Batch Queue Management and the Magic of '2'

Cary Millsap/*Hotsos Enterprises, Ltd.*

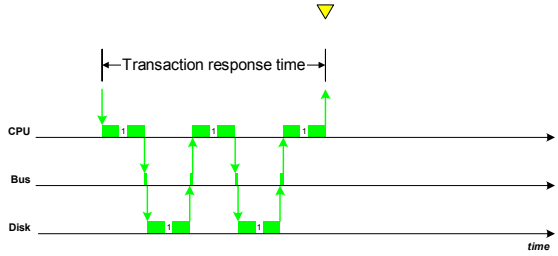
Introduction

Trying to run too many simultaneous Concurrent Manager jobs hurts system performance. The temptation to make this error is powerful: the Concurrent Manager lets you specify any number of queues that you want, and certainly your intuition might tell you that more queues might yield better batch throughput. But executing more than about two simultaneous batch jobs on a CPU degrades response times for everyone and actually reduces the total amount of work the system can do for your business. In this presentation we discuss the science behind why the number 2.0 is the magical key to batch queue management that maximizes performance for everyone.

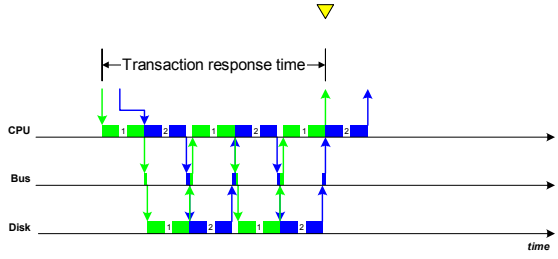
How Does Workload Affect Response Times?

When a system becomes too busy, it slows down. We relearn the lesson every time we navigate rush hour traffic. Nearly everyone understands the connection between "too busy" and "poor response times." The concept is a cornerstone of system performance management. Yet, batch workload overload has been a common cause of system performance problems in the hundreds of Oracle Applications customers that my colleagues and I have visited in the past ten years.

We use a notation similar to one used in [Menascé (1998), 36] to examine the interrelationships among resources inside an Oracle Applications system. In the following picture, we can see a single batch job requesting and receiving service from three system components, a CPU, a system bus, and a disk drive.

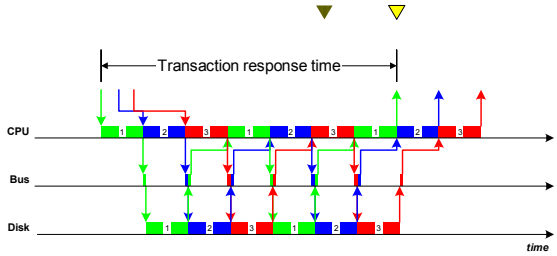


If we add second batch process to the mix, we can see the interrelationships between two processes competing for the same resources.



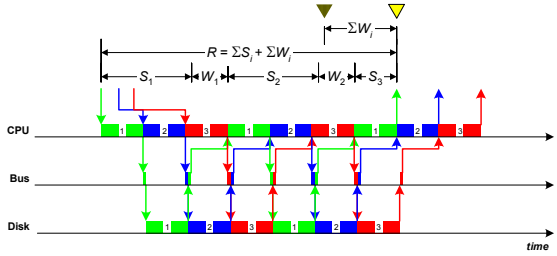
In this notation, a crook in a resource request arrow denotes queueing for a resource. Job 2 in this drawing queues briefly during its first request for CPU, at a time when job 1 is receiving CPU service. However, during most of the observation interval shown here, there is no queueing for any resource. Job 2 experiences only a single very brief stay on the CPU queue, and job 1 experiences no queueing delay whatsoever.

If we add a third batch process to the mix, we see much more queuing in the system.



In this scenario, job 3 causes a bit of performance trouble for everyone because its presence motivates a CPU queue length of one throughout almost the entire observation interval. The only CPU service request that does *not* wait in the CPU queue is the first CPU service request by job 1. All subsequent CPU service requests must queue for CPU because the CPU is busy serving another request each time a request is made.

