The Evaluation of Process Accounting Data to Analyze and Pinpoint Performance Bottlenecks in a Job Mix

Jeffery A. Kuehn, National Center for Atmospheric Research,
Scientific Computing Division, Boulder, Colorado

ABSTRACT: *When a supercomputer logs large amounts of system or idle time, expensive resources are being wasted. Normal system monitoring tools oftenfail to expose the source of this waste because the problems do not always lie within the operating system or its configuration. The evaluation of process accounting data provides many detailed insights into the performance problems of a job mix. Not only can process accounting data be used to trace resource utilization, it can also help to pinpoint performance problems, and in many cases suggest solutions to those problems. This paper discusses process accounting data analysis, as well techniquesfor understanding the hidden messages in the data.*

1 Introduction

During the day-to-day management of large computer systems, there is frequent monitoring of system utilization through the use of tools such as the sar(l) command. Ideally, the reports should reflect no idle time and user utilization of 95% or greater. When this is not the case, conventional wisdom suggests a need to "tune the system". But adjusting kernel and NQS configuration parameters does not always produce significant improvements because the problems often lie outside of the operating system software within the system's applications. When NCAR initially embarked on system tuning efforts for an eight processor Y-MP, adjusting the system configuration allowed for best case sar(l) reports with user time percentages in the middle eighties range and idle time noticeable even on the best days. After migrating toward an application-centered approach to system tuning, the sar(l) reports reflected utilization percentages in the middle nineties with little or no idle time apparent for the general purpose machines, and percentages in the high nineties for the more lightly loaded special purpose machines. In short, on an eight processor machine, NCAR was able to reclaim approximately one additional processor of utilization beyond a best effort system tuning approach by monitoring and tuning the applications on a regular basis .

This paper does not suggest hard numbers for limits on application performance as that is not possible, but rather provides guidelines for comparing the jobs in a particular site's job mix so that the worst performers can be incrementally improved. At NCAR, the sar(I) reports are used to identify when applications need tuning. The Technical Consulting Group within NCAR's Scientific Computing Division is then responsible for ferreting out the poorly performing applications and improving their performance by working directly with the user community. In the three years since this policy was implemented, NCAR's user community has responded very positively. Furthermore, the application tuning operations have frequently resulted in reduced charges. However, since NCAR's user community is somewhat transient and the applications are the result of evolving research, continuing application tuning efforts are required to prevent system performance from sliding back down into the middle eighty percent range.

The key to the success of these continuing efforts has been to identify those applications which can benefit most from tuning and to pursue improving them first. This has the effect of bringing system performance in line with goals within the period of a few weeks. The process accounting records, maintained for job charging, are the primary source of information used to identify problem applications.

2 Quick overview of process accounting

A comprehensive discussion of the process accounting system is beyond the scope of this paper, however a quick overview will be helpful. The process accounting records contain a summary of the accumulated resources used by a process during its execution. This summary is the basis for the reports printed by the ja(l) command, but contain more information than ja(1) typically displays. Most of the information in the process accounting records comes from the kernel's process table, while other pieces come from the kernel's job table. The records are written during the exit of a process (or job) and thus the accounting records are only available for post-mortem analysis. The records contain no time-series information, but do contain some timememory integrals which are useful for charging but

not very helpful for tracking performance problems. The available information includes user id, job id, elapsed time, user time, system time, memory high water mark, and others. For a complete list of the fields available see /usr/include/sys/acct.h.

Because the process accounting records contain information about resource utilization, they can expose problems associated with an "imbalance" in this area. Since the idea of a balance in resource utilization is difficult to define, it will be described in relation to other processes running on the system for the purpose of this discussion. With this in mind, process accounting can help to identify programs which are either I/O bound or CPU bound when compared to other processes on the system. Additionally, the records can point out, to varying degrees, programs which have inefficient I/O, inefficient multitasking, and inefficient memory management. Since the process accounting records evaluate the process from the kernel's perspective, problems such as low MFLOP rates, lack of vectorization, or inefficient algorithms cannot be discerned, but because CPU bound processes could suffer from one or more of these problems, the need for further analysis may be implied.

3 Performance problems exposed by process accounting

As mentioned previously, process accounting clearly exposes several problems related to I/O performance which directly contribute to high system time and idle time percentages observed in sar(l) reports. By focusing optimization efforts on the processes reporting the highest system times, system performance numbers can be quickly affected. Since I/O problems are also at the root of system idle time, improving the I/O performance of applications can positively affect idle time as well. The use of process accounting records to recognize three classes of I/O performance problems are examined in this section.

3.1 I/O buffers sized incorrectly

When the library I/O buffers are misconfigured for an application, large I/O requests are split into several smaller I/O requests by the application libraries. This means that a single I/O request at the user level may generate many calls into the operating system, each transferring a subset of the data.

