



TMurgent Technologies, LLP

Your Destination for Application Virtualization

TMurgent Technologies

White Paper

Perceived Performance

Tuning our systems for what really matters

Fourth Revision
May, 2011

Preface

This is the fourth revision of this paper. In the original paper, published some 8 years ago, I coined the term *Perceived Performance* in an attempt to differentiate how I looked at system performance from how I saw most professionals act when working on performance issues. The concept of starting the performance measurement process by looking at the end-to-end performance results was natural to me from the years of experience I had working in the communications industry prior to returning to my roots in the systems space. In dealing with WAN issues, it was always best to start with end-to-end performance measurements, and to then break things down into isolated subset measurements. Often, when there was an issue detected end-to-end, we needed to isolate out parts of the internetwork for further testing, for example isolating the LAN from the WAN. It was from that background that I began looking at the end-to-end performance of terminal services from the end-user's point of view. The intended point of the first version of the paper was simply no more than that we need to start by looking at the end-user's view before diving into the system component details. Perceived Performance was introduced as a methodology for use to look at complex system performance.

But, as sometimes happens, a second point also appeared in that first paper; one needed amplification and shortly after the original paper I produced an update that eventually somewhat overshadowed the original intent. In measuring performance of Terminal Servers from the end-user point of view I found that the obvious value to measure, delays that occur between a common user generated action and the visible response by the system, was more complicated than we might think. In particular, the paper pointed out, the variation in delays had a greater impact on the user's perception of performance than the actual average delay. This second version of the white paper coincided with a toolkit I produced that was available as a free download to generate tests and display results on your own.

In the third version of the paper, updated in 2007, we were seeing the emergence of server virtualization. At the point of writing the paper, the consensus was that virtualizing back end server loads was a good thing to do, but virtualizing terminal servers was not. This consensus was not based on exhaustive analysis, but on some very good testing done by a small number of experts, such as my friend Ron Oglesby. The second paper also introduced a visualization of the perceived performance measurements. This visualization, which I called the "Perceived Performance Profile", allowed a human to view the effect of varying system performance over a range of changed conditions, such as adding additional load to the system(s).

It is now 2011. The server virtualization platforms, as well as underlying hardware in general, has vastly improved over the last 6 years. Virtualizing terminal servers over a server virtualization hypervisor has become common place for any new deployments – without harming the user. The "new thing" is now using the server virtualization hypervisor to host virtual desktop operating systems. While we can casually call these virtual desktops nothing more than "single user terminal servers", the fact that there are now so many VMs on the hypervisor, plus the usual externalization of the disk subsystems make these deployments very different indeed. In this version of the paper, after reviewing the original works, we desire to look at performance measurement for these virtual desktop loads. We also need to look at how other changes, especially due to the mass adoption of server virtualization, in our systems and system architectures affect how we need to look at performance. Finally, the paper will foreshadow what will probably be a fifth version of the paper at some point in the future – looking at Perceived Performance in Cloud Computing.

Introduction

The purpose of this paper is to update our definition of the 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 better 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 poorly understood in detail. But this is a far cry better than in 2003 when the concept was not understood at all!

It is rare that an IT professional actually has the time (or budget) to become truly trained in performance tuning. Often when dealing with system performance, the IT Professional tends to focus on things they have read and things they have heard from their peers in the past. 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, and sometimes end up being incorrect due to miscommunication along the way. While advances in Operating Systems and advances in Hardware account for much of this misguided efforts, the different challenges associated with virtualization 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 systems. 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 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 the number of spreadsheets re-calculated, and reports generated, and print jobs printed, and users rotating 3d-images (or just playing solitaire if the system isn't locked down).

In focusing on computational performance, one generally analyzes counters that provide detail on specific system activities, such as 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.

One common mistake made in Computational Performance is in not really understanding the counters. Especially with the Microsoft Operating Systems, we often see professionals look at the wrong counters. An example might be looking at the “Page Faults/Second” counter when they should really be looking at the “Pages in/Second”. This mistake is like your doctor measuring your overall cholesterol number rather than your “bad cholesterol”. The soft page faults that are included in the page faults counter are like the “good cholesterol” and indicate a healthy, well running, system. Another is when people look at the “Disk Queue Length”, when perhaps the “Disk Read Queue Length” might be a better counter to observe.

Computation Performance is an important technique you will use to optimize your systems. However, if you are optimizing a multi-use systems such as a Terminal Server or multiple VMs on a hypervisor, 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

Examples of Perceived Performance

Sometimes some examples illustrate a point better. Here we present two examples of using Perceived Performance.

Example 1: Context Switching

One example we often used to run into with IT Professionals is what we describe as being overly concerned with context switching. [Editorial note: In the time since the paper was originally published, the underlying CPUs have gotten so much faster, and we have more of them on our systems, leading to CPU related issues as being of less interest. This results in professionals no longer concerning themselves with Context Switching. This example, however, does illustrate the kind of problem that comes from focusing in on a particular counter without really understanding it].

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 range¹.

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).

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

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

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, what we call *serialized delay*.** 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.

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.

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.

Example 2: Delay Profile Graph

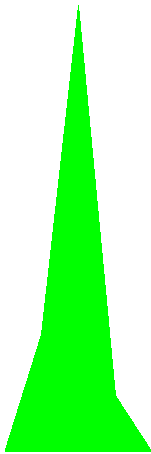
When publishing the second version of the paper, some additional work had gone into measuring the effect of serialized delay imposed on an application by others sharing the same system. This work was done in the context of investigating how a tool, such as triCerat Simplify Resources or the Citrix CPU management software they acquired from Aurema, could be used to improve performance on a Terminal Server, as perceived by the users. The impact of adding such software to a server can be quite positive (depending on the applications used), and in publishing an updated paper we demonstrate a way to view these results.