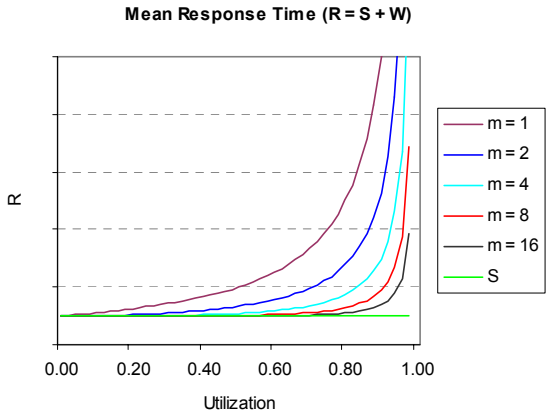
Note from the position of the triangle that the introduction of job 3 into the system has degraded the response time of job 1. The exact amount of response time displacement is equal to the queueing delays incurred by job 1. The body of mathematical study called *queueing theory* in fact allows us to predict the expected waiting duration for a given measure of system busy-ness.



In queueing theory, we generally denote response time as R, service time as S, and wait time as W, so $R = S + W$. In our example, job 1 receives three contiguous resource accesses (labeled S_1, S_2, and S_3), and job 1 queues for service twice (labeled W_1 and W_2). The total response time for job 1 is therefore:

$$R = S + W = S_1 + W_1 + S_2 + W_2 + S_3 .$$

The job 1 response time degradation that is attributable to the presence of other jobs on the system is exactly the sum of that job's wait time, $W = W_1 + W_2$. The "magic of queueing theory" is its ability to predict this queueing delay for a given level of workload. The graph shown here relates average response time (R) to the average utilization of a resource.



In this graph, the parameter m represents the number of resources serving a single queue. For example, we can model an 8-CPU symmetric multiprocessor (SMP) computer using $m = 8$. The mathematical formulas required to produce such a graph are available for the price of a good book. Especially good sources are [Gunther (1998)] and

[Jain (1991)]. Hotsos offers an easy-to-use Microsoft Excel queueing theory workbook that is available online [Millsap (2000)].

Note that regardless of utilization, average service time (S) for a given resource is constant. The vertical distance from S to R is of course $W = R - S$. When two identical jobs yield different response times, the difference is because of the difference in workload.

For non-SMP computers, this difference is striking throughout the utilization range. For example, a job on the $m = 1$ curve above will take about twice as long to run when utilization is 50% as it will on an unloaded system.[1] It will take about four times longer to run if it is run on a system with 75% average utilization.

On SMP computers, you can heap a lot more workload onto the system without noticing response time degradation. For example, a job on the $m = 16$ curve above will have almost exactly the same response time at 80% utilization as it did at 10%. However, for SMP systems, the wait times take a more violent swing upward when utilizations reach their critical range.

This "critical range" is a function of SMP order (number of CPUs, or m):

Number of CPUs (m)	Critical CPU utilization value
1	75%
2	82%
4	88%
8	92%
16	95%

The moral is this: It is *very* important for your system's management team to keep system utilization to the left of its critical zone if you intend to provide consistent response times to your user community [Millsap (1999), 7–9]. This job is important on non-SMP systems because they are so sensitive to workload changes everywhere from 0% to 100% in the utilization domain. And the job is important on SMP systems because they are highly sensitive to workload changes near their critical zones.

The Importance of Batch Queue Discipline

A system's management team has two fundamental batch queue goals.

1. As long as there are jobs awaiting execution, you want to run as many of them as you can. You don't want to waste the precious CPU capacity that you've bought and paid for. After all, once a CPU time slice goes unused, it can never be reclaimed.[2] It is a mistake to allow too many CPU cycles to go idle if you have jobs that are sitting in a *pending* state in your batch queue. Meeting this goal appeals to a batch queue manager's senses of generosity and kindness.
2. But you don't want to run so much workload that you degrade response times for your system's interactive users. As we'll see in this paper, overloading your system with batch jobs will actually degrade your system's ability to do useful work for your business. It's a situation from which absolutely nobody benefits.[3] Many

[1] But that's okay if it runs twice as fast as you need when utilization is near 0%.

[2] However, this does *not* mean that you want to run your system at 100% utilization. Remember that response times get flaky when a system nears its critical utilization zone. Hence, in an environment where there's a mixture of batch and online workload, a system manager will *try* to leave some CPU cycles idle in order to maintain better response time consistency for the online users on the system. Such idle CPU cycles spent in this manner aren't wasted; they're spent to obtain the value of consistent response times.

