UNICOS System Tuning Improvements

Donald Lee, Cray Research, Inc., Eagan, Minnesota

ABSTRACT: *The area of performance improvement, a primary focus of Cray Research, has had many changes in the last two years. Some basic tuning techniques and assumptions have become obsolete. UNICOS fine-grained multithreading, system buffer cache improvements, primary/secondary file systems, ldcache improvements, and SDS packing all have significance beyond their basic functions. This paper discusses the potential impact of these features from a global system performance and tuning perspective.*

1 Introduction

System tuning is the process of applying the available resources of a computer system to the problems presented for solution.

One of the great strengths of UNICOS is the number and quality of the mechanisms available to an administrator or user for this task.

For example, users can improve the efficiency of applications by using main memory directly as workspace. If memory is considered too precious, it can be used to improve I/O performance. Choices include use as buffer space, as library cache, or as memory-resident file space. Administrators have two types of cache available. System buffer (kernel) cache can be added to be used to advantage by the whole system. Memory can also be configured as a memory-resident file system or as memory-ldcache on a disk file system to improve performance.

The SSD has similar flexibility and power. SSD space can be used as workspace for very large arrays (auxiliary arrays), as direct file space, as file-specific cache, as a file system, as an 'extension' of main memory (swapping), or as ldcache.

Specific knowledge of the workload, and of the performance characteristics of the resources at hand guide the process of making choices between competing uses of system resources in the hope of making optimal use of those resources towards the site's goals.

Over the years, experience has taught us that certain characteristics of customer workloads can be safely assumed, and certain performance characteristics have become folklore. As a result, there are many tuning tips that currently circulate among CRI analysts and customers. However, much of this folklore has been rendered obsolete by recent developments.

This paper will discuss some of the more interesting recent changes in UNICOS and described their impact on tuning issues.

2 Fine-grained Multithreading

Probably the largest single change in UNICOS 8.0 is the addition of fine-grained multithreading. This is not an entirely new feature. Some multithreading was present in earlier versions of UNICOS, but it was not very effective. In almost all cases, the practical limit on the available system CPU power was one CPU. (On a four-CPU system, this would be indicated by a system CPU time of 25% on sar(1).)

This limit had a predictable effect on systems with multiple CPUs. If the aggregate system CPU usage exceeded one CPU, all other system calls had to wait on system semaphores. System call service was delayed, and overall throughput was reduced. Semaphore wait time was also increased because system calls are not always initiated at convenient times.

For example, if two processes make two system calls per second, each requiring exactly a quarter-second of CPU time, the total system time is a half-second per process, per elapsed second. However, this occurs only if each process makes its system calls when the other process is not already in the kernel with the semaphore set. If the two processes issue their system calls at the same time, at the start of each second and at the half-second, then the rate of system CPU usage is 1.5 seconds per elapsed second. This is because one of the two processes must always wait for the other to finish before performing its system call. The wait time is clocked as system CPU time and is spent waiting for ownership of a system semaphore.

If the aggregate system CPU time was appreciably more than one CPU, then the cumulative effect of this semaphore wait time could be severe. System CPU time could rise dramatically as multiple processes 'pile up' on system semaphores, and system throughput would fall. In severe cases, systems would see idle time because the kernel could not service system requests in a timely fashion.

Even if the actual system call CPU time was less than one CPU, the random nature of the arrival of system calls could cause inflated system CPU time due to collisions on the system semaphores.

The fine-grained multithreading feature in UNICOS 8.0 was designed and implemented to address these problems. The result has been even better than many had hoped. For multi-CPU systems, especially for systems with more than four CPUs, the reduction of system CPU time has often been dramatic.

Prior to UNICOS 8.0, sites were strongly encouraged to do everything possible to reduce system CPU time. A number of sites made choices based on the absolute requirement to reduce system CPU time. Now that UNICOS 8.0 has arrived, excessive system CPU time is still undesirable, but is no longer a dominant issue in system tuning.

3 System Cache

UNICOS 8.0.2.1 contains a set of changes in the system buffer cache that have a dramatic effect on system CPU time and I/O efficiency through the cache.

The changes were made as a result of the discovery of the large amount of system CPU time being used in the system cache management routines.

The direct result of the changes was a large reduction in the timings of certain system calls using the system cache. On systems with average-sized system cache, the direct savings in system CPU on a `read`(2) system call could be more than 2 milliseconds of system CPU time. By comparison, a minimal `read`(2) system call takes under 150 microseconds of CPU time. In addition, the I/O rate of data through the system cache was dramatically improved in some cases. Some `write`(2) system calls showed improvements in the area of 100%.

