Tuning Your I/O Performance On MPE/iX Systems !

Paper number: 1028
Paul Wang
SolutionSoft Systems, Inc.
2350 Mission College Blvd. Suite 715
Santa Clara, CA 95054
paulwang@netcom.com
408.988.7378

Abstract

System performance tuning maximizes the return on your hardware and software investments. A key component of that tuning is I/O optimization. Many low throughput and long response time problems are due to I/O bottlenecks.

The CPU speed doubles roughly every 18 months. On the other hand, disk speed only improves by 30 percent or so in the same time frame due to its mechanical nature. As the gap widens between CPU and disk speed, I/O tuning becomes increasingly more important. While the same is true for low and middle level systems, the problem is particularly daunting for high-end systems where multiple CPUs are furiously pumping data at the potential I/O bottlenecks.

This paper gives an overview on disk I/O and MPE/iX I/O subsystem. It also provides tips and guidelines on do's and don'ts of preventing I/O bottlenecks and maximizing I/O throughputs.

MPE/iX I/O Subsystem Overview

To better understand how to tune your I/O performance on MPE/iX systems, it is very helpful to first understand the I/O subsystem. There are three levels of the I/O subsystem: the physical hardware level, the I/O disk manager level, and the file system level. Let’s examine them one by one.

Physical Hardware Level

To perform an I/O to a disk, the total time it takes is the sum of seek time, rotation delay time, and data transfer time. Seek time is for the disk read/write head to move the target track. Once the head is on the right track, it then must wait until the target sectors spinning underneath the head, which is the rotation delay time. Finally data are transferred to or from the disk, which is the data transfer time.

Seek time is the most time consuming operation. However, if the head is already on the target track, then no seek is necessary and the seek time will be zero. This is key for disk I/O performance. Please note that the impact of seek distance is negligible on modern disks. They have a very small disk surface to begin with (3.5 and 5.25 inches) and they are getting smaller every day. Furthermore, the seek time is dominated by the head to "ramp up" and "ramp down" (command processing, acceleration and deceleration, settling time and servo reacquisition), the actual traveling time for the

head is negligible. Putting critical data in the middle of the disks are once popular techniques in the 70's. However, this particular tuning technique is out-of-dated and has no effects on system performance.

The faster the disk spins, the smaller the rotation delay time is. The average rotation delay time after a seek is half of the disk rotation time. Another key for disk I/O performance is to eliminate rotation delay. Since there are gaps between sectors and disk can prepare next I/O while it is processing the current I/O, issuing 'sequential' I/Os on contiguous sectors (on the same track) requires no rotation delays.

I/O Subsystem Level

The basic I/O unit is a page, or 4 Kbytes, or 16 sectors. The size of any I/O must be in a multiple of pages. Currently an I/O is between one to sixteen pages. In addition, multiple I/Os can be linked together so that there is only one interrupt to the OS when all the linked I/Os are completed.

MPE/iX I/O subsystem is priority based. Each disk is associated with its own disk manager, which handle all I/Os against that disk. Each disk manager has 32 queues. Depending on the priority of the I/O request, it is queued to the corresponding queue. Requests within the same queue are serviced first-in-first-out (FIFO). The disk manager always services the first request on the first non-empty queue. Finally, disk managers are independent of each other. Concurrent I/Os can be in-progress on multiple disks the same time.

In this way, urgent I/Os are serviced immediately. On a heavily loaded system, I/Os from more important processes take precedent of lesser ones. The down side is possible starvation for lowest priority I/Os. To this end, the disk manager will boost an I/O request priority (by moving it to a higher queue) if the blocked issuing process's priority is boosted.

File System Level

File system has very sophisticated algorithms to issue I/Os. Let's examine how read, write, and checkpoint I/Os are handled.

For a random read (such as FREADDIR), file system issues an I/O that covers the record of interest, which is most likely a one page I/O. For a sequential read (such as FREAD), file system prefetches data ahead and recycles data behind. By bringing in "new" data ahead of time, chances are they will be available (already in memory) once it is needed. By making "old" data overlay candidates for the memory manager, more memory will be available for other processes. The file system prefetches eight pages ahead initially. If it detects the process is consuming faster than it prefetches, then file system increases the prefetch amount by one page, up to the maximum of sixteen pages. On the other hand, if it detects the prefetched pages disappear (by memory pressure) before it is used, file system decreases the prefetch amount by one page, down to the minimum of one page. Finally, if a process is using random intrinsic (such as FREADDIR) access data sequentially, file system will detect that after 4 tries and switch to sequential mode. Once the sequential access pattern is violated, file system immediately switch back to random mode.