[3] Well, maybe your hardware vendor will benefit if you can be convinced that your problem is insufficient capacity. But before you buy, explore the two usually higher value avenues of (1) balancing your workload more efficiently in a longer duration time window, and (2) eliminating unnecessary workload such as that caused by unoptimized SQL.

> people overlook or ignore this goal, because enforcing it is difficult. Enforcement requires the right blend of intellectual self-confidence, stubbornness, and persuasiveness.

Being Santa Claus is more fun than being the bearer of bad news. But failing to execute the disciplinary responsibilities imposed by constraint number two can cause your system performance to go very, very wrong for *everyone*. The big problem is this: If doing your job means that you have to discipline your boss' boss based on a theory that you really can't convincingly explain, then you're not going to be able to do your job. The goal of this paper is to help you understand the consequences of ignoring the disciplinary side of batch queue management so that you will provide the best performance possible to everyone on your system.

Let's address one of the most difficult challenges that your system's batch queue manager will have to face. If you're not your system's batch queue manager, then imagine that you are. And imagine that the following words are coming from the mouth of your CFO:

> "Why are you letting my staff's jobs sit *pending* in the Concurrent Manager for 30 minutes before you'll permit the system to even begin executing them? At least let their jobs start running so that they can be making progress toward completion. As it is, their reports are getting *no* CPU cycles, and we're not getting anywhere this way. [*Expletive deleted*], go add another batch queue."

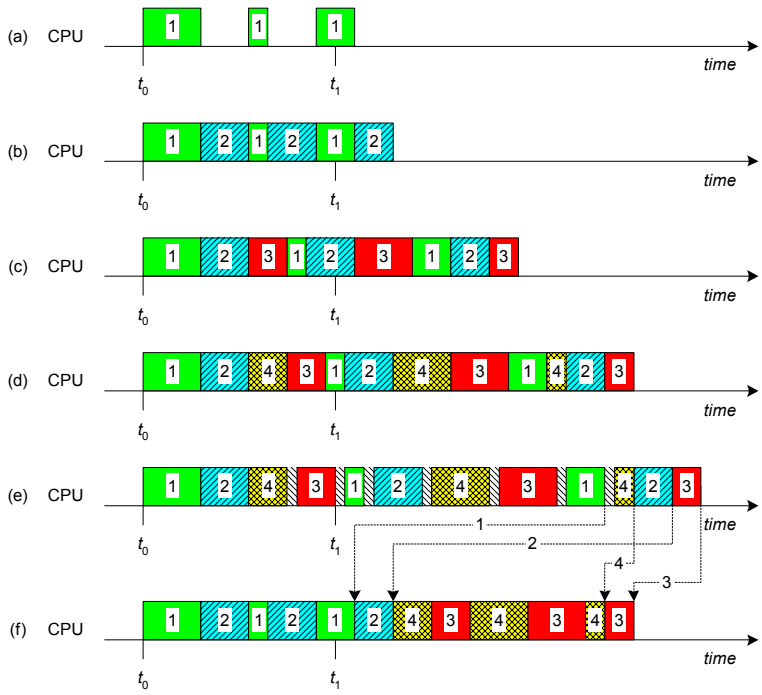
So, why not just dump everything onto the CPU queue and let the operating system figure things out? It would take some of the pressure off you, because after all, then it would be the system's fault that the job isn't finished. If you force the job to queue in the concurrent manager, then it looks like it's *your* fault it isn't finished.

There are two reasons that you should not succumb to this argument. First, the CPU scheduler will not prioritize jobs the way that *you* would prioritize jobs. The Oracle Concurrent Manager gives you extraordinary control over job prioritization that cannot be duplicated at the operating system layer.[4]

Second, and more importantly, long CPU queues degrade the system's ability to do work. When there are more than about three processes queued up for a CPU, the operating system scheduler begins to steal a disproportional amount of CPU capacity from the applications you're trying to run. So instead of an operating system (OS) overhead of 5% or less, you begin to see OS overhead of 10% or more. Workload overload begets diminished capacity, which diminishes the system's ability to get workload off the queues, which begets further diminished capacity, and so on. System-wide performance can become horrific and remain so until someone relieves the system of workload.

