Jon's Performance Musings: Hanging by a Thread

Welcome to my first column on various performance/capacity issues affecting our lives. With over 40 years in the industry (yup, I'm one of those greybeards) and 15+ years on NonStop I have some tales to tell and some wisdom to share (you can be the judge of that). I welcome feedback and questions. Reflecting what many of you are experiencing in your professional lives, my consulting practice covers multiple architectures, including NonStop, Windows, VOS, and Unix. My job, and I'm good at it, is helping clients anticipate and prevent slowdowns and outages. I work with clients where there are literally thousands of dollars of transactions each second at stake.

I'll try to give you some of the benefit of my experience. I hope that you'll take away at least one tidbit to think about from each column.

The NonStop architecture has always demanded an awareness of processing threads. The loosely-coupled multiprocessor nature of the NonStop forces the application designer to think about parallelism. The floor is littered with large, monolithic applications that, when ported to the NonStop, didn't take advantage of the parallel nature of the architecture and failed to perform. Those apps had to be broken into smaller pieces that could be replicated and distributed across the resources available. Once that was done there was much better hope that the application would be well matched to the architecture and perform well.

Now with the J-series which incorporates multiple cores into each logical processor blade, the necessity of designing for parallelism becomes even more important. Within each blade one finds symmetrical (shared memory) multiprocessing, as well as the traditional asymmetric software-based fault tolerance.

Even with well-designed applications there are occasionally situations where avoidable bottlenecks occur. The basic question is: "How many instances of process X or connection Y do I need to support the traffic I'm expecting?"

I like to use an analog to the highway system. The difference between performance and capacity:

> *Performance* is designing a highway, curvature, elevation, grade separation, signage, etc. so that cars can drive it at 60 mph.
>
> *Capacity* is designing that same highway so cars can drive it at 60 mph during rush hour.

Obviously, you need more lanes to handle the traffic. So, too, do you need more server processes, logical connections, etc. to handle the expected workload. But how many?

The issue is not strictly throughput. Throughput is a factor, of course. And you must design for the *anticipated* peak demand, not the average demand. As a friend said, "Never design a bridge for the average height of a sailboat's mast. That guarantees problems."

When you know the throughput target, the real issue is concurrency. At 100 tps, how many transactions will be "in flight" within the system at any given moment, and does the system have the resources to

handle them? The in-flight (concurrent) number is the product of the throughput and the response time. So, if your target is 100 tps, and the average response is 0.5 seconds, you can expect (and must design for) 50 concurrent transactions.

But wait! I said never design for the average. If you are going to design properly, design for the worst case response, so that if your disks or back-end authorizer slow down, you will still be able to process the transactions, just more slowly. If your experience shows that, for example, 4 seconds is still a reasonable response, and you've seen that from the back-end at busy times, then you need to design for 400 concurrent transactions.

> To some of you this may be an obvious discussion, but I can assure you that this is a real issue at some sites. We're successfully completing a project with a Southern bank where the bottleneck was the capacity, both bandwidth and concurrent LU2 sessions, of an SNA link to their host.
>
> *Remember legacy comm, like SNA or X.25? I realized that there was a generation gap when I was talking to a younger gentleman at the HPTF conference and he asked "What's SNA?"*

Designing for concurrency is a mandatory part of configuring the system. Making sure that there are enough server processes and connections to handle the anticipated peak is absolutely key. (For more about this issue, refer to my article "At The Mercy Of The Arrival Rate" http://www.banbottlenecks.com/mercy.html .)

One more, frequently-overlooked issue re parallelism: Batch!

Unfortunately, batch programs tend to be monolithic. They tend to be single-process program sets which do heavy data manipulation in a sequential manner. As such, they don't take full advantage of the NonStop architecture. Sometimes, they are in danger of breaking their "window," i.e. running longer than they should and missing deadlines for completion.

So take the time to examine the batch stream and redesign portions of it so that the inherent concurrency in NonStop can be used to advantage. Look for programs in the batch sequence that can be run at the same time as other steps.

If you're using indexed and partitioned files, or SQL, examine the batch process to see if it can be broken apart and executed in parallel based on a key range. We've seen dramatic improvements in execution time with some judicious redesign of the basic batch stream.

The NonStop is an industry leader in high throughput and fault-tolerance. Its architecture is perfect for parallel OLTP processing. When the application is configured to take advantage of the architecture, the sky's the limit for scalability and throughput.

Jon E. Schmidt
President, Transaction Design, Inc.
542 San Pedro Cv
San Rafael, CA, 94901, USA
1.415.256.8369

jon@banbottlenecks.com

Jon is the founder of Transaction Design, Inc., a consulting firm located in the San Francisco Area which specializes in performance and capacity studies with clients worldwide. He is the creator of the Ban Bottlenecks® service and has an extensive background in the implementation, testing, and tuning of high-availability systems.