VDBench: A Benchmarking Toolkit for Thin-client based Virtual Desktop Environments

Alex Berryman, Prasad Calyam, Matthew Honigford, Albert M. Lai

Ohio Supercomputer Center/OARnet, VMware, The Ohio State University,
berryman@oar.net; pcalyam@osc.edu; mhonigford@vmware.com; albert.lai@osumc.edu

***Abstract*—The recent advances in thin client devices and the push to transition users' desktop delivery to cloud environments will eventually transform how desktop computers are used today. The ability to measure and adapt the performance of virtual desktop environments is a major challenge for "virtual desktop cloud" service providers. In this paper, we present the "VD-Bench" toolkit that uses a novel methodology and related metrics to benchmark thin-client based virtual desktop environments in terms of scalability and reliability. We also describe how we used a VDBench instance to benchmark the performance of: (a) popular user applications (Spreadsheet Calculator, Internet Browser, Media Player, Interactive Visualization), (b) TCP/UDP based thin client protocols (RDP, RGS, PCoIP), and (c) remote user experience (interactive response times, perceived video quality), under a variety of system load and network health conditions. Our results can help service providers to mitigate over-provisioning in sizing virtual desktop resources, and guesswork in thin client protocol configurations, and thus obtain significant cost savings while simultaneously fostering satisfied customers.**

I. Introduction

Common user applications such as email, photos, videos and file storage are already being supported at Internet-scale by "cloud" platforms (e.g., Amazon S3, HP Cloud Assure, Google Mail, and Microsoft Azure). Even academia is increasingly adopting cloud infrastructures and related research themes (e.g., NSF CluE, DOE Magellan) to support various science communities. The next frontier for these user communities will be to transition "traditional distributed desktops" that have dedicated hardware and software installations into "virtual desktop clouds" that are accessible via thin clients. The drivers for this transition are obvious and include: (i) desktop support in terms of operating system, application and security upgrades will be easier to manage centrally, (ii) the number of underutilized distributed desktops unnecessarily consuming power will be reduced, (iii) mobile users will have wider access to their applications and data, and (iv) data security will be improved because confidential user data does not physically reside on thin clients.

The recent advances in thin client devices and the push to transition users' desktop delivery to cloud environments have opened up new challenges and will eventually transform how we use computers today. One major challenge for a "virtual desktop cloud" service provider will be to handle desktop delivery in a scalable manner to provision and adapt the cloud platform for an increasing number of users. Given the fact that memory is the most expensive and possibly the most contended resource in virtual desktop clouds (i.e., users will idle their CPUs but will keep their applications always open on a desktop), suitable "overcommitted" memory sizing for virtual desktops based on user activity profiling is vital. Another major challenge will be to ensure satisfactory user experience when accessing desktops from remote sites with varying end-to-end network path performance.

This material is based upon work supported by the Ohio Board of Regents, VMware, Dell, and IBM.

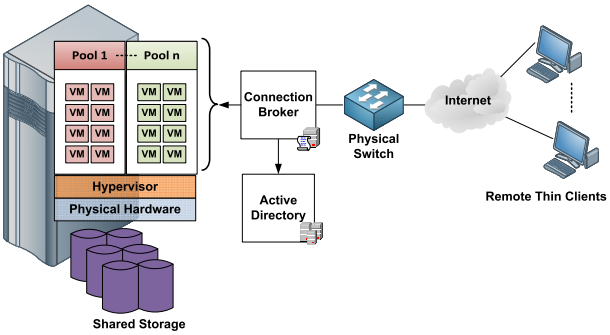

Fig. 1. Virtual desktop cloud system components

Figure 1 shows the various system components in a virtual desktop cloud. At the server-side, a hypervisor framework (e.g., VMware ESXi, OpenVZ, Xen) is used to create pools of virtual machines (VMs) that host user desktops with popular applications (e.g., Excel, Internet Explorer, Media Player) as well as advanced applications (e.g., Matlab, Moldflow). Users of a common desktop pool use the same set of applications, but maintain their distinctive datasets. The VMs share common physical hardware and attached storage drives. At the client-side, users connect to a server-side broker via the Internet using various TCP (e.g., VNC, RDP, RGS) and UDP (e.g., PCoIP) based thin client devices. The connection broker authenticates users by active directory lookups, and allows users to access their entitled desktops.

Our work is motivated by the fact that service providers need frameworks and tools today that can enable them to build and manage virtual desktop clouds at both staging-scale and Internet-scale. To cope with increasing user workloads, extensive work has been done to efficiently manage server-side resources based on CPU and memory measurements [1 - 4]. However, there is surprisingly sparse work [5] [6] on resource adaptation coupled with measurement of network health and user experience. It is self-evident that any cloud platform's capability to support large user workloads is a

978-0-7695-4302-4/10 $26.00 © 2010 IEEE
DOI 10.1109/CloudCom.2010.106

function of both the server-side desktop performance as well as the remote user-perceived quality of experience. Hence, lack of proper "human-and-network awareness" in cloud platforms inevitably results in costly guesswork and over-provisioning while managing physical device and human resources, which consequently annoys users due to high service cost and unreliable quality of experience.