The following sequence of CPU timelines from the notation introduced above shows what happens when we keep adding batch workload to the system to the point of overload and beyond.

[4] If you're thinking that you can prioritize jobs at the OS level by using, for example, the UNIX *renice*(1) command, stop now. Incrementing or decrementing the priority of Oracle server processes leads to terrible performance problems. Every Oracle server process obtains and releases latches, locks, and enqueues. When you decrement the OS priority of any Oracle process, then you extend the duration during which that process will hold a latch, lock, or enqueue needed by the very processes to which you're trying to grant higher priority. The result of manipulating OS process priorities is that you retard the speed of the very processes you're trying to be "nice" to.



In scenario (a), we see a system with a single batch queue running one batch job. CPU utilization for the observation interval $[t_0, t_1]$ is roughly 50%. The approximately 50% CPU idle is time consumed by the batch job visiting other resources in the system (bus, disk, network, and so on, which are not shown in the drawing). Assuming that there are no unnecessary bottlenecks, users will perceive system performance in scenario (a) to be excellent.

In scenario (b), we add a second batch job to the system. This job makes effective use of the idle time left in scenario (a) to double system throughput for the observation interval $[t_0, t_1]$. System performance is almost identical to that obtained in scenario (a). It is a good thing to have added the second concurrent batch queue, because we have used idle CPU capacity to do valuable work without inducing a performance problem for anyone.

In scenario (c), we add a third batch job. This third job introduces queueing for the CPU (which is not explicitly drawn, but which is evident, for example, from the displacement of the second CPU request for batch job 1). The queueing in turn degrades performance in a way that is very noticeable for the first two batch jobs. Their response times are noticeably worse in scenario (c) than they were in scenario (b). Total system throughput remains at five units of work completed in the interval $[t_0, t_1]$, but the individual users' perception of work accomplished per unit of time has degraded.

In scenario (d), we add a fourth batch job. This job intensifies the CPU queueing and causes further response time delays for all the jobs in the system. Because the average CPU queue length has reached 3 for much of the picture, the real situation is even worse than scenario (d) lets on. Scenario (e) shows what actually happens. The long CPU queues cause an extra CPU scheduler overhead, which is denoted by the gaps between batch job request services. The extra CPU scheduling overhead further degrades system response times.

In scenario (f), we can see what would happen if were to use only two batch queues and force the third and fourth jobs to wait their turn in one of these two queues. The result, perhaps surprisingly to many people, is that *all* jobs, including even the ones required to sit pending in a batch queue, would finish sooner than if all the jobs had been dumped onto the system for the CPU scheduler to figure out. The moral: Restricting batch workload so that you *avoid excessive CPU queueing* makes the system produce results sooner for everyone, even the people whose jobs are required to wait their turn in the batch queue.

How Many Simultaneous Batch Jobs Can I Run?

Now it is time to answer the question, "How much batch can my system stand?" The answer is amazingly simple: you should attempt to run *no more than two* simultaneous batch jobs per CPU on your system. This upper bound value of 2.0 is in fact special, as we shall see. You will probably achieve optimal system performance by running no more than about 1.8 batch jobs per CPU available to your batch processing.

Since the mid 1990s, my colleagues and I have made empirical recordings of how many batch jobs our consulting clients could run on various types of systems. Before we began collecting data, we estimated that the key parameters were (1) the number of CPUs on the system, (2) the speed of those CPUs, and (3) the character of the batch jobs being run. What we observed startled us:

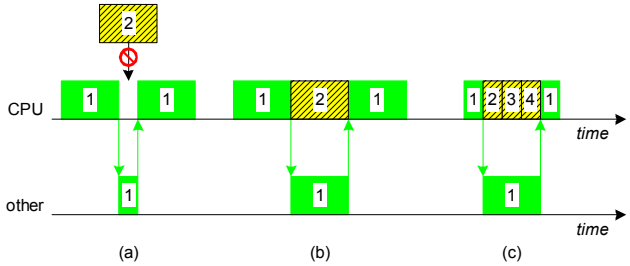
- The number of simultaneous batch jobs that maximizes Oracle Applications system throughput varies by the number of CPUs available to the applications, and it varies by the CPU intensity of the batch jobs being run. The number does *not* vary as a function of CPU speed.
- The optimal number of simultaneous jobs per CPU varies as a function of CPU service time for the job divided by total service time for the job. For CPU intensive batch jobs, the optimal ratio is near 1.0. For I/O intensive jobs, the optimal ratio is closer to 2.0.