We begin by considering a relatively static performance scenario. In the first example we described a test to measure the delay of a very simple program that goes through a few hundred context switches to perform a task. Such a test is appropriate as a measure of the Perceived Performance felt by a user in a shared user environment like a terminal server. Each time there is a context switch, there is a chance that another program thread will run instead our intended application. When this occurs a small, incremental, delay occurs. *Serialized Delay* is the term used for the sum of these small incremental delays. Serialized Delay in general, however, occurs due to delays from many sources, not just CPU related ones. Network and File/IO also cause user delays in many ways, as so other applications or other virtual machines.

If we run our relatively static test many times on an otherwise idle system, we are likely to find that for any given test run, the Serialized Delay will vary from one run to another. If we run the enough times (say a thousand times) on the lightly loaded system we can determine a minimum period possible for the test. This minimum period may, for the purpose of displaying test results, be designated as a serialized delay value of zero. The serialized delay of any other run of the test is determined as the test time minus the minimum value.

Performing a sufficient number of test runs, and calculating the serialized delay for each run allows us to create what we call a *Delay Profile*. This Delay Profile illustrates the likelihood that the user will experience a delay of a given amount. This may be displayed as a graph to provide a visualization of the perceived performance effect that a user feels when other software is running on the server.

Figure 1 illustrates the Delay Profile Graph of a given test application on an unloaded system. The test application was run 600 times and the serialized delay calculated for each test. The height of the chart indicates the number of times that this serialized delay was measured to be the delay indicated (in 100ths of a second on the horizontal axis). This allows us a quick visualization of not only the average delay (the peak), but of the variation in delay.



As can be seen in the graph, even on a stable and virtually idle system there is a small variation in the serialized delay due to background system software and events. Both the average delay and variation in delay are key parameters in determining the performance that a typical user will perceive. A user can adjust to a 50 millisecond delay between typing a key and seeing it display on the screen when the delay is consistent. But the performance seems so much worse when the delay varies between 20 and 200. The variation shown in Figure 1 is small enough that the user would not notice.

Figure 2 illustrates two Delay Profiles of the same system when additional loads are added to the system under test

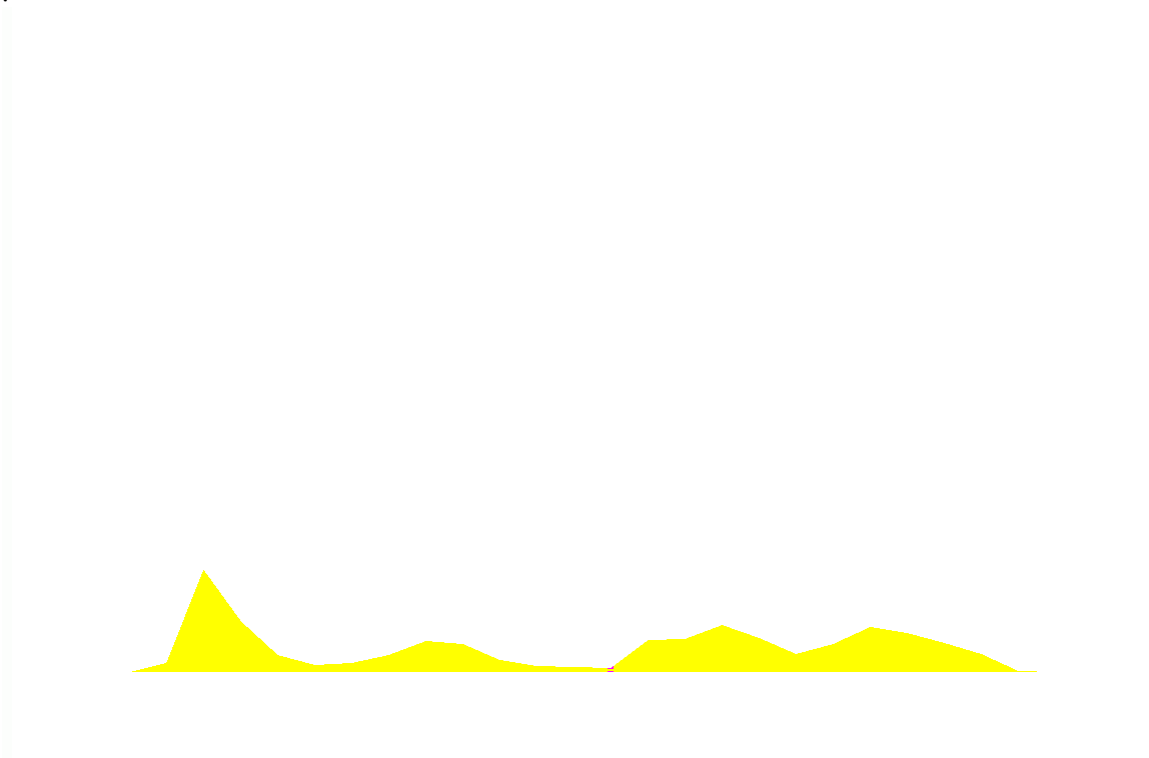


Figure 2 - Delay Profile with and without load

Each added load attempts to consume a complete CPU on the system (this system was a dual processor with Hyper-Threading enabled on Windows Server 2003, effectively having four processors). As is shown in yellow, not only is the average serialized delay larger when load is added, it is also quite variable. Numerically, this variability can be described by a calculating standard deviation; however, I happen to like a nice picture instead.

Clearly a user would perceive the performance of the situation shown in yellow in Figure 2 as being worse than the green situation; not only is the average delay greater, but the delay variability is much greater. Especially when considering repetitive short duration user tasks, such as typing documents and emails where users typing speed is affected by the visual feedback delays, the performance of the system as perceived by the user (as well as their productivity) would be affected more by the delay variability than the absolute delay. One interesting study that looks at user perception in the face of different delay patterns is “*Rethinking the Progress Bar*”³

³ *Rethinking the Progress Bar*, Harrison, Amento, Kuznetsov, & Bell; Proceedings of the 20th annual ACM symposium on User interface software and technology.