From a global system tuning perspective, the implications of the changes are even more dramatic. One of the fundamental assumptions about the optimal size of system buffer cache has been eliminated. In the past, system cache was known to be a system bottleneck. If too much I/O passed through system cache, it caused excessive system time. Applications using the cache were likely to have a poor elapsed time. The larger the system cache, the poorer the performance.

Although system-buffered I/O is still not optimal for most applications, the size of the cache now has no link to the system time of the machine. The system cache is no longer the major performance bottleneck it once was.

In theory, a much larger system cache can now be considered in system configurations to reduce disk access and improve system throughput. It is possible that a sufficiently large system cache could replace the ldcache currently used on most systems for the / (root) and /`usr` file systems. Administrators no longer need to be as careful about configuring a minimal system cache.

Unfortunately, current testing has been exclusively focused on the speed of the I/O through the cache and on the system CPU overhead of I/O system calls. While reassuring, these results do not indicate the extent of the advantages that might be gained by making full use of the system cache changes.

4 Primary/Secondary File Systems

In UNICOS 7.0, `mkfs`(8) was enhanced to allow the separation of the partitions of a file system into primary partitions and secondary partitions. The primary partitions hold all file system metadata and all “small” user files. The secondary partitions are reserved exclusively for user data.

This feature effectively separates the small-granularity I/O activity from the large-granularity activity on the file system. This separation allows the two different types of partitions to be optimized for each type of I/O by using different disks, channels, and parameters.

Small-granularity I/O (such as inode access, bitmaps, and small files) works best on devices with low seek and latency costs. Disk bandwidth is of minimal importance because the cost of each I/O operation is often dominated by latency time. Devices with smaller sectors and faster rotational speeds excel in this case. Fragmentation of available space is also of minimal importance.

Efficiency of large-granularity I/O is normally achieved by maximizing bandwidth. With a sufficiently large I/O granularity, latency issues become insignificant. Large-granularity allocation is important in this situation because file fragmentation interferes with maximizing bandwidth. Striping is an attractive option in this environment.

Primary/secondary allocation allows the administrator to separate configuration of small- and large-granularity I/O based on the size of the files contained in the primary and secondary partitions of the file system.

Although file size is an imperfect basis on which to make judgements about I/O granularity, it appears in practice to work well.

The separation of configuration options not only allows the administrator to choose near-optimal parameters for the two partition types, but also does a good job of reducing hardware contention between the two types of I/O. The small-granularity I/O does not interrupt streaming disks and stripe-group I/O, and large-granularity I/O does not delay access to inodes and other data requiring rapid response.

The result of this capability is to free the administrator from tradeoffs that previously prevented use of techniques such as striping to increase bandwidth on file systems.

For instance, the previous lack of effective separation between small- and large-granularity I/O made striping impractical on any file system with non-trivial quantities of small-granularity I/O (which interferes with efficient striping).

With the primary/secondary feature, a file system with mixed-granularity I/O could be configured, for example, as four primary partitions and 12 secondary partitions, each consisting of a stripe group (sdd). In such a file system, the smaller-granularity random accesses tend to be confined to the primary partitions. I/O for the larger files, which is likely to be larger-granularity I/O, is confined to the striped secondary partitions.

Other benefits of this feature include the reduction of the number of partitions that need to be updated at `sync`(2) time, and the ability to choose the number of available inodes on a file system independent of the major allocation granularity of the file system.

The net result is that substantial flexibility and potential performance gains are now offered to administrators who wish to take advantage of this capability.

5 Ldcache Trickle-sync

When used appropriately, ldcache can be of tremendous advantage to the performance of a system. Ldcache is included in this paper not because recent changes have had a large impact, but because old ideas still appear to be obstacles to efficient use of ldcache.

The ldcache trickle-sync feature was first introduced in UNICOS 6.1.7. However, it is still not widely used; where used, it is often not well understood. It remains a feature that can have a major positive impact on system performance.

The trickle-sync feature includes:

- Changes to the algorithm that chooses ldcache units to sync to disk.
- The `-x` *max,min* option which controls trickle-sync by cache data age
- The `-h` *high,low* option which controls trickle-sync by amount of 'dirty' data

5.1 Ldcache algorithm changes

Prior to the addition of trickle-sync, the ldcache algorithm to select which data in the cache to sync to disk was a simple LRU (least recently used) algorithm. Dirty and non-dirty data was handled in the same way, and the least recently used portion of the cache was always the one reused when new cache space was demanded.

