CITI Technical Report 97–4

Modeling and Measurement of the PeopleSoft Multi-Tier Remote Computing Application

Yi-Chun Chu
ycchu@citi.umich.edu

Charles J. Antonelli
cja@citi.umich.edu

ABSTRACT

The Center for Information Technology Integration has undertaken a study of the potential impact of PeopleSoft software on U-M production computing environments, focusing on the measurement and modeling of the Peoplesoft 6 architecture in order to provide results useful in sizing various components of the M-Pathways campus deployment.

We have constructed a closed queueing network model of the two-tiered deployment architecture and evaluated the model using mean-value analysis. We also built a testbed and a database exerciser at CITI that were used to measure model parameters and perform model validation. Our testbed also yields preliminary data for a three-tiered deployment architecture planned by M-Pathways, which forms the basis for the next phase of our work.

December 1997

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

Executive Summary

In this first phase of work we have developed a Queueing Network Model for a two-tiered PeopleSoft 6 (PPS6) client-server system, built a performance modeling tool to evaluate the model via Mean Value Analysis, and developed *Stress*, a load generator that exercises Oracle database servers. The model outputs closely matched the measured values in those cases where the CPU and disk service demands remained constant; however, in several cases the service demands were found to vary with the number of clients, causing major discrepancies. We plan to refine the model in the next phase of work by dissecting the measured service demands into smaller components, using the *tkprof* tool.

We then performed some preliminary measurements when running PPS6 in a three-tier Citrix environment, both by monitoring network traffic and by measuring CPU utilization. First, we found that the average network traffic rate between Citrix WinStation clients and the WinFrame server consumes about 0.6% of a 10 Mbps Ethernet, so the network is not likely to be a bottleneck. Second, we measured the network communications overhead in terms of CPU time expended on client and server machines, and found that the Citrix client consumes about 96% of the client machine CPU; in contrast, the Citrix server consumes about 19% of the server machine CPU -- about as much as *Stress* does when run on the same platform. Finally, we noted a 14-17% increase in completion time when running *Stress* in the 3-tiered as compared with the 2-tiered environment. Our measurement tools do not allow us to account for CPU time charged to individual processes, so we cannot as yet correlate the increased completion time with the ICA server overhead.

We were not able to obtain an accurate characterization of typical PPS6 application workloads for inclusion in this study. Reasons for this include the inability to access real PeopleSoft applications running against real databases. As a result our models give results in terms of Oracle database operations instead of PeopleSoft panel accesses. In the next phase of work we plan to measure actual PeopleSoft applications and to record client, network, and server utilizations, as well as the stream of Oracle requests generated by each PeopleSoft panel. This request log will be replayed against a real database copy running on the real hardware, using a modified version of *Stress*. This work will require close coordination with M-Pathways development staff.

Modeling and Measurement of the PeopleSoft Multi-Tier Remote Computing Application

Yi-Chun Chu and Charles J. Antonelli

December 1997

1. Introduction

The Center for Information Technology Integration has undertaken a study of the potential impact of PeopleSoft software on U-M production computing environments. This study has focused on the measurement and modeling of the Peoplesoft 6 architecture [13] in order to provide results useful in sizing various components of the M-Pathways campus deployment [12].

Our strategy has been to construct both an analytic model of the M-Pathways deployment architecture and a representative testbed at CITI. Our testbed permits the measurement of relevant model parameters, which are then used to drive the model to generate predictive estimates.

In Section 2 we present the analytic modeling techniques used to model a two-tiered Peoplesoft 6 architecture. In Section 3 we outline the testbed and database exerciser we built to validate the model parameters and gather performance data. In Section 4 we present the results of some preliminary investigations of the Citrix WinFrame product [5], which is used to build a three-tiered architecture on top of Peoplesoft 6. Finally, in Section 5 we discuss the results obtained and set the stage for future work, and conclude in Section 6.

2. Analytic Modeling of Client-Server Systems

The performance evaluation of the PeopleSoft 6 distributed application (henceforth called PPS6) uses a methodology common to many performance evaluation projects [8,11]. The procedures comprising the methodology are:

- understanding the software architecture of PPS6,
- creating an analytic model of PPS6,
- building a performance modeling tool to evaluate the analytic model,
- measuring the modeling parameters,
- validating the performance model, and
- using the model to evaluate system performance of varying configurations.

We describe these procedures in detail in this and the following sections.

2.1 PPS6 Software Architecture

PPS6 is a two-tier client-server distributed application [13]. The future deployment of PPS6 will be augmented with the Citrix WinFrame [12], which adds a distributed Windows presentation capability. The deployed PPS6 architecture hence resembles a generic three-tier client-server application.[1] The software architecture of PPS6 augmented with Citrix WinFrame is shown in Figure 1.

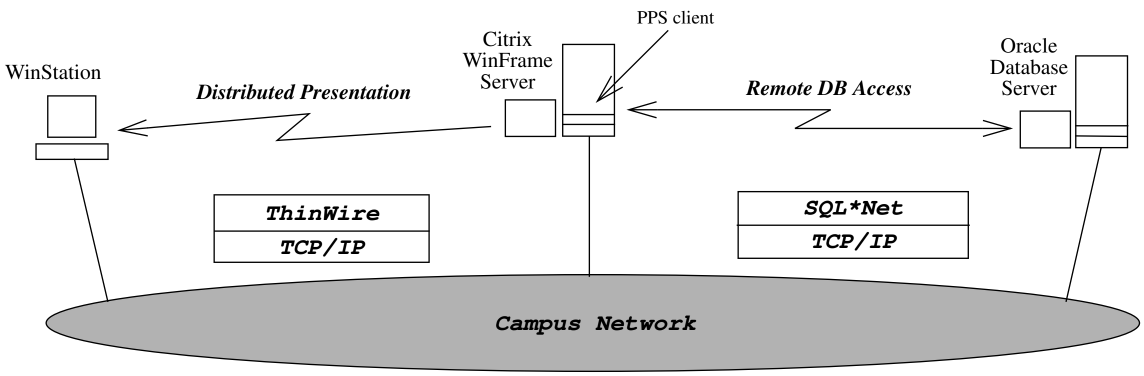


Figure 1. Software architecture of PPS6 with Citrix WinFrame extension.

PPS6 clients are "thick" clients. Each client program, called a *panel* in PeopleSoft terminology, contains business rules for data processing and provides a graphical user interface (GUI). PPS6 clients access remote Oracle databases by means of SQL*Net [15], which is the foundation of Oracle's family of networking products, providing network connectivity in distributed, heterogeneous computing environments.

Citrix WinFrame provides a distributed presentation service with its Intelligent Console Architecture (ICA) [4]. ICA, a general-purpose presentation protocol for Windows applications, is conceptually similar to the X-Windows protocol in UNIX platforms. Its efficient and secure presentation service is implemented in the ThinWire protocol, which supplements the underlying transport service with framing, compression, and encryption capabilities.

2.2 Analytic Modeling of PPS6 Client-Server Systems

It quickly became clear that developing a single model for the deployed PPS6 and Citrix WinFrame components would be very difficult, because a single model would be too complicated to build and impractical to evaluate with efficient algorithms. Therefore, a general approach for modeling a complicated system following the "divide-and-conquer" principle was used in this project [11]. First, we built an analytic model for PPS6. As a consequence, we will be able to model the entire deployed system as the WinFrame clients interact with a "flow-equivalence" service center representing the PPS6.