In this paper, we present the "VDBench" toolkit that uses a novel methodology and related metrics to benchmark thin-client based virtual desktop environments in terms of scalability and reliability. The methodology involves creating realistic workflows in order to generate synthetic system loads and network health impairments that affect user-perceived 'interactive response times' (e.g., application launch time, webpage download time). In addition, the methodology allows correlation of thin-client user events with server-side resource performance events by virtue of 'marker packets' that leverage and extend our earlier research on slow-motion benchmarking of thin-clients [7] [8]. The marker packets particularly help in the analysis of network traces to measure and compare thin-client protocols in terms of transmission times, bandwidth utilization and video quality. Further, the methodology lends itself for generation of resource (CPU, memory, network bandwidth) utilization profiles of different user applications and user groups. Such profiles can be used by service providers to optimally categorize applications into desktop pools, allocate system-and-network resources, and configure thin-client protocols. In addition to describing the VDBench methodology and related metrics, we also describe how we used a VDBench instance to benchmark the performance of: (a) popular user applications (Spreadsheet Calculator, Internet Browser, Media Player, Interactive Visualization), (b) TCP/UDP based thin client protocols (RDP, RGS, PCoIP), and (c) remote user experience (interactive response times, perceived video quality), under a variety of system load and network health conditions.

The remainder of the paper is organized as follows: Section II provides a background and describes related work. In Section III, we present the VDBench methodology and metrics for user-load simulation based benchmarking and slow-motion application interaction benchmarking. Section IV presents our simulation results to validate the VDBench methodology. Section V concludes the paper.

II. Technical Background

In this section, we provide the technical background relating to our implementation of the VDBench toolkit that is based on memory management capabilities of VMware ESXi Server, TCP/UDP based thin-client protocols, and slow-motion based thin-client benchmarking principles.

A. Memory Management

The memory management capability of VMware ESXi Server optimizes the utilization of physical memory [9]. Each VM is allocated a specified size of memory, an optional minimum reservation, and a small amount of virtualization overhead. The ESXi server attempts to allocate memory to each VM up to the specified limit. In cases of memory over commitment, occurring when the sum of the total memory specified exceeds the amount of physical memory, each VM is guaranteed at least the reserved amount of memory, and receives additional memory based on the current load on the ESXi server. A taxing policy is used to create an additional cost for inactive memory pages, thus exhausting the VM's memory share at a higher rate and triggering the memory management tools of ESXi sooner than if all the memory pages were active. The ESXi server must reclaim allocated memory from a VM that has exceeded its amount of memory shares in order to redistribute the memory to an under-allocated VM. This process is accomplished by either invoking a memory "ballon driver" that is installed on the VM or having the ESXi server swap the contents of its' physical memory to a swap file on the hard disk. The balloon driver is installed on the guest operating system within a VM as part of the VMware tools software package. The balloon driver is controlled by the ESXi Server and forces the guest operating system to free up the pages using the guest operating system's native memory management algorithms and returns them to the ESXi server for redistribution. The balloon driver reports to the guest operating system in the VM like a normal program that has higher and higher memory utilization. The memory usage of the balloon driver triggers the native memory management algorithms which uses garbage collection to remove pages, or swaps them to the VM's virtual swap disk if the pages are still being used.

B. Remote Display Protocols

User experience is the dominating factor in determining the configuration of the underlying remote display protocol. Remote display protocols are used to transmit the visual components of the virtual desktop to the client. The remote display protocols have different methods of determining the most optimum way to encode and compress the data in order to transport and render it at the client side. Different protocols are designed to optimize different display objects like text, images, video and flash content. Each protocol has a different impact on the system resources (CPU, memory, I/O bandwidth) that are used to compress the display data on the server side. Some protocols handle the compressions of text better than others whereas, some protocols handle the compression of multimedia content better. These display protocols also exhibit different levels of robustness in degrading network conditions; some are more adaptive than others. This robustness can come from the underlying transmission protocol (TCP/UDP), or the protocol's ability to adapt and scale its compression to fully utilize all of the available network path. Examples of TCP/UDP based thin-client protocols include Microsoft Remote Desktop Protocol (RDP via/TCP), HP Remote Graphics Software (RGS via/TCP), and Teradici PC over IP (PCoIP via/UDP). The level of compression done on the server side of the thin-clients must be reversed on the client side in the task of decompression. High levels of compression on the server side can cause less network resources to be consumed, but the client is required to consume additional system resources in order to rebuild the transmission. A optimal relation between compression, network availability, and client side computing power must be set to ensure satisfactory user experience.

