allow the write data to reside in the write cache until forced out by a least recently used algorithm. While this is good when an application working set fits in the write cache, it effectively eliminates much of the speed matching and concurrency

© Copyright Hewlett-Packard Company 2002

effects of the write cache. The write cache remains full all the time and the controller must first free a cache page by writing some data to the disks before a pending write from the host can be completed. The cache write rate effectively becomes the disk write rate. In this case, write cache is of marginal value.

Tip: It may be useful to selectively switch between High Performance mode and Normal mode to improve performance without significantly risking data loss. On the initial load of a system, when new (allocating) writes are common, switch to High Performance mode to improve performance, then for production switch back to Normal mode for increased reliability. This can improve the initial load but yet not risk significant loss of work due to an unlikely failure scenario.

normal mode
The Normal mode will periodically (once every four seconds) check for data, or map meta-data older than four seconds then queue that data to be written to the disks. This mode has many advantages. First, a catastrophic NVRAM failure will not result in significant data loss. Only the data that was in the write cache – which was the data written in (approximately) the last four seconds of operation – will be lost. Second, older data that has not been accessed will be written to the disk and kept in the cache for a read or write hit, or as space for a new cache write miss.

Prior to firmware version HP14, Normal mode enforced the SCSI Force Unit Access (FUA) tag. FUA requires that data be stored on or retrieved from the disk media rather than cache memory before an I/O is completed to the host. This eliminates the performance benefits of cache.

Tip: When using a firmware version prior to HP14, workloads with a high percentage of FUA tags are candidates for High Performance mode so the performance benefits of cache can be obtained.

With firmware version HP14 and later, Normal mode no longer enforces the FUA tag. This makes it unnecessary to use High Performance mode with a workload containing a high percentage of FUA tags. Starting with firmware version HP14, the Restricted Normal mode became available for applications that require FUA enforcement.

The performance log can be used to determine whether a workload has a high percentage of FUA tags.

restricted normal mode
The Restricted Normal mode is the same as Normal mode except the SCSI Force Unit Access (FUA) tag is enforced. Restricted Normal mode first became available in firmware version HP14.

secure mode
In Secure mode, write cache is disabled. Writes must complete to the disk before they are completed to the host. Map changes are committed to the disk before any I/O that generated the map change is completed. Catastrophic NVRAM failures will not result in any data loss. Secure mode offers the highest level of data security and the lowest write performance.

performance guideline #5: LUN configuration
The LUN configuration of an array refers to the number, capacity and redundancy group (or disk group) placement of the storage volumes configured in an array. The term LUN is often used to refer to a storage volume in an array. LUN is a SCSI term that stands for Logical Unit Number. In some array designs, a storage volume and a LUN are identical. They are the exact same entity in the configuration. In other designs, a LUN is a separate entity and represents a host access path to a storage volume. In either case, a LUN is the means by which hosts access the storage volumes in an array.

The LUN/storage volume configuration is significant to array performance in a couple of ways. Since the LUN is the host access path, some host operating systems allocate resources needed for data access on a per LUN basis, and those resource allocations may be subject to limitations. Examples of such resources are: SCSI queue tags, data buffers and driver I/O control structures. The limited allocation for a single LUN or a small number of LUN's may not be sufficient to allow the necessary level of workload to reach the array so the array can achieve its full performance capability. In this case, a larger number of LUN's will need to be configured on the array and accessed by the host(s) so that sufficient resources are allocated by the operating system to provide the array with a workload to match its full performance capability.

The LUN configuration also determines the mapping of the workload onto the disk groups in the array. Some arrays provide for only a single or a limited number of LUN's per disk group. Others allow an arbitrary number of LUN's to be configured on a disk group. At a minimum, at least one LUN should be configured and used on each disk group in the array or there will be no way to utilize the capacity and performance resources of all the disks. It is important to spread the workload as evenly as possible across all the disks in the array so that the performance capability of each disk can equally

© Copyright Hewlett-Packard Company 2002

contribute to the overall performance of the array. This corresponds to spreading the workload across the disk groups in proportion to the number of disks in each of the groups and thus, to spreading the workload across the LUN's in a way that achieves that proportion. The more evenly the workload is spread across the disks, the less chance that a single disk can become the performance limitation.