The analytic model of PPS6 which we built is a closed queueing network model (QNM) with a fixed number of customers (PPS clients). The choice of a closed model with fixed customer population reflects the situation when PPS6 is in production. A user submits a request by filling out a panel, waits for the result, and "thinks" a while before submitting the next request. A pictorial presentation of the model is shown in

1. A three-tier client-server distributed application has three software components. The first tier is composed of "thin" clients which provide the user interface (presentation). The second tier, called the application server, contains the business rules for data processing. The data are stored in the third tier, which is usually composed of database servers. The "thin" clients in the three-tier architecture contain the user interface only; in contrast, the "thick" clients in a two-tier architecture contain both the user interface and business rules for data processing.

Figure 2. It includes a fixed number of clients, a local network, and a database server (modeled as a server with one CPU and one disk).

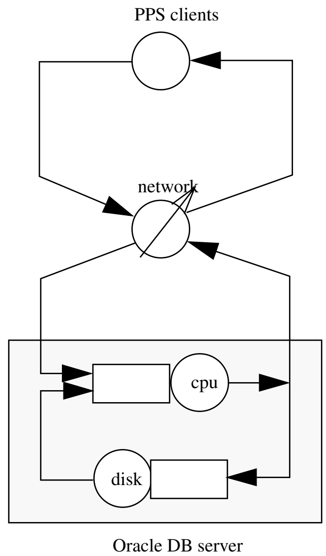


Figure 2. Queueing network model for the PPS 6 application.

The model in Figure 2 assumes that PPS clients and the database server reside on the same Ethernet network. Therefore, the local Ethernet can be modeled as a load-dependent service center[1] accessed by clients and the server. PPS clients are modeled as delay centers with fixed population and a constant think time (the elapsed time from receiving a previous reply to submitting another request). The database server is modeled as a load-independent service center with two devices: a server CPU and a server disk. Each class of client request[2] is carried out with a certain amount of server CPU time (CPU service demand) and server disk time (disk service demand), both assumed to be exponentially distributed. Therefore, client requests in the closed QNM are characterized as multi-class homogeneous workloads. The solution technique for the model should thus be able to evaluate QNMs with multiple job classes.

2.3 Solution Techniques for Evaluating QNMs

Mean-value analysis (MVA) is an efficient algorithm for evaluating closed QNMs with exact solutions [8, 10,11]. MVA computes only mean performance measures; variances of performance measures can not be computed with this technique. MVA can be enhanced to evaluate more complicated QNMs. First, it has been extended to evaluate closed QNMs with multiple job classes. Second, an evaluation technique for service centers representing load-dependent devices has been developed and integrated into the MVA algorithm. For QNMs with large numbers of job classes, however, the exact MVA algorithm could require excessive time and space to run. Therefore, an approximate solution technique, called the approximate MVA, is usually used in practice. Since the approximate MVA is quite accurate, it is useful as a general technique, even for QNMs that could be solved exactly [10].

In this project we have built a performance modeling tool, called *MVA*, for evaluating general closed QNMs. The tool provides a Tcl/Tk user interface for constructing analytic models and displaying model outputs after analysis. Its core evaluation technique is based on a multi-class, approximate MVA algo-

1. The network is modeled as a load-dependent service center because Ethernet causes varying delays for packet transmission under different workloads (network traffic).
2. The class of client requests is classified by the specific PPS6 panel generating the request.

rithm, called the *linearizer* [3]. We further enhance the linearizer to evaluate Ethernet networks (load-dependent devices). The following subsections present the modeling parameters and performance measures in our tool.

2.3.1 Modeling Parameters of Multi-Class MVA Algorithms

A multi-class closed QNM is described as a set of modeling parameters to be evaluated by a multi-class MVA. These modeling parameters are:

K	The number of servers.
C	The number of closed classes.
N_c	The population of closed class c. (c = 1,2,..., C)
Z_c	The average think time of closed class c. (c = 1,2,..., C)
$D_{c,k}$	The average service demand for a class c customer at server k.
	The service demand is the total service time required by a class c customer at a
	server k considering all visits made by the customer to that server and
	$= V_{c,k} * S_{c,k}$

where,

$V_{c,k}$	The average number of visits of a class c customer to server k per invocation.
$S_{c,k}$	The average service time for a class c customer when visiting server k.

MVA evaluates a QNM by determining the following performance measures:

$R_{c,k}(\overline{N})$	The average residence time for class c customers at server k with population $\overline{N}$, where $\overline{N}$ is the network population vector, $(N_1, N_2,..., N_c)$.
	It includes both queueing and service time.
$R_c(\overline{N})$	The average residence time for class c customers with population $\overline{N}$.
	It is defined as $\sum_{k=1}^{K} R_{c,k}(\overline{N})$.
$X_c(\overline{N})$	The total throughput of class c customers with population $\overline{N}$.
	It is defined as $\frac{N_c}{R_c(\overline{N}) + Z_c}$.
$Q_{c,k}(\overline{N})$	The average queue length for class c customers at server k with population $\overline{N}$.
$Q_k(\overline{N})$	The average number of customers at server k with population $\overline{N}$.
	It is defined as $\sum_{c=1}^{C} Q_{c,k}(\overline{N})$.
$U_{c,k}(\overline{N})$	The utilization of server k by class c customers with population $\overline{N}$.
$U_k(\overline{N})$	The utilization of server k with population $\overline{N}$.
	It is defined as $\sum_{c=1}^{C} U_{c,k}(\overline{N})$.

2.3.2 Model Representation in the *MVA* Tool

The *MVA* tool provides a friendly user interface for model construction. The modeling parameters of a closed QNM can be directly converted into equivalent modeling parameters in the tool. The equivalent model representation in the *MVA* tool for the analytic model introduced in Figure 2 is shown in Figure 3.

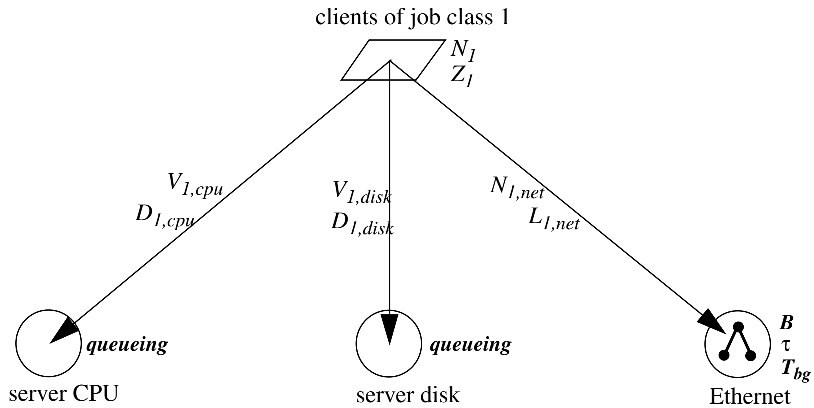


Figure 3. Model representation of the PPS6 in the *MVA* tool.

A job class is represented as a parallelogram with two parameters: the client population (N) and the average client think time (Z). A device is represented as a circle with its service discipline, either a *queueing* or *delay* service center, as its single parameter. A link connecting the parallelogram and a circle contains information about the service requirement for a job class visiting the device: its visit ratio (V)[1] and the average service time for each visit (D).