Perceived Performance Profile

While superimposing two Delay Profiles for comparison is illuminating, if we want to view the continuum of Delay Profiles as loading increases a different form for display is needed. I call this a *Perceived Performance Profile*. Let us consider the following test using the tools from the Perceived Performance Toolkit:

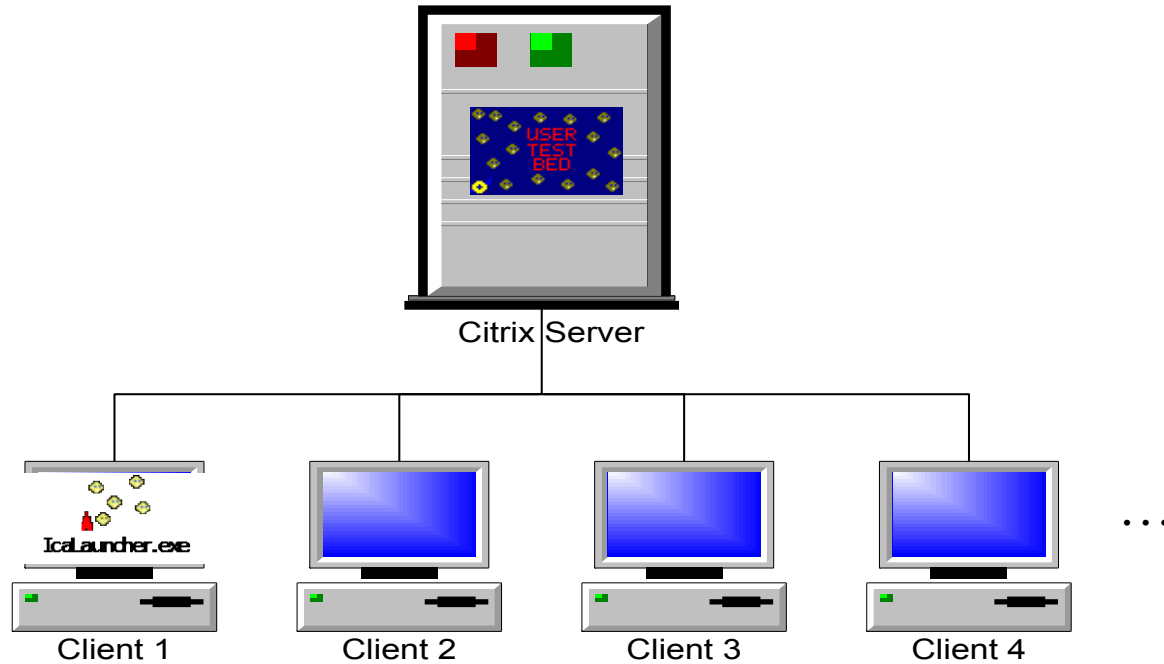


Figure 3 - Variable load test

Each client uses a remote launcher/measurement program called “IcaLauncher.exe”. Upon running IcaLauncher, a remote session to the Citrix Server is established and a published application on the server is run. This application, ServerTestApp, runs to completion without human intervention and is designed to produce a delay that is affected by serialized CPU delays. The IcaLauncher measures the delay from when the launch request is made until the program runs to completion. It is important to note that the delay is measured from the client side and not on the server:

- This allows delays imposed by the intervening network (which could be a WAN) to be included.
- This decreases measurement error that would occur if the measurement was made on the server that might be imposed by performance issues on the server.

A series of launches are made from a single client, each after waiting a small settlement time after completion of the prior test, and measurements (starting with the second launch) of the delay are recorded. These results may be compiled into a Delay Profile as previously described.

Next, a second series of launches are made using an additional test client running at the same time. Additional test series and Delay Profiles are generated.

In Figure 4, the complete results from all series of tests are shown. In this Delay Profile Graph, one can see how both the average delay and variability in delay tend to increase. This graphically illustrates the performance that end-users perceive when they are performing typical tasks such as word processing.

