Modeling Remote Desktop Systems in Utility Environments with Application to QoS Management

Vanish Talwar, Klara Nahrstedt, Dejan Milojicic

HP Labs and UIUC

Email: {vanish.talwar, dejan.milojicic}@hp.com,
klara@cs.uiuc.edu

Abstract

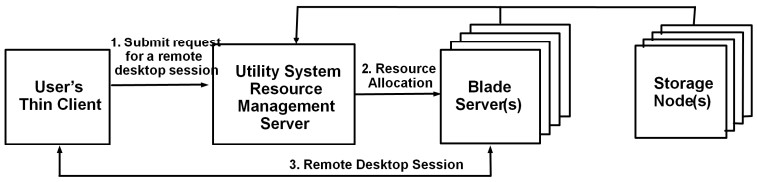
A remote desktop utility system is an emerging client/server networked model for enterprise desktops. In this model, a shared pool of consolidated compute and storage servers host users' desktop applications and data respectively. End-users are allocated resources for a desktop session from the shared pool on-demand, and they interact with their applications over the network using remote display technologies. Understanding the detailed behavior of applications in these remote desktop utilities is crucial for more effective QoS management. However, there are challenges due to hard-to-predict workloads, complexity, and scale. In this paper, we present a detailed modeling of a remote desktop system through case study of an Office application – email. The characterization provides insights into workload and user model, the effect of remote display technology, and implications of shared infrastructure. We then apply these learnings and modeling results for improved QoS resource management decisions – achieving over 90% improvement compared to state of the art allocation mechanisms. We also present discussion on generalizing a methodology for a broader applicability of model-driven resource management.

Table of Contents

Emergence of Remote Desktop Utility Systems

- Today's desktop systems in enterprises
  - are deployed as *fat clients*, heterogeneous, & spread across organizations
  - have high maintenance, management and security costs
- An alternative new computing model is emerging
  - consolidates desktops in virtualized shared resource pools of compute and storage servers; on-demand allocation of resources for a desktop session
  - has advantages of reduced infrastructure costs as well as maintenance costs
  - user interacts with his applications through remote display technologies, e.g. , Microsoft RDP, HP RGS, Citrix, VNC

Fig 1



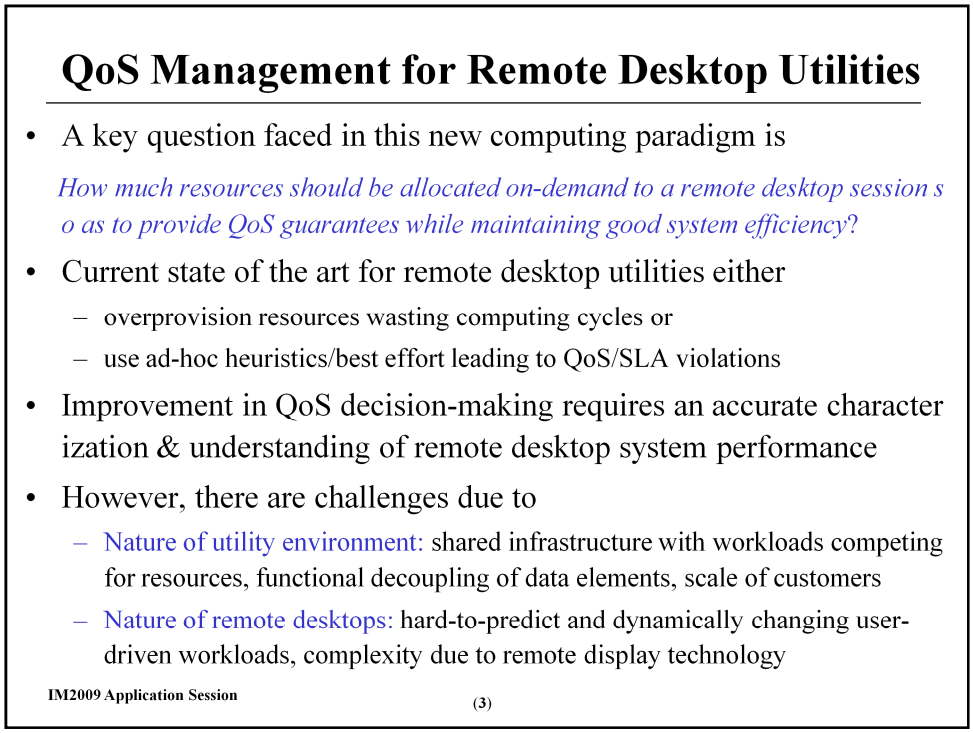
IM2009 Application Session

(2)

1. Introduction (Part 1 – Motivation/Background)

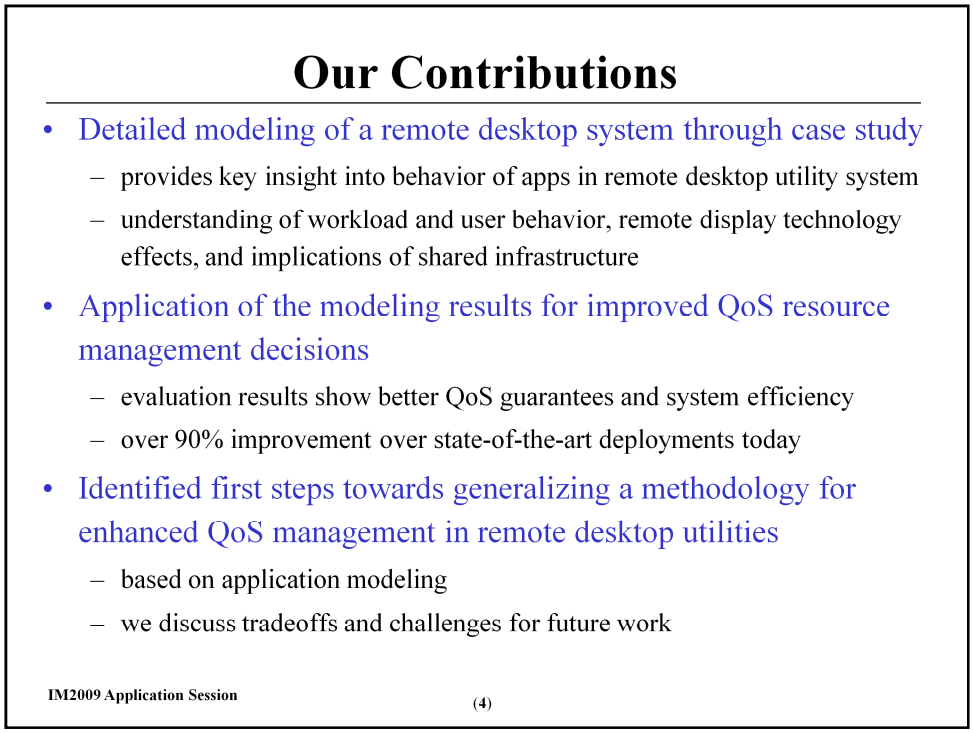
Traditionally, desktop systems in todays' enterprises are deployed as *fat clients* spread across organizations. These clients are heterogeneous in hardware and operating system, and there are hundreds of applications deployed on them. However, fat clients model has led to high maintenance, management, and security costs. With the growing complexity of software and hardware, such costs are expected to rise exponentially. A new computing model is emerging that proposes to consolidate the desktops in virtualized resource pools of compute and storage servers. The resource pool is *shared* across users, and resources are allocated *on-demand* to user applications. We term such a computing model a *remote desktop utility system*. Such a remote desktop utility computing model is envisioned to reduce infrastructure costs through sharing of resources; as well as management and maintenance costs by locality of on site repair, and eliminating duplication of processes and management of superfluous resources. The paradigm of a remote desktop utility system has been gaining popularity in various forms. Enterprises - in particular those in the financial, health-care, insurance, and design automation markets - are adopting such a system to consolidate desktops/workstations, and thereby improve manageability, resource utilization, and save costs [1, 2, 3, 4]. Example desktop applications that run in the utility are Office applications (Word, Excel, Outlook), financial stock broker and trading applications, and CAD/CAM applications.

