Increasing Performance

Alleviating the Pain

Every now and then I hear the comment "We don't have any performance problems." But I always have to add the response "Yet." I'm always happy to hear that a shop is doing well. But I'm a pessimist (realist) at heart. I'm also a perfectionist. I have yet to see a site or application where things couldn't be done better. And my long experience has proved that these laws apply:

> *Parkinson's Law: Work expands so as to fill the time available for its completion.*
> *Parkinson's Law Corollary: Data expands to fill the space available for storage.*
> *Parkinson's Second Law: Expenditures rise to meet income.*

In the best of all possible worlds we would have infinite money and time to throw at improvements. You can fix a lot of issues by "throwing hardware at it." Experience has shown that a hardware upgrade is usually effective, but in some instances it can be wasted money. A CPU upgrade will not help if the application is disk bound. A disk upgrade will not help if the application is memory bound. And so on.

Every application will benefit when its development team, building on the knowledge gained from the initial project, designs and implements "Version 2.0". Unfortunately, most applications never get to that stage. V1.0 may go through incremental improvements, but the big redesign doesn't happen for time and cost reasons until the next generation architecture arrives. Hopefully the original team is still around and has become knowledgeable enough to take advantage of the new architecture and to build on their previous experience with the application.

Where's the Pain?

If I'm going to commit resources, where do I commit them, and what kind of resources do I commit? We all support multiple applications across multiple architectures, and our CST ("copious spare time") is non-existent, so there has to be a good reason to start a project to fix things.

Where's the pain? Chances are you know. Your help desk has told you that they get complaints. The batch window is broken every first of the month. Transactions slow down around 5 o'clock every Friday. Your job is to prioritize the pain, and then to alleviate the most painful spots first.

There are lots of common things that you can check and adjust without starting a mega-project. Here are some things to look for.

Saturation

CPU technology continues to evolve. One of the ways the technology has evolved is by implementing CPUs which use multiple cores and processing threads, instead of ever-faster single-threaded processors. Unfortunately, we can no longer rely on the single "cpu busy" metric for our processors to determine if there is trouble. We now have to look at each processor thread's utilization, searching for a pegged process. Any processor thread that is

consistently above 90% busy deserves a second look to see what's pushing it, even though the whole CPU may be 25% busy (for a quad-core example). Usually it means that a process needs to be replicated or a job needs to be partitioned.

All Things In Their Time

It's rare that an application owns exclusively all the resources that it needs. Sharing CPU, disk, memory, network, etc. can create slowdowns.

Online applications usually have batch processes using the same data. Batch is intended to use all the resources available to it, but this frequently slows down online. The question is whether the batch processes are (or can be) scheduled with minimal impact to the peak processing times of the online system. Review your batch environment, looking at a) time of day schedule and b) process priority. One of the easiest wins I had at an account was to convince them to ask their bank if they could move the daily extract from noon to 3pm.

Batch Thrash

While we're talking about batch, take a look at your overall batch schedule. I have seen too many sites where an entire suite of batch jobs (including backups) are fired off simultaneously at midnight or noon or whenever. Having all these jobs start at the same time can put tremendous pressure on CPU, memory, and disk unnecessarily. That conflict can hurt the online system badly. See if it doesn't make better sense to schedule some jobs sequentially, and certainly spread them across processors and disks.

Throw Cache (Not Cash) At It

As CPUs get more powerful the bottleneck moves. Usually to the disk farm. With the new CPU technology there is usually a corresponding increase in available memory. Systems go from 4GB per processor to 20+GB or more. Use performance tools to see how busy the disks are, and then add cache to each disk to reduce physical disk accesses. We prefer to see our clients' disks in the high 90 percentile hit ratio, ideally above 95%. 80,000 or more blocks of cache memory per disk are not unusual.

This works if the disks are read-intensive. However, it doesn't work if the reads are to a very large file being processed sequentially, perhaps multiple times. If the disks are write-intensive you may need to move files.

Spread Writes

Disks are mechanical beasts. They have rotating platters and moving heads. They also have lots of local cache in the drive electronics, in the SAN (if you are using one), and more cache in the disk process. That doesn't help if your application is writing gigabytes of data. If it's writing fast enough, all levels of cache will fill up and you *will* be waiting for the physical components to do their job. While modern disks have lots of storage capacity, major transaction systems need multiple spindles and high rotational speed to have enough moving parts to keep up with the required throughput.

If you have a disk that is too busy and has a lousy cache hit ratio as a result of a high write rate, use performance tools to determine which files or indexes are the most write-active, and look for someplace else to put them. Partition the data and indexes and spread them across drives and CPUs.

Also pay attention to the file and disk options that the OS may give you. For the NonStop look at Buffered, Auditcompression, Refresh, Serial Writes, Verified Writes, Icomp/Dcomp, etc. These all affect performance, and must be used intelligently.

Review Indexes

I once went to a software house to help them with a performance problem. When I dug into the application, I discovered that the principal table, which was updated constantly by the online application, had 14 indexes. Fourteen! It seemed that their design discipline was non-existent. Any programmer who needed to write a report added an index to the file so that they didn't need to sort the file. Ouch! A single insert resulted in 15 physical writes to the disk (the file was transaction-protected).