Tip: One possible way to achieve an even balance is to create disk groups with equal numbers of disks and place equal numbers of LUN's on them. Then use volume manager software in the operating system to stripe data across all the LUN's in the array.

The number of disk groups in an array and the number of disks in a disk group can be fixed by the array design or may be configurable. At a minimum, the number of disks in a group should not be any more than can be effectively utilized by the number of LUN's that are allowed given any per LUN resource limitations imposed by the operating system.

Application note: HP StorageWorks Virtual Array model VA7100 implements a single disk group. Models VA7400 and VA7410 implement two disk groups. The number of disks in the groups can be configured within defined ranges. The number of LUN's that can be configured on a group is not limited other than the overall limitation on the number of LUN's supported by the whole array. For the VA7100 this is 128 LUN's. For VA7400 and VA7410 it is 1024 LUN's. The LUN's on each disk group span all the disks in the group. It is not necessary to have more than one LUN per group (except for any operating system based limitations) to fully utilize the performance of all the disks.

The traditional approach to database tuning by assigning the different components of the database (table space, log, etc) to different LUNs may not result in an even distribution of I/O to the LUN's. However, with the VA7400 and VA7410, spreading the workload between the two disk groups and accessing each LUN through its primary controller can accomplish this.

performance guideline #6: usage history

Traditional architecture disk arrays provide a static layout of data on the disks. There is no migration of the data over time so the performance characteristics of the array are essentially "stateless." That is, current performance has little dependence on the usage history once a configuration is established. Performance may be affected by a temporary state change in the array such as disk failure and rebuild in progress, but once that condition is cleared, the array performance characteristics return to "normal."

This is in contrast with the latest technology array architectures that incorporate advanced virtualization features. In many cases, the advanced virtualization features are enabled by the ability of the array to automate data placement choices and migrate data transparent to the host systems and applications. The current placement of the data is partially dependent on the prior usage history of the array.

Application note: In certain situations, there will be a performance advantage to account for the current data layout when using the HP StorageWorks Virtual Array.

When a VA LUN is first created, the virtual capacity for the LUN is reserved. No data is actually written to the disks. As data is written, the first write to a cluster (a contiguous 256K block on a disk) will allocate the physical space for the cluster on the disk, format the cluster and then perform the write. These are called "allocating" writes. This process takes more time than subsequent rewrites of the cluster. This has implications for performance benchmarking (see performance guideline below about performance benchmarking) and for real application workloads. However, the effect may be less pronounced for real application workloads. In real application use, the array is only filled with data once, and then the workload consists primarily of reads and rewrites of data that has already been written at least once.

Tip: Pre-filling a LUN with data on a VA prior to use may be a useful technique to avoid the allocating write "penalty" for performance benchmarks (see performance guideline below regarding benchmarking for more details) but, there are performance benefits for not pre-filling a LUN when the VA is configured for AutoRAID operation. It is recommended that LUN's not be pre-filled prior to real application use.

Two methods can be used to determine if allocating writes are affecting performance. First, Command View SDM will show the allocated capacity for each LUN. If this capacity is growing over time, then there is some performance impact due to allocating writes. Second, a metric in the performance log (armperf command) maintains a count of allocating writes. If this metric continues to have counts in it for each performance log sampling interval, then there is some performance impact due to allocating writes.