In a remote desktop utility system, users request a remote desktop session through thin clients (see Figure 1). The management system allocates resources from the shared server pool on-demand for the session. After the allocation is successful, a connection is established between the users' thin client and the allocated compute server using remote display technologies such as Citrix, Microsoft's RDP, HP RGS [2], and VNC [5]. Once connection is established, the end-user may interactively start one or more applications within the session. Keyboard and mouse events are sent from the users' thin client to the remote compute server in the utility, and the output of the applications is exported from the remote compute server to the users' thin client.



1. Introduction (Part 2 – Problem)

The management system is thus a key component of the utility system responsible for allocation of appropriate resources to the remote desktop session meeting QoS guarantees and maintaining good system efficiency. Traditional resource management solutions used in commercial products today such as CCI, ClearCube [2,3] typically allocate an entire blade to the remote desktop session. While this provides good QoS guarantees, it leads to over-provisioned resources resulting in system inefficiency. On the other hand, other deployments, for example, using VDI deployments [2,4] use ad-hoc heursitics or best effort allocation of resources leading to under-provisioned resources and hence unsatisfied service guarantees. For a wide-spread adoption of a remote desktop utility system and to obtain the benefits of a consolidated infrastructure, it is important to improve the QoS management decision-making. However, this requires an accurate understanding and modeling of the behavior of applications in a remote desktop utility paradigm. Such an accurate characterization is challenging due to the nature of utility environment and remote desktops. A shared utility infrastructure leads to workloads competing for resources, scale of customers accessing shared services, and functional disaggregation of compute and storage elements. Additionally, remote desktop sessions have hard-to-predict user-driven workloads and increased complexity due to remote display technology effects.



1. Introduction (Part 3 – Contributions)

We make the following key contributions through this paper:

▪We present a detailed modeling of a remote desktop system through a case study of Office application. The modeling exercise has provided key insights into application behavior in remote desktop utilities, specifically improving the understanding of the workload and user behavior, remote display technology and implications of shared infrastructure.

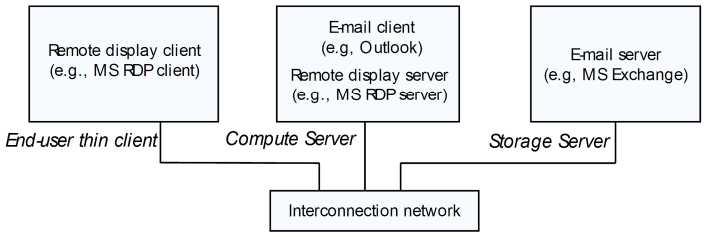
▪We applied the results of the modeling exercise to conduct an experiment showing the benefits of characterization results for QoS resource management decisions. The experiment involved the provisioning of email client sessions, and the perceived QoS and system efficiency were measured for a given user category. The results show that we can improve over 90% in system efficiency compared to typical commercial allocations while maintaining QoS guarantees. At the same time, we do better than ad-hoc best effort allocations by showing reduction of 90% SLA violations with our approach.

▪We also present discussion towards generalizing a methodology for providing enhanced QoS management in remote desktop utilities. Our case study evaluation enabled us to develop first steps in this direction and we identify tradeoffs and challenges for future work.

Modeling Illustration through Case Study

- Candidate use cases for remote desktop utilities fall under
  - market segments: health-care, government, financial, design automation
  - application types: office apps, financial apps, CAD tools
- We picked Office apps (specifically email) for our case study
  - Popular application and easy to setup
  - Less sophisticated than other enterprise applications (such as financial) but serves as a good starting point for a broader study

Fig 2



IM2009 Application Session

(5)

2. Remote Desktop Modeling

We approach the modeling derivation by choosing a case study example. While remote desktop utilities are gaining popularity in a variety of industries, for simplicity, we pick Office application for our case study. Within Office applications, we picked email. This application is relatively easy to setup even though not as sophisticated and complex as some other enterprise applications such as financial. Nevertheless, it provides a good starting point for the modeling research and as can be seen in the rest of the paper, it did provide sufficient complexity and representation for the purposes of our study.

The email application in the context of a remote desktop utility systems has the following components to consider: (i) a remote desktop client that sends keyboard and mouse events, (ii) a remote display server that exports graphics screen of applications, (iii) the email client application and associated processes hosted on the compute server, and finally (iv) backend servers hosting the email server to which the email client synchronizes to send and receive email. The email server also stores a permanent archived copy of the emails.

