Jon's Performance Musings: Your 2011 Capacity Plan

Jon E. Schmidt
Transaction Design, Inc.
San Rafael, CA, 94901, USA
1.415.256.8369
inform@banbottlenecks.com

Jon is the founder of Transaction Design, Inc. (TDI), 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.

Picky, Picky, Picky

…Is a good thing. Otherwise known as attention to detail, the focus on the small items will pay off in the long run. We all want our systems to be stable, predictable, boring even. I'm happiest when I can tell a client that their systems are boring, that there were no surprises in the last month, and that they have comfortable margins of capacity in all the categories for the foreseeable future.

Boring is Achievable and Desirable

Only a few among us enjoy getting alerts. Adrenaline junkies may like to be constantly pinged so that they can charge in on their white horse and fix a problem, but that isn't the way a professional should work. I've even known certain types to set up artificially low alert thresholds so that they can do their "white knight" stuff.

If you're going to do it right, you need to anticipate and prevent issues, not react to them. Yes, it takes work. You've heard me say that it's not rocket science, just due diligence. It takes a constant effort to find defects and deviations, and correct them. It takes dedication from the whole team, and from management. If you are successful, you and your clients will win. Your service levels will climb to an "excellent" rating. And your stress level will fall.

Heinrich, Again

In our childhood we played with wooden blocks. We piled them on top of each other, eventually reaching a height where the column was unstable and fell. Each block was small, but as each block was added to the top of the column, it added to the instability of the column.

So too do small problems add up in our systems. The Heinrich Ratio[1] makes a strong case for a program of constant improvement. The alternative is an accumulation of errors which will result in a catastrophic failure, eventually. Our job is to find and fix all of the small errors that creep into our systems over time. Each new feature, report, or other enhancement has the ability to complicate and de-stabilize the overall environment. OS and utility software releases can add to the problem. Changes in staff don't help either.

What's Normal?

To be able to tell what's wrong, you need to know what's normal. How should the application work? What background jobs are necessary, how long do they usually take to run, and what resources do they use? What time are backups run and how long do they take? Extracts? FTP's?

When are the periods of peak online demand? Does peak demand move from month to month, season to season? (I can guarantee you it does.) What resources does the application use during peak? Are there constraints? (I can guarantee you that there are.) When does the application traffic begin to approach those constraints?

What are your service levels? What kind of response are you providing your clients? Is it the same throughout the day, the weekend, and the month? What are the transaction success/denial rates? Reversal rates? What kind of response are you receiving from your back-ends, partners, or SOA services? Is it stable?

What's Rational?

When you get to this level of understanding an application, good things can happen. We recommend what we call "rationality checks" as we look at a system. For example, we examined the transaction rates and queue traffic of an OLTP application. One of the things we noticed was a strange multiplier in the queue traffic. For a rate of 10 transactions per second, we noticed that one server class was handling over 100 messages per second. We questioned it, and yes: "Bug."

Another example was a batch job that seemed to take too long for what it was supposed to do. Looking at the file data, we discovered that the job apparently was reading over a million records from a file that only had 100K records in it. "Bug."

And the Irrational

It's those unexpected blips on the chart that must be investigated. Maybe page faults shot up for a moment. Or free memory dropped. Or one of the CPU cores pegged. Or there was a burst of event messages...

These blips are real, caused by something, and will probably come back if not fixed. That's the part we want to prevent. If they come back during the online traffic peak, they can hurt.

1 http://en.wikipedia.org/wiki/Herbert William Heinrich

To Begin

I've actually been told, "No, we don't want to spend any time on that issue, because it happens too often." D'oh! Computer systems are deterministic. They only do what someone has told them to do. In the case of errors, that "someone" might be an OS designer, an untrained operator, a developer, or similar. In any case, if uncorrected, it will repeat.

Remember Finagle's Law of Dynamic Negatives: If something can go wrong, it will go wrong, and at the worst possible moment.

So, learn your system. Take a close look at the time of peak online demand for the recent period. I suggest looking at half-hour period. Look at everything. Look at CPU, cores, memory, paging, processes, disks, communications, queue traffic, events, and legacy comm. Look at the application traffic and service levels. What are the message rates? Do they make sense? Are there transient processes? Why? Are there background jobs running at this time? Why? Can they be moved? How much are they conflicting with the online system? Are they using the same disks? Is their priority properly lower than the online system? How's the balance of the system? Where are the hot spots? Cores? Disks?

Taking It Seriously

Effective performance/capacity management is a discipline. It's also a team effort. In most shops you have multiple departments, each responsible for some aspect of the processing environment. When you find a problem, even though it's a transient problem, institute formal procedures for its investigation and resolution. Assign tasks; follow-up; fix it. Otherwise it will come back to bite.

As part of that discipline, start collecting data for each month's application traffic peak half-hour. You will want to have a history of these peak analyses so you can tell when things change, and whether they changed for the better or for the worse.