A write or update intensive application will suffer proportionally to the number of indexes. Keep in mind that any write, insert, or update will result in the data *and* the indexes being updated. If a commit is involved, all of those updates will have to wait for the safe-store.

I recommend a "Is this index necessary?" review of your designs. Also, I definitely recommend using a large blocksize for the index blocks when there is a choice.

Speaking of Commits

In both batch and online applications, think about when a commit must be issued. A commit per transaction is traditional and in many cases necessary. But not always. A server-based transaction stream could be committed every 10 transactions, or every 0.1 seconds, whichever is sooner. This could cut down significantly on the safe-store activity to the disks.

Rationality

While you're looking at file usage, this is a good time to do a rationality check. I know that your software doesn't have any bugs, but it may have a couple of "unique features." Compare the transactions per second to the messages per second through the application and disk processes. See if it's what you expect. Is the ratio of TPS client-to-server calls one to one, or something greater?

For batch jobs, compare the number of reads and writes to each file or table to the number of records or rows in the input file being processed. Do the numbers add up? Also, are the numbers of reads to a particular file or table far greater than the number of records in the table? To say it a different way, does it appear that the file/table is being traversed several times in one processing cycle?

And in terms of really obvious checks, verify how many times a file or index is opened by an application. It should be once per application start. Rarely, but it happens, poor application

code will open and close a file per transaction. This is especially the case if a transaction causes a process start, which then does its own open/close. Bring a service up once, and then leave it up!

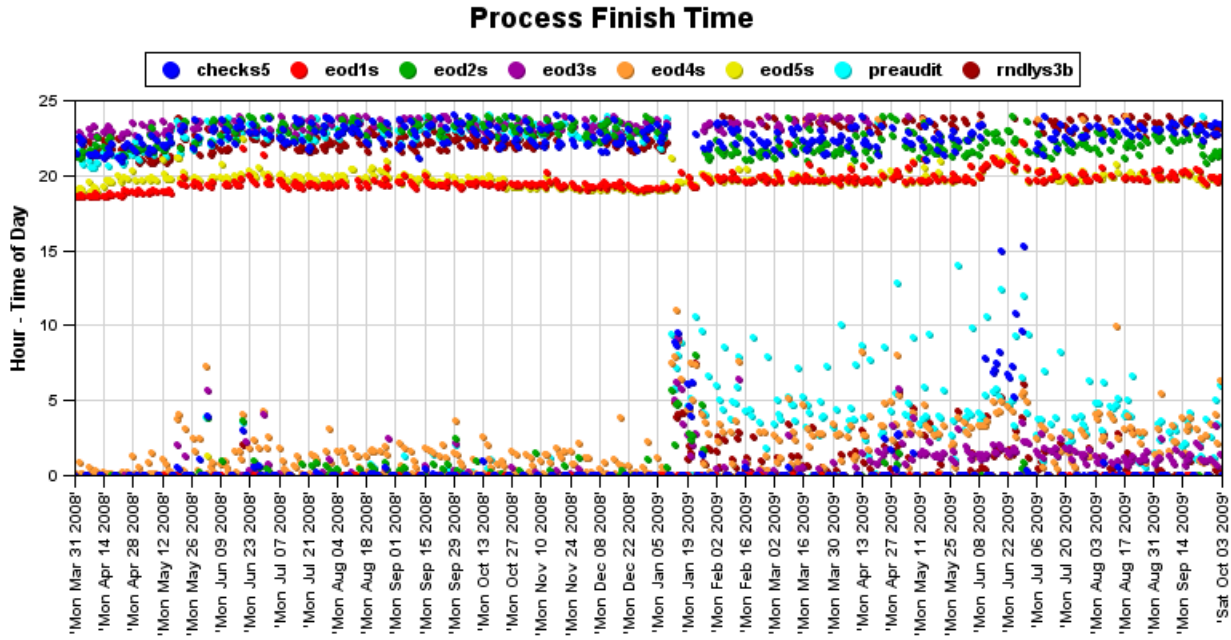
SANity

Storage Area Networks are shared resources. That's the good news and the bad news. Always look for contention within the SAN. Make sure that your transaction files are on dedicated drives, ideally connected by a dedicated path through the fabric. Watch for slowdowns: I/Os taking longer to complete, even when the I/O rate is the same.

One major performance advantage that SANs have is data striping: One logical disk may span multiple physical drives such that a single file can spread across multiple sets of spindles and heads. So a "hot" drive can be cooled down by adding mechanicals to it.

The Net

One must realize that the network may be very high speed, but it's still one bit at a time. Messages are broken up into packets, and they take time to arrive at their destination. Lots of things can go wrong in the mean time. Over the course of a relatively short time, conditions can change dramatically. Contention and re-routing are constantly affecting the round-trip time (RTT). Take a look at the *Jon's Performance Musings* column in this issue for more discussion of networking.



Increasing Performance

Application performance is limited by the physical characteristics of the hardware its software and data occupies. However, the proper design and good tuning can make a significant difference.

Jon E. Schmidt
Janis K. Kuritsubo
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 capacity/performance 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. Janis K. Kuritsubo is the Chief Technical Officer for Transaction Design.