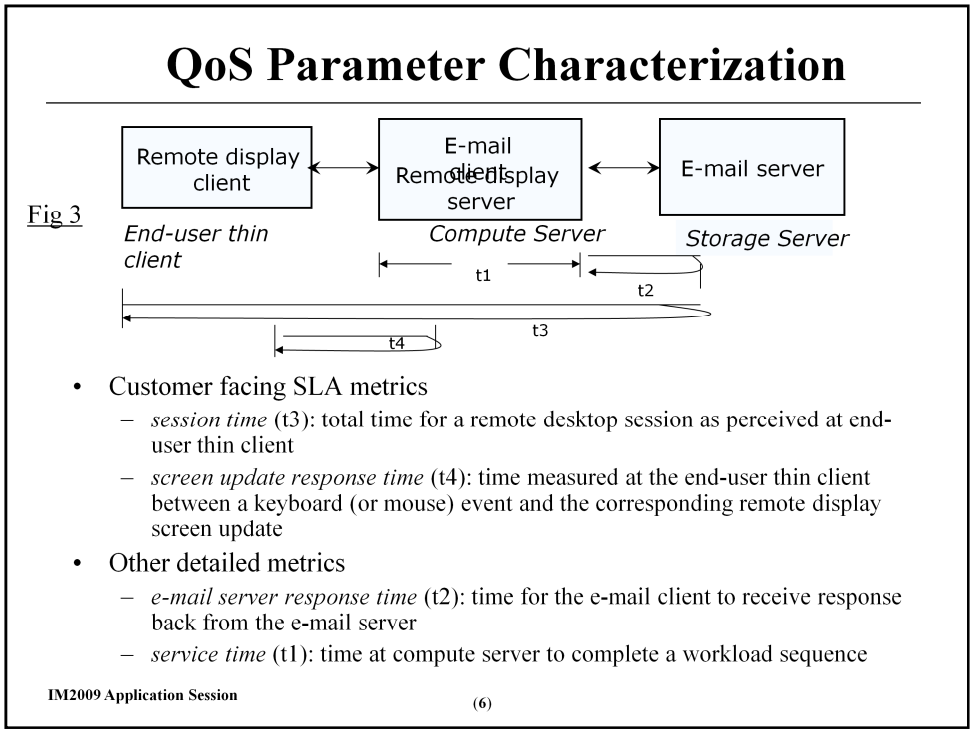
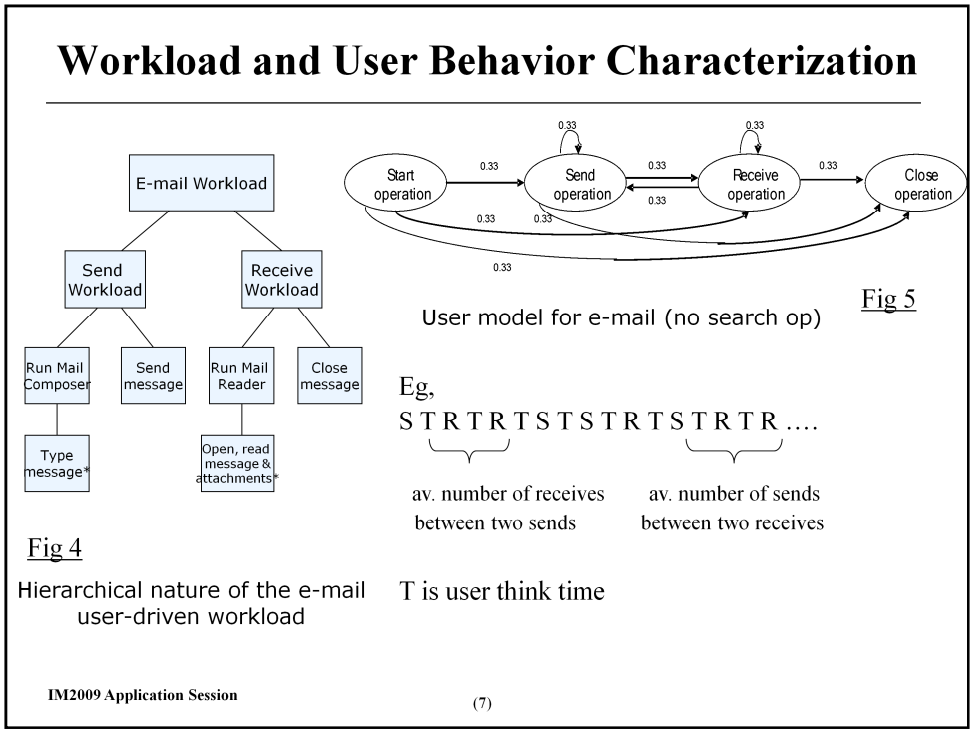


Fig 3

2.1 QoS Parameter Characterization

QoS parameters represent the performance metrics of interest. These parameters can be expressed at various levels. We find that for a remote desktop session, there are multiple QoS metrics to consider. At first glance, response time would seem to be the SLA metric that customers would most care about. However, there are two types of response times in remote desktop scenario. The first one relates to the responsiveness of the screen update. Specifically, this is the time between a keyboard (or mouse) event and the corresponding remote display screen update, and would include time spent by the application at the thin client, network, compute server, and optionally storage server in response to the keyboard (or mouse) event. We term this QoS parameter as *Screen Update Response Time* (Q1) and is measured at the end-user thin client. Another response time metric relates to the effectiveness of the remote compute server to complete the applications' tasks. At a coarser granularity, we can measure the time to complete an entire desktop session, inclusive of all of the activities – initiation, executing of multiple workload sequences, exit. We term this time as *Session Time* (Q3) and is the total time for a remote desktop session as perceived at end-user thin client. At a finer granularity, one can measure the time to complete a subset of a particular workload sequence, i.e., a set of requests that signify a significant task. Although this time can be measured at the thin client or at the compute server each with their own connotations, we consider measuring it at the compute server since parameter (Q1) is already capturing the effects of remote display protocol issues, and hence measuring the time at the compute server would capture the thin client independent aspects. We term this QoS parameter as *Service Time* (Q2). Finally, for bottleneck analysis, an administrator might be interested in the time to access shared storage services, in this case the email server. So, accordingly, *E-mail server response time* (Q4) – is the time for the e-mail client to receive response back from the e-mail server for a single invocation. These four parameters, termed (Q1) to (Q4) are the application QoS parameters that would represent characterizing an email remote desktop session. Among these, the session time and screen update response is what we would consider representative of customer facing SLAs. In addtion to these application QoS metrics, there are *system QoS* parameters too that would be interest, for example, to capacity planners. These parameters include resource utilizations (Q5) – those of CPU, memory, network, disk I/O. One imporant aspect to note is that unlike traditional applications, in a remote desktop session, we need to measure the utilizations of the remote display server as well in addition to the application usage. As would be seen in the remaining sections, this is an important aspect that cannot be ignored during resource management decisions.