Tip: When a VA is used in AutoRAID mode and has been in service for a while, most of the data will reside in RAID 5 DP (firmware version HP14 and later). If, when in this state, there is a need to restore or reload data

© Copyright Hewlett-Packard Company 2002

for a whole LUN, there will be a performance benefit for the reload to delete and recreate the LUN (or format the LUN) prior to the reload. Deleting or formatting the LUN erases the old data in the LUN and frees up the space that was used to store the data. This allows the array to optimize the write processes used during the reload. If the old data is not removed, it is likely that much of the reload writing will take place using the relatively inefficient RAID 5 DP read/modify/write process.

performance guideline #7: I/O routing

While not specifically a configuration of the array itself, I/O routing refers to the configuration of connections and SAN infrastructure between host systems and the array and to the software pathing configuration of the host software. The design of some arrays results in performance that is independent of I/O routing. That is, the performance is the same no matter what path the I/O takes. In such systems, path load balancing software like HP AutoPath can be beneficial to distribute the data traffic and aggregate the performance of multiple physical links between hosts and array.

Dual controller array designs like the HP StorageWorks Virtual Array usually fall into one of two categories in regards to pathing: active-active or active-passive. In an active-passive design, only one of the two controllers is able to respond at any given time to requests for access to particular LUN's. In this case, the I/O's can only be routed to that controller. Path load balancing software is not beneficial and does not function correctly on an active-passive array.

Application note: The HP StorageWorks Virtual Array implements the active-active model. Access to any LUN can be routed to either controller but a noticeable performance improvement is obtained when an I/O is routed to the primary controller for the LUN. When a LUN on the Virtual Array is created, it is allocated from a particular redundancy group (or disk group). Disk groups are associated with one of the two controllers as a primary controller for the disk group. Some VA models have only one disk group so all I/O's should be routed through that disk groups primary controller. Other VA models have two disk groups so the host pathing should be configured to route I/O's to the primary controller for the disk group containing that LUN.

The Command View SDM 'armtopology' command will display the primary controller information for each LUN.

Tip: When configuring pvlinks for HP-UX connected to the Virtual Array, the primary path for each LUN should be to the primary controller, and the fail-over path to the secondary controller. Similarly, AutoPath should be configured the same. It is recommended that path load balancing not be used. Path load balancing is not required to achieve optimum performance on the Virtual Array.

workload

The second category of performance guidelines relates to the nature of the workload presented to the array. The workload is often largely determined by the application with little room for any kind of adjustment by the system administrator or end user - other than to change their system usage patterns. In some cases, the workload is heavily influenced by middleware and operating system software layers between the application and the array. It may be possible to alter the workload characteristics by configuration of these intervening software layers. The workload guidelines may be as useful to application and system software developers as they are to system administrators and end users because large-scale changes in storage workload characteristics may require changes in the software design.

performance guideline #8: concurrency

An array consists of many disks. To realize the full performance potential of an array, all of the disks must be kept busy with useful work all of the time. The disks must operate in parallel. The buffering and speed matching behavior of the cache will achieve this effect to a certain degree. For some workloads, the cache provides sufficient concurrency to achieve parallel disk operation and realize full performance from the array. For other workloads, caching alone is not sufficient. For these workloads, there must also be concurrency inherent in the application workload itself.

Ultimately, concurrency in an application workload results from multiple users performing independent tasks. Some applications may be of a nature such that there is no natural concurrency in the workload. This would typically be a single user application performing some sort of serialized sequence of activity. In this case, there is no ability to more fully utilize the performance of an array.

A multi-user application (a server for example) will usually be capable of generating concurrency in the workload. It is important that the concurrency in the application workload actually be presented to the array for full array performance to be achieved. Concurrency to the array may be limited even though a sufficient level of concurrency exists in the application. The application software may be configurable in some way to control the number of concurrent software processes. If there are fewer software processes than there are number of users, user requests will be queued waiting for an available process. In this case, increasing the number of software processes will allow more concurrency to propagate beyond the application software.