For a write, file system encourages concurrent I/Os by "post ranges". It first divides the write into ranges (up to 32 ranges), where a range is a virtual address range that covers a single disk. It then issues I/Os concurrently on all the ranges. The number of I/Os it issues for each range depends on the number of outstanding I/Os against that disk at the time to prevent flooding. Those I/Os are linked and when they are completed, file system will then issue the next bunch. If there are more than 32 ranges, file system will pickup the rest of the ranges when the current 32 ranges are all done.

At checkpoint time, all updated Transaction Manager (XM) data, such as IMAGE/SQL, KSAM, User Logging files, associated with the log are posted to disks. This is done periodically so that XM can safely reuse the log. Checkpoint is a massive I/O bound operation. It is done 4 files at the time until all dirty XM files are posted to disks.

As far as I/O priority is concerned, it usually inherits the process priority. An exception is when closing a file, which produces the lowest priority. Finally the checkpoint I/O priority is at CQ base. As a result, it should have unnoticeable I/O bandwidth impact for system processes (AQ and BQ) and interactive users (CQ). Batch processes (DQ and EQ) which are I/O bound might see slower response time during checkpoint time.

Tips on MPE/iX I/O Tuning

The CPU speed doubles roughly every 18 months. On the other hand, disk speed only improves by 30 percent or so in the same time-frame due to its mechanical nature. As the gap widens between CPU and disk speed, I/O tuning becomes increasingly more important. While the same is true for low and middle level systems, the problem is particularly daunting for high-end systems where multiple CPUs are furiously pumping data at the potential I/O bottlenecks.

To access same amount of data and faster, we can either reducing the number of I/Os (i.e. increase the size of average I/O) and/or balancing concurrent I/Os across multiple disks. The placement and size of file extents are the most dominant factors for I/O performance tuning. The following are some tips to increase I/O performance.

Combine Extents for Files

Generally speaking, a large extent is better than many small extents with the same accumulated size. One large extent uses fewer system resources. The system also performs better with a large extent during virtual address-to-disk address translation (for example, servicing a page fault), since fewer extents need to be searched. This is noticeable when servicing a page fault causes still more page faults on the corresponding system translating structures. Furthermore, one large extent discourages disk fragmentation at extent deallocation time.

Most importantly, a large extent increases I/O effectiveness by encouraging bigger and fewer I/Os. For example, considering the cases of reading from a 64 Kbytes file with either one 64 Kbytes extent or sixteen 4 Kbytes extents. The former would take 16 I/Os and the latter just 1! Although the amount of data transferred is the same (64 Kbytes), the one extent case is 16 times faster! The reason is due to disk seek time and rotation delay time for each I/O, which are the major components of disk I/O time. Of course, in addition to the response time savings, don't forget about CPU time savings! It takes an order of magnitude of the number of CPU instructions to generate sixteen I/Os vs. one I/O.

All small files, which are less than 512 Kbytes, should be combined. All physically contiguous files (extents are contiguous on disk) should also be combined. This happens a lot due to extent faults and extent placement always going for the same, most empty disk. All program files as well as data files with dominant sequential access patterns are also excellent candidates to be combined. In general, a combined file is better as long as I/Os against the file would not cause a bottleneck. Please note that in the sequential access case, the file system performs automatic "prefetch ahead" and "post behind" for users so I/O is not a problem.

Spread Extents for Files

On the other hand, a very large extent (tens of megabytes) with an intensive multi-user random access pattern can be a performance problem. Page faults or prefetches against the same extent are all targeted for a single disk. An I/O cannot be serviced until all higher priority I/Os and previous same-priority I/Os are completed. The problem is worse when a checkpoint (posting all dirty data) is in progress. Hundreds of checkpoint I/Os are generated continuously and they may also be competing for the same disk.

In this case, it is more advantageous to spread the space evenly across multiple disks. Since multiple I/Os can be serviced by multiple disks concurrently, the throughput and response time are greatly improved. For example, the spreading of extents for the Debit/Credit benchmark's "Account" dataset across seven disks (except the master volume) on a HP 3000 Series 960 with 128 megabytes of memory boosted the throughput by 50 percent compared to the non-spreading case. In fact, the response time criterion (90 percent of transactions are completed within two seconds) cannot be met without spreading the extents!

Please note that in the previous example, avoiding the master volume allows us to separate the transaction manager's log I/Os from the database I/Os. This is especially important for ldev 1, since I/Os against system libraries and data structures also consume I/O bandwidth. In general, if the number of volumes within a volume set is greater than five, it is best to spread across all member volumes without the master volume; otherwise, do spread across all volumes.