2.2. Workload and User Behavior Characterization

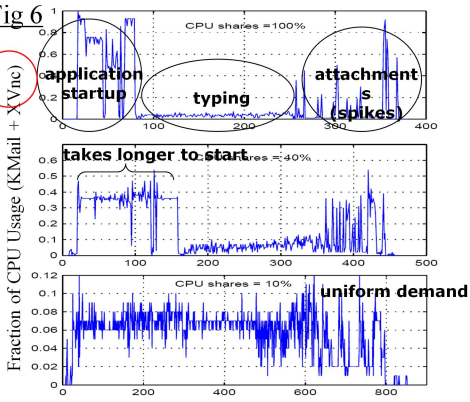
Modeling the workload characteristics is critical for the purpose of profiling the application performance. A remote desktop system provides challenges due to the user-driven nature of the workload. To understand this, we analyzed the email workload sequence and hierarchically represented it's interactive workload into subworkloads as shown in Figure 4 (send and receive). Each of these subworkloads can in turn have their own subsub-workloads. For example, a send workload consists of opening mail composer, typing a message, and then sending the message. While there can be other sub-workloads possible too with the email workload such as search, calendar etc., for simplicity and for the purposes of understanding, we restricted ourselves to just send and receive workloads. Now, an email workload alternates between various sub-workloads. This can be captured as a Markov model representing the user transitions among the top level subworkloads, e.g., sends and receives (see Figure 5). Thus, according to this abstract representation, an email workload can be considered as a sequence of send and receive operations Start S1 C R1 C R2 C S2 C S3 C R3 C S4 C R4 C R5 ... Close where C is cycle time (user think time). From an administrator and management perspective then, the important workload and user model parameters to consider given an email application hosting request are: *E-mail activity level:* this would translate into the probability distribution for the number of sends (or receives) between two receives (or sends) in the Markov model as well as the think time distribution; *Message activity level:* this would translate into load activity of the workload in terms of message size distribution for sends and receives (inclusive of attachments), as well as the typing and reading speed of email messages affecting CPU activity. Although these parameters are highly variable and user dependent, in practice, we believe enterprise users could be classified into well defined user categories based on the typical values for the above characteristics. Early attempts have been made to do so, e.g., knowledge worker, high-performance worker [15]. However, they have been ad-hoc so far and can be improved further using more formal parameters and models such as attempted above.

Varying the Degree of Sharing in Utility Infrastructure

CPU Utilization Characterization

- We vary the CPU shares (percentile) allocated to the session – less the shares allocated, more the number of desktop sessions that can be fitted onto the server (and thus more the degree of sharing of the utility infrastructure)

Fig 6



- Highly variable resource demands - even for a such a micro-benchmark
- Well correlated with user actions

Fig 7

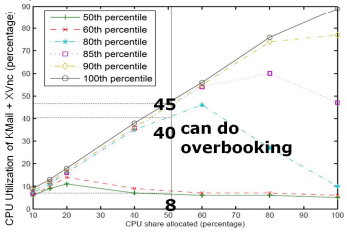
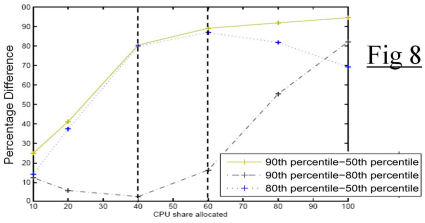


Fig 8



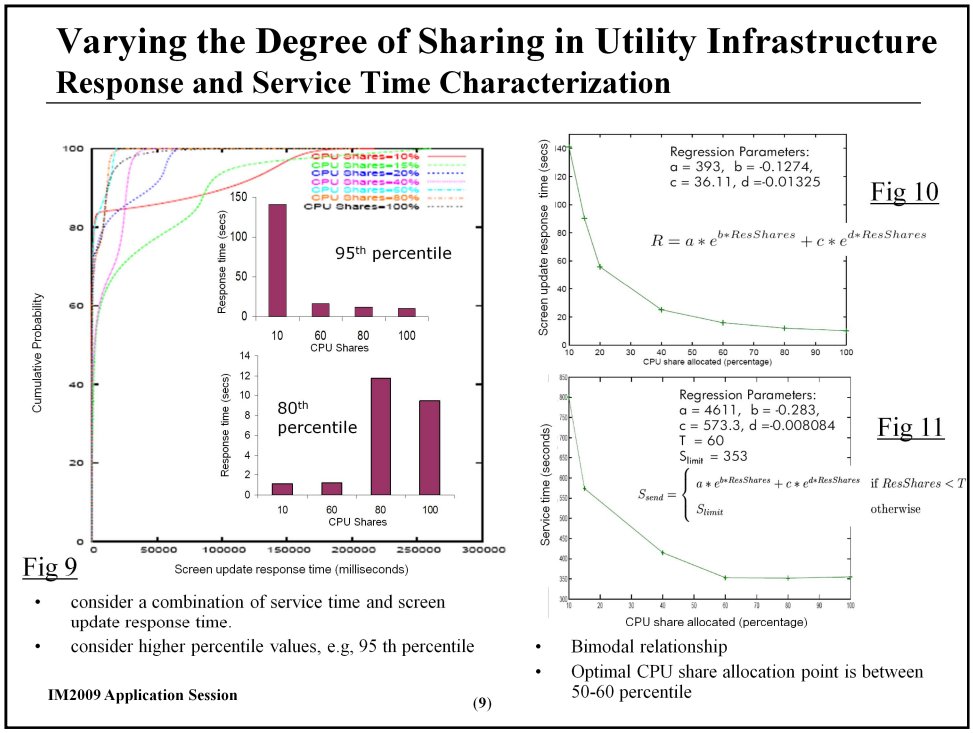
With increase in CPU share allocation, only a small fraction of requests utilize the available CPU utilization (optimal CPU share allocation point is between 40-60 percent)

2.3 Characterizing the Effect of Shared Utility Infrastructure

Next, we characterize the implications of a shared utility infrastructure for remote desktops whereby multiple sessions a re hosted on the same blade. Our study was to see the performance implications of this and if we can determine the optimal allocation points so that multiple sessions could be shared without interference. Our experimental testbed was Linux-based with TightVNC as the remote display technology, KDE KMail as the e-mail client, Postfix and Dovecot POP3 server as the e-mail servers. HP PRM [6] was used as a resource partitioning software to provide isolation among the multiple sessions. This software also provides knobs to control and guarantee the amount of CPU shares allocated to a session, enabling us to do controlled experiments for our characterizations. Networking effects were ignored as all the machines were connected to a local switch. We created our own benchmark suite based on a send workload at a typing speed of 40 words/minute. The benchmark opens the KMail composer, fills out headers, types an e-mail message , inserts attachments, and then sends the message. VNCPlay [8] was used for workload replays.