C. Thin-client Performance Benchmarking

There have been studies of performance measurement using slow-motion benchmarking for thin-client systems. The slow-motion benchmarking technique was used to address the problem of measuring the actual user perceived performance of client by Nieh et. al. [8]. This work was focused on measuring the performance of web and video applications on thin-clients through remote desktop applications. Lai, et. al. [7] [8] used slow-motion benchmarking for characterizing and analyzing the different design choices for thin-client implementation on wide-area networks. In slow-motion benchmarking, an artificial delay in between events is introduced, which allows isolation of visual components of those benchmarks. It is important to ensure that the objects are completely and correctly displayed on the client when benchmarking is performed. This is because the client side rendering is independent of the server side processing. Existing virtual desktop benchmarking tools such as "Login VSI" [10] do not take into consideration this distinction between client side rendering and server side processing and hence are not relevant when network conditions degrade. Note that we combine the scripting methodology of Login VSI that provides controllable and repeatable results for the execution of synthetic user workloads on the server side, and the introspection that slow-motion benchmarking provides into the quality of user experience on the client side. This combination allows us to correlate thin-client user events with server-side resource performance events. Earlier works on thin-client benchmarking toolkits such as [5] and [6] have several common principles that are used in VDBench, however they are focused on recording and playback of keyboard and mouse events on the client-side and do not consider synchronization with server-side measurements, as well as user experience measurements for specific applications (e.g., Spreadsheet Calculator, Media Player).

III. VDBench Methodology and Metrics

In this section, we present the VDBench toolkit methodology and metrics for user-load simulation based benchmarking and slow-motion application interaction benchmarking.

Figure 2 shows the various VDBench physical components and dataflows. The hypervisor layer is used for infrastructure management. The hypervisor kernel's memory management functions are invoked during the user-load simulations and the virtual network switch is employed in the slow-motion application interaction measurements. Our *VDBench Management* virtual appliance along with a fileserver/database, as well as the desktop pools containing individual VMs are provisioned on top of the hypervisor.

A. User Load Simulation based Benchmarking

The goal of our user load simulation is to increase host resource utilization levels so as to influence interactive response times of applications within guest VMs. In our first trial, we created synthetic memory loads in a controllable manner by having a small number of VMs running large matrix operations that consume host resources. We expected resources to be starved away by these large matrix operations from a VM under test. However, we contrarily observed

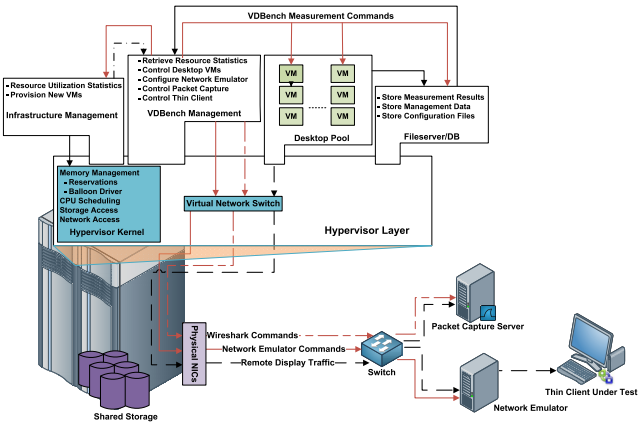

Fig. 2. Components of VDBench and data flows

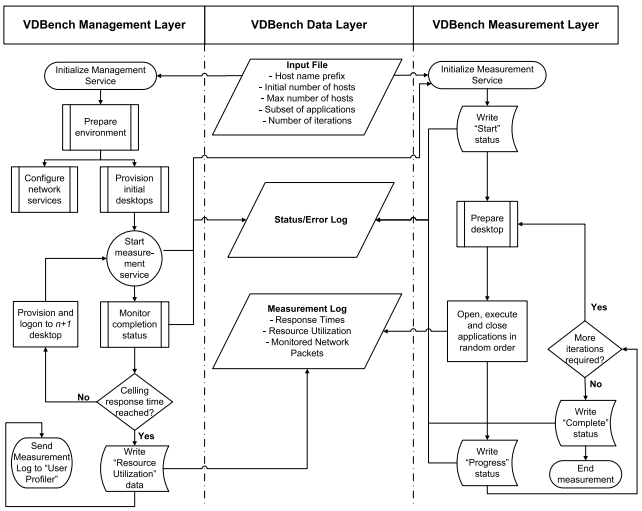

Fig. 3. VDBench control logic for benchmarking

that our efforts were superseded by effective implementation of memory mangement tools in the hypervisor. The inital application response time results did not exhibit the expected increasing trend in correlation with increasing system load.

In our subsequent trial, we developed a different load generation method shown in Figure 3 that models real users' workflows by concurrent automation of application tasks in random across multiple VMs. With this approach, we were able to controllably load the host machine and correspondingly obtained degrading application response time results.

Figure 3 shows the logical connection between the management, data and measurement layers of our VDBench virtual appliance. The *management service* is responsible for the provisioning of desktops, launching the load generation scripts on the VMs, monitoring their progress, and recording results of the measurements. An experiment begins when the VDBench management service provisions the first VM, and then spawns a *measurement service* on the VM, which then starts running the load generating script in order to establish a baseline of application response times. The load generation script automates