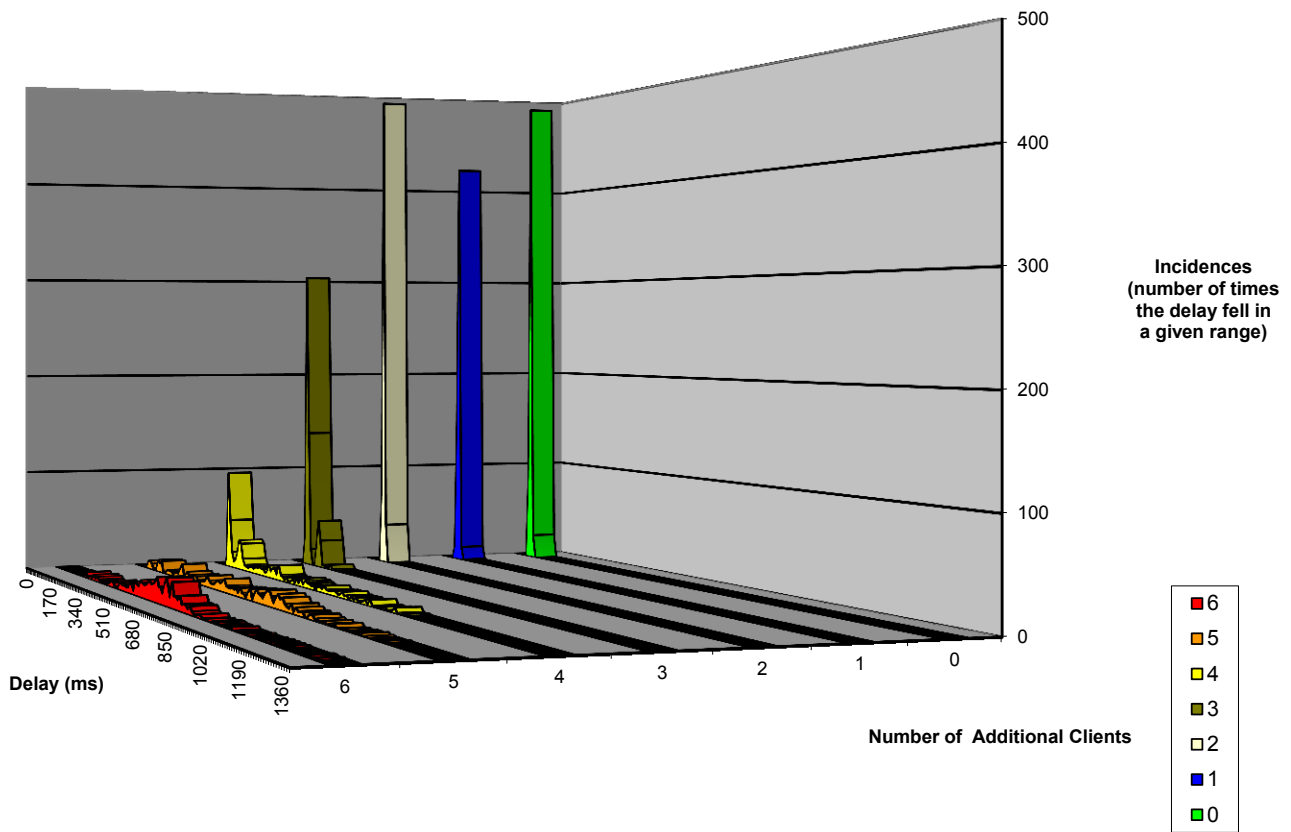


Figure 4 – Perceived Performance Profile under loads

As stated earlier, the goal of *Perceived Performance* is to improve user productivity by measuring end user performance and then making modifications aimed at reducing delays in responsiveness to user actions.

In Figure 5, we present the same tests after adding some additional software which modifies the thread scheduling algorithms of the operating system. In these tests, we used Simplify Resources 3.0 from triCerat. By implementing a Quality-of-Service (QoS) model on thread scheduling, this software improves both the average serialized delay and variability of a “typical user task”.

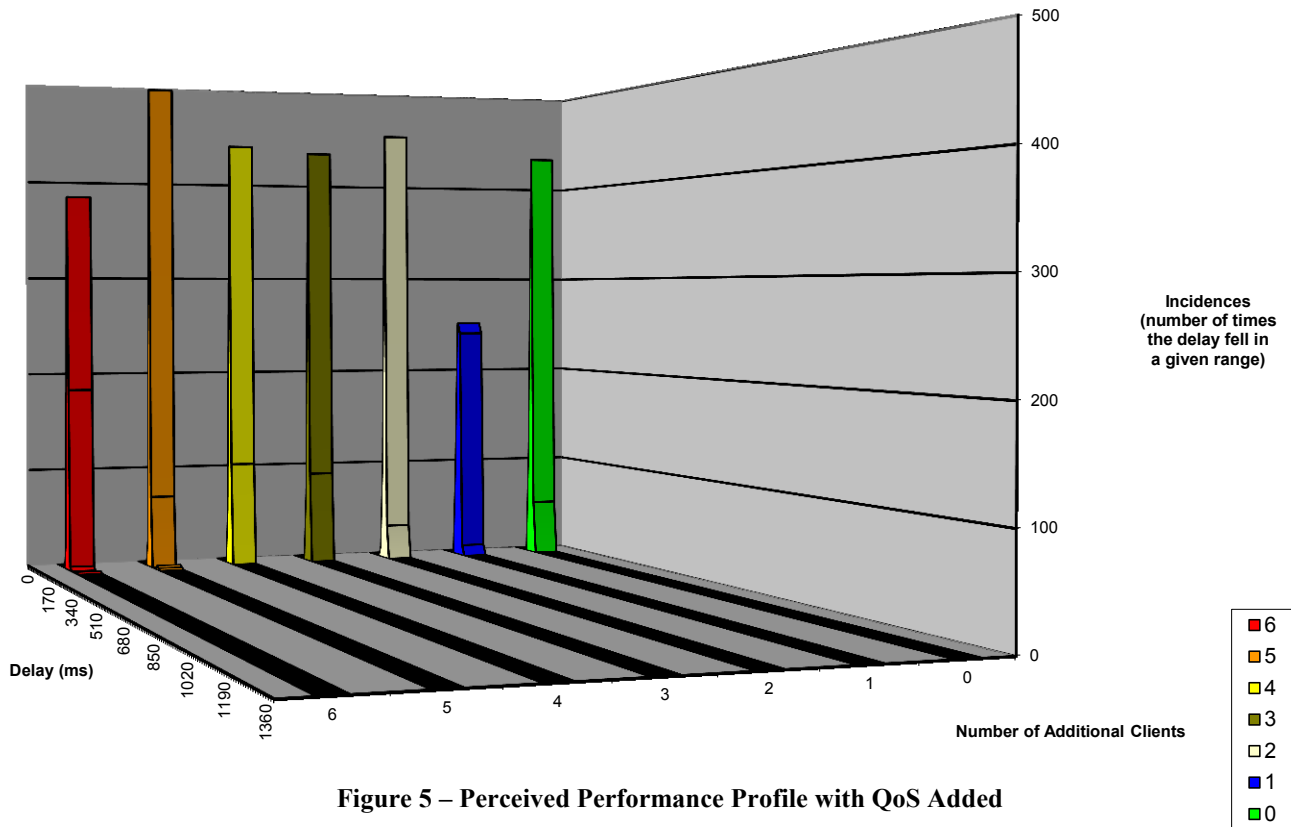


Figure 5 – Perceived Performance Profile with QoS Added

Adding the QoS algorithms software to the system improved the performance perceived by the users, even under extraordinary loading conditions. In cases where user performance due to CPU consumption limits scalability of a Terminal Server, this can allow more users to be supported on a given system before users begin to complain.