2.3.1 CPU Utilization Characterization

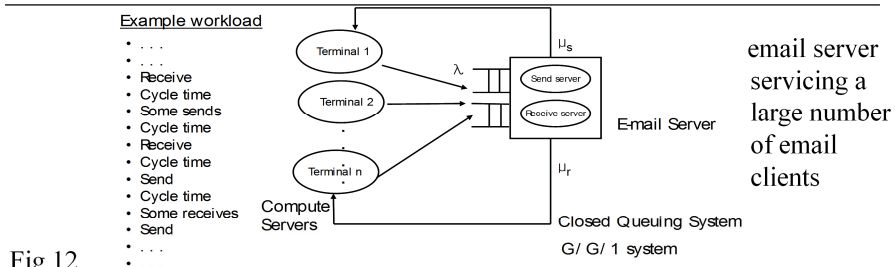
We performed the histogram and CDF analysis on the aggregated utilization measurements. Some key observations are summarized. *First*, Figure 6 shows the time plot of CPU utilization for the send benchmark, measured for different CPU resource share allocations. A visual analysis can show that even for such a small workload sequence, there is so much variation in the resource demands and sensitivity to user behavior. This high variance implies that merely considering a n average utilization as an estimate of resource demands would be deceptive and could lead to wasted cycles when there is a sudden change in demand (eg. during typing). At the same time, we also see that the demand variation is correlated to user actions – there is high utilization during application startup, low and flat utilization during typing phase, and spikes during final attachment steps. Combining the two, it can be recommended to have a white-box technique that is a ware of application behavior and used at runtime for fine-grained QoS management. *Second,* we draw inferences on the behavior of CPU utilization at various CPU share allocation points. See Figure 7 that shows the various percentiles for the CPU utilization of KMail+Xvnc (the VNC server process) for send benchmark. An interesting statistical inference from Figure 7 is illustrated in Figure 8. As we increase the CPU share allocation, the percentage difference between the higher percentile value (e.g., 90th percentile) and the lower percentile value (e.g., 50th percentile) keeps increasing at an exponential rate. The percentage difference between the 90th percentile and 80th percentile keeps decreasing till 40 percent CPU shares, after which it starts increasing. If we look at the percentage difference between the 80th and 50th percentile, it increases till 40 percent CPU shares, before slowing down and then eventually decreasing after 60 percent CPU shares, thus coming closer to the 50th percentile values. Figure 7 thus implies that with increase in CPU share all ocation, only a small fraction of requests utilize the peak available CPU utilization implying that one can obtain a good enough performance even with lower CPU share allocation. More importantly, the figure provides statistical estimates on what should the CPU share allocation be while satisfying resource demand needs of majority of KMail operations (between 40 and 60 percent in this case). This knowledge would be useful for allocation decisions as would be illustrated later on.



2.3.2 Response and Service Time Characterization

Figure 9 shows the CDF plot of screen update response time for different CPU shares. Some key inferences: *First*, the graph in Figure 9 shows an interesting anomaly – one would normally expect that as the CPU share allocation decreases, the response time should increase due to limited resource availability. This is reflected well for the 95th percentile screen update response time plot. However, if one plots the 80th percentile values, one sees an opposite behavior, specifically, the screen update response time at 10 percent CPU share is lower than that at say 100 percent ile CPU share. Let's see why. As the CPU share allocation is reduced, the user interactions with the system slows do wn since it now takes a longer time for any task to complete. The user thus adapts his interaction behavior, for example, with slower responsiveness, the user may start typing fewer characters before waiting for a screen update. This implies that there will now be a lot greater number of screen updates at lower CPU share allocation, each with a small latency value. As a result, we see in our example that at 80th percentile, the number of actual screen updates at 10 percent allocation is much larger than those at 100th percentile, and each of those screen updates have a very small response time value explaining the opposite behavior. However, there will still be some operations, for example, the initial application screen bring-up which would be unaffected by the user's adaptive behavior. For those small fraction of screen updates, the expected behavior would still hold true as is seen for the 95th percentile value. The important take-away is that even though the average screen update response time measured at lower percentiles may go down with decreased CPU share allocation, the service time is still going up and hence overall it is still taking a longer time for the entire workload sequence to complete. Hence, it is important to consider the characterization for both the screen update response time AND service time, i.e multiple QoS parameters, since their combination characterizes the responsiveness and the overall throughput of the system accurately. Considering only the screen update response time would deceive the system to do incorrect resource allocation. This inference ties back to the QoS characterization where too we realized the need to have multiple QoS parameters for remote desktop sessions. *Second inference*, we analyze further the 95th percentile screen update response time for different CPU shares (see Figure 10) and fit the curve using regression models. The relationship is exponential in nature with the response time starting to flatten out after about 60 percentile value; the equation is shown in the graph Similarly, we plotted the service time relationships (Figure 11) for different CPU shares. A visual analysis of this figure shows a clear bimodal relationship and after the CPU share allocation of 60 percentile, the service time maintains a constant value, thus even with a greater CPU allocation, the responsiveness is not affected . The service time values for CPU shares below 60 percentile is fitted using a regression curve and was also found to be exponential. The conclusions from these two graphs correlate well with those obtained from the utilization plots and that is -- an optimal allocation point can be found for the email application such that even if there are multiple sessions on the compute server, it will give good responsiveness (that optimal value is around 60 percentile in this case).

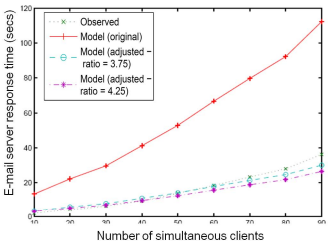
Characterizing the Effect of Scale and Dependencies

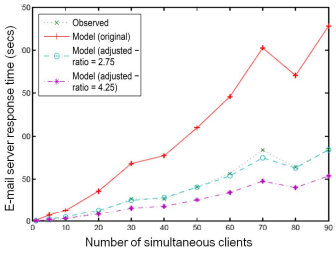


email server servicing a large number of email clients

Fig 12

$$R_{server} = \frac{N - S_{av} \pm \sqrt{(S_{av} - N)^2 + 4\rho\mu\ NS_{av}}}{2\rho\mu}$$