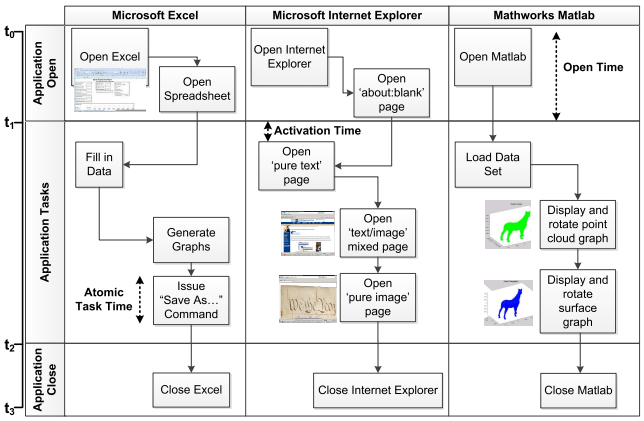

Fig. 4. Random progression of application tasks

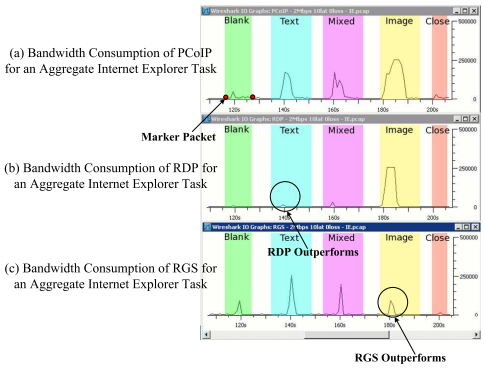

Fig. 5. Example traces to illustrate slow-motion benchmarking

the execution of a sample user workflow of *application tasks* as shown in Figure 4. The workflow involves simulating a user launching applications such as Matlab, Microsoft Excel, and Internet Explorer in a random sequence. Once all of the the applications are open, different application tasks are randomly selected for execution until all of the tasks are completed. Next, the script closes all of the launched applications in preparation for the next iteration. A controllable delay to simulate user *think time* is placed between each of these steps, as well as the application tasks. An exaggerated user think time is configured in VDBench in order to incorporate slow-motion principles into remote display protocol experiments. This process is repeated 40 times for each test so that a steady state of resource utilization measurements can be recorded in the *measurement log*.

Once the initial baseline is established, an additional VM is provisioned by the management service and the load generation script is run concurrently on both VMs while application response time measurements are collected in the measurement log. This pattern of provisioning a new VM running the load generation script, and collecting application response time data is continued until the response times hit the *response time ceiling*, representing an unacceptable time increase in any given task execution (e.g., a user will not wait for more than 2 seconds for the VM to respond to a mouse click), and subsequently the experiment is terminated.

The application response times can be grouped into two categories: (i) *atomic*, and (ii) *aggregate*. Atomic response time is measured as the time taken for an intermediate task (e.g., "Save As" task time in Microsoft Excel shown in Figure 4 or web-page download time in Internet Explorer) to complete while using an application. The atomic response times can also refer to an application's *activation time*, which is the time for e.g., taken for the appearance of dialogue boxes in Excel upon "Alt+Tab" from an Internet Explorer window. Aggregate response time refers to the overall execution time of several intermediary atomic tasks. One example of aggregate response time calculation is the time difference between t_3 and t_0 in Figure 4.

B. Slow-motion Application Interaction based Benchmarking

Our aim in VDBench development in terms of application interaction benchmarking is to employ a methodology that only requires instrumentation at the server-side and no execution on the client-side to estimate the quality degradation in desktop user experience at any given network health condition. The capability of no-execution on the client-side is critical because thin-client systems are designed differently from traditional desktop systems. In thin client systems, the server does all the compression and sends only "screen scrapes" for image rendering at the client. Advanced thin-client protocols also support screen scraping with multimedia redirection, where a separate channel is opened between the client and the server to send multimedia content in its native format. This content is then rendered in the appropriate screen portion at the client. The rendered output on the client may be completely decoupled from the application processing on the server such that an application runs as fast as possible on the server without considering whether or not the application output has been rendered on the client. Frequently this results in display updates being merged or even discarded. While these optimization approaches frequently conserve bandwidth and application execution time may seem low, this does not accurately reflect the user perceived performance of the system at the client. Further, no-execution on the client-side is important because many thin-client systems are proprietary and closed-source, and thus are frequently difficult to instrument.

To address these problems and to determine the performance characteristics of each of the remote desktop protocols (i.e., RDP, RGS, PCoIP considered in this paper), we employ the slow-motion benchmarking technique. This technique employs two fundamental approaches to obtain an accurate proxy for the user-perceived performance: monitoring server-side network activity and using slow-motion versions of on-screen display events in applications. Figure 5 shows a sample packet capture of a segment of a slow-motion benchmarking session with several on-screen display events. We provide a brief description of our specific implementation of this technique below. For a more in depth discussion, please refer to [7] [8].

Since the on-screen display events are created by inputs that are scripted on the server-side, there are several considerations that must be acknowledged in our benchmarking technique.

First, our technique does not include the real-world time delay from when a client input is made and until the server receives the input. It also does not include the time from which a client input is made and the input is sent. Lastly, it does not include the time from when the client receives a screen update and to the time the actual image is drawn on the screen. We approximate the time omitted by the first limitation in VDBench by adding the network latency time to the measurements. However, the client input, and display processing time measurements are beyond the scope of our current VDBench implementation. Note that we also assume thin-client protocols do not change the type or amount of screen update data sent and captured in our tests, and that any variances in the data sent are due to ensuring reliable transport of data, either at the transport layer in the case of TCP-based protocols (RDP, RGS) or at the application layer in UDP-based protocols (PCoIP).