Example 3: Perceived Performance Profile and Virtual Machines

In the third version of the paper, we started using the Perceived Performance Profile (abbreviated as “P3”) to look at the performance of virtual machines. Because we have more than a single shared operating system to consider, being able to visualize the performance as certain conditions change becomes important.

In Figure 6, the P3 of a terminal server deployed directly to bare metal (without server virtualization) is shown. In this test, the perceived performance is measured as before, although different means were used to generate the additional simulated CPU load than in the prior tests.

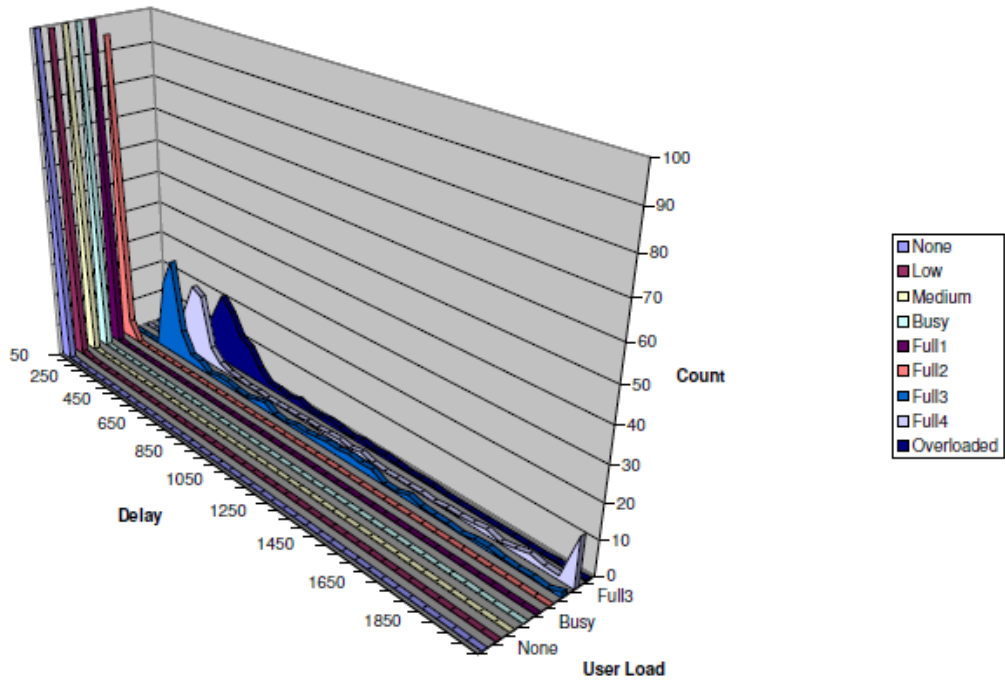


Figure 6 - P3 of a Physical System

In Figure 7, the same test was run on the same hardware, but in adding an early virtualization system (Microsoft Virtual Server in this case).

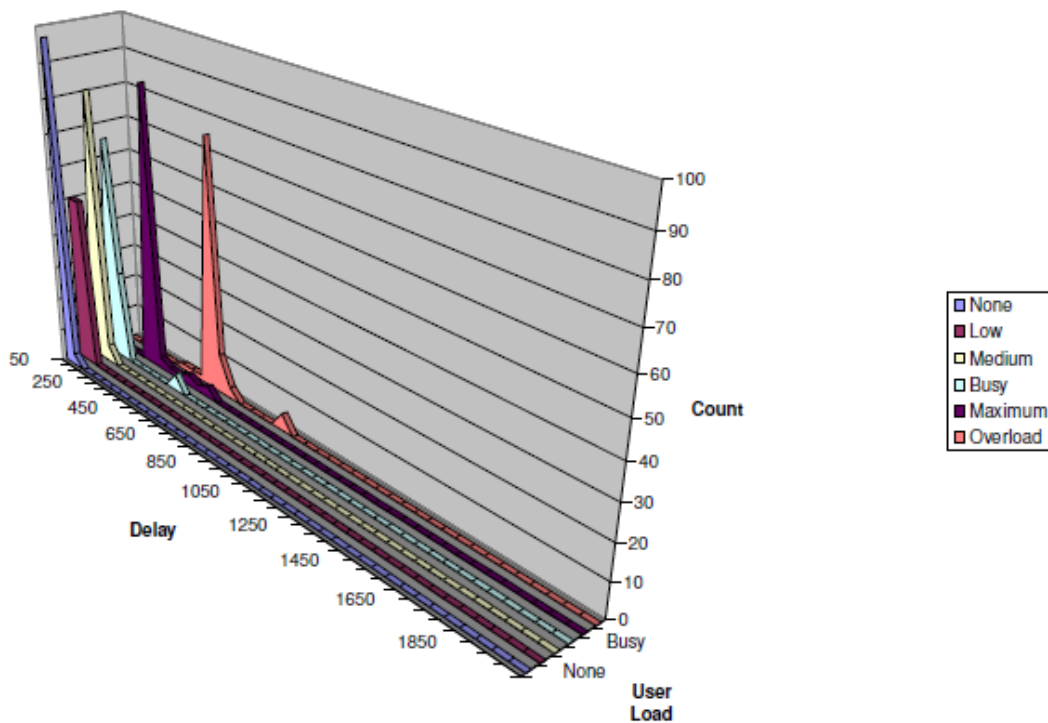


Figure 7 - P3 with Virtual Server (single VM)

This test in the VM demonstrated average degraded performance, but with a more variable delay when lightly loaded but markedly less variability than the physical system when under stress.