3.1.1 Characteristics

Small I/O buffers leave a telltale fingerprint in the process accounting records. For processes using a significant amount of resources, both in terms of accumulated system CPU time and in terms of the amount of data transferred, the following characteristics can identify those processes which may benefit from I/O buffer size adjustments.

- A relatively large percentage of total CPU time is system CPU time.
- A relatively large percentage of the elapsed time is I/O wait time.
- Much of the I/O wait time is logged as "locked I/O" wait time.
- The number of logical I/O requests divided by the number of physical I/O requests will be greater than I .0.
- The average transfer size is relatively small.
- A rough estimate of the transfer rate formed by the quotient of data transferred and I/O wait time is significantly less than the device speed.

3.1.2 Further analysis and improvement

The CRI procstat(1) utility can be used during program execution to determine which files may benefit from larger I/O buffers. Analysis of the I/O statements in the source code will show the actual transfer sizes. These transfer sizes should be considered as the minimum size for an I/O buffer. In the case of sequential access files, performance can be further improved by making the I/O buffer size a multiple of this minimum.

3.2 I/O routed through the system cache

The Fortran libraries routinely attempt to route data transfers around the kernel's buffer cache, but in many cases, such as for text I/O, Standard C I/O, and illformed raw requests, the data is sent through the system's buffer cache. For requests on small files and files which are accessed from many different programs, passing the data through the buffer cache works to the advantage of overall system performance. But in the case of very large datasets associated with grand challenge applications, this is often an impediment to performance. If such files are read or written only once, caching them does little good. If they are read or written more than once, the buffer cache must typically be inordinately large to ensure reuse of the data, reducing the total memory available for large codes. Finally, the kernel's buffer cache management algorithms are geared towards acceptable performance in the general case, whereas an application that carefully manages its own I/O can almost always improve on the general case LRU-style algorithms used in the kernel.

3.2.1 Characteristics

Data movement tbrough the operating system's buffer cache also leaves a telltale fingerprint in the process accounting files. For processes using a significant amount of resources, both in terms of accumulated system CPU time and in terms of the amount of data transferred, the following characteristics can identify those processes which are moving data through the kernel's buffers.

- A relatively large percentage of total CPU time is system CPU time.
- A relatively large percentage of the elapsed time is I/O wait time.
- Much of the I/O wait time is logged as "unlocked I/O" wait time.
- The average transfer size is relatively small.
- A rough estimate of the transfer rate formed by the quotient of data transferred and I/O wait time is significantly greater than the device speed.

3.2.2 Further analysis and improvement

As with the case of small I/O library buffers, a code passing data through the system cache can be run under the control of the CRI tool, procstat(l), to isolate which files are at issue. Re-routing this data around the system cache may be straightforward, but several cases in which improving the code required significant rewrites have been encountered. As always, the costs and benefits must be weighed.

3.3 Inefficient SSD usage

The SSD is an often misunderstood tool for improving I/O performance. It usually shows best for out-of-core solutions where the working data for a program are stored in the SSD and the program transfers small pieces into memory, cycling repeatedly through the working data. In these cases, each word of data on the SSD is read and written many times during the execution of the program. The worst case use of the SSD is for files that are read once and never written. Files which are written once and never read are generally not appropriate for the SSD except in special cases where it is necessary to disencumber memory as quickly as possible so that computation may be resumed. In all cases, larger transfers to the SSD are more efficient than small transfers. Therefore, preference should be given to larger data transfers.

3.3.1 Characteristics

As with other I/O processes, SSD usage or misusage leaves a clear trail in the process accounting records. For processes using a significant amount of resources, both in terms of accumulated system CPU time and in terms of the amount of data transferred to and from the SSD,the following characteristics can identify those processes which are not reusing the data they have stored in the SSD or those which are transferring data in small chunks.

- A relatively large percentage of total CPU time is system CPU time.
- The average size of an SSD transfer is small.
- The ratio of total SSDI/O to SSD high water mark is small.

3.3.2 Further analysis and improvement

As was the case with other I/O problems, further insight into performance on a file-by-file basis can be gained by running the code in question under the control of procstat to isolate the SSD file accesses requiring optimization. Restructuring the code to make use of the SSD through larger transfers will reduce the system time. For those files which are not reusing the data in the SSD, moving the files back to disk is worth considering.

4 Performance problems implied by process accounting

Several performance problems which are not clearly exposed by the process accounting records can still be inferred from process accounting. As with the I/O problems, the key to recognizing this new class of problems is again the percentage of the time spent in the operating system. Once a suspicious process is identified, other performance analysis tools can be employed to isolate and fix the problem.