We now explain our network traces handling to obtain the performance metrics supported by VDBench. We benchmarked a range of thin-client protocol traces (RDP, RGS, PCoIP) to compare their performance under a variety of conditions. The PCoIP protocol traces exhibited a reluctance to quickly return to idle traffic conditions. This is most likely due to monitoring and adaptation algorithms used in the auto-scaling of the protocol. Judicious filtering process based on the volume of idle-time data allowed us to successfully distinguish the data transferred for the pages from the overhead. This lack of peak definition was exacerbated by the deterioration of network conditions in case of all the protocols. As latency and loss increased, the time taken for the network traffic to return to idle also increased, and correspondingly resulted in degraded quality of user experience on the remote client-side. We express this observation as *transmission time*, which is a measure of the elapsed time starting from the intiation of a screen-update and ending when the nework trace has returned to idle conditions. The initiation and completion of screen-events are marked in VDBench by the transmission of *marker packets* shown in Figure 5 that are sent by the VDBench automation script. A marker packet is a UDP packet containing information on the screen-event that is being currently displayed. The *transmission time* can be visualized based on the duration of the peak between marker packets. Over this *transmission time* interval, the amount *data transmitted* is recorded in order to calculate the *bandwidth consumption* for an atomic task.

We use these metrics in the context of a variety of workloads under various network conditions. In our slow-motion benchmarking of web-page downloads of simple text and mixed content shown in 5, the initiation and completion of each web-page download triggers a transmission of a marker packet. The marker packet contains information that describes the event (in this case, which web-page is being downloaded), and a description of the emulated network condition configured. We start packet captures with the thin-client viewing a blank desktop. Next, a page containing only a text version of the US Constitution is displayed. After a 20 second delay, a web page with mixed graphics and text is displayed. After another 20 second delay, a web page containing only high-resolution images is displayed. Following another 20 second delay, the browser is closed and displays a blank desktop. This process is repeated 3 times for each thin-client benchmarking session and corresponding measurements are recorded in the VDBench measurement logs.

For the slow-motion benchmarking of video playback workloads, a video is first played back at 1 frame per second (fps) and network trace statistics are captured. The video is then replayed at full speed a number of times through all of the remote display protocols, and over various network health conditions. A challenge in performance comparisons involving UDP and TCP based thin-client protocols in terms of video quality is coming up with a normalized metric. The normalized metric should account for fast completion times with image impairments in UDP based remote display protocols, in comparison to long completion times in TCP based thin clients with no impairments. Towards meeting this challenge, we use the video quality metric shown in Equation (1) that was originally developed in [7]. This metric relates the slow-motion playback to the full speed playback to see how many frames were dropped, merged, or otherwise not transmitted.

$$\textit{Video Quality} = \frac{\frac{\frac{\textit{Data Transferred (aggregate fps)}}{\textit{Render Time (aggregate fps)}}}{IdealTransfer(aggregatefps)}}{\frac{\frac{\textit{Data Transferred (atomic fps)}}{\textit{Render Time (atomic fps)}}}{IdealTransfer(atomicfps)}} \quad (1)$$

IV. Performance Results

In this section, we present simulation results to validate the VDBench methodology for user-load simulation based benchmarking and slow-motion application interaction benchmarking. In our user benchmarking experiments presented below, we used a VM testbed environment running VMware ESXi 4.0, consisting of an IBM HS22 Intel Blade Severs, installed into IBM Blade Center S chassis. The blade server has two Intel Xeon E5504 quad-core processors and 32GB of RAM, with access to a 9TB shared SAS. Each VM ran Windows XP and was allocated 2GB of RAM. The network emulation is done using NetEm, which is available on many 2.6 Linux kernel distributions. NetEm uses the traffic control `tc` command, which is part of the 'iproute2' toolkit. Bandwidth limiting is done by implementing the token-bucket filter queuing discipline on NetEm. Traffic between the client and server is monitored using a span port configured on a Cisco 2950 switch. The span port sends a duplicate of all the packets transmitted between the VM and the thin client to a machine running Wireshark to capture the packet traces and to filter out non-display protocol traffic.

A. User Load Simulation Results

Figure 6 shows percent of increase of the memory utilization after the balloon driver has already began to reclaim memory. Note that the *Memory Balloon* measurements majorly influence the memory usage statistics. The balloon driver engaged with 17 VMs running since the amount of memory allocated to VMs exceeded the amount of physical memory and preceded to increase by 750% with 46 VMs running. The value of the balloon size increased with a steady slope as the number of active VMs increased. The *Memory Granted*,