For models with multiple job classes, each job class is represented by a separate parallelogram. In addition, its service requirement is specified by links connecting to visited devices.

2.4 MVA Extension with the Ethernet Device

The analytic model of PPS6 includes a network device (a local Ethernet network) connecting client and server machines. The modeling parameters of an Ethernet device are determined differently from those of the previous section. Ethernet performance varies with network traffic due to access contention caused by the link-layer protocol, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) [6,7]. One approach for assessing the per-packet delay in the Ethernet network is to model the network as a load-dependent (LD) device, and solve the model by the extended MVA with LD service centers [10,11]. The corresponding LD service center models Ethernet contention as a certain number of stations simultaneously accessing the network. One disadvantage of this approach is the difficulty of assessing the per-packet delay when considering the interference traffic from sources outside the system under evaluation.[2] The background traffic problem can be solved by modeling an Ethernet network with respect to the overall packet rate and the average packet length rather than number of stations accessing the network [1,7,9].

Assessing the per-packet delay in Ethernet with respect to the average packet rate can be performed by the Ethernet delay submodel of NetMod [1]. The analysis technique of NetMod is based on decomposition. NetMod decomposes a network model with interconnected subnetworks into several related sub-

1. Visit ratio is the same as $V_{c,k}$, the number of visits of a customer to a server per invocation.
2. It is difficult to estimate how many stations besides the system under evaluation will simultaneously access the Ethernet and cause contention.

models, analyzes each submodel with corresponding closed analytic expressions, and then consolidates the results. The Ethernet submodel in NetMod applies Lam's formula for the per-packet delay [7], and Hammond and O'Reillys' formula for Ethernet utilization [8]. Lam's Ethernet delay model estimates per-packet delay with four input parameters: the transmission speed of Ethernet, the one-way propagation delay of the Ethernet network, the average packet rate, and the average packet length. The packet rate and packet length parameters are averaged from all traffic sources accessing the Ethernet. Background traffic is thus taken into account while estimating the per-packet delay.

Khandker has validated the feasibility of the integration of the linearizer and the analysis algorithm of NetMod [9]. He combines the above two algorithms with a small program which converts the output of the linearizer to NetMod, and vice versa. The combined method iterates one algorithm after the other, and uses the outputs of one algorithm as input parameters of the other until the two algorithms converge. In our implementation of the linearizer, we enhance his method by incorporating the Ethernet delay model into the core iteration of the linearizer directly. This new algorithm generates the same performance measures, but converges faster because fewer iterations are needed.

The enhanced linearizer introduces new input parameters for an Ethernet device:

B	The transmission speed (network bandwidth) of Ethernet. (10 or 100 Mbps)
τ	The one-way propagation delay of the Ethernet network.
T_{bg}	The background traffic of Ethernet in Mbps.

The visit ratio and service time of an Ethernet device are replaced by:

$N_{c,k}$	The average number of packets generated for class c customer per invocation.
$L_{c,k}$	The average packet length of all packets generated by a class c customer per invocation.

The average packet length, $\bar{L}_k$, for an Ethernet device k is estimated by the following formula:

$$\bar{L}_k = \frac{\sum_{c=1}^{C} X_c(\bar{N}) \times N_{c,k} \times L_{c,k}}{\sum_{c=1}^{C} X_c(\bar{N}) \times N_{c,k}}$$

The average packet rate, $\bar{R}_k$, is estimated as:

$$\bar{R}_k = \frac{T_{bg} + \sum_{c=1}^{C} X_c(\bar{N}) \times N_{c,k} \times L_{c,k}}{\bar{L}_k}$$

Both formulas use $X_c(\bar{N})$, the estimated throughput for a class c customer in the previous iteration, to compute the per-packet delay of Ethernet in the current iteration.

3. Measurement Methodology and Model Validation

A performance model aims at representing the behavior of a real system in terms of its performance. The input parameters for a performance model describe the hardware configuration, the software environment, and the workload of the system under study. The representativeness of a model depends directly on the quality of its input parameters [11]. This section presents two important steps for obtaining the input parameters for our analytic model: performance measurement and parameter estimation. We start this section with a short description of our measurement testbed.

3.1 Measurement Testbed

A measurement testbed for the PPS6 has been set up at CITI. The testbed is composed of a WinStation client (Windows NT Workstation 4.0 equipped with a 200 MHz Pentium processor and 32 MB RAM), a WinFrame server (Citrix WinFrame Server 1.6 equipped with two 200 MHz Pentium processors and 128 MB RAM), and a database server (IBM RS/6000 Model F30 running AIX 4.1 and Oracle server 7.3.1, equipped with 256 MB RAM). These three machines are attached to a private 10-Mbps Ethernet through a Cisco 1900 switch which also provides connectivity to the campus network.

A database exerciser, called *Stress*, is used as the load generator against the Oracle server in the testbed. *Stress* provides two sets of functions. First, it allows manual selection of database access functions. Second, it supports automatic request generation for synthetic database workloads. When used in this mode, *Stress* can compose a mixed workload from four SQL database calls (insert, delete, query, and update a record), each with different weights, and then run the workload repetitively against the specified database while collecting performance statistics.

3.2 Performance Measurement

The purpose of the performance measurement step in our methodology is to provide measurement data for determination of model parameters. Our analytic model requires two sets of input parameters: the server CPU and disk service demands for carrying out a *Stress* request; and the number of packets generated by the request and the average packet length. The first set of parameters can be derived from the CPU and disk utilization measured by *iostat* - a system monitoring utility. The number of packets and the average packet length can be obtained with *tcpdump* and *tcptrace* - packet capturing and analysis tools.

Iostat is a general UNIX monitoring utility available across different vendor platforms. It can report system statistics of CPU and disk utilization at specific time intervals with the granularity of seconds. Our measurement plan monitors the following performance measures:

- *%user* — The percentage of time the system unit spent in execution at the user (or application) level.
- *%sys* — The percentage of time the system unit spent in execution at the system (or kernel) level.
- *%idle* — The percentage of time the system unit was idle with no outstanding disk I/O requests.
- *%iowait* — The percentage of time the system unit was idle waiting for disk I/O to complete.
- *%tm_act* — The percentage of time the physical disk was active (busy).
- *tps* — The number of disk transfers per second.

The above performance measures can be converted directly into CPU and disk utilization. For example, the server CPU utilization (*%cpu*) is the sum of *%user* and *%sys*; and the server disk utilization (*%disk*) is the same as *%tm_act*.

Tcpdump is a portable UNIX tool for displaying the headers and payload of packets captured on a network interface. *Tcptrace* is a UNIX tool for analyzing the packet traffic captured by *tcpdump*; for example, it can reconstruct and compute statistics about TCP connections established between pairs of hosts. For each type of *Stress* request, the number of SQL*Net packets ($N_{c,k}$) generated and the average packet length ($L_{c,k}$) are derived from the packet trace collected by *tcpdump*. Since each type of *Stress* request always generates the same number of SQL*Net packets, we count each packet and average the packet length manually. For SQL requests generating a large number of packets, *tcptrace* can produce a statistical analysis of the packet trace.