We could also use the P3 to compare different systems. The P3 in Figure 8 represents the same terminal server test, but using VMware's GSX virtualization software.

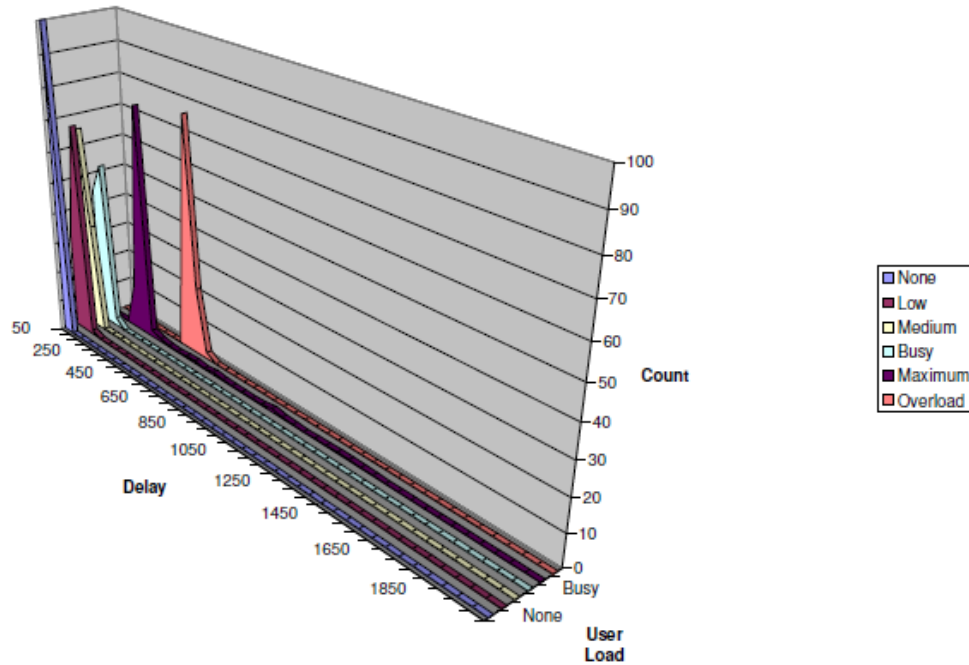
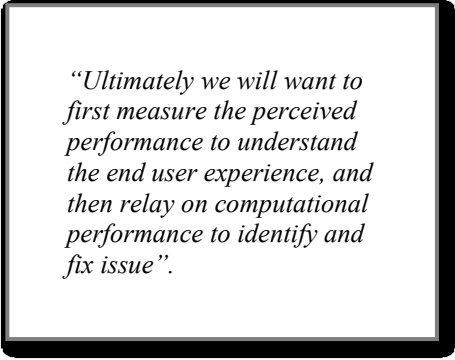


Figure 8 - P3 Using VMware GSX

Perceived Performance and VDI

Considering Performance Perception in Virtual Desktop scenarios (where many desktop operating systems are hosted on a server virtualization infrastructure), I believe there is much more to consider than for a terminal server on bare metal, or even in a virtual server such as we were considering 4 years ago. The differences are fourfold.

- The performance of the hypervisor has improved significantly. Also, multiple relatively inexpensive multicore processors have become the norm aided by considerably increased memory capacities. These advances have led to the common practice of placing more virtual machines on a single hypervisor, as CPU and Memory bottlenecks have been eliminated (or at least moved).
- The architecture of the virtual server infrastructure being implemented is vastly different today. A well designed virtualized server for VDI is likely to use an external shared storage (SAN via fiber channel or iSCSI) than to use local disks (although some systems are advancing to a hybrid approach using shared storage for permanence with local cached copy for performance).
- The use of “desktop” operating systems in the VMs, rather than server operating systems which typically have higher peak loading, is leading to a much greater VM per hypervisor density, and even VM per logical CPU ratio.
- Users of VDI desktops tend to use their computer session differently than those of Terminal Services. While we think of “task-based workers” as the norm for Terminal Services, these users tend more to be “knowledge workers” or even “power users”. Even among users of the same ilk, users today tend to do more multitasking, which affects how they perceive performance.



“Ultimately we will want to first measure the perceived performance to understand the end user experience, and then relay on computational performance to identify and fix issue”.

If we look back at the performance work on Terminal Servers, we can see that early on in the technology there was considerable variability in how the systems were deployed and this required that significant analysis of the deployed system. Later on, when everyone had settled on similar hardware platforms and standard “tweaks” to aid the performance, the notion of measuring end-to-end performance made more sense.

When it comes to VDI we are still very much in that early phase, and until we settle on standardized architectures and more mature software platforms and configurations, we will likely be able to product significant performance impact by making a few well-chosen guesses at where to attack performance and applying Computation Performance techniques. Ultimately we will want to first measure the perceived performance to understand the end user experience, and then relay on computational performance to identify and fix issues. But for now we can have such a tremendous impact by just adding more disk spindles or replacing out a badly performing anti-virus solution.

When looking at performance in these VDI scenarios, the order of guessing where to look for performance has significantly changed from that of the old Terminal Server days. Under TS, you worried first about CPU, then Memory, then IO. With VDI on these modern platforms, the order is most likely reversed so that we worry first about IO (especially file IO), then Memory, and then CPU.

This order also affects how we must consider Perceived Performance. The assumptions made previously that delay variation was more important than absolute delay, was made under an assumption that the user was experiencing many short duration events. This is no longer the case. The user’s perception of performance will be a summation of perception of both short and longer term events.