The Numbers

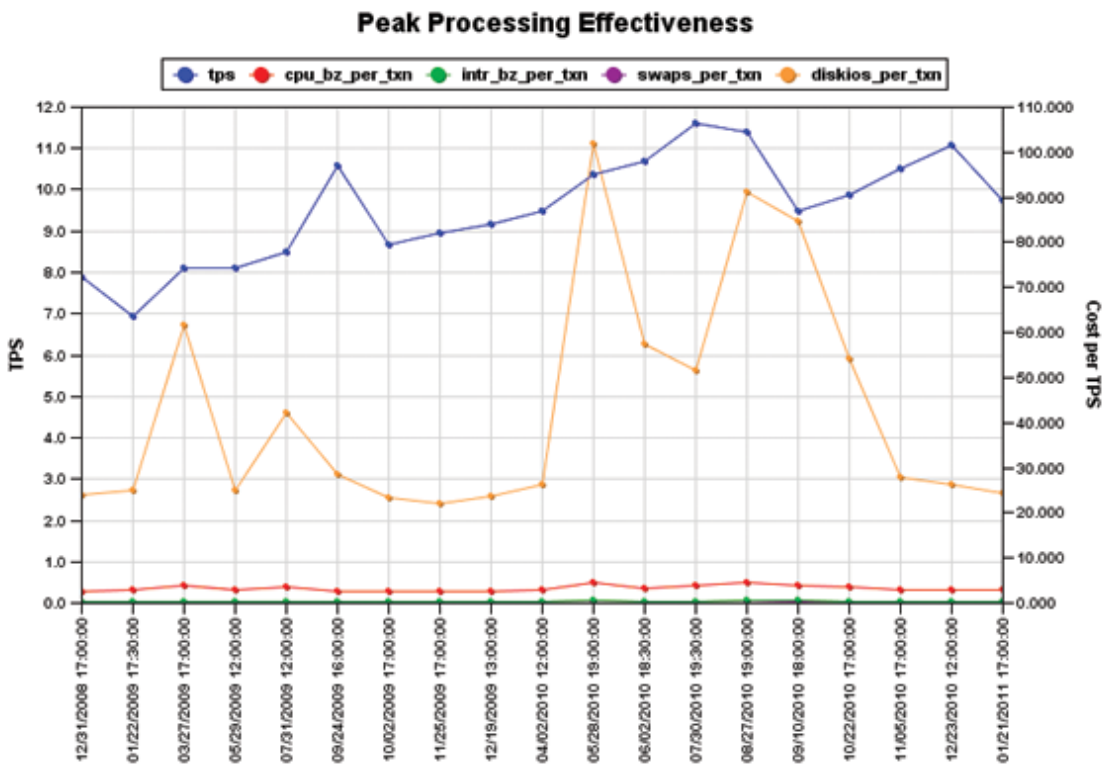
Here's what I recommend that you start collecting and tracking for each month's peak:

Transaction counts:

Very, very basic requirement. You should also track response times, response distributions, success ratios, and back-end response if available. If you don't have a way to count actual business transactions, count something you know will be directly proportional to the transactions, such as queue messages or IP packets.

CPU and Core Utilization

Obvious. What's not so obvious is a) computing and tracking CPU per transaction or Core milliseconds per transaction, and b) CPU/Core balancing. The workload should be balanced equally across the CPUs/Cores. Over time one would hope that the CPU usage per transaction will remain steady. Most likely it will not, as features are added and background jobs compete. If your tools provide you with workload statistics, use those in addition, but the



gross numbers will flag you to competing processes and unexpected additions to the workload.

Disk Reads and Writes

The computation of I/Os and reads/writes per transaction will help to detect utilization per transaction (UPT) creep over time, and/or competition from other processes.

Swaps

There is no such thing as a good page fault. A well-designed transaction system should have no transients, and no page faults. Tracking swaps during peak will let you detect transient processes, low memory, and bad design.

Free Memory

There is no substitute for real memory. Memory imbalance, low memory with swapping, and memory leaks all need to be detected before they cause problems.

Disk Drive Activity

Here's where you'll find most of the bottlenecks and slowdowns. Disk busy, disk queue depth, cache size, reads and writes should be tracked. Hot spots must be identified before they slow down processing.

IP Traffic

Look here for competing FTP jobs that are sucking up the backbone and perhaps competing for disk I/O. FTP should be moved away from the peak online time.

The chart above shows the application peak half-hour over the last 18 months of a system.

Legend:

tps:	Transactions per second for the node for the half-hour (left scale)
cpu_bz_per_txn:	CPU busy percent for the node divided by the transaction rate
intr_bz_per_txn:	Interrupt busy percent for the node divided by the transaction rate
cpu_bz_per_txn:	Swaps for the node divided by the transaction rate
cpu_bz_per_txn:	Disk I/Os for the node divided by the transaction rate

But I Only Changed… And It Shouldn't Have Affected…

A regional bank decided to synchronize their current activity file between their north and south data centers in near time. They wrote the code, tested it, and when they placed it into production the systems slowed unacceptably. Since we had a baseline going back several months, we were able to quickly show them what had changed and its impact. A single write to the local CAF file had become a write to the local CAF, a write to the queue file, a read from the queue file, a transmit, and a write at the remote CAF. Application features will creep. As performance/capacity engineers we need to understand and plan for their impact.