Some of The Foibles of Performance Monitors

A number of questions that I have been asked recently on the subject of Performance Monitoring have convinced me of the value of a short piece on the idiosyncrasies and issues that surround the subject. For example, vendors' explanations of individual counters are frequently terse and sometimes non-existent. Often, the unit size is not provided, e.g. is this memory counter in bytes, kilobytes or megabytes?

This is not a treatise on performance monitoring techniques *per se*, merely a discussion of some of the problems that surround it and of issues that relate to the subsequent use of collected metrics. As monitoring can be carried out during benchmarking, testing and production running, the topic is relevant to many chapters in part 1 of the book.

General Comments

Standard system level monitoring tools, typically supplied by the OS vendors, tend to provide the most credible metrics simply because they are at the heart of the kernel and therefore their relative accuracy can usually be relied upon. Specified counters are logged every *n* seconds (termed the collection interval). The length of a monitoring session is configurable; it may indeed run 24 hours per day on a production system. This type of monitor incurs a modest overhead (typically less than 1-2% CPU), as it is simply collating kernel counters. I am assuming here that the collection interval is set to a reasonable figure; let us say no less than 15 seconds. The use of smaller intervals will increase the overhead and may impact on performance. The size of the collection interval should reflect your objectives, e.g.:

- 1-5 minutes may be sufficient for standard production system monitoring, as part of the overall Capacity Management system
- 10-20 seconds may be adequate during benchmarking or testing activities
- <=5 seconds should only be used when necessary, e.g. to identify troublesome spikes in performance.

If more than one server is being monitored, particularly during a benchmark or a stress test, ensure that the time is synchronised across all servers. Although this advice may seem obvious, I have spent many a mind-numbing hour trying to analyse metrics from servers with different time settings!

Make sure that you know whether the timestamp that appears on the output relates to the start or end time of the collection interval – it may be important!

On many systems, e.g. Unix and Windows, the first collection interval will contain metrics from the last time that the server was booted. This information should usually be ignored, unless you have a specific need for it.

Some counters, e.g. CPU time will contain actual usage during the collection interval; this will typically be done by subtracting the total time (since last boot) at the end of the interval from the total time at the start of the interval. Other counters that do not increase inexorably, e.g. the amount of free memory, may be based on a single sample taken at the end of the interval.

One or more collection intervals may contain weird looking figures; they can be extremely low, extremely high, or even negative! This can be caused by the monitor struggling to complete its work because one or more parts of the system is highly stressed. It is best to ignore such intervals.

System-Level Monitoring

CPU

If you are using discrete elements of CPU usage, e.g. user, system or kernel time, *et cetera*, beware that some monitors may exclude elements such as the time spent handling interrupts. In this example you could derive the interrupt usage by total CPU – (User CPU + System CPU).

Some Unix monitors produce a "waiting for IO" element of CPU usage. This is **NOT** CPU usage and should not be included in the total CPU utilisation; it is an indication that the CPU could be used if one or more IOs completed, i.e. it is a potential to use the CPU; it can be a sign that there is an IO bottleneck. The usual foolproof method of calculating the actual total CPU utilisation is 100% minus the CPU idle time percentage.

On multiprocessor systems, some monitors may produce an aggregate CPU usage figure, e.g. if a server has 4 CPUs up to 400% usage may be shown, as opposed to an averaged figure across the 4 CPUs.

Disk

Disk monitoring can be confusing. Back in the 1980s disk subsystems were relatively unsophisticated. A device driver in the OS would queue requests for a specified disk, releasing only one at a time. This made monitoring fairly straightforward: the number of requests handled; the time that the disk was busy (from the perspective of the device driver); the average service time per request; and the time spent queueing in the device driver. The arrival of SCSI, which allowed the device driver to have multiple requests outstanding (see Chapter 17 – Hard Disk Basics), has somewhat muddied the waters. Nevertheless, the standard system monitors will still provide useful metrics on how busy the disk is. Monitors within the disk subsystem itself (usually only on the more sophisticated subsystems) can provide more detailed information.

It is common to see periodic high service times (100ms or more) on what appears to be a lightly loaded disk. This can be caused by a short flurry of system activity, e.g. a filesystem flush.

Memory

Although there are a large number of memory-related counters there is no single counter that indicates the general health of the memory management system. Ultimately, it is safer to analyse the effect of any memory problems on response times or throughput in the application. Adrian Cockcroft's book *Sun Performance and Tuning (2nd edition)* contains an excellent description of memory management P.326-336. While it is Sun specific the majority of the processes that are described apply to most virtual memory management systems.