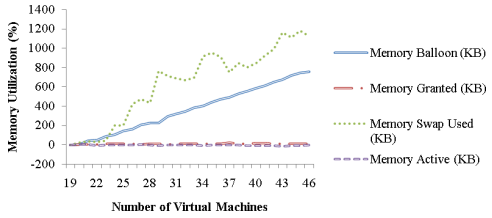

Fig. 6. Memory Utilization with increasing system loads

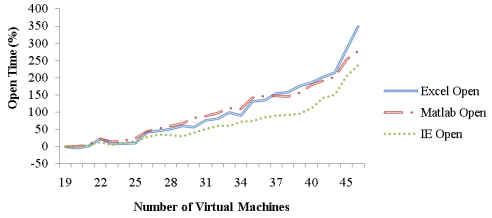

Fig. 7. Application Open Times with increasing system loads

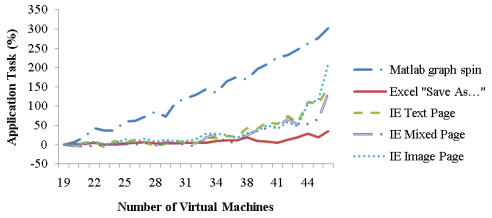

Fig. 8. Application Task Times with increasing system loads

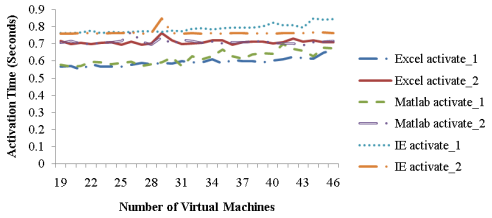

Fig. 9. Application Activation Times with increasing system loads

and *Memory Active* measurements have a negative slope since the memory overhead is increasing with the number of VMs, thus reducing the total amount of memory available to be used by the guest operating systems in the VMs. The actual value of *Memory Swapped* started at 240*MB* and increased to 3050*MB*, corresponding to a 1120% increase. At first the pages being swapped are inactive pages and are not likely to affect performance, but as the amount of swapping increases, the likelihood of active pages being swapped starts to negativly impact performance.

The time taken to open applications clearly increased with the increasing load as shown in Figure 7. The loading of applications is heavily dependent on transferring data from the hard disks into memory. When the memory granted to a VM is constricted due to the balloon driver, the VM must make room for this new application in memory. If the VM has not exhausted its memory shares in resource pool, the memory management tools and balloon driver of ESXi Server will decrease the memory pressure on a VM, thus granting the VM more memory to use for applications. However, if the VM has exhausted its share of the memory, the VM must invoke its own memory management tools and start deleting old memory pages using a garbage collection process, or swap them to its own virtual disk. These processes take time to complete, thus extending the application open times at the load increases. Excel, Internet Explorer, and Matlab went from 1.3*sec*, 2.3*sec*, and 10.8*sec*, to 5.9*sec*, 7.7*sec*, 38.5*sec* corresponding to 472%, 301%, 359% increase, respectively.

The time taken for actual tasks to complete within an application are shown in Figure 8. The task titled '*Matlab Graph spin*' first involved spinning a point-cloud model of a horse, and then pre-computing the surface visualization of the point-cloud data. The data sets are precomputed in order to limit CPU utilization and consume more memory. The task initally took 34*sec* and grew to take 127*sec*, corresponding to a 273% increase. This result highlights the fact that applications such as Matlab are highly sensitive to resource over-commitment and need special desktop provisioning considerations.

The Internet Explorer tasks involved loading a page with different types of content. The time taken to load a page of only an image saw the biggest increase starting at .75*sec* and grew to 2*sec*. The other two page types both remained under .5*sec* to complete, even under the highest system load. This increase, while statistically significant, is not obviously perceivable to the user. The task titled '*Excel Save*' is the time taken for 'Save As...' dialog box to appear. This Excel task originally took .9*sec* and later took 1.3*sec* only showing a 44% increase. Further, this Excel task is not heavily affected by the load increase and exhibited results very similar to the results in next section on application 'activation time' (illustrated in Figure 4) measurements.

The activation times taken to switch in-between applications are shown in Figure 9. The increase in activation times is on the order of one to two-tenths of a second and did not exceed one second. This is a very small change and is likely not perceivable to an user. The large percentage increases are due to the small timescale. The results are fairly random, but overall show a marginally increasing trend.

B. Slow-motion Application Interaction Results

Figures 10 - 13 show results from our slow-motion benchmarking testing of RDP, RGS, and PCoIP under a combination of both a range of network latencies (0ms, 50ms, and 200ms) as well as no packet loss and a relatively high packet loss of 3%. These network were selected because they provide insights into how these thin client protocols may behave on actual LAN, WAN and wireless last-mile connections.