Analytical model validation

Fig 13

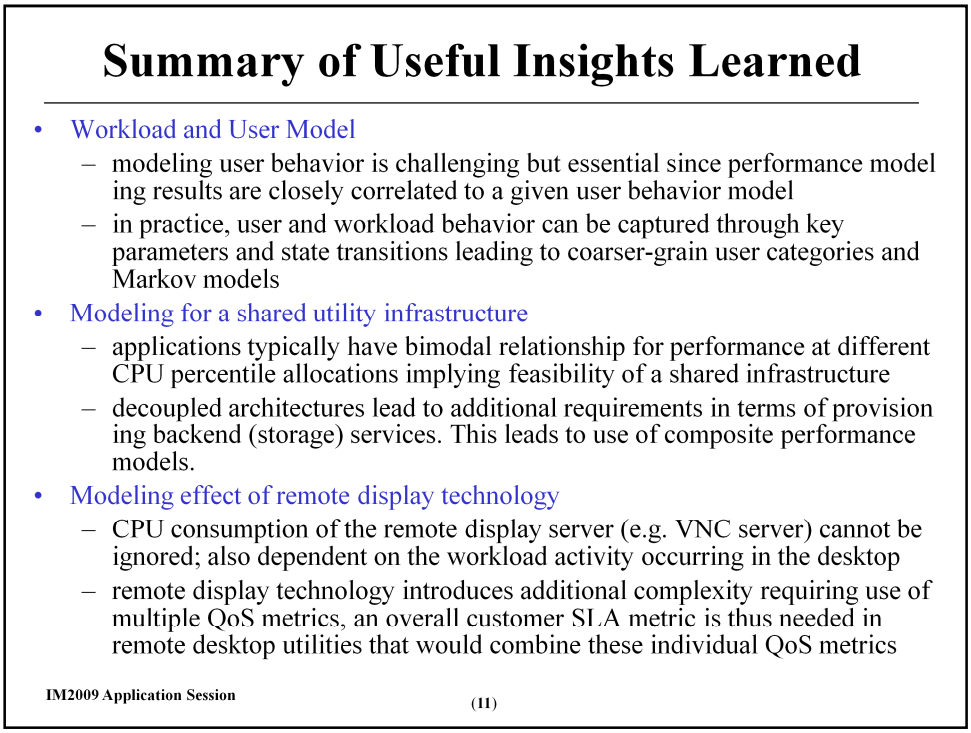
with empty message (630 bytes)

with message of size 123 KB (10

2.4 Characterizing the Effect of Scale and Dependencies

Given the decoupling of compute and data elements in the utility system, and the shared nature of backend storage services, we would like to characterize the effect of scale and dependencies to access such shared services. In the case of email, this implies characterizing the response time from email server. To determine the effects of scale, we use queuing theory to derive an a nalytical model for the response time from the e-mail server in the presence of N e-mail clients. Fig. 12 shows the queuing diagram for e-mail application - the customers to the e-mail server a re N e-mail clients hosted on the compute servers. The workload generated by them consists of a combination of sends and receives. Each e-mail operation (send or receive) waits for a response back from the email server before proceeding to the next operation in the workload. This behavior leads to a closed queuing system. We consider general service times at the e-mail server, and a general arrival rate, and hence the queuing system is a G/G/1 system. For lack of space, we don't show the detailed derivation of the email server response time, but the final equation is stated in Figure 12. This equation has been validated through an experimental setup in which we execute 1-100 simultaneous e-mail sessions with Postfix and Dovecot as the backend email servers (Fig. 13). Detailed explanation is omitted due to lack of space, but from the figure, one can observe that the response time predicted by the model, and that experimentally observed match and follow similar pattern. This implies that one can predict to sufficient accuracy the effect of scale using analytical methods in a remote desktop utility, and can use this prediction for allocation and placement decisions to meet desired QoS requirements for response time.

The question however is how this email server response time effects the end-to-end customer facing SLA metrics. In the presence of the functional data dependencies, we see that the total service time for a send request is the sum of (i) time taken at the compute server, and (ii) the e-mail server response time. Now, we have derived the former using the statistical performance model equation described in Section 2.3, and the latter has been derived using the analytical performance model equation described in this section. Hence, the extended performance model equations are given by $S'_{send} = S_{send} + R_{server;}$ $S'_{receive} = S_{receive} + R_{server.}$ The interesting implication to note is that this overall equation combines terms derived from statistical and analytical relationships – leading to composite performance models. With increasing complexity in applications and data centers such as that exhibited by the decoupled architecture, we expect this to take place more often especially as we try to increase the accuracy of characterization.



2.5 Summary of Useful Insights Learned

While the previous subsections have described in detail various interesting characterization results, we thought it useful to highlight and summarize a few key insights next.

Workload and User Model: Remote desktop systems do present several challenges in modeling their workload and user behavior and the process can be state-space explosive. However, it is important to model them as we found that in reality, the performance modeling results are closely correlated to a given user behavior model and must be considered so during any QoS allocation decision. We found that in practice certain key user parameters can be identified for a workload, and that similar patterns across classes of users can group users into categories. Similarly, detailed workload behavior can be captured formally as Markov models and this formal representation can be applied for runtime QoS management.

Modeling for a shared utility infrastructure: We found the email application to have a bimodal relationship in terms of performance at different CPU percentile allocations. If this we re true of other desktop applications too (preliminary analysis confirm this), then this is a good indication that sharing of infrastructure is quite feasible since overbooking could be performed without loss of performance guarantees. Formal statistical relationships can be easily formulate d and used for QoS allocations. Among other results, we did find however that changing the CPU percentile allocation can affect client perceived response time. This could in some situations affect the user reaction and behavior and could undermine the model predictions. The practical approach to handle this would be through runtime management wherein feedback on user behavior would be mapped to different workload states. Lastly, we find that decoupled architectures lead to additional requirements in terms of QoS provisioning of backend (storage) services to effectively handle scale – we also found that this can lead to development and use of composite performance models consisting of statistical and analytical relationships.

Modeling effect of remote display technology: We found remote display technology to be influential for performance characterization even under good network conditions. e.g., CPU consumption of the remote display server (e.g. VNC server) cannot be ignored and it is also dependent on the workload activity occuring in the desktop. Remote display technology also introduces additional complexity thus requiring use of multiple QoS metrics, this was determined both intuitively and practically validated. There is thus a need for an integrated cust omer SLA metric within remote desktop utilities that combines the individual QoS parameters.

Application to QoS Resource Management

- Scenario is to allocate 40 email client sessions in a remote desktop utility system
- Model-driven approach uses the characterization results and insights to decide on the resource allocation decisions
- QoS metric measured is response & service time; system efficiency is measured in terms of server utilization

