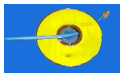

TMurgent Technologies

White Paper
Perceived Performance
Tuning a system for what really matters

September 18, 2003

Introduction

The purpose of this paper is to explain a subtle difference between tuning a computer for maximum computational performance, and tuning it for what we call "Perceived Performance". *Perceived Performance* is about tuning so that the average user on a multi-user system perceives that the system is operating faster for them.

This is not a paper about specific tricks and techniques to optimize your system. It is a paper to explain our concept of, and importance of, Perceived Performance. In working with IT professionals, we have found the concept to be ill understood. Often when dealing with system performance, they focus on things they have read and things they have heard in the past. It is rare that an IT professional actually has the time (or budget) to become truly trained in performance tuning. The techniques and objectives that they have picked up along the way may or may not be appropriate for what they are trying to accomplish. While advances in Operating Systems and advances in Hardware account for much of this misguided efforts, the different challenges associated with Terminal Services (and the tendency for otherwise excellent sources to provide only passing reference to the difference when operating in this environment) require a different mind-set, and a somewhat different set of goals.

Hopefully we can explain this concept sufficiently here. The paper does not represent an end goal, but a part of the education process to understanding how to tune your system. The IT professional need also look to more practical texts, which help you to organize your efforts and describe tools you can use to measure cause and effect. In addition there are countless website containing tricks and tips. With an understanding of *Perceived Performance*, you should be able better understand why and when specific techniques are appropriate for your Terminal Server environment.

Computational Performance

Computational Performance is the term we refer to in order to describe a methodology where one analyzes the system with a goal of improving the peak capacity by eliminating unnecessary actions that reduce the overall computational capacity of the system.

Computational capacity is the total amount of useful work that can be accomplished on a system in a given fixed period of time. It is spreadsheets re-calculated, and reports generated, and print jobs printed, and users rotating a 3d-image (or just playing solitaire if the system isn't locked down).

In focusing on computational performance, one analyzes where CPU cycles are spent and where more (or occasionally less) hardware or software resources provided to the system can increase the computation capacity.

Most reference books on performance use computation performance as the methodology to help you organize your performance related efforts. These books, the methodologies they teach, the tools they train you to use, and, the tricks they teach, are excellent guides and are of immense use to you.

A simple example might be as follows. The subject: memory. The tools: Use the performance monitor to look at the memory utilization and paging activity on a system. The solution: add more physical ram to the system. The result: less paging activity, which frees up the system to perform more useful work, rather than wasting all that effort (CPU Cycles, Bus, and Disk Activity) just sliding things in and out of physical memory.

Computation Performance is an important technique you will use to optimize your systems. However, if you are optimizing a multi-user system such as a Terminal Server, then you need also consider (and temper your solutions with) *Perceived Performance* thinking.

Perceived Performance

Perceived Performance is the term we use to describe a methodology where one analyzes the system with a goal of improving user productivity by focusing on issues that affect the performance of the system as perceived by the users using it.

Few IT professionals run their server farms at 100% CPU loading (and none should). Typically, the farms are set up with a goal of an average loading as a much lower value. Maybe it is 60%, maybe it is 30%. Much depends upon the environment and budget.

Quite often, the reality is that IT starts out planning and deploying to such a figure, however more servers are added not because of exceeding a magic number (although the number may be used in purchase justification) but because the users are complaining about how slow it is to do their job.

A focus that uses perceived performance achieves optimal user productivity. While accomplished using primarily the tools and techniques of computational performance, the thought process and analysis under perceived performance is geared toward a different objectives to reach a different goal. **Table 1**, below, summarizes the key difference in goals and objectives of these two methodologies.

	Computational Performance	**Perceived Performance**
Goal	Improve peak capacity.	Improve user productivity
Objectives	Eliminate unnecessary actions that reduce the overall computational capacity of the system.	Modifications aimed at reducing delays in responsiveness to user actions.

Table 1 - Goals and Objectives of Performance Methodologies

Example of Perceived Performance

One example we often run into with IT Professionals is what we describe as being overly concerned with context switching.

Context Switching is the overhead that occurs when the operating systems switches between different tasks. (See our white paper "*Scheduling Priorities: Everything you never wanted to know about OS Multi-tasking"[*Ref 1*]* for a simple description of how the OS multi-tasks). Each time a CPU switches from working on one thread to another the CPU must save information necessary so that it can later resume processing of that thread exactly where it left off. Typical information includes the contents of CPU registers, including instruction address and stack pointers.

Computational Performance dictates that by reducing the number of context switches, you reduce the overhead associated with the switching, and thus increase the overall computational capacity of the system. Context switching overhead discussions have a deep history. In years gone by, a class of operating systems known as "Real-time Operating Systems" was rated almost entirely by their context switch time. Today IT professionals attempting to tune their system often use the Performance Monitor to see the number of context switches per second appearing on their server as a measure of system performance.

Perceived Performance thinking is not concerned with "reasonable" levels of context switching. Why? Because the ratio of CPU cycles per Context Switch cycles have been reduced significantly on our systems over the years.

In part, OS/compiler vendors have learned how to produce an absolute minimum context switch time. In part, chip manufacturers like Intel have made the processors more capable as well. And finally, the constant doubling of processor speeds have massively increased the number of CPU instructions that take place in a given period of time while the average thread run time (the average clock time that a thread runs without being interrupted) is going down.