Figures 10 and 11 show the total amount of data transmitted in bytes for each screen update. RDP transports the 'text only' web page with very minimal data transmitted and thus has a higher 'coding efficiency' for text. Both RDP and RGS maintain a relatively constant amount of data transmitted across increased latency and loss. The UDP-based PCoIP also in most cases exhibited a significant increase in the amount of data transmitted as latency and packet loss was increased. Despite this increase in data transmission, the user experience

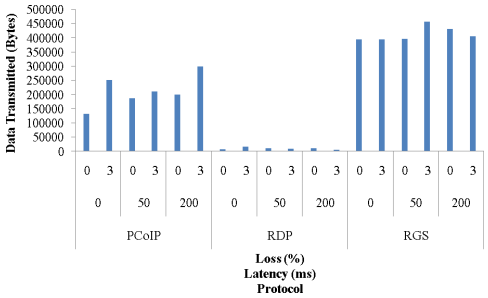

Fig. 10. Data Transferred for Text page under network emulation

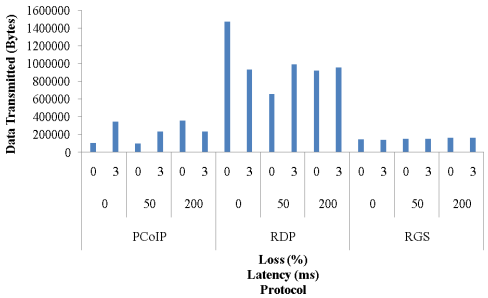

Fig. 11. Data Transferred for Image page under network emulation

subjectively is still very acceptable for PCoIP at relatively high packet loss rates of 3%. While we are viewing these thin-client protocols as generally black boxes, because the user experience remained acceptable at these high packet loss rates despite increased data transmission, we surmise that this spike in data transmission is most likely due to the addition of diagnostic data allowing PCoIP to adapt to the degrading network conditions. In the corresponding transmission time results, we observed an increase in latency for each of the protocols as network conditions degraded. While the TCP-based protocols, RDP and RGS, were affected by both increased latency and loss, the UDP-based PCoIP was largely unaffected by increases in latency.

Figures 12 and 13 show the bandwidth utilized in Megabits per second for each screen update. The ratio of the data transmitted and time taken are used to calculate the 'data transmission rate' or bandwidth consumed. We can also see that in general, there was a reduction in bandwidth utilized for

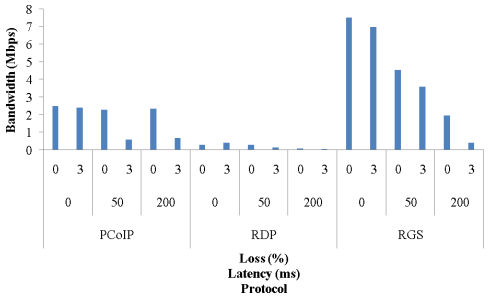

Fig. 12. Bandwidth consumed by Text page under network emulation

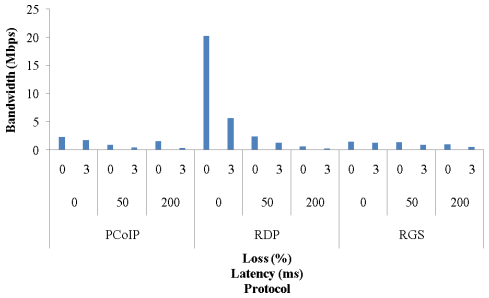

Fig. 13. Bandwidth consumed by Image page under network emulation

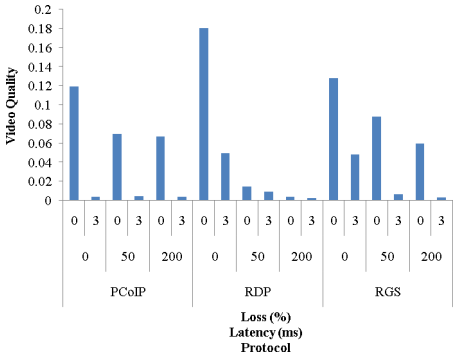

Fig. 14. Video Quality comparison under network emulation

each of the protocols when comparing low-latency, no loss with low-latency, and high loss network conditions. This is not surprising as this likely indicates a throttling of bandwidth utilization as packet loss increased. In addition, the bandwidth utilization was affected by increased transmission times under degraded network conditions.

Figure 14 shows the video performance of PCoIP, RDP, and RGS under a variety of network conditions. PCoIP under no loss conditions showed relatively stable performance with respect to increased latency. However, under high loss conditions, PCoIP suffered a severe drop in video quality. This behavior is most likely due to the display protocol being based on UDP, with recovery from loss being difficult. While RDP performed the best under idealized conditions, RDP suffered dramatically under either loss or high latency conditions. RGS was not as dramatically affected as RDP by increased latency. However, under high loss conditions, RGS had significantly reduced video quality performance.

C. *Performance Mapping to User Pool Profiles*

Service providers frequently make virtual desktop resource provisioning decisions based on *user application profiles* and *user group profiles*. To cater to such a need, we developed a feature in VDBench that can generate resource (CPU, memory, network bandwidth) utilization profiles for a sample scenario of different user applications (Matlab, Microsoft Office, Internet Explorer, Windows Media Player) and user groups (Engineering Site, Distance Learning Site, Campus Computer Lab) at sites accessing the virtual desktop cloud resources. The individual application profiles are combined to form user

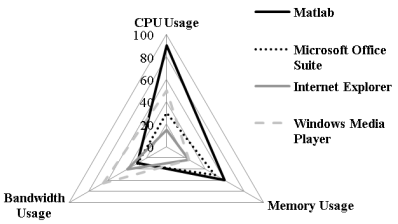

Fig. 15. User Application resource consumption profiles

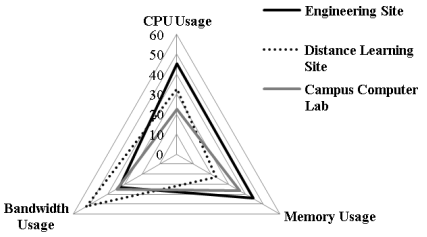

Fig. 16. User Group resource consumption profiles

group profiles by identifying the applications the user group uses, and by averaging the corresponding applications resource consumption values together.