© Copyright Hewlett-Packard Company 2002

The operating system or device driver is another place where concurrency might be artificially limited. The default configuration of the driver path to an array LUN may restrict the amount of host system resource (examples: queue tags, I/O buffers, I/O control structures) available for use by I/O to that LUN. These kinds of restrictions are usually in place to prevent a single storage device or LUN from consuming all the I/O resources available in an instance of the operating system. In that case, the options are to reconfigure the driver path to allow more resource or, if that's not possible, use more LUN's to access the disk group. If more LUN's are used, then some means is needed to evenly distribute the workload across those LUN's so the resources reserved for each LUN are evenly utilized (see performance guideline above about LUN configuration).

The level of concurrency necessary to achieve full performance from an array is proportional to the number of disks in the array. For a small block random workload, there should be at least one I/O active in the array (on average) for each disk in the array. Three I/O's per disk (one active and two queued per disk) will allow seek sorting algorithms in the array controllers or the disk controllers to do some optimization.

For sequential workloads, it is more difficult to estimate the necessary minimum number of active I/O's in the array to optimize performance although it is still roughly proportional to the number of disks. Read ahead and write back caching can be very effective with a sequential workload making it unnecessary to have a high level of concurrency in the workload presented to the array. As a rough estimate, there should be a few (three or four) sequential I/O's active per disk group in the array. Published performance data for an array may be helpful to determine if there is any additional benefit to higher concurrency for sequential workloads.

There is one note of caution. While an array requires a high level of workload concurrency (demand) for best performance, care must be taken to avoid over demanding the array. If the array is provided with a higher number of active I/O's than it can effectively service, many of the I/O's will remain inactive for a relatively long time in an incoming I/O queue in the array. This will be reflected as higher response time for each I/O with no significant increase in I/O throughput. When this point is reached, the array is saturated. More concurrency will only result in longer response times and not higher performance. As the demand increases even more, the response times can become so long that host driver timeouts occur.

performance guideline #9: synchronous I/O

Synchronous I/O refers to I/O's that request synchronization of the cache before they are completed. Applications (databases in particular) may use synchronous I/O to implement certain I/O ordering constraints that are necessary for data recovery at the application level in the event of the loss of write cache. Synchronous I/O is implemented at a low level by the SCSI Force Unit Access (FUA) command tag.

> **Application Note:** The HP StorageWorks Virtual Array implements resiliency modes in which FUA is enforced and modes in which it is ignored (see performance guideline above about cache operating parameters). The choice is a trade-off of performance and data reliability/availability. The trade-off should be made carefully.
>
> **Tip:** Unix file systems typically implement an "OSYNC" file access attribute. OSYNC forces file I/O to be synchronous (tagged with FUA). For best performance, OSYNC should be avoided if possible. Regardless, for VA, the response to OSYNC (and FUA) is determined by the resiliency mode.

performance guideline #10: I/O block size

Data blocking factors and alignment boundaries exist in the design of disk arrays that tend to make the performance of I/O's more or less efficient depending on these characteristics of the workload. These factors usually exist in the implementation of the cache (cache page size) and in the implementation of the striping layout of the data on the disks (striping segment size). Some array implementations allow these parameters to be selected when the array is configured. Others implement fixed sizes.

> **Application note:** The HP StorageWorks Virtual Array does not allow the selection of cache page size and striping segment size. The VA cache page size is 64K and the striping segment size is 256K.

Generally, best performance is achieved by matching the I/O block size and alignment to both the cache page and the striping segment if possible. This allows each I/O to be serviced by a separate cache page and by a separate disk. Bandwidth oriented workloads will benefit more if the I/O block size is a significant portion of or even a whole disk stripe. However, this may not always be possible. Implementations of RAID level 3 tend to use a relatively small striping segment size so performance of bandwidth oriented workloads can be optimized this way.