In [Ref 2], Dr Bradford shows a method that attempts to measure the context switch time on a Windows 2000 Server. In this article he indicates that it runs about 1.8 usecs on a 650 Mhz processor. (NOTE: a usec is 1/1000 of a msec, or one millionth of a second). I analyzed his test and made a few minor changes, both to reduce potential unnecessary paging overhead and to account for start/end thread swaps that were not accounted for in the calculations. We tested this on different speed systems (ranging from 666Mhz to 2.4Ghz) and both 2000 and 2003 server OSs, and generally came up with figures in the 1.6 usecs/context_switch range[1].

So the bottom line is that if a change is made to a modern server that even doubled the number of context switches per second you do reduce the computational capacity by a small amount. For example, by lowering the maximum run time of a thread (under Microsoft Windows this is referred to as a number of "quantum") you potentially increase the number of context switches that would added in the case of a CPU bound thread that would normally run for the full quantum limit. For example, moving the quantum value from 36 (roughly 180ms on a multi-processor) to 6 (30ms)[2] in the presence of a CPU-bound thread would add a context switch overhead of 0.005% (see Equation 1).

$$(6cs * 1.6u\,\text{sec}/cs)/180m\,\text{sec} = 0.005333\%$$

Equation 1 - Added context switch overhead (worst case scenario)

[1] This value still contains much more than the context switch time because it captures other OS activities including interrupts, APCs and DPCs, but it is a "good enough number". The actual value would be lower.

[2] Changing from "Background Services" to "Application" mode.

However we run our servers with large amounts of spare capacity. 60% loading plus 0.005% loading still looks like 60% loading. The flip side of reducing the quantum (which increased the overhead ever so slightly) is that responsiveness is improved. Instead of Task-A waiting up to 180ms for a competing Task-B to complete its quanta before Task-A can begin its work, it waits for 30 to 60ms.

Can the user tell the difference between a 60 and a 180ms delay? Yes. And the reason the answer is yes is because Task-A processing will likely swap out multiple times to complete the work of a very simple request. **The responsiveness the user feels is the result of multiple small delays that add up!** A blink of an eye is about 100ms. When the user clicks on a button and has to wait 500ms to see the menu appear they feel it. After 750ms they might click on it again, thinking they missed the target they tried to click.

Perceived Performance is about tuning the system to reduce the delays that decrease responsiveness to user actions. And the effect of responsiveness can be measured. Either empirically (how many users can you log onto the system before they complain), or experimentally (measuring responsiveness to a particular action).

We often measure experimentally using the following test. We have a small Win32 based GUI executable which we use. The test consists of a small cmd file that repeatedly measures the amount of time to launch the executable and have it shut down. This involves the system allocating a new process, the loader loading up the executable from the disk, a few threads are spawned and a window is presented. Then the GUI sleeps (waits on a timer) for a short period and terminates itself. Even on an idle system this process will cause hundreds of context switches. We know, from running on an idle system for thousands of passes, what the best possible time to complete this process is. We run this test under a given load with certain system parameters and determine an average completion time. From this value we subtract the minimum time and we have a measure of the current responsiveness of the system. The lower this value is (the closer to the minimum time) the more responsive the system is. Change system parameters and run the same test. Did responsiveness improve or get worse?

Such a test allows us to accurately measure whether a change to the system improves perceived performance or not. As is often the case when it comes to performance, the logical result may not be what you find if you actually run the test.

Ultimately, however, it is the empirical test that proves whether the change is important. Does this change allow more users to share this system? Changes that individually (or as a group) do not increase the user capacity of the system, even if the experimental tests show some improvement are in the long run probably a bad idea. They make the system more complex to set up and more complex to maintain. The problem is in devising an empirical test that can be measured.

We can devise a measurable empirical test using the experimental test we described above. In the experimental test, we run our cmd file on a system with different number of users, until the responsiveness exceeds some magic value.

What is the magic value? You need to determine that for your situation. Fortunately, this determination need only be made once. To determine, you start by using a given system configuration. You need only add more users until you subjectively determine that the performance is no longer acceptable in actual tasks. Once you find the maximum number of users, run the responsiveness test with that many users running and you have your magic value.

Now you can make system changes and measure the number of users that can run until the magic value is exceeded.

Conclusion

In this paper, we make a distinction between two methodologies to improving system performance of Terminal Server systems. In doing so, we by no means intend to imply that Computation Performance is bad. Our purpose is to guide the reader to use their head for what is important to their system, rather than what was in the head of the author who is tuning a very different system without Terminal Services.

The two remaining references in the paper are well worth the effort to get to know if you are performance tuning a Terminal Server.

- The O'Reilly "Stickleback fish" book "*Windows 2000 Performance Guide" [*Ref 3*]* is a thorough guide to general performance tuning of a Windows 2000 computer, whether desktop or server.
- Brian Madden's White Paper [Ref 4] is a refreshing look at specifically tuning a Terminal Server.

References and useful links

The following are useful references, some of which were used in the development of this White Paper. Often, additional useful links may be found in these references.

Ref 1 ***Scheduling Priorities: Everything you never wanted to know about OS Multi-tasking*****,** TMurgent Technologies, July 2003, **http://www.tmurgent.com/SchedulingWP.htm**

Ref 2 ***Context Switching Part 1: High performance programming techniques on Linux and Windows*****,** Bradford IBM DeveloperWorks, July 2002, ***http://www-106.ibm.com/developerworks/linux/library/l-rt9/?t=gr,lnxw02=RTCS***

Ref 3 ***Windows 2000 Performance Guide*****,** Friedman & Pentakalos, O'Reilly & Associates 2002.

Ref 4 ***Terminal Server Performance Tuning*****,** Madden, September 2003, **http://www.brianmadden.com/papers**