When spreading a file, try not to spread one extent per disk. For example, a 50% capacity dataset spreading this way may only utilize half of the disks! Even though there are extents on the other half of the disks, there are no data in them yet!

Users are advised to always spread their performance-critical files (with intensive multi-user random access pattern). Since hashing and B-tree accesses are by nature random (i.e. 99% of on-line access to KSAM, IMAGE/SQL or ALLBASE/SQL), performance-critical databases are almost always excellent candidates to be spread.

Defragment Disks Regularly

Since extents are of different sizes, as extents are allocated and deallocated, the average size of free space tends to become smaller and very tiny fragments of free space may be created. This is called disk fragmentation. Disk fragmentation encourages small extents, which in tern promotes more I/Os and less efficient I/Os (small I/Os). In addition, disk fragmentation wasted disk space. Any free space less than 64 Kbytes are wasted and free space between 64 and 512 Kbytes are not available to allocation request bigger than 16 Mbytes.

The solution is to proactively defragment the disks. Rather than waiting for the fragmentation to accumulate and cause problems, we always keep the disks clean. It is recommended to defragment all disks at least once each week. This can be easily automated into daily or weekly batch processing.

Checkpoint Considerations

Faster checkpointing is very desirable. Faster checkpointing discourages checkpoint collision, where the current checkpoint can not proceed until the previous checkpoint is completed. Checkpoint collision can be a very serious performance problem where users pause for a long time periodically. It usually occurs with big memory system (more dirty pages to post) coupling with unbalanced and fragmented disks (takes longer to checkpoint). Secondly faster checkpointing has less impact on batch jobs and CQ base users (due to CQ base priority checkpoint I/Os). Finally, faster checkpointing

encourages faster recovery time after system interruption. Why? If the system crashes while no checkpoint is in progress, then recovery manager only needs to recover the current log-half up to the crash point. On the other hand, if the system crashes while checkpoint is in progress, then recovery manager must also recover the entire previous log-half in addition to the current log-half. The latter case on average triples the recovery time of the former case (1.5 log-half vs. 0.5 log-half)!

Checkpointing is an I/O bound operation, where all dirty pages associated with the log-half are posted to disks (four files at a time). The first three tips are critical to speedup checkpoint. To reduce the number of I/Os or to increase the average size of I/Os, we combine small files and periodically defragment the disks. To balance concurrent I/O for all disks, we spread medium to large files.

Finally for serious checkpoint collision problem, consider partitioning the applications on the volume set into multiple volume sets. Now each volume set will take longer to checkpoint and there are less dirty data to be checkpointed. In addition, multiple user volume sets increase application resiliency. If a disk crashes, only one volume set will be affected and it takes less downtime to rebuild the volume set (less data to be restored).

Use Store XL and COPY Command to Copy Files

FCOPY, DSCOPY, and CM Store copy data by sequentially reading and moving data in blocks. It has many undesirable effects: A fill-disk operation is performed each time an extent is allocated for the target file. Since the copied data will override the fill pattern, fill-disk operations are unnecessary. It degrades response time and doubles the number of I/Os for the target file (fill-disk plus posted data). For sparse files, "non-existent" data (fill pattern) in extent gaps will be copied and cause target files to be fully allocated, wasting disk space. Worse yet, source files will also no longer be sparse files after copying, and will be fully allocated due to reads! Finally, the algorithm tends to generate many extents for target files. These extents are allocated dynamically as data is copied.

There are many ways to copy files. In particular, Store XL and the COPY command generate desirable extent distributions since they use both the contiguous block and automatic extent spreading algorithms. On the other hand, FCOPY, DSCOPY, and CM Store generate many smaller extents. They not only foster disk fragmentation but also perform more slowly. This is especially noticeable with sparse files.

Of course, those subsystems have unique features that are not provided by Store XL and the COPY command. Users should continue to use those subsystems when it is necessary, but using Store XL and the COPY command will be both faster and better for your disk usage.

Avoid Double I/O

Initial allocation is always more efficient than allocating dynamically later and avoids costly fill-disk operations. If the disk space requirements are known in advance, it is a good practice to initially allocate the space.

If the disk space is not pre-allocated, then it is advisable first to write data beyond the file EOF, and to extend the EOF (FCONTROL or FCLOSE) only after the writing is done. This way, costly fill-disk operations are eliminated. This is particularly valuable with initial file loading.