This algorithm had some very nice performance benefits. Unfortunately, it also had some undesirable side-effects on data integrity.

This algorithm guaranteed that the most frequently used data in the cache was the least likely to be synced to disk. File system metadata (like inodes, bitmaps, and directories) are often very frequently accessed. Because these are seldom the least recently used blocks, the algorithm seldom selected them to be synced. Therefore, when the most critical filesystem data was updated, it was left in cache and almost never written to disk. This meant that ldcache relied heavily on the `ldsync`(8) command, which is periodically performed by `init`(8), to force this data to be written to disk.

Unfortunately, this presented administrators with a difficult choice between performance and reliability. The ldsync interval was the only tunable parameter for controlling the ldsync behavior. Short ldsync intervals tended to prevent file system corruption in the event of an interrupt, but had very negative performance impacts. Long ldsync intervals tended to improve performance characteristics, but left the file systems vulnerable to corruption in the event of an interrupt. Even with a longer interval, there was still a large 'spike' of disk activity when ldsync was executed. The default of 120 seconds seemed to be too long for most critical file systems, but too short to satisfy performance concerns. Some sites even developed hybrid schemes using `cron`(8) and periodic reallocation of ldcache to avoid this tradeoff.

The addition of trickle-sync code to UNICOS changed the sync selection algorithm. The selection of cache units to sync is no longer made on the basis of how recently they have been accessed, but on the basis of how long it has been since they have been changed (made dirty). The result is that the sync operations from cache to disk now match the order of operations from user to cache much more closely, resulting in reduced chance of corruption in the event of a system interrupt.

5.2 Ldcache -x option

The `ldcache`(8) `-x` option improves the flexibility of ldcache by allowing the administrator to choose the maximum allowable age of dirty data in the cache. This option accepts the *max,min* parameters. The *max* parameter represents the maximum age, in seconds, of cache data. If any unit in the cache reaches this age, all units more than *min* seconds old are synced to disk.

This behavior has several advantages over the periodic `ldsync`(8) command. It is file system specific. Critical file systems can be configured with short *max* values to ensure that data on these file systems is written to the underlying disk in a timely fashion. Scratch file systems can be configured with large values to avoid disk I/O as long as possible, which improves performance. If the *max* and *min* values are close together (such as 30,28), only a small amount of data is synced every two seconds, and the load on the underlying disk is 'smoothed' very nicely (that is, the spikes caused by ldsync are avoided).

5.3 Ldcache -h option

The `ldcache`(8) `-h` option is probably the least understood ldcache option, but it is arguably one of the most useful for ldcache performance management.

This option regulates the amount of data in the cache that can be 'dirty' at one time. It is used to maintain the primary advantage of the cache: keeping frequently used data in the cache so it can be repeatedly accessed.

Without this option enabled, processes that write large quantities of data to ldcache have the ability to dominate the cache. This option allows the administrator to limit the proportion of the ldcache that is available for write activity.

The `-h` option accepts the *high,low* parameters. The *high* parameter specifies the maximum number of ldcache units (as specified also on the `-n` option) that are permitted to be 'dirty'. The *low* value is used as a trigger to start disk I/O. Once this *low*

threshold is reached, each time an ldcache unit is made dirty, I/O is started on the oldest dirty unit in the cache.

This means that if the number of dirty units in the cache is greater than *low*, and *current* is the current number of dirty units in the cache, then outstanding I/O is active on at least *current* - *low* units.

Early problems with this code have been fixed and the performance advantages of this feature have been shown over and over again.

If trickle-sync is configured on all ldcache allocations, periodic ldsync operations can be almost entirely eliminated.

The most visible and dramatic performance advantage of this feature is the ability to smooth out bursts of ldcache disk I/O, which, in severe cases, can briefly make a system appear hung. This problem is all too common on systems with very large ldcache that do not properly use trickle-sync.

This feature is reliable, fairly simple, and supported in the install tool. It offers great advantages for any site using ldcache.

6 SDS packing

UNICOS 8.0.3.1 introduces SDS packing, which is simply the ability of the kernel to move, or 'pack', SDS space directly, without using checkpoint or suspend operations.