Paging activity provides one important view of memory performance. The overhead that is involved in resolving a page fault depends whether it is a "soft" or "hard" fault. If a page has been inactive for a while it will end up being added to the free list (pages that can be redeployed). When the process from which the page has been removed requires it back, it is possible that the page is still on the free list. If so, it is simply a case of altering the status of the page and adding it back in to the process's working set. This "soft" fault, as it is called, is relatively cheap to resolve in resource terms. "Hard" faulting is significantly more expensive. In this case, the page is no longer on the free list. If it has been modified it will have been necessary to write it out to the paging backing store on disk. When the process requires it again it must be retrieved from backing store, necessitating further disk IO. Hundreds or in some cases thousands of soft faults per second can be handled without any noticeable effect on performance (Microsoft software in particular seems to generate many soft faults). Hard faults can soon affect performance if there is any appreciable volume of page reads / writes over a sustained period. While it will vary from OS to OS, ideally page reads should not exceed 10 per second for sustained stretches; similarly, writes should not exceed 3-4 per second.

On some systems, e.g. Unix, the rate at which the page out scanner is looking for inactive pages can provide indications that there is a memory shortage.

The total size of virtual memory may provide some clues on memory performance. Once again it will vary by OS and by the type of work but the size of virtual memory should not be more than twice the physical memory size, preferably less.

Always be careful to understand the unit size that is used in memory counters, sometimes they can be bytes and sometimes pages, and in the latter the size of page can vary from system to system.

Process Level Monitoring

Process level information will be mostly derived from information held within the kernel. Some OSs do not provide IO counters on a per process basis. Another issue can be the ability of the kernel to attribute the cost of IO interrupt handling to the process that invoked the IO.

Some monitors may have difficulty picking up processes that are created after the monitoring session has started.

Some systems provide an "accounting" system, whereby the start / end time of each process, along with the resources used (potentially subject to the restrictions mentioned above) can be logged and subsequently analysed.

Transaction / Product Monitoring

You need to establish how credible any monitor is. For example, I have seen metrics for SQL statements that were blatantly inaccurate. The usual issue for such monitors is that they can only see "what goes past their nose" and there can frequently be usage that is performed in another process or system component that the monitor simply cannot see.

In these days of multi-tiered architectures, the ideal method is to track resource usage across threads, processes and servers. This can be done manually but it is inevitably slow, cumbersome and labour intensive. On bespoke systems it may pay to develop an "angiotrace" facility, which will automate this process. Where third-party products are used this may be difficult if not impossible to do.

Handling "Uncaptured" Usage

The basic issue is to understand what elements of resource usage (say the CPU) are captured, and more importantly what elements are not captured. As mentioned previously, IO interrupt handling may not be captured at the process level. When using such data for sizing, modelling or capacity planning purposes it is obviously important that all resource costs that are used directly or indirectly by a process (or thread) are accounted for. In the interrupt handling example, it may be possible to calculate the total system-wide interrupt handling cost and to prorate it across processes on an IO basis. Alternatively, in lieu of any IO counters by process, it may be necessary to perform the prorating on the basis of the amount of CPU consumed by each process.

Issues Surrounding The Use of Time-based and Event-based Monitors

It is very easy to encounter problems when using time and event-based monitors, depending on what your objectives are:

- Time-based monitoring, typically at the system level, is satisfactory when simply keeping an eye on the performance of a production system over time

- The use of time-based monitoring when testing individual transactions or jobs, possibly as part of a sizing exercise, requires care. If you are trying to isolate the resource usage of transaction a from transaction b, it follows that they must not reside in the same collection interval; therefore, they must run sequentially and there must be a delay so that transaction b does not start until the commencement of the next collection interval

- Where event-based monitoring is used, e.g. process, transaction, job or SQL statement, for a defined period of time, be aware that at the specified finish time there may well be "events" that are still in progress and you may get incomplete figures or (depending on the monitor) no figures at all for these outstanding items.

Logging and Traces

Logging at the transaction level may not be too much of an overhead although it is important to ensure that no IO bottlenecks occur. The use of dedicated disk(s) for logging or some form of asynchronous logging where IOs are batched can minimise any problems.

Tracing can be a horrendous overhead, quite possibly consuming 20-30% of the CPU. It follows that its use should be limited, particularly on production systems.

Further Reading

It will naturally pay to invest in a solid book on performance monitoring and tuning. Examples include:

Waters, F., *AIX Performance Tuning*, Prentice Hall, ISBN 0133867072

Cockcroft, A., Pettit, R., *Sun Performance and Tuning (2nd ed.)*, Prentice Hall, ISBN 0130952494

Shee, R., Deshpande, K., & Gopalakrishnan, K., *Oracle Wait Interface: A Practical Guide to Performance Diagnostics & Tuning*, Oracle Press, McGraw-Hill/Osborne 2004, ISBN 0-07-222729-X