What surprised us the most was that the CPU speed of an Oracle Applications system does not influence the number of batch jobs per CPU that you should attempt to execute in parallel. On a 16-CPU system, you should run no more than an absolute maximum of 32 simultaneous batch jobs. On a 1-CPU system, you should run no more than an absolute maximum of 2 simultaneous batch jobs, even if that CPU is incredibly fast. Startling.

Seeing the same batch-to-CPU ratios over and over again, with a clear upper bound of 2.0 sparked a curiosity, which, I am happy to report, we have been able to research since starting our own company in October 1999. From that research, we have generated a useful and reasonably amazing result:

> Define ρ (rho) as follows:
>
> $$\rho = \frac{\text{number of simultaneous batch jobs}}{\text{number of CPUs available for batch processing}}.$$
>
> If you can get better system throughput with $\rho > 2$ than you do with $\rho \leq 2$, then your system can be made to run faster *without requiring a CPU upgrade*.

There are only three relationships that the average CPU service time (S_{CPU}) and the average service time for everything else (S_{other}) can have: either (a) $S_{CPU} > S_{other}$, or (b) $S_{CPU} = S_{other}$, or (c) $S_{CPU} < S_{other}$. The picture below depicts these three possible relationships using our now-familiar notation.



In scenario (a), $S_{CPU} > S_{other}$. In this case there is insufficient idle CPU capacity to add a second simultaneous batch job without incurring CPU queueing delays to the first job. On a system with CPU intense batch jobs, the optimal ratio value is $\rho < 2$. If your batch jobs are CPU intense, then you should run *fewer than two batch jobs* per CPU available for batch processing.

In scenario (b), $S_{CPU} = S_{other}$. In this case there is just enough idle CPU capacity left over to insert a second simultaneous batch job without inducing excessive CPU queueing.

In scenario (c), $S_{CPU} < S_{other}$. In this case, a job requires so much service time from resources outside the CPU that there is plenty of idle CPU capacity left over to service other batch jobs' CPU requests. In this picture, it is actually

possible to put four simultaneous batch jobs onto the system without incurring average CPU queue lengths greater than one.

On first glance, scenario (c) might appear to be an optimal goal state. After all, shouldn't a system be more productive if you can run a larger number of things simultaneously? But scenario (c) is actually a picture of a system with a performance problem. In scenario (c), a batch job suffers from a bottleneck at one or more components outside of the CPU. Hence the performance of scenario (c) can be improved *without upgrading CPUs*.

This is an important result. CPU is the most expensive resource in a computing system. Not only are CPU upgrades more expensive than most other system component upgrades, but CPU is also one of the least scalable resources on a system.[5] Your hundredth disk drive adds almost exactly as much I/O call rate capacity as your first disk drive, but your hundredth CPU would bring no more computing power to your system than an extra SQL Server manual.[6] If you're already at the high end of your computer vendor's product line, then CPU is definitely your most expensive resource, because in order to upgrade, someone has to invent something.

Rather than rejoice in a fine-sounding batch-to-CPU ratio, the appropriate reaction to scenario (c) is to solve the problem that is prohibiting your batch jobs from using your available CPU power the way they should be. The answer is likely that there is a disk or network bottleneck, the repair of which will result in improved system throughput at reasonably low cost. Finding that two or more batch jobs can execute upon one CPU without driving it to 100% utilization is red flag that almost always means you can improve performance without upgrading CPUs.[7]

What Do I Type?

Now that we know that the magic ceiling is $\rho \leq 2$ jobs per CPU, how do we apply this knowledge to our batch queue management using the Concurrent Manager? A simple and effective algorithm for Oracle Applications batch queue management is as follows:

1. Determine how many CPUs you have available to the instance of Oracle Applications that you are analyzing. Call this number n. If you have more than one instance of Oracle Applications on your system, determine how many CPUs' worth of capacity you wish to give to each.
2. Determine how many CPUs' worth of system capacity you can dedicate to your batch workload. Call this number $(n - k)$, where k is the number of CPUs' worth of capacity that you dedicate to interactive workload.[8] Usually, you must consider your interactive users first and "reserve" as much CPU capacity as necessary to keep interactive response times stable. However, sometimes (e.g., month-end close), you may need to allocate more capacity for batch processing than your interactive users would normally tolerate.