6.1 SDS history

SDS was developed to make good use of the SSD and the VHISP (Very High Speed) channel. At VHISP speeds, even a delay of a few microseconds could substantially reduce the effective transfer rate of a request. The system calls created to support SDS gave the user direct access to the SSD with the lowest possible overhead (latency). In theory, this would allow even small requests to get a reasonable portion of the theoretical speed of the VHISP.

In fact, the SDS mechanism turned out to be very effective. The transfer rates attainable with SDS are higher than those of an SSD file system, and far higher than those of ldcache. Clearly, if peak performance is desired, SDS is the method of choice.

Unfortunately, to get these high transfer rates, the SDS mechanism had to be very simple and very fast. Contiguous, monolithic allocation was chosen for its speed and simplicity. As a result, fragmentation was very likely. The management of this resource was difficult, and has historically been relatively poor.

The quickfile daemon was developed to try to alleviate these problems. It used checkpoint/restart to preempt SDS so as to decrease fragmentation and manage oversubscription. However, this mechanism proved to be inadequate. Checkpointing turned out to be a very expensive management mechanism. Also, limitations on checkpoint meant that many jobs could not be managed by the quickfile daemon.

In UNICOS 8.0, the SDS manager was changed to use suspend, which had fewer limitations but still had drawbacks.

Suspend has almost none of the restrictions of checkpoint, but is a similarly expensive operation. Both checkpoint and suspend depend on copying the SDS image to a secondary storage device, which can be prohibitive with large SSDs.

Because the SDS relocation mechanisms are expensive, design decisions were made in the I/O libraries to avoid changing SDS segment sizes. The maximum amount of SDS space permitted for the job was always allocated, regardless of the actual amount requested.

An additional problem with SDS is that the SDS arena is shared with ldcache. It is often desirable to reallocate ldcache dynamically to respond to changes in workload. If done while jobs using SDS are running, SDS arena fragmentation is likely.

6.2 UNICOS 8.0.3.1 SDS changes

To improve SDS management, two changes were made to the UNICOS kernel:

- ldcache now allocates SDS space starting at higher addresses, or 'top down', rather than starting at address zero and working toward higher addresses.
- The concept of 'gravity' has been introduced to the SDS arena. Processes with SDS allocations 'slide' their allocations toward lower SDS addresses when SDS demand dictates.

The first change allows the reallocation of ldcache without direct conflict between SDS allocations and the ldcache space.

The second change causes the user SDS allocations to migrate toward address zero in an orderly fashion as SDS is allocated and deallocated.

The actual movement of SDS segments is done using main memory and the VHISP channels. It is a very efficient and simple routine that allows the rapid relocation of SDS with negligible impact on the overall system. It is completely transparent to the application and has no dependence on available swap or checkpoint space.

6.3 SDS packing benefits

The SDS packing feature is of immediate and obvious benefit to all sites that either use, or want to use, SDS.

This feature tremendously reduces the reliance on suspend to perform SDS management. In cases where SDS is not actually oversubscribed, SDS packing can often be done without any need for suspend operations.

For a site that actually oversubscribes the SDS resource, the job of the SDS manager is much easier, because remaining SDS allocations are packed in low SDS, leaving a single contiguous SDS region to be managed. This behavior reduces the amount of swap space required to support SDS oversubscription.

Full advantage has not yet been taken of this feature. Given this more efficient packing of SDS, the libraries could be changed to manage SDS as they were originally intended, allocating only the SDS being used rather than the NQS limit from start to finish as is currently being done. Interactive use of SDS could be accommodated.

Although this change does not eliminate the chance of ldcache fragmentation, a careful administrator can now change ldcache at any time on a running system, without taking extreme measures such as suspending all SDS work. The final result is that SDS usage has been made much more attractive.

7 Conclusion

Important performance characteristics of UNICOS have changed in recent releases. Changes with the most impact on tuning folklore include:

- Fine-grained multithreading improves overall kernel performance and reduces the need to lower system CPU time.
- System buffer cache changes improve I/O performance and allow the administrator more freedom in choosing system cache size.
- Primary/secondary partitioning improves tuning precision and flexibility in disk configuration.
- The ldcache trickle-sync feature improves the efficiency and reliability of ldcache.
- SDS packing addresses longstanding SDS management issues, making SDS usage much more attractive.

It is time to update the ideas of optimal tuning parameters in order to take advantage of the opportunities for improved system performance offered by these changes.