An application has an I/O block size that is naturally most efficient for the application. However, this size may not be the most efficient for the array. It may or may not be possible to adjust the application to match the array. Likewise, it may or may not be possible to adjust the array configuration to match the application. If it is possible to adjust the I/O block size of the application, it will be a compromise between that which is naturally best for the application and that which is best for

© Copyright Hewlett-Packard Company 2002

the array. Making this adjustment (if possible) to achieve an optimum result can be a complex task. Because of this complexity, often times there is no attempt made to adjust I/O block size to match the array configuration.

Some file systems provide parameters that can be configured to adjust the I/O block size that results from file I/O. These parameters include the file system buffer cache page size and the file system data block and data fragment allocation sizes. Many databases have similar configuration parameters. Configuring these parameters so that I/O block sizes match array block sizes may not optimize performance at the application level. If the application has a natural I/O block size that is different, changing the configuration of the file system or database may result in useless transfer of extra data between host and array that is not needed by the application. In this case, the performance may appear to be optimized at the array level but it is not optimized at the application level.

Tip: For the HP StorageWorks Virtual Array, it has been found through benchmarking that on HP-UX using the VxFS file system, single stream sequential file I/O performance can be optimized by configuring the file system to generate 64K I/O's rather than the default 8K I/O's. This can be achieved using this command:

vxtunefs -o max_buf_data_size=65536 /mnt/newfs_mount_point

The following set of VxFS configuration parameters are optimized for sequential file I/O on a VA7400 with 45 disks:

max_buf_data_size = 65536 (set the maximum transfer size)
read_unit_io=65536 (set the hint for Vxfs on read size)
read_pref_io=65536 (set another hint for Vxfs on read size)
write_unit_io=65536 (set the hint for Vxfs for write size)
write_pref_io=65536 (set another hint for Vxfs write size)
read_nstream=45 (when doing read-ahead, do enough to parallelize across the 45 spindles in the VA7400)
write_nstream=45 (when doing write-behind, do enough to parallelize across the 45 spindles in the VA7400)
max_diskq=104857600 (tell Vxfs not to throttle the write behind quantity for any given process)

The following set of VxFS configuration parameters are optimized for small, random file I/O on a VA7400 with 45 disks:

max_buf_data_size = 8192 (set the maximum transfer size)
read_unit_io=8192 (set the hint for Vxfs on read size)
read_pref_io=8192 (set another hint for Vxfs on read size)
write_unit_io=8192 (set the hint for Vxfs for write size)
write_pref_io=8192 (set another hint for Vxfs write size)
read_nstream=1 (shutdown read-ahead because the workload is random)
write_nstream=45 (when doing write-behind do enough to parallelize across the 45 spindles in the VA7400)
max_diskq=104857600 (tell Vxfs not to throttle the write behind quantity for any given process, if not set then it can throttle NFS daemons)

performance guideline #11: volume manager striping

Some operating systems include volume manager software to assist in the management of storage volumes. The volume manager software will often support a striping function to allow data from the "logical" volumes visible at the application level to be evenly distributed across the "physical" volumes accessible by the host. The physical volumes comprise any storage device accessible by the host such as individual disk drives and LUN's presented by an array.

Striping across volumes is intended to distribute the workload as evenly as possible across all the storage devices so that no one device will become a performance limitation. In a storage configuration with mixed devices, this may not be appropriate since some devices may have higher or lower performance capability than others.

Application note: The HP StorageWorks Virtual Array implements striping within a redundancy group (or disk group). All LUN's allocated on a disk group will stripe all the disks in that group. Therefore, there is no need to use volume manager striping to distribute the workload across disks within a disk group. However, there may be a need to stripe across multiple LUN's within a single disk group to avoid per LUN I/O path resource limitations that may be implemented by the operating system (see performance guideline above about LUN configuration).

It is beneficial to use volume manager striping across VA LUN's contained on different disk groups or different arrays. The purpose is to distribute the workload across disk groups and across arrays, so that no one disk group or array becomes the main performance limitation. This must be planned carefully though. Disk groups containing different numbers of disks will have different performance characteristics. If that is the case, an even distribution of the workload may not be the most optimal configuration.