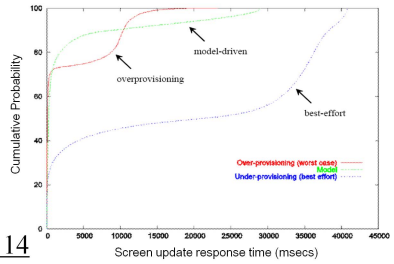


Fig 14

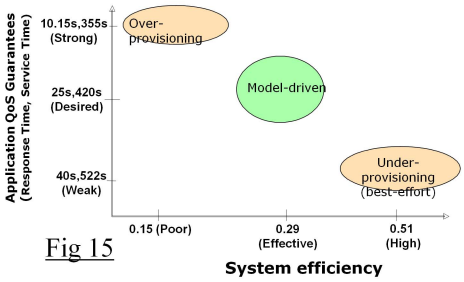


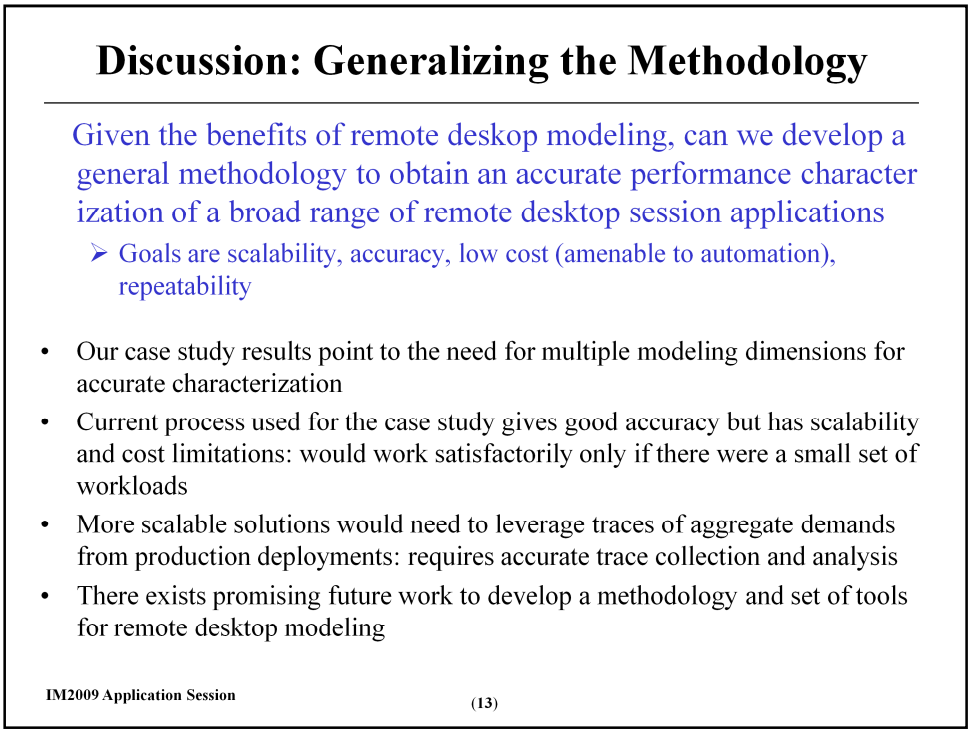
Fig 15

- ~93% improvement in system efficiency over widely-used commercial schemes that provide dedicated blades to customers, ~90% reduction in SLA violations compared to ad-hoc/best effort resource allocations

IM2009 Application Session (12)

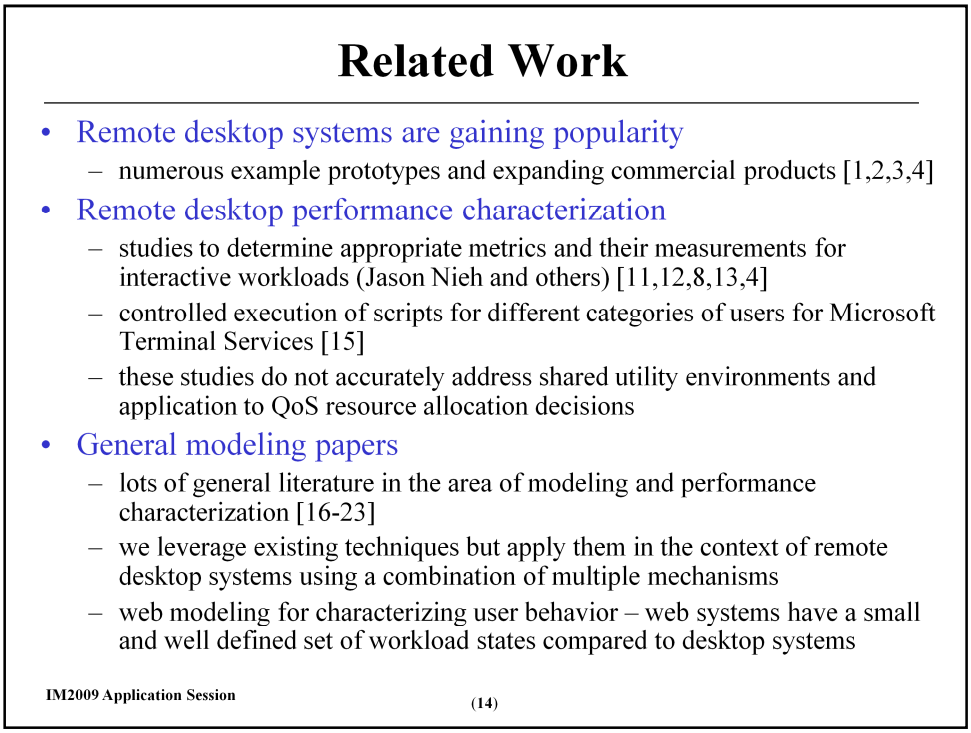
3. Application to QoS Resource Management

We conducted an experiment to show the benefits of the obtained characterization results for QoS management. The scenario is to host 40 e-mail client sessions in a remote desktop utility. The SLA desired for each session is 25 second or less for the screen update response time, and 6-7 minutes or less for the service time. We use three approaches – over-provisioning wherein a single blade is allocated to each session in the manner done today by many commercial deployments, best-effort where client sessions are allocated to servers using greedy heuristics and rules of thumb, and model-driven wherein the characterization results and equations are used for allocation decisions. The user model and workload is kept the same for the all the three approaches. We measured the QoS metric in terms of screen update response time and service time; average CPU utilization of the servers was also measured for inferring system efficiency. Figures 14 and 15 show the results. As expected, the over-provisioning case does poorly in system efficiency (0.15), model-driven does better (0.29), and best effort provides the best system efficiency (0.51). In terms of screen update response time, Figure 14 shows the cumulative probability distribution for the three approaches showing over-provisioning to do best with respect to QoS guarantees, best-effort the worst, and model-driven in between. Figure 15 combines the two results well and shows a visual comparison of the three approaches against application QoS metric (95th percentile) and system efficiency. As shown, with best effort, the response time is quite high and a bove the desired SLA. On the other hand, with over-provisioning, one achieves a very good screen update response time much below the desired SLA. Thus, with over-provisioning, the user has a very good experience and the QoS requirements are well satisfied. However, this has come at the expense of system efficiency. As pointed out earlier, with over-provisioning, the average CPU utilization is only 0. 15. With the model-driven approach, the 95th percentile response time value is 25.68 seconds as had been predicted using the model equations. Thus, with the model-driven approach, one not only satisfies the SLA desired, but also achieves a better system efficiency than over-provisioning, thus providing a better *combined* application QoS guarantees and effective system efficiency. More specifically, it achieves a 93% improvement in system efficiency over the over-provisioned single-blade allocations. Further, from Fig. 14, we see the model-driven approach satisfies the desired SLA for 95% of requests compared with 50% for best-effort – a reduction of 90% in SLA violations with a model-driven approach.