---

[5] What, you might ask, is a less scalable resource than CPU? Serialized operations within the database such as latch, lock, or enqueue acquisition. The only way to make serialization operations more scalable is to eliminate them.

[6] …At least in year 2000 symmetric multiprocessing technology on the floor in most computing centers running Oracle Applications.

[7] There is one exception. If the sequence of resource requests labeled "other" is truly optimized, and the CPU service time is *still* less than the "other" service time, then the optimal value of ρ will be $\rho > 2$. I have never heard of this phenomenon happening in real life, but it happens regularly in competitive Oracle product benchmarks, which yield optimal ρ values of 5 to 10 or more. These benchmarks are so well optimized that many "other" service requests consist of a single-block I/O request (possibly using very small blocks to minimize data transfer time). In this case, the "other" service request cannot be further optimized; for example, it would be inefficient to stripe a single block of Oracle I/O across multiple disks.

Managers of these benchmarks also tune out as much CPU workload as possible, and they use systems with very fast CPUs. These two factors make it possible for the CPU service time to be less than the "other" service time, even when the "other" service time is optimized. The result is optimal ρ values that reach values well above $\rho = 2$. The chances of these circumstances being present in your real-life application are practically zero. Oracle batch jobs execute a lot of code path for every I/O call, which virtually assures the utility of '2' as your magic number for batch queue management.

[8] Of course, if $(n - k) < 0$, then you have a problem requiring better application optimization or capacity planning before investigating better batch queue management.

3. Configure your Concurrent Manager to run about $1.8(n - k)$ simultaneous batch jobs. That is, set the number of processes for each work shift so that the sum of processes across active work shifts at any given time is $1.8(n - k)$. For instructions on how to configure Concurrent Manager work shifts and the number of processes per work shift, see [Oracle (1998), 7-83].
4. If your CPU utilization is consistently well left of its critical range when there are jobs waiting in a batch queue, then consider adding a queue by incrementing the number of processes executed by one or more work shifts. If you can keep adding queues until your ratio ρ of batch jobs to available CPUs exceeds 2.0, then investigate why your system is bottlenecked on a resource other than CPU.
5. If you see either CPU utilization of 100% for extended durations or unacceptable online response time variances, then reduce the number of simultaneously active batch queues by decrementing the number of processes executed by one or more work shifts.

Note that your value of $(n - k)$ will change for different times of the day, week, or month. Customize work shifts to account for this. For example, on a 12-CPU system, you might require only 20% of capacity to keep your users happy on Mondays ($k = 0.20 \times 12 = 2.4$ CPUs), but you might require 35% of capacity to keep them happy on Fridays ($k = 0.35 \times 12 = 4.2$ CPUs). Then define work shifts so that $1.8(12 - 2.4) \approx 17$ batch processes are simultaneously active on Mondays, and $1.8(12 - 4.2) \approx 14$ batch processes are simultaneously active on Fridays.

Summary

My colleagues and I have made remarkable throughput and response time improvements at dozens of Oracle Application sites by imposing an appropriate batch queueing discipline via the Concurrent Manager. When you see a 16-CPU computer buckling under the weight of 96 concurrent jobs, reducing the number of batch queues can be a difficult message to sell. I hope that you can see why this difficult and often counterintuitive step is the right way to break the cycle of system performance pain.

- Restricting batch workload so that you *avoid excessive CPU queueing* makes the system produce results sooner for everyone, even the people whose jobs are required to wait their turn in the batch queue.
- With real-life Oracle Applications batch workloads, the ratio ρ of batch jobs to available CPUs that maximizes system throughput is $\rho \leq 2.0$, no matter what kind of system you have. Your optimal ratio is probably near $\rho = 1.8$.
- If your system throughput with $\rho > 2$ simultaneously executing batch jobs per CPU exceeds your system throughput with $\rho = 2$ simultaneously executing batch jobs per CPU, then investigate why your system is bottlenecked on a resource other than CPU.