I would propose that while for an expected short duration event, delay variation is more important than the average delay, for a very long term event the opposite is true. I further propose that users are inherently more multi-tasking than they were a number of years ago. When a user performs a task, whether the task is

typing a letter on the keyboard or clicking the mouse on a button to perform a complicated task, the reasonably experienced user has an expectation about how long that task *should* take.

If that expectation is short, such as typing the letter, the user will stay concentrated on that task and I suggest that the delay variation is vital to the user's perceived performance. If, on the other hand, the expectation of how long the task should take exceeds some magic value for that user, the user will become distracted with another task, such as checking email or a web page. Obviously, the "magic value" will be different for each user, but each will have one, but once distracted, the delay variation will be unlikely to have any impact because the task will complete before the user looks back anyway. Thus if the two delay graphs shown in Figure 2 occur after 10 seconds rather than a sub-second period, the user might not notice the difference.

While these are extreme cases, the relationship between absolute delay and delay variation is not as simple as one or the other. The user's perception of performance will be a combination of both delay types. More research will be needed to determine how best to relate these two delay results under different conditions.

State of the Art for Perceived Performance in VDI

The current "state of the art" for measuring perceived performance in a VDI environment at the time of this writing appears to be Project VRC⁴. The goals of project VRC are to provide a repeatable framework that attempts to measure user perceived performance in a comparable way, so that two hardware or software setups could be tested and compared.

Project VRC uses a test automation framework that simulates user activity using standard office applications, occasionally measuring the delay between a user request and the system response. Because a human comparing two P3 graphs might be subjective, the project set out to produce a single number that represents the performance of the system under test under varying conditions. If chart in Figure 7 can be boiled down to a single number and chart in Figure 8 to another number, then we can compare those two numbers to determine which has better performance.

The single number derived in this project is called the VSI MAX, or Virtual Scale Index Maximum. VSI MAX does not represent an exact number of users that if working simultaneously on the platform would consider the performance reasonable, but it is a number that when compared to the number produced by another system is comparable to know which would support more users.

VSI MAX is determined by examining the results of perceived performance testing. Although the test framework produces a graph showing the minimum, average, and maximum delay for a given series of tests, the determination of VSI MAX does not consider delay variability at all; the method used to determine VSI Max has been tweaked in each version, but currently it focuses on only the maximum value in a test series. Some of the recent improvements include throwing out the highest and lowest two results in a test series (as well as any obviously bad results caused by issues in the framework). Also, the cutoff for what bad performance is now based on a baseline of adjusted worst performance for the first 15 test series. Using this as a baseline, if the worst case delay is more than 2 seconds longer, VSI Max is declared.

While this test technique is fairly well designed⁵, and the analysis of results is probably sufficient for what we need today, when performance tuning involves making changes that have significant differences in performance, we will eventually need improved measurement analysis that takes into consideration all of the test results (rather than "worst case only") and even understands better how to relate the absolute and

⁴ <http://www.projectvrc.nl>

⁵ One possible exception is that the delay measurements are made directly on the system under test (inside the VM) instead of at the client. In investigating this, I found that this had little effect on the measurements made except under test conditions that would have been unacceptable to the user anyway, however, this design precludes certain kinds of perceived performance tests.

variable delays into the calculation. Nobody knows how to do this today, so please do not take this as a criticism of the VSI Max calculations!

Boiling Down to Two Numbers

The concept of boiling down the complex performance to a single number itself is of considerable interest. In that regard, I suggest that there are two numbers that we can calculate today that would be very important components of an “improved” single number than we have today with VSI Max.