For model validation purposes, three different types of *Stress* requests - *Stress insert*, *Stress update*, and *Stress delete* - have been measured in the testbed. Each performance measurement is conducted over 10,000 consecutive requests to minimize the boundary effect of measurement. The elapsed time of 10,000 requests is measured by *Stress* itself, and is used to derive the average completion time per request (the elapsed time between *Stress* generating a database call and *Stress* receiving the last result packet from the Oracle server) and the server throughput. At the same time, we measure the average server CPU utilization and disk utilization with *iostat*. They are used to estimate the CPU and disk service demands per request in the next subsection.

For each type of *Stress* request, we repeat the same measurement with different numbers of concurrent clients. The measurement data are presented in Tables 4-6. For the second set of model parameters, Table 7 shows the number of SQL*Net packets per *Stress* request and the average packet length for each type of request.

Table 4. Measurement data for *Stress insert*.

# of clients	%cpu	%disk	C (ms)	X	D_{cpu} (ms)	D_{disk} (ms)
1	20.19	57.66	59.79	16.73	12.07	34.47
2	32.35	90.70	75.43	26.52	12.20	34.21
3	39.31	90.38	92.46	32.45	12.11	27.85
4	45.89	91.21	106.20	37.67	12.18	24.21
5	52.04	91.07	115.44	43.31	12.02	21.03
6	56.45	89.66	134.73	44.54	12.68	20.13
mean					12.21	26.98
stdev					0.24	6.31

Table 5. Measurement data for *Stress update*.

# of clients	%cpu	%disk	C (ms)	X	D_{cpu} (ms)	D_{disk} (ms)
1	24.74	40.99	79.58	12.57	19.69	32.62
2	44.40	69.59	96.72	20.68	21.47	33.65
3	55.62	78.43	117.93	25.44	21.86	30.83
4	65.18	80.52	137.78	29.03	22.45	27.24
5	72.23	81.16	158.23	31.60	22.86	25.68
6	76.24	81.11	184.41	32.54	23.43	24.93
mean					21.96	29.24
stdev					1.31	3.66

Table 6. Measurement data for *Stress delete*.

# of clients	%cpu	%disk	C (ms)	X	D_{cpu} (ms)	D_{disk} (ms)
1	24.38	42.85	82.44	12.13	20.10	35.33
2	42.43	69.91	99.83	20.03	21.18	34.89
3	54.07	79.98	122.04	24.58	21.99	32.54
4	62.64	81.89	143.76	27.82	22.51	29.43
5	68.70	82.63	166.44	30.04	22.87	27.50
mean					21.73	31.94
stdev					1.11	3.41

C: request completion time.
X: server throughput.

Table 7. Other *Stress* request parameters.

request type	N_{pkt}	L_{pkt}	D_{client}
Stress insert	6	115.67	16.12
Stress update	16	123.94	26.46
Stress delete	16	113.44	28.51

3.3 Parameter Estimation

Parameter estimation deals with the determination of input parameters from measurement data. Many times, performance measurement tools do not provide enough information for calculating the input parameters required by a performance model. Therefore, inferences have to be made to derive the desired parameters [11]. In this subsection we show how the CPU and disk service demands can be derived from the measurement data in the previous subsection.

For each performance measurement, we set up *Stress* to generate synchronous consecutive requests. At the same time, the number of requests (n) carried out within the measurement interval (T) is measured along with the server CPU utilization and disk utilization statistics. Therefore, the average completion time (C) per request is estimated as T/n and the average server throughput (X) is estimated as n/T requests per second (rps). The average CPU service demand is estimated from the $\%cpu$ as:

$$D_{cpu} = \frac{\%cpu \times T}{n}$$

Similarly, the average disk service demand is estimated from %disk as:

$$D_{disk} = \frac{\%disk \times T}{n}$$

The estimated service demands for each type of *Stress* request are also shown in Tables 4-6. For each type of *Stress* request, the number of *Stress* clients varies from one to six except for *Stress delete*.[1] The mean and standard deviation of the estimated service demands are listed at the end of each table.

Model parameters, such as CPU or disk service demand, usually use the mean of all measurement data available. The *MVA* tool can determine accurate performance measures if the variance of measured service demands is relatively small in comparison to their mean. For *Stress* requests, however, the measured service demands vary with the number of concurrent *Stress* programs: the CPU service demand increases slightly with more concurrent clients and the disk service demand apparently decreases with more concurrent clients.[2] This undesired phenomenon will cause a discrepancy between the measurement data and the model outputs because the *MVA* tool evaluates analytic models with fixed service demands for all client populations.

3.4 Model Validation and Performance Evaluation

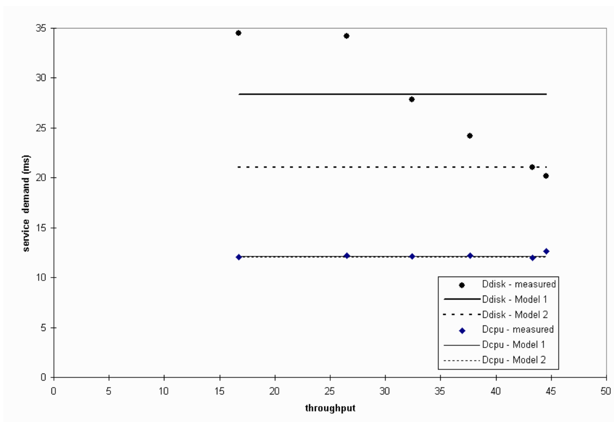
Considering the disparity in the measured service demands, we evaluate the analytic model with two sets of model parameters. The only difference between the two sets of parameters is their CPU and disk service demands: one uses the mean value of all measured service demands (called Model 1) and the other uses the exact measures obtained with five concurrent *Stress* clients (called Model 2).

When the input parameters are applied, the *MVA* tool determines the performance measures for a fixed client population (number of clients) with zero think time. We increase the client population one at a time until the disk is saturated. The model outputs and the measurement data are compared with three perfor-

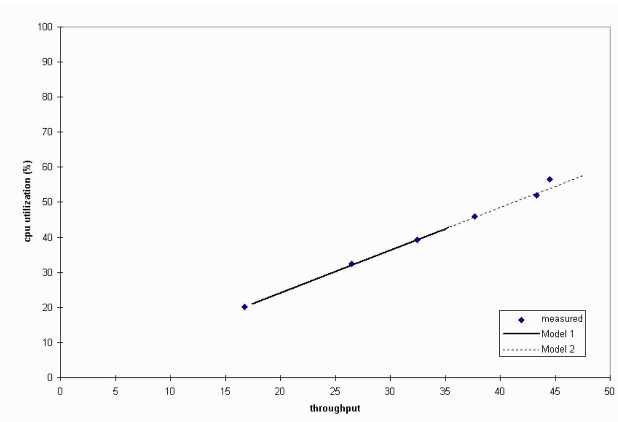
1. Performance measurement for *Stress delete* with six clients fails because one client machine is saturated.
2. We are still investigating the reason why the disk service time decreases while the CPU service time increases with more concurrent clients accessing the Oracle server. We believe several reasons may cause the phenomenon. First, it is most likely due to the Oracle database caching. Second, certain disk optimizations could reduce the disk service time in the presence of increased pending disk I/O. Finally, the increasing CPU service time is probably due to the lock contention for several concurrent Oracle connections accessing the same database.

mance curves: CPU utilization, disk utilization, and completion time (the average residence time in a job class) against throughput. These performance curves are shown in Figures 8-10.