4.1 Multitasking problems

Multitasked codes can exhibit unusually high system CPU times for several reasons, the most common of which stem from granularity problems, both too coarse and too fine. Furthermore, if a program requests more processors than it is able to use, an increase in system time will be evident. Also, seemingly innocent constructs in which a system call occurs within a guarded region can cause a cascade of deadlock interrupts and thus high system time.

4.1.1 Characteristics

The first step to identifying poorly multitasked codes in the accounting records is to make note of those codes which are flagged as multitasked through the presence of a valid multitasking structure. Though this structure contains information about the connect time to N CPUs, this information is of limited usefulness when collected from a process running in a batch environment. The fingerprint of multitasking problems includes combinations of the following characteristics in processes using a significant amount of CPU time resources .

- A relatively large percentage of total elapsed time is system CPU time.
- A relatively large percentage of total CPU time is system CPU time.
- A relatively large percentage of total elapsed time spent in semaphore wait.

4.1.2 Further analysis and improvement

Although the accounting records are not helpful in locating the source of multitasking problems, CRI' s atexpert(l) tool can quickly localize the problem and provide insight into its cause. Depending on the type of problem encountered, a solution may simply require a reduction in the number of CPUs called, or it may require some recoding.

4.2 Poor memory management

Codes which make requests to the operating system for increases in memory will incur a great deal of overhead if memory must be reshuffled to grant the request. This can become a problem if an application makes several small memory requests to the operating system rather than aggregating the requests. This occurs most often when the application is explicitly managing its own memory.

4.2.1 Characteristics

The fingerprint of poor memory management is, like that of multitasking problems, rather generic~ The characteristics to watch for are:

- A large otherwise unexplainable system CPU time.
- A relatively large I/O swap count.

4.2.2 Further analysis and improvement

The CRI procstat(1) command is also useful in tracking problems with memory allocation. If memory allocation is in fact a problem, it can be easily resolved through loader directives which set the initial size and the default increment for dynamically managed memory.

4.3 CPU bound processes and low MFLOP rates

A CPU bound process is one which spends most of its time running in user space. Frequently, this is caused by slow and/or inefficient algorithms which the compiler fails to vectorize. The accounting records reveal nothing about the MFLOP rates achieved by a code, but a disproportionately large percentage of the total CPU time in user space execution can imply that there may be a problem. Further analysis with more specialized tools will be necessary to determine if a particular CPU bound code will be responsive to optimization.

4.3.1 Characteristics

Unlike the other performance problems discussed in this paper, the hallmark characteristic of a CPU bound process is an unusually small percentage of the total CPU time spent in the operating system. Other clues that a code is CPU bound might include an I/O wait time that accounts for an unusually small percentage of the elapsed time. While these could indicate an efficient but computationally intensive process, it is usually worth further analysis.

4.3.2 Further analysis and improvement

Most of CRI's performance analysis tools are geared towards analyzing CPU bound codes, but the first and simplest tool is the hardware performance monitor. This tool can provide a snapshot of the code's actual MFLOP rate which can be directly compared against expected performance numbers for the machine in question. Should the code be running at well below the expected performance, additional tools such as the Perftrace libraries can be used to instrument the code and locate the performance bottlenecks for optimization.

5 Is once enough?

How frequently to monitor system utilization will obviously depend on the user community and job mix. If the community is transient, or if the job mix varies greatly, it is worthwhile to perform frequent examinations of the process accounting. If the job mix is more stable and the users are veterans, less frequent monitoring will suffice. At NCAR, the sar(l) reports help to determine the appropriate level of system monitoring and user contact required for optimum system utilization. When the sar(1) reports appear good, monitoring is given a low priority. When the sar(1) reports look poor, monitoring demands a higher priority. At this point, a few of the worst performers will be contacted each day for a couple of weeks. Most of the users contacted have responded well and were willing to cooperate. Working with the users representing the worst offending codes is usually sufficient to bring the machine's performance back into line with expectations in a very short period of time. Over the past few years, the method for contacting users who appear to be having problems has been streamlined down to a short email with an offer of assistance and an attached explanation of simple changes aimed at correcting the problem, such as resizing I/O buffers.

6 Summary

There is more to tuning a system than adjusting the kernel configuration parameters. To achieve the highest possible system utilization numbers, it is also necessary to tune the applications running on the system. To put this another way, a well-tuned operating system cannot compensate for a poorly-tuned job mix. Process accounting records can provide valuable clues about which applications are performing poorly and, in some cases, can also provide clues as to why an application is performing poorly. Regular monitoring of the process accounting records and contact with the user community can be very effective tools in maintaining a well-tuned and efficient system. While application tuning is not a panacea for system performance problems, it is an integral part of maximizing the available resources on a supercomputer.