© Copyright Hewlett-Packard Company 2002

performance guideline #12: performance benchmarking
The section below in the glossary about performance benchmarks gives general background and guidelines about performance benchmarking for arrays. Some specific guidelines about benchmarking should be noted.

> **Tip:** As described above in the performance guideline about usage history, the first time data in a LUN on the HP StorageWorks Virtual Array is written after it has been formatted, the write process will be somewhat slower than for subsequent rewrites due to the cluster allocation process. It may be possible to achieve a better benchmark result by "conditioning" the VA prior to running a benchmark. The purpose of conditioning is to eliminate the first write performance "penalty" from the benchmark results. This may be reasonable since in real application use, the data is usually only loaded onto the array one time. The VA can be conditioned either by pre-filling the LUN's being used by the benchmark or by repeating the benchmark (assuming that the benchmark will use the same area of the array in each run).

It is sometimes tempting to use simple data copy programs like the Unix "dd" command or Windows Explorer to do a quick and easy performance benchmark. However, it may not always be informative. Typically, these programs do not generate sufficient workload concurrency to demonstrate the full performance capability of an array. However, multiple, simultaneous copies can create a sufficient demand. Scripting the execution of many concurrent dd copy operations can demonstrate the full performance capability of an array.

The system cache and the array cache can give misleading benchmark results if they are not accounted for in the benchmarking process. Small data sets may present a workload that fits entirely in cache (very small working set). If that is the case, the measured performance will be that of the system cache or the array cache and not that of disk access in the array. Real application workloads very rarely fit entirely in cache so the result will not be representative of the array performance capability that is available to real applications. dd to the raw device file will avoid the system cache but not the array cache. dd to the block device file or to the file system will demonstrate the system buffer cache performance.

glossary of performance terms and concepts

application
The particular task or tasks being accomplished by the computer system. The application is often supported by one or more application software programs that implement many of the functions of the application. However, the application is not the software alone. It is the interactive usage of the application software to accomplish business objectives.

workload/demand
The amount and type of activity in a computer system requested by the application. The workload can be measured at different points in the system and has different characteristics and units of measure based on where it is measured.

The workload of a system can be described at a high level by the class of application and the number of users. For example: a mail server with 400 mailboxes. The workload is specifically the rate at which individual requests for activity (tasks, jobs, transactions) arrive at a particular point in the system. This is to be distinguished from the rate at which those requests are being completed. Demand is synonymous with workload but carries an emphasis that it is the rate of requested activity being measured, not the actual throughput being achieved.

stream/process
A serialized sequence of workload activity created by a background software process or by an interactive user session. A new job in the sequence will not be requested until the previous one is completed. This term is most commonly applied in a storage context to the workload of a storage device when that workload is sequential but use of the term is not limited to that concept.

open loop workload/demand
A class of workload in which the creation of new jobs is not dependent on the completion of previously requested jobs. The rate of job arrival has no relationship to the rate of job completion. Open loop workloads can occur when the potential number of system users is very large such as in a network server application. In this case, it is the unbounded arrival of users that can create the open loop characteristic rather than the workload created by each user. If the open loop workload at a particular point in a system exceeds the throughput for a sustained period of time, the number of job requests waiting at that point in the system will accumulate without bound and some kind of interruption in system operation (such as a timeout) may occur.

closed loop workload/demand
A class of workload in which the creation of new jobs is dependent on the completion of previously requested jobs. This type of workload will occur when there is a limited number of processes each of which is creating a serialized sequence of

© Copyright Hewlett-Packard Company 2002

activity. The processes operate such that a new job in the sequence will not be requested until the previous one is completed. The workload is limited to be the number of processes that are concurrently in operation. A process corresponds to either a background software process or an interactive user session.