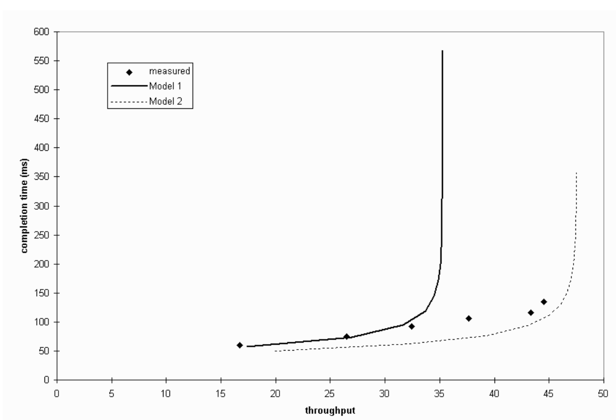
Figure 8a shows the service demands of *Stress insert* from the measurement data, and from the input parameters of Model 1 and Model 2. Among three different *Stress* requests, *Stress insert* has the largest disparity in its measured disk service demand: it drops from 34.47 ms with one client down to 20.13 ms with six clients. This causes about 7.32 ms of difference in disk service demands between Model 1 and Model 2. The model outputs for disk utilization (in Figure 8d) and completion time (in Figure 8b) thus reflect this disparity and cause a major discrepancy between model outputs and measurement data However, since *Stress insert* has a very small disparity in the measured CPU service demands, the CPU utilization from the model outputs closely matches the measured CPU utilization (in Figure 8c).



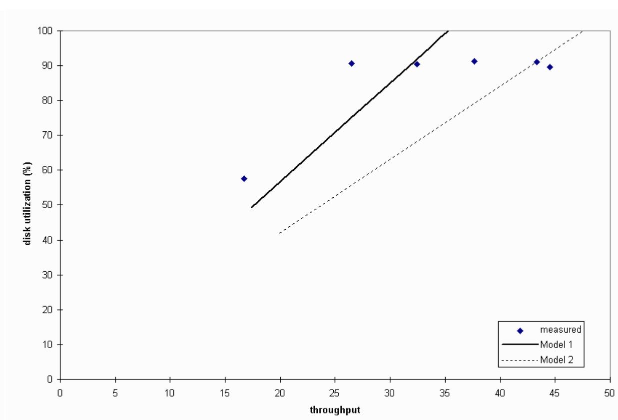
(a) service demands



(c) CPU utilization



(b) completion time



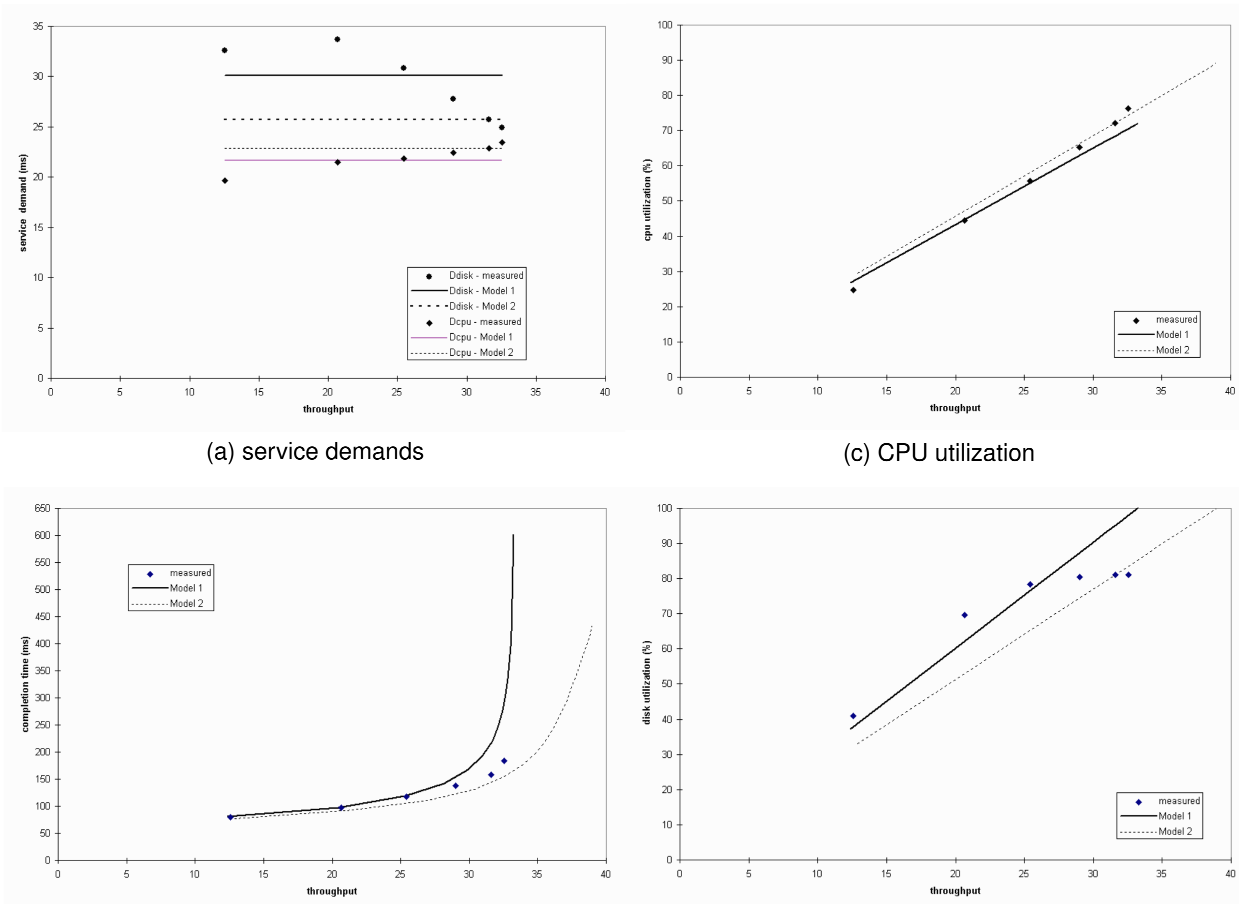
(d) disk utilization

Figure 8. Comparison of measurement data and model outputs for *Stress insert*.

The *MVA* tool determines that the Oracle server saturates with throughput of 35 rps for Model 1 and of 47.55 rps for Model 2. The maximum throughput can also be derived from the inverse of the disk service demands (1/0.028 and 1/0.021) applied to the model because the disk is the bottleneck device in this case.

For *Stress update* and *Stress delete*, the disparity among all measured disk service demands is similar to, but not so significant as in *Stress insert*. However, another undesired phenomenon arises in that the measured CPU service demand increases with more concurrent clients. This causes a small discrepancy in

CPU utilization between the model outputs and the measurement data. In general, the same analytic model does a better job of predicting performance measures for *Stress update* and S*tress delete* than for *Stress insert* because the former have a smaller disparity in measured service demands.