Figures 15 and 16 show the user application and group profiles, respectively for our sample scenario. The CPU and memory usage values reflect true percentages of the VMs resources that each application used. The bandwidth usage is relative to each other using the Excel task as the baseline for the value. The majority of Matlab tasks use only CPU and memory resources for its' calculations. The bandwidth utilization of Matlab is low since most of the processing is done behind the scenes and results are displayed at the end. Windows Media Player uses the highest amount of bandwidth since it is transmitting full screen updates for the duration of the video. It also sees moderately high CPU usage for the decoding of high resolution video files, and compression of screen updates that are sent to client. The Microsoft Office Suite uses a moderate amount of all of the resources. Office Suite documents usually contain more rich multimedia elements such as diagrams or color images, resulting in similar bandwidth usage to Matlab. With the exclusion of Flash content, Internet Explorer uses minimal CPU and Memory, but it can use a notable amount of bandwidth depending on the richness of the web page content. The Engineering group see the highest CPU, and memory utilization with their heavy usage of Matlab and the Office Suite. The Distance Learning group consumes relatively the most bandwidth because it makes heavy usage of Windows Media Player to watch videos, and displays rich content in Internet Explorer and Office Suite.

The application and user group profiles are important to service providers because they can be used to infer how resource allocations can be balanced to ensure group profiles harmoniously share host/cluster resources. In the case of our sample scenario, the *Engineering Site* group profile indicates high CPU and memory utilization with low bandwidth consumption, and could be provisioned on the same host/cluster as the *Distance Learning Site* since its group profile has low CPU and memory utilization with high bandwidth consumption.

V. Conclusion

In this paper, we presented a virtual desktop performance benchmarking toolkit viz., "VDBench," which is used to simulate thin client user activity profiles and analyzes resource consumption characteristics. The toolkit uses a combiation of novel methodologies to automate scalability testing of server side hardware and reliability measurement of the network utilization of multiple display protocols using slow-motion benchmarking at different network health conditions.

Our work is unique as it is one of the few studies investigating the impact of increasingly constrained memory and network health conditions on the performance of various application tasks in a virtual desktop cloud environment. In contrast to slow-motion benchmarking studies previously conducted, we used server-side monitoring of the network to characterize the level of bandwidth required in order to deliver an optimal user experience under non-ideal network health conditions. We also investigated the CPU utilization in a variety of application tasks. By combining these CPU characterizations with memory and thin-client bandwidth related metrics, we were able to circumscribe the kind and amount of resources required to deliver adequate performance (i.e., satisfied user experience) for both individual applications as well as entire user groups. With the use of these benchmarking methodologies and metrics developed in the VDBench toolkit, service providers looking to deploy thin-clients based virtual desktop clouds will be able to greatly reduce the amount of costly guesswork and over-provisioning commonly encountered in this domain.

References

[1] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, A. Kemper, "Adaptive Quality of Service Management for Enterprise Services", *ACM Transactions on the Web*, Vol. 2, No. 8, Pages 1-46, 2008.

[2] P. Padala, K. Shin, et. al., "Adaptive Control of Virtualized Resources in Utility Computing Environments", *Proc. of ACM SIGOPS/EuroSys*, 2007.

[3] B. Urgaonkar, P. Shenoy, et. al., "Agile Dynamic Provisioning of Multi-Tier Internet Applications", *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 3, No. 1, Pages 1-39, 2008.

[4] H. Van, F. Tran, J. Menaud, "Autonomic Virtual Resource Management for Service Hosting Platforms", *Proc. of ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009.

[5] N. Zeldovich, R. Chandra, "Interactive Performance Measurement with VNCplay", *Proc. of USENIX Annual Technical Conference*, 2005.

[6] J. Rhee, A. Kochut, K. Beaty, "DeskBench: Flexible Virtual Desktop Benchmarking Toolkit", *Proc. of Integrated Management (IM)*, 2009.

[7] A. Lai, J. Nieh, "On The Performance Of Wide-Area Thin-Client Computing", *ACM Transactions on Computer Systems*, Vol. 24, No. 2, Pages 175-209, 2006.

[8] J. Nieh, S. Yang, N. Novik, "Measuring thin-client performance using Slow-motion benchmarking", *ACM Transactions on Computer Systems*, Vol. 21, No. 1, Pages 87-115, 2003.

[9] C. Waldspurger, "Memory Resource Management in VMware ESX Server", *ACM Operating Systems Review*, Vol. 36, Pages 181 - 194, 2002.

[10] R. Spruijt, J. Kamp, S. Huisman, "Login Virtual Session Indexer (VSI) Benchmarking", *Virtual Reality Check Project - Phase II Whitepaper*, 2010. (http://www.projectvrc.com)