4. Discussion: Generalizing the Methodology

As we have seen, modeling provides administrators with useful insights which when reflected in QoS management decisions improves customer satisfaction while also improving data center efficiency. The obvious question then is: could we generalize a methodology for modeling in remote desktop systems, and could we build automation tools to aid administrators during the process? While this is a very desirable goal, it is a hard problem primarily due to two key factors. *First,* our study has shown that an accurate characterization needs to cover diverse needs, specifically, those of QoS characterization, workload and user behavior, dependency characterization, and characterizing effect of resource shares and of scale. Some of these need application domain knowledge, others experimental evaluation, and still others analytical mechanisms. While a lot of general literature on characterization and prediction mechanisms have existed and practiced along each of these dimensions [e.g., 16,17,18,19,20], they have been employed individually. From our case study example, it is clear that even though the characterization along each of the five points mentioned above will provide us insights into the behavior, on their own individual accord they do not provide all the details needed. Thus, a generalized methodology would need to consider multiple modeling dimensions. *Second,* any generalized methodology would need to be scalable to the number of applications and be cost effective, in addition to improving accuracy. Thus, questions such as: Can the methodology be applied to hundreds of applications? Can the process be fully automated and if so what is the complexity of the modeling mechanisms/algorithms? need to be addressed. The process used and described in our case study, for example, can be used to provide good accuracy but it has challenges when it comes to scalability. Nevertheless, it provides a good starting point and can be used for accurate characterization when the application set is small. For large scale, we would need to leverage a trace-driven approach where traces from past runs of the applications in remote desktop utilities are used to determine patterns if any. These can be combined with a profiling approach to provide cost and scale. However, a traces-based approach has the obvious challenge of obtaining accurate and detailed traces from production data centers. With rising popularity of the remote desktop computing paradigm, we are hopeful this problem will be addressed shortly.



5. Related Work

Remote desktop systems are gaining popularity and numerous commercial products exist in this space [1,2,3,4] providing motivation for our work. Research efforts have also been made to build complete desktop systems such as SLIM [9] and Collective [10]. In terms of remote desktop characterization, Jason Nieh et. al have conducted performance comparisons of various thin client systems [11, 12], and studies have been conducted to determine appropriate metrics and their measurements for interactive workloads [8,13]. There also exist controlled execution of scripts for different categories of users (knowledge, high performance etc.). to aid in capacity planning of Microsoft Terminal Services [15]. These studies though related do not address modeling of remote desktop systems in shared utility infrastructures in the manner that we consider. They also don't focus on applicability to resource allocation systems.

A lot of general literature exists on characterization mechanisms. [16] discusses histogram analysis, markov models, PCA analysis, regression models, queuing theory. Among other examples, trace-based profiling and workload characterization for multi-tier enterprise workloads has been undertaken by Rolia et al. [17]. application profiling has been pursued by Urgaonkar et al.[18], analysis of personal computer traces has been conducted by Smith et al. [19], analytical model based techniques has been pursued by Urgaonkar et al.[20]. In our study, we leverage these mechanisms and apply them to characterize a remote desktop utility system.

Conclusions and Future Work

- Remote desktop utility systems an emerging computing model
- Modeling remote desktop systems in such environments an important problem for effective QoS management
- We presented in this paper
  - remote desktop modeling results using case study example of Office apps (email)
  - showed benefit of applying results to QoS management decisions
- Future Work
  - model the effects of network latency and memory, impact of virtual machines
  - more evaluations on the applicability of modeling results for QoS management including for runtime management
  - develop a scalable and accurate methodology and automation tools for modeling applications in remote desktop utilities; leverage aggregated demand traces

6. Conclusions and Future Work

With growing popularity of the remote desktop utility systems, there is a need to understand the behavior of application performance in this paradigm and apply that knowledge for crucial QoS resource management functions. Through a case study example of an Office application (email), we walked through a detailed modeling of a remote desktop system and presented various insights along the way. We also applied the characterization results towards a resource allocation system and showed the benefits of the modeling results towards significantly improving the state of the art mechanisms. For future work, we plan to continue our modeling exercise to also characterize the effects of other resources (network, memory), as also the impact of virtual machines. Further, we plan to gather learnings from our modeling methodology, enhance it, and develop a set of scalable tools that would automate the derivation of models in remote desktop utilities.

References

[1] Forrester Research. Desktop Virtualization Is the Future of The Corporate PC. January 2006.
[2] HP. Consolidated Client Infrastructure. http://www.hp.com/go/cci; Virtual Desktop Infrastructure; http://www.hp. c om/go/vdi. Remote Graphics Software: http://www.hp.com/go/rgs
[3] ClearCube. http://www.clearcube.com.
[4] VMWare. Virtual Desktop Infrastructure. http://www.vmware.com/products/vdi/.
[5] T. Richardson, et. al. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.
[6] HP. Process Resource Manager (PRM). http://www.hp.com/go/prm
[7] K. Yaghmour et al. Measuring and Characterizing System Behavior Using Kernel-Level Event Logging. *Usenix* '00 .
[8] N. Zeldovich et. al. Interactive Performance Measurement with VNCplay. *Freenix* '05.
[9] B. K. Schmidt et. al. The Interactive Performance of SLIM: A Stateless, Thin-Client Architecture. *SOSP* '99
[10] R. Chandra et. al. The Collective: A Cache-Based System Management Architecture. *NSDI* '05.
[11] A. Lai and J. Nieh. Limits of Wide-Area Thin-Client Computing. *SIGMETRICS*'02.
[12] R. Baratto et. al. THINC: A Virtual Display Architecture for Thin-Client Computing. *SOSP*'05.
[13] J. Nieh et. al. Measuring Thin-Client Performance using Slow-Motion Benchmarking. *ACM TOCS*, 21(1), 2003.
[14] Y. Endo et. al. Using Latency to Evaluate Interactive System Performance. *OSDI*'96.
[15] Microsoft Corporation. Windows 2000 Terminal Services Capacity Planning. *Technical White Paper, 2000*.
[16] R. Jain, editor. *The Art of Computer Systems Performance Analysis*. Wiley, 1994.
[17] J. Rolia et. al. Statistical Service Assurances for Applications in Utility Grid Environments. *MASCOTS*'02.
[18] B. Urgaonkar et. al. Resource Overbooking and Application Profiling in Shared Hosting Platforms. *OSDI*'02.
[19] M. Zhou and A. J. Smith. Analysis of personal computer workloads. *MASCOTS '99*.
[20] B. Urgaonkar et. al. An Analytical Model for Multi-Tier Internet Services and Its Applications. *SIGMETRICS*'05.