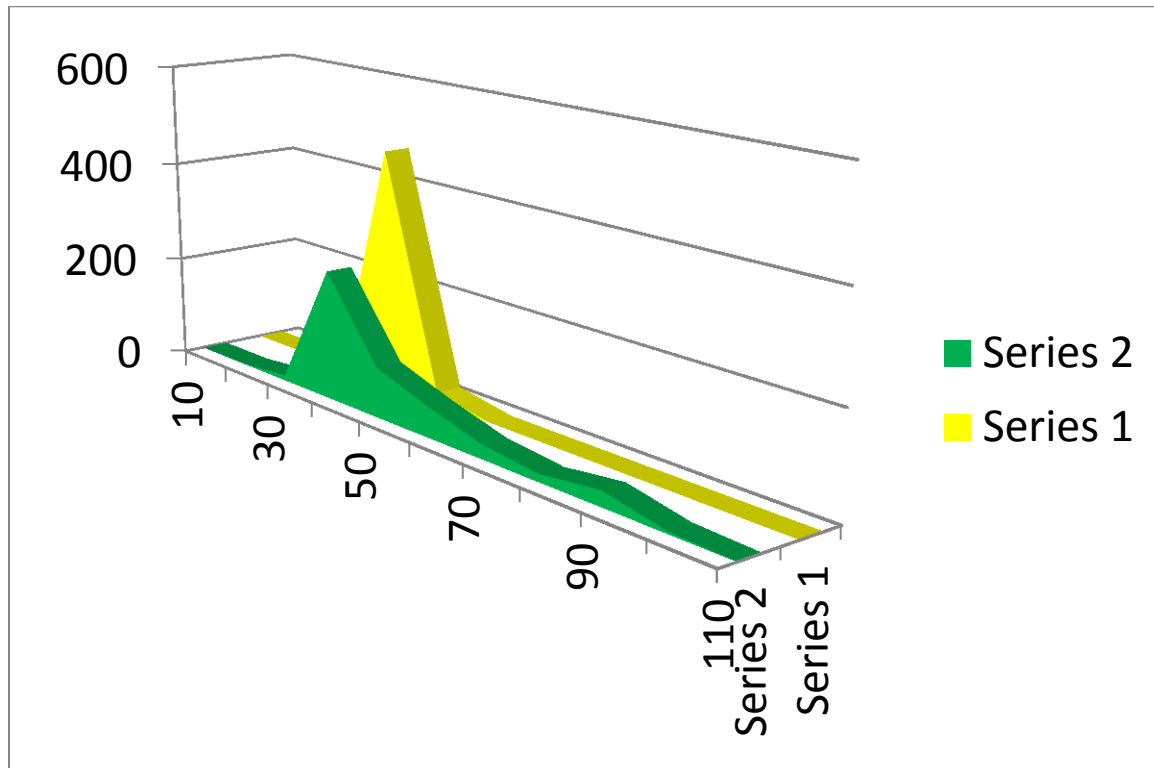


Figure 9 - Performance Calculated as 2 Numbers

Figure 9 represents the Perceived Performance Profile of two series of tests. Using this data, we can calculate two numbers:

- Average Delay Value
- Delay Variation

For the average delay, one could choose the median or mean delay value. My preference is to use the mean.

For delay variation, I have given to using a calculation referred to “mean average deviation”. This calculation is slightly more complicated than the better known “standard deviation”, but provides a clearer number. This calculation is performed by first determining the mean value, and then determining the absolute value of the difference of each sample from this mean, and finally averaging out those absolute differences to generate a number that represents the average difference of any sample from the mean. In the case of Figure 9, these calculations for the two series are:

- Series 1: Average Delay = 40.2ms, Delay Variation=.784ms
- Series 2: Average Delay = 50.64ms, Delay variation=11.882ms

The question of how to apply these two numbers into a calculation of a single number will require additional study. For now, I can suggest two obvious situations to consider:

- For highly repetitive event-response scenarios where the expectation of the user is that the response should occur in a short period of time (for example in under 2 seconds), delay variability should have a larger impact on the perception of performance by the user than average delay. These scenarios include typing and mouse clicking to perform simple actions.
- For event-response scenarios where the expectation of the user is that the response should occur after a significant period of time (for example, over 20 seconds) and especially if also not a repetitive event, delay variability will have little impact on performance perception and average delay would be more important.

Combining Perceived Performance with Computational Performance for VDI

Measuring performance from the end user viewpoint is a necessary first step to identifying and solving performance issues. That we choose to deploy software to users using VDI does not change this. Such testing allows us to compare two different possible implementations, or to quantify tested performance today versus that of tomorrow. Such measurements will eventually need to give way to more detailed testing using conventional “computational performance” techniques.

In addition, A VDI session consists of many different phases, connecting, logging into a VM (which includes significant layering or user profile movement activities), a work phase, and a save work/logoff phase. The test today is focused on the work phase. We may find we want different test methods to determine performance of the other phases.

The Coming of the Cloud

If/as public cloud based computing becomes more relevant, work in performance will get far more spicy. Not only is Cloud Computing based upon Server Virtualization technology, it is also always remotored over a variable WAN network. Add to this that the Cloud provider will only allow a very limited access (if any access at all) to performance indicators taken from within the cloud infrastructure. Measuring from the user to the cloud and back will need to be the first line of defense. Tests that can segregate measurements into three parts, Local LAN, WAN, and Cloud, will become necessary. The addition of this highly variable WAN component will also likely make delay variability more important. The limited access that we will have to the Cloud end will also become an issue. APIs for that access are still under development and appear to be headed towards proprietary APIs rather than a single “standardized” API, which will make things all the more interesting!

Conclusions

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 measure the perceived performance, then make changes, and then measure again to verify whether or not the change helps.

Specifically when it comes to VDI implementations, the biggest bang for the buck today is to focus on the storage systems. The key computational performance counter referred to by most experts is the Disk Queue Length counter for the system. This counter provides a snapshot of the average number of disk I/O requests that are held in queue, waiting for a previous request to complete. The “conventional wisdom” is to monitor the DQL and compare it to the number of spindles in use. This conventional wisdom holds that you don’t want to constantly run the system with more than a DQL of 1 for each spindle.

While this conventional wisdom is a good start, there is more to be considered. The DQL is not a counter of how many requests are in the queue currently, but an average number. Thus a DQL=1 means that

something was almost always waiting in the queue without being processed due to an earlier I/O request. The first thing to be considered is that, especially in a VDI scenario, if you have a DQL of 8 on a system containing 8 spindles, it is unlikely that the queue represents one queued item for each spindle. More likely you have a disk queue length of a larger number on one of those spindles, and thus an unhappier user waiting for that I/O operation to complete. So while an overall measurement might be a good place to start, a detailed look might also be needed. Also consider how to determine how many spindles are effective – especially when various RAID configurations are in play. In these complicated scenarios, measurement of I/O latency (or perhaps latency relative to the transfer size) might be more appropriate. In addition, consider that the performance impact of a high latency response to a disk write is may be negligible when it comes to perceived performance to that of a high latency disk read because so many disk writes are. Rather than look at the total DQL the Disk Read Queue Length might be more interesting except during the save work/logoff phase.

Much more work needs to be done to evolve the concepts described in this paper, especially as our systems and needs continue to evolve. Stay Tuned!

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.turgent.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, [no longer available online].

Ref 5 *Rethinking the Progress Bar*, Harrison, Amento, Kuznetsov, & Bell; Proceedings of the 20th annual ACM symposium on User interface software and technology [Available in the ACM Digital Library]

Ref 6 *Faster Progress Bars: manipulating perceived duration with visual augmentations*, Harrison, Yeo, & Hudson, Proceedings of the 28th international conference on Human factors in computing systems. [Available in the ACM Digital Library]

Ref 7 *Login VSI 3.0 Administrator Guide*, Login Consultants and PQR, [available with free online registration at www.loginconsultants.com]