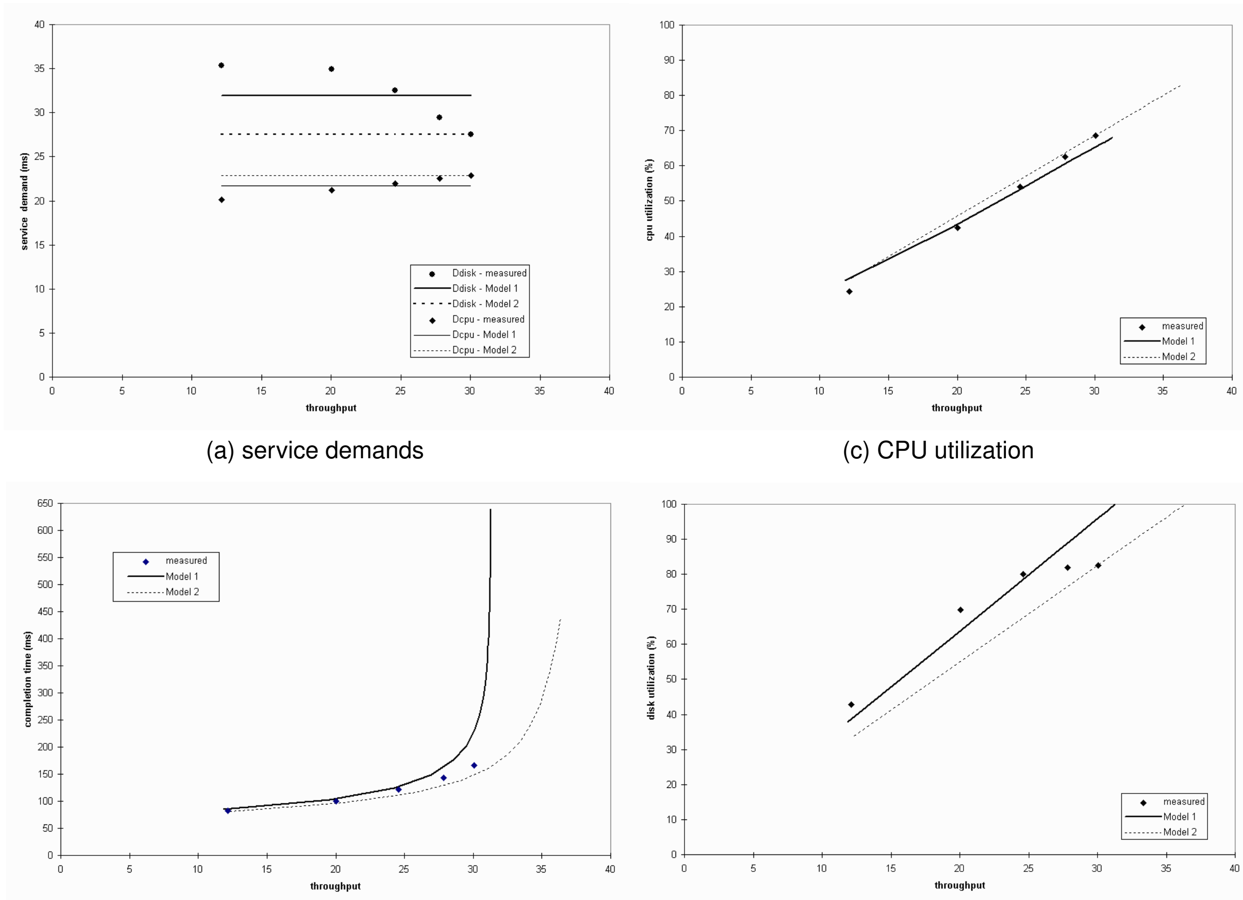
(a) service demands (c) CPU utilization

(b) completion time (d) disk utilization

Figure 9. Comparison of measurement data and model outputs for *Stress update*.

From the above performance experiments, we find the analytic model does not predict the performance satisfactorily because of the discrepancy between the model outputs and the measurement data. More specifically, the general model assumption about the fixed service demands is not valid because the measured service demands actually vary with the number of concurrent clients. As a result, the first step for model refinement is to identify the actual cause for the disparity in service demands at the Oracle server. This task requires better monitoring tools capable of decomposing the execution of an SQL request into smaller components. In addition, we also have to characterize how service demands vary with the number of concurrent Oracle connections. We discuss a plausible approach in our future research plan in Section 5.

For capacity planning purposes, values for two more parameters are required to complete our model: the client population and the average "think" time for each job class. Unlike the previous model parameters, they are usually determined by a capacity planner based on his or her experience and intuition. However, statistical analysis of measurement data from production systems can help determine these two parameters more accurately. Finally, we do not evaluate the analytic model with multiple job classes (such as mixed *Stress* requests) because our measurement techniques cannot provide the performance measures to validate the model outputs. Since capacity planning usually evaluates the analytic model with multiple job classes, we are investigating the Oracle accounting facility to fulfill this purpose.



(a) service demands

(c) CPU utilization

(b) completion time

(d) disk utilization

Figure 10. Comparison of measurement data and model outputs for *Stress delete*.

3.5 Regression Analysis of Service Demand*s*

Regression analysis is a general technique for finding a formula consistent with a set of data that can be used for predicting values outside the range of the original data [8]. The technique is applied here to estimate service demands with respect to the number of concurrent clients. In this subsection, we exercise this technique with the performance measures of *Stress delete* and compare the result with the previous model outputs.

As mentioned in the previous subsection, the *MVA* tool evaluates the model with different client populations (number of concurrent clients), but with the same CPU and disk service demands. Since the measured service demands actually vary with the client population, it is reasonable to change them as we change the client population. However, an immediate problem arises because we do not have a complete set of performance measures for model evaluation. Therefore, we apply the regression analysis to estimate the service demands that could not be measured (for six or more concurrent clients) based on the service demands we measured (for one to five concurrent clients).

To discover the trend of the service demands for *Stress delete*, the measured values are plotted against the number of concurrent clients (the dots in Figure 11). This helps us to determine what kind of regression methods should be chosen for each case. In this example, we chose linear regression for the CPU service demand, (the dashed line in Figure 11) and curvilinear regression for the disk service demand (the bold curve in Figure 11). The regression analysis thus provides the *MVA* tool with the estimated service demands of different numbers of concurrent clients for model evaluation. The regression model outputs are shown in Figure 12.

# of clients	measured		regression	
	D_{cpu}	D_{disk}	D_{cpu}	D_{disk}
1	20.10	35.33	20.35	36.91
2	21.18	34.89	21.04	33.88
3	21.99	32.54	21.73	31.49
4	22.51	29.43	22.42	29.54
5	22.87	27.50	23.11	27.91
6			23.79	26.53
7			24.48	25.33
8			25.17	24.28
9			25.86	23.36
10			26.55	22.53
11			27.24	21.78
12			27.92	21.10
13			28.61	20.49
14			29.30	19.92
15			29.99	19.40
16			30.68	18.91
17			31.36	18.47
18			32.05	18.05
19			32.74	17.66
20			33.43	17.29