Frequently Asked Questions About the Paper

More CPUs

Q: What if I buy a system with more CPUs? Shouldn't that raise the number of batch jobs that I can run simultaneously?

A: Yes, having more CPUs' worth of capacity will increase the total number of jobs you can run simultaneously. But the number of jobs *per CPU* that you allow to run simultaneously should still hover around $\rho = 1.8$, and the optimal ratio for you will not exceed two unless you are bottlenecked on a resource other than CPU.

Faster CPUs

Q: What if I use faster CPUs? Shouldn't that raise the number of batch jobs that I can run simultaneously?

A: The answer is no. Using faster CPUs will reduce a job's CPU service time, which might reduce the job's total response time.[9] If a job finishes more quickly, then the next job can begin sooner, and the system's throughput will exceed that of a similarly configured system with slower CPUs. But regardless of CPU speed, you should not attempt to execute more than two jobs per CPU in parallel.

Rounding Error

Q: Using your algorithm with $\rho = 1.8$, I computed that I should run 13.2 batch processes at one time. But I cannot enter "13.2" in the "processes" field of the Oracle Applications Work Shifts Window; I must specify an integer. How many Concurrent Manager "processes" should I configure? 13? Or 14?

A: The batch-to-CPU ratio $\rho = 1.8$ is a typical optimal value for a healthy Oracle Applications system. No matter which integer number of simultaneous processes you choose, examine your system performance and manage your batch accordingly. If online response times are rock-solid and your CPU utilization is consistently left of the critical zone for your system's number of CPUs, then consider adding another batch queue. If you have inconsistent online response times, or if your CPU utilization is consistently near the critical zone for your number of CPUs, then reduce your number of simultaneously active batch queues.

Low Batch-to-CPU Ratios

Q: My system's optimal ratio ρ seems to be much less than 2.0. Is this a problem?

A: A system's optimal ratio will be less than 2.0 if its batch jobs are CPU intensive. Oracle Applications Financial Statement Generator reports are in fact particularly CPU intensive, so running a lot of FSG reports will drive your batch-to-CPU ratio closer to $\rho = 1.0$. If you have any batch jobs that use parallelism within the job (e.g., if you use Oracle Parallel Query), then your optimal batch-to-CPU ratio can even drop to $\rho < 1.0$. The figure $\rho = 1.8$ that I've recommended in this paper accounts for a typical mixture of Oracle Applications jobs, including FSG reports.

References

N. Gunther. *The Practical Performance Analyst*. New York NY: McGraw-Hill, 1998.

R. Jain. *The Art of Computer Systems Performance Analysis.* New York NY: John Wiley, 1991.

D. Menascé, V. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Upper Saddle River NJ: Prentice Hall PTR, 1998.

C. Millsap. *Performance Management Myths & Facts*. Oracle Corp, 1999. Available at www.hotsos.com.

C. Millsap. *Queueing Theory Workbook*. Hotsos, 2000. Available at www.hotsos.com.

Oracle Corp. *Oracle® Applications System Administrator's Guide, Rel. 11*. Redwood Shores CA: Oracle Corp., 1998.

About the Presenter

Cary Millsap is the manager of Hotsos LLC, a company dedicated to improving the self-reliance of Oracle system performance managers worldwide through classroom education, information and software delivered via www.hotsos.com, and consulting services. Mr. Millsap served for ten years at Oracle Corporation as a leading system performance expert, where he founded and served as vice president of the System Performance Group. He has educated thousands of Oracle consultants, support analysts, developers, and customers in the optimal use of Oracle technology through commitment to writing, teaching, and speaking at public events. While at Oracle, Mr. Millsap improved system performance at over 100 customer sites, including several escalated situations at the direct request of the president of Oracle. He served on the Oracle Consulting global steering committee, where he

[9] Using a faster CPU will always mean reduced CPU service time for a given job. However, using a faster CPU will not necessarily result in reduced *response time* for a job. A faster CPU will result in reduced response time only if the job has a CPU bottleneck.

was responsible for service deployment decisions worldwide. He is the inventor of the OFA Standard, creator of the original APS toolkit, a Hotsos Tools designer and developer, editor of *hotsos.com*, and the creator of Hotsos Clinic.