System operation in response to a closed loop workload is self-limiting. That is why it is called "closed loop." The demand will be throttled by the lowest point of throughput in the system. All components of the system will have the same measured throughput because the demand is being limited to be the lowest throughput of all the components. Therefore, it is difficult to discover by measurement which component in the system is limiting throughput because all components in the system will have the same measured throughput. It may appear that performance of a closed loop workload is low because the application is not creating a high demand but some other component of the system may actually be the limiting factor.

workload intensity
The number of processes in a closed loop workload.

performance benchmark
A software program and testing procedure that creates a simulated workload. The goal of a performance benchmark is to provide a common basis for performance comparison between different system configurations or products. There is a wide variety of performance benchmarks and they create a wide variety of workloads. Some performance benchmarks attempt to simulate as faithfully as possible the workloads created by particular classes of applications. Others focus on demonstrating the maximum performance characteristics of particular system components such as the maximum data throughput of a storage device. The workload created by a performance benchmark can approximate the workload created by a real application. In some cases, it is a poor approximation. This can lead to false conclusions about expected performance of the real application. The uncertainty can sometimes be addressed by using the real application in a non-production mode as the benchmark. Even then, the nature of the real application may change over time so that the original benchmark results no longer represent actual performance expectations. Performance benchmarks are an important tool in system performance analysis and need to be used carefully.

response time
The real elapsed time from the arrival to the completion of a particular job at a particular point in the system. The response time includes queue waiting time that may occur because the resources needed to complete the job are unavailable at job arrival plus the actual time required by the hardware to perform the task.

service time
The real elapsed time required by the hardware to perform a particular task. The service time is the response time less the queue waiting time. Some applications require storage system response time to be near the service time for best application performance.

throughput
The overall rate at which work is being completed at a particular point in the system. Units of throughput measurement that are typically used in a storage context are MB/sec (mega-bytes per second), IO/sec or IOPs (input/output operations per second) and TPS (transactions per second).

MB/sec
Mega-bytes per second. A unit of measurement that is often applied to a storage device to measure demand or throughput. It measures the rate at which data is transmitted to and from the storage device. The measurement is often qualified by some characteristics of the workload such as reads or writes or a combination of reads and writes of a certain length. MB/sec throughput is usually measured using a large I/O request size (64k bytes or above) because data transfer bandwidth tends to dominate the performance characteristics of a storage device for this type of workload. However, MB/sec throughput can be measured and can have meaning for any type of workload.

IO/sec (IOPs)
Input/output operations per second. A unit of measurement that is often applied to a storage device to measure demand or throughput. It measures the rate at which read and write requests are arriving at or being completed by the storage device. The measurement is often qualified by some characteristics of the workload such as reads or writes or a combination of reads and writes of a certain length. IO/sec throughput is usually measured using a small I/O request size (2k to 16k) because I/O command processing time tends to dominate the performance characteristics of a storage device for this type of workload. However, IO/sec throughput can be measured and can have meaning for any type of workload.

TPS
Transactions per second. A unit of measurement that is usually applied to a database management system to measure demand or throughput. It measures the rate at which database transaction requests are arriving at or being completed by

© Copyright Hewlett-Packard Company 2002

the database management system. This is a relevant measurement for storage because storage is often an important factor in the performance of database management systems.

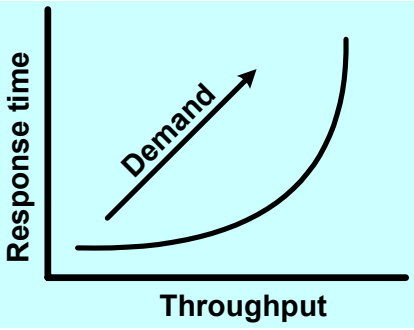
sequential
A particular type of storage workload in which the addresses and lengths accessed by a sequence of requests specifies in combination a contiguous range of data. This is an important type of workload because many storage devices implement optimizations to accelerate performance of sequential workloads. Data access software such as file systems and databases often include algorithms that attempt to generate sequential workloads to take advantage of the storage device optimizations. It is also representative of the workload created by certain kinds of applications.