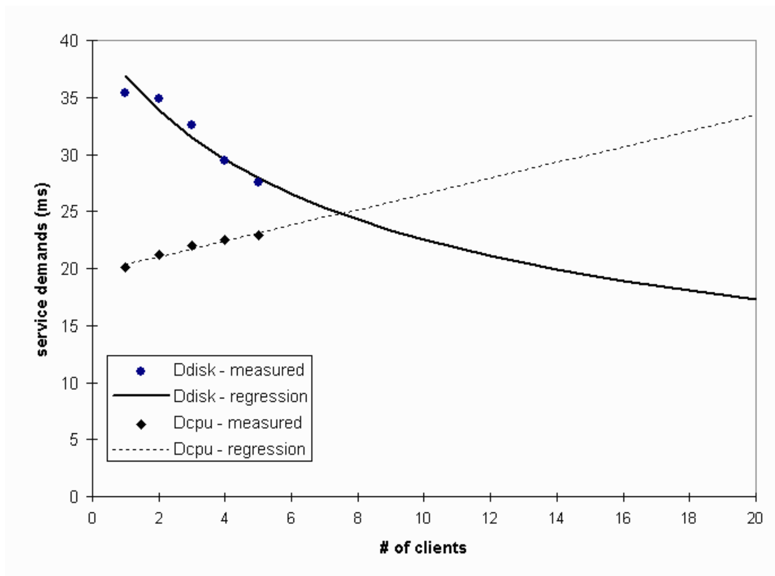


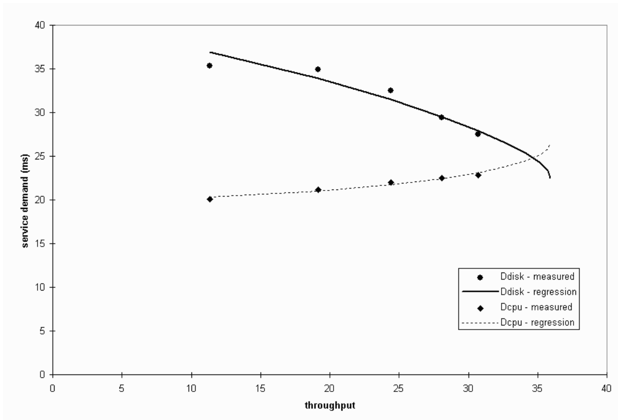
Figure 11. *Stress delete*: regression analysis for service demand estimation.

Figure 12a plots the estimated service demands against the throughput, which provides another way for judging the trend predicted by the regression methods chosen. From the performance curves in Figure 12b-d, the regression model outputs match the measurement data much better than do the outputs of Model 1 and Model 2. However, the regression model outputs at higher throughput values are not very satisfactory because the error in the predicted value grows larger as the independent variable increases.

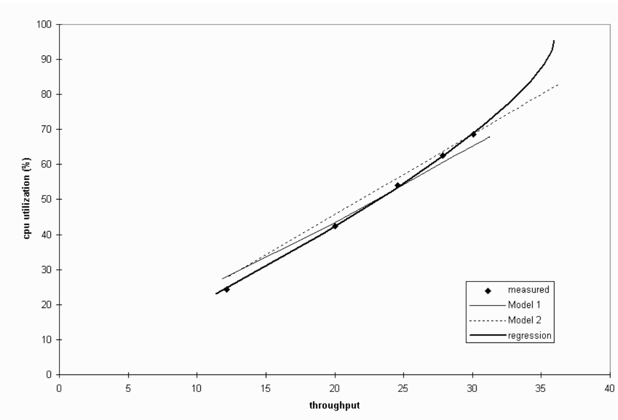
From the above experiment, we learned the regression analysis is an attractive alternative when performance measures do not provide enough information for estimating model parameters. This method nevertheless has restrictions. First, we have only exercised this method with a single job class and a zero think time because our measurement techniques cannot provide performance measures in a mixed workload environment. It is still inconclusive if multiple job classes and a non-zero think time can affect the accuracy of this method. Second, we can only apply regression analysis with respect to the number of concurrent clients, instead of the client population at the service center.[1] However, modeling a LD service

1. For this task, we need to measure how service demands vary with the number of pending disk requests at the Oracle database server.

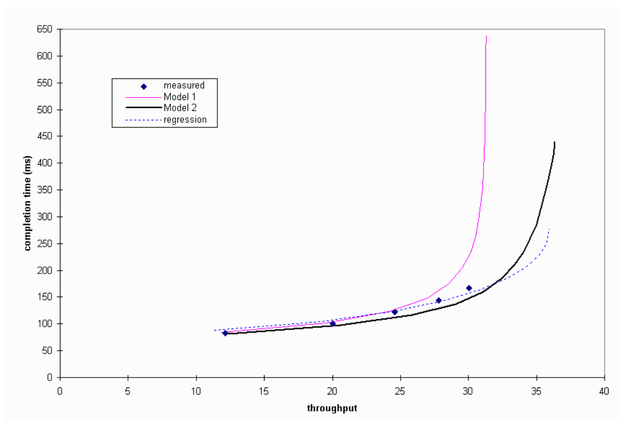
center requires information about how its service demand varies with the client population at the service center. These issues will be addressed in our future research plan.



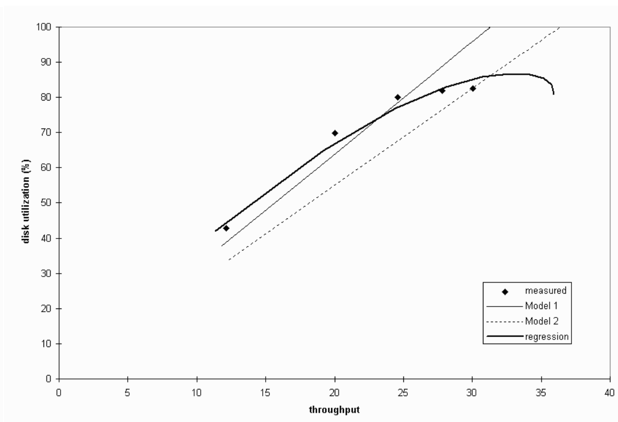
(a) service demands



(c) CPU utilization



(b) completion time



(d) disk utilization

Figure 12. Comparison of performance measures and three model outputs for *Stress delete*.

4. Preliminary Investigation of Citrix WinFrame Servers

There are two major benefits to extending PPS6 with Citrix WinFrame servers. First, it eases the administration effort for distributing the PPS6 client programs, since they no longer need be stored on the widely-dispersed Citrix client machines. Second, it provides a more secure environment for running the PPS6 panels because the application binaries are stored on the Citrix server machines, which reside in locked rooms. From the performance perspective, however, stockpiling several PPS6 panels in a single WinFrame server can make the server itself a performance bottleneck. In addition, the extra ICA traffic between WinStations and WinFrame servers can impact the network. Therefore, performance measurement of Citrix WinFrame servers includes:

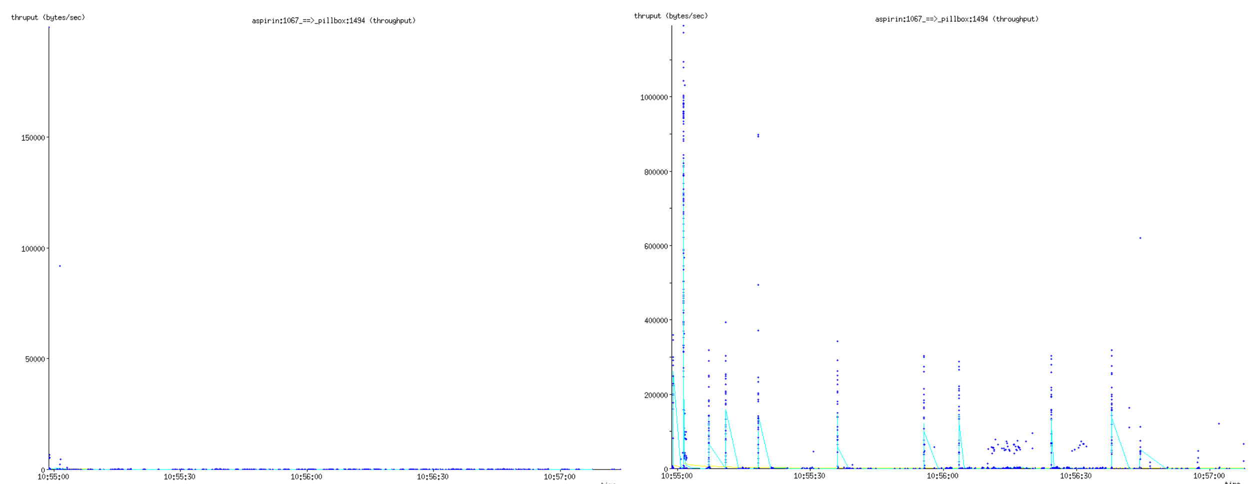
- estimating the capacity of a WinFrame server,
- estimating the network impact of ICA traffic, and
- investigating the performance impact for running PPS6 panels on WinFrame servers.