Disk Array Considerations

With the advent of disk arrays (C2252, C2254, C2258), disk capacity grows dramatically. C2252's capacity is 2.7 Gbytes, C2254's capacity is 5.4 Gbytes, and C2258's capacity is 8 Gbytes.

To balance the I/O subsystem, it is recommended that users not mix disks of greatly differing capacities within a volume set. The reason is to avoid disk space allocation dominated by big disks, which could become an I/O bottleneck and would foster unbalanced I/O by nature.

Consolidating a volume set from many disks into few C2254s or C2258s deserves special attention. Even though they have much better seek time and transfer time than "conventional" disks, the number of disks within a volume set could decrease six to twelvefold! As a result, if the environment requires heavy random-access I/Os, a few C2254s or C2258s might perform more poorly. For example, consolidating 12 C2202s with I/O load averaging 10 I/Os per second per disk into 2 C2254s generates unreasonable I/O demand of 60 I/Os per second per disk! This demand is obviously beyond the C2254's capacity and the system response time would suffer as a result. In such a case, consolidating into 4 C2252s might be a better option.

Add New Disks

Adding new disks into a volume set may easily create an I/O bottleneck. This is especially true when the volume set is full or close to capacity.

MPE/iX extent placement algorithm places new extents on the disk with most free space. When a new disk is added, it will be THE place where new extents are allocated. Due to data locality, those new data tend to be the most active data. Worse yet, if any performance critical databases are restored or reorganized, then they will be allocated solely onto the new disk! Clearly the I/O demand for the new disk may easily exceed its bandwidth. As a result, the system throughput and response time may suffer. It is recommended to re-balance the disk allocation immediately after adding new disks into a volume set. This will not only prevent those new disks from becoming an I/O bottleneck, but also improve system performance by increasing I/O concurrency across all disks in the volume set.

It is of critical importance to populate the volume set to rebalance the disk allocation immediately after adding a new disk. This will not only prevent the new disk from becoming an I/O bottleneck, but also improve system performance by increasing I/O concurrency across all disks in the volume set.

Small System Volume Set

Since ldev 1 is treated specially for extent placement algorithm such that it is always placed at the end of the sorted eligible list, small system volume sets with only two or three disks require special attention.

For a small system volume set, this tends to fill up the member volumes and leave lots of free space on the master volume, ldev 1. As a result, extents for a performance critical file tends to be all on the member volumes or all on ldev 1. This may become an I/O bottleneck. In addition, a full or very fragmented member volume may cause system errors described in the severe disk fragmentation case mentioned previously.

Users are advised to re-balance the disk allocations once a disk is, or close to, full. This is especially true for a small system volume set, which is more likely to generate full member volumes. Please note that "small" here means the number of disks rather than the disk capacity. In fact, a member volume could be a disk array, which could be as big as 8 Gbytes.

Not to exceed Channel bandwidth

Even though the I/O channel adapter physically can connect up to 15, 7, 8 devices for the Fast/Wide SCSI, single-ended SCSI, and Fiber Link respectively, doing so would almost certainly cause I/O performance problems. The combined I/O load on all devices attached to the channel may easily exceed the channel bandwidth and the channel becomes the I/O bottleneck. A good hint is when there are excessive number of processes waiting for I/Os and the average I/O queue length on disks is not very high.

Although the exact number of disks a channel can connect without impact performance depends on user environment, a good rule of thumb is not to exceed 9, 4, 4 disks for the Fast/Wide SCSI, single-ended SCSI, and Fiber Link respectively. Be extra careful when the channel is also shared with tape devices.

Conclusion

System performance tuning maximizes the return on your hardware and software investments. A key component of that tuning is I/O optimization. Many low throughput and long response time problems are due to I/O bottlenecks. I/O optimization is becoming increasingly more important as the gap widens between CPU and disk speed.

MPE/iX systems have sophisticated mechanisms, such as post ranges, intelligent prefetch and checkpointing, to handle I/Os effectively. It performs best when extents are of large size, disks are clean, and active extents are balanced across all disks. As more loads and data are added to the system, situation deteriorates and eventually I/O bottleneck emerges.

Rather than suffering or fixing it when the bottleneck occurs, let’s apply the I/O tuning tips proactively to prevent it from ever happening. Best of all, many of the tips can be automated into weekly or monthly jobs. Let’s always keep our disks clean, data balanced, and I/Os streamed.

Biographic

Paul Wang is the president of SolutionSoft Systems, Inc. He is a software developer and consultant, specialized in transaction management, system performance, file system internals, data base, disk space management and On-line transaction processing. Previously, he was an internal architect of transaction management in HP's Core MPE/iX lab.