multi-stream sequential
This is a workload which comprises multiple, concurrent sequential workloads. Typically, each sequential stream in the workload is created by a separate process. It can be difficult to recognize the individual sequential streams when they are interleaved at a single measurement point. Without careful observation, a multi-stream sequential workload can appear to be a random workload.

random
A particular type of storage workload in which the addresses specified by a sequence of requests appears to be randomly distributed over a particular range of addresses. This is an important type of workload because it tends to defeat optimizations implemented in storage devices to accelerate performance in response to a sequential workload. It can be representative of the workload created by multiple, independent users of a system. Even if the workload created by each individual user is sequential, the combination of all the users can appear to be random at a storage device.

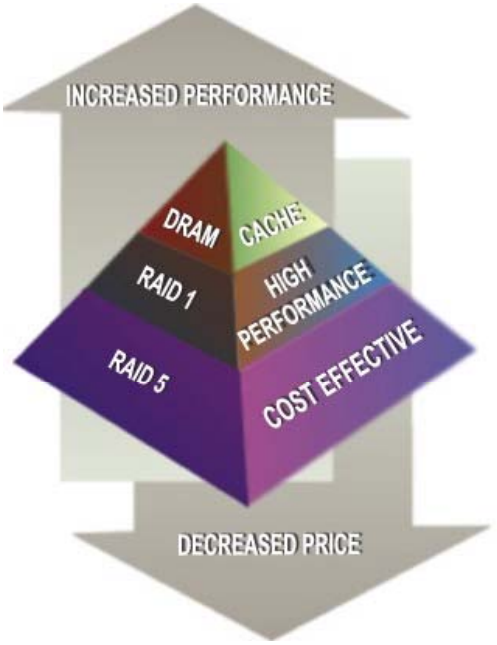
utilization
The degree to which a particular component or resource in a system is used. It is measured as an average over a certain period of time and is expressed as a percentage of component utilization over that time period. Zero percent means no use over the measurement period. 100% means fully busy for the entire measurement period. Near 100% utilization indicates a performance constraint in the system. High utilization and high queue depth both are indications that a component or resource is near saturation.

saturation
A condition in which the demand on a particular component or resource in a system is equal to or greater than the throughput of that component or resource. The condition is characterized by asymptotically increasing response time as the demand increases with no further significant increase in throughput. The increase in response time is due to queue waiting time. The saturated component or resource will have near 100% utilization and will have a relatively deep queue.



© Copyright Hewlett-Packard Company 2002

storage hierarchy
An architectural arrangement of storage methods such that the faster, higher cost methods are closer to the application and the slower, lower cost methods are farther away. The storage methods are arranged in rank according to speed and cost. The goal of a storage hierarchy is to provide storage that appears to an application to have the speed of the more costly methods and the cost of the less costly methods. HP's StorageWorks Virtual Array products include AutoRAID technology which implements a storage hierarchy using DRAM cache memory, RAID 1+0 and RAID 5DP.



working set
A concept that refers to address locality in the workload. The working set comprises a set of data addresses that have been accessed over some measurement period of time. The working set size is usually compared to the storage capacities at various levels in a storage hierarchy. If the working set fits into one of the levels, it will stay resident at that level and the performance of the storage method at that level will be realized over the measurement period of the working set.

thrashing
A condition in which the working set does not fit into a particular level of a storage hierarchy. The result is that the data contents at that level are constantly changing and the performance of that level is not realized.

for more information

For additional information on HP StorageWorks Virtual Arrays and other HP storage products and solutions, please call your local HP sales representative or visit the HP storage Web site at http://www.hp.com/go/storage.

All brand names are trademarks of their respective owners.
Technical information in this document is subject to change without notice.
© Copyright Hewlett-Packard Company 2002
08/02

© Copyright Hewlett-Packard Company 2002