To date, several monitoring tools running on Windows NT have been exercised for measuring the overhead caused by the WinFrame extension; analytic modeling of the extension will be our goal.

4.1 Performance Measurements of ICA Traffic

It is difficult to quantify the workload of a distributed presentation service. One way to study the service is by monitoring the communications between its components; in this case, the ICA traffic. ICA generates *window frames* (network traffic) along both directions of a WinStation connection. At the WinStation, window frames carry key-strokes and mouse movements executed by a user; at the WinFrame server, window frames carry display updates on behalf of a program's GUI.

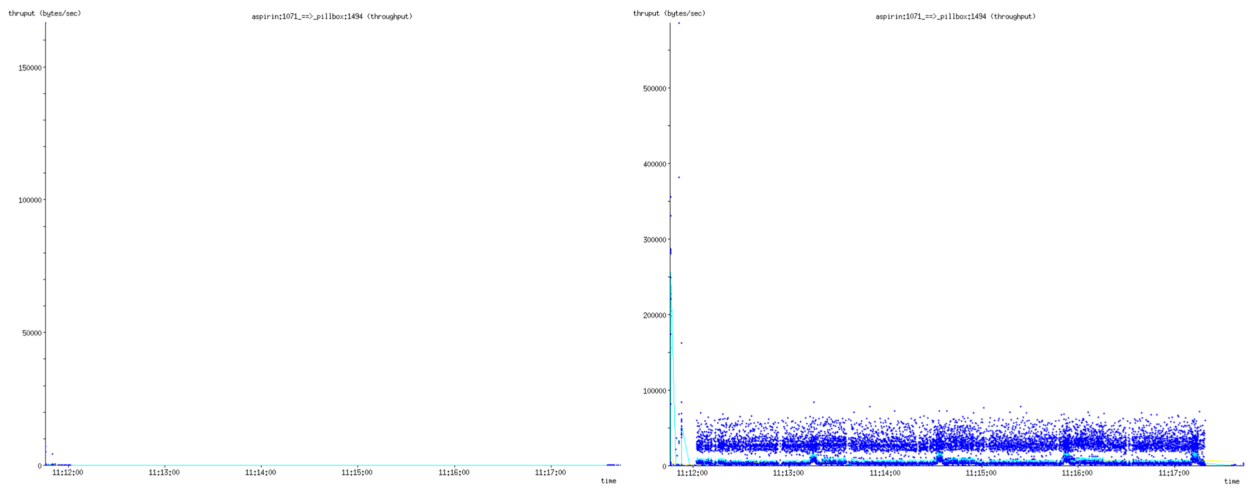
We use two Windows programs, Lotus *WordPro* and *Stress*, to study the ICA traffic. Figure 13 shows the ICA traffic for editing a small document with Lotus WordPro. Each dot in the figure represents an instantaneous TCP throughput value in one direction of the WinStation connection. It can also be interpreted as the number of data bytes being transported in that direction. Figure 13a shows ICA traffic for carrying key-strokes and mouse movements, and it is not surprising that the packet rate is very low. Figure 13b shows ICA traffic for carrying *WordPro* display updates. The highest peak is at the very beginning of the connection, probably due to bringing up the entire GUI for the first time. The rest of the ICA traffic is bursty with a few smaller peaks.



(a) WinStation --> WinFrame server (b) WinFrame server --> WinStation

Figure 13. ICA traffic for editing a small document with Lotus *WordPro*.

Figure 14 shows the ICA traffic for running *Stress* with 4 repetitions of 1000 database calls per repetition. No ICA frames are sent by the WinStation (see Figure 14a), except at the very beginning and the very end, because *Stress* requires no input after the load generation starts. Similarly, the highest peak at the very beginning of Figure 14b is probably attributable to bringing up the *Stress* panel for the first time. After that, a large amount of display updates are sent by the WinFrame server, which is shown as a band of dots spreading along the time axis in Figure 14b.



(a) WinStation --> WinFrame server (b) WinFrame server --> WinStation

Figure 14. ICA traffic for running *Stress*.

Table 15 lists some statistics, from the *tcptrace* reports, about the above two WinStation connections. In the *Stress* case, only 72 out of 5424 packets sent by WinStation actually carry data; the rest only carry TCP acknowledgments. In the *WordPro* case, more than half the packets (428 out of 743) sent by WinStation carry data, which is probably for key-strokes and mouse movements for document editing.

Table 15. Comparison of ICA traffic for *Stress* and *Lotus WordPro*.

	Stress		***Lotus WordPro***	
	WinStation -->	<-- WinServer	WinStation -->	<-- WinServer
total packets	5424	10728	743	909
actual data packets	72	10691	428	750
actual data bytes	1750	2117356	4593	144273
average segment size (bytes)	24	198	10	192
RTT average (ms)	113.7	1.4	86.4	33.9
throughput (bytes/sec)	5	5381	34	1065
connection elapsed time (sec.)	357.02		135.49	

4.2 Characteristics of ICA Overhead

Running an application remotely at the Citrix WinFrame server requires extra CPU time to manage this distributed presentation service. We call this extra CPU time the *ICA overhead* to distinguish it from the actual CPU time consumed by the application. ICA overhead also applies to the WinStation on which the result is displayed. Therefore, we further distinguish the *ICA-client overhead* from the *ICA-server overhead* at the WinFrame server.

The amount of ICA overhead associated with an application depends on many factors. From the ICA perspective, the two most important factors are the encryption and compression setting in a WinStation configuration [4]. The ICA encryption setting affects the amount of overhead associated with each ICA frame. A WinStation configuration allows three levels of encryption setting: no encryption at all, encryption during the authentication phase only, or encryption of each ICA frame. Similarly, the ICA compression setting affects the overhead associated with each ICA frame. A WinStation configuration can set compression on or off, with the default compression scheme or others.[1] Other configuration factors affecting the ICA overhead include the size and the color depth of the window displayed at the WinStation.

Application-specific and user-specific factors also affect the ICA overhead. Different Windows applications have different GUIs. They require different user inputs and generate different display outputs. Similarly, different users of an application can execute key-strokes and mouse movements differently. Since the ICA overhead is sensitive to these dynamic factors, they make the ICA overhead difficult to predict. The discrepancy of ICA traffic between Figures 13 and 14, and the *tcptrace* statistics in Table 15 help support the above arguments.

4.3 Performance Measurements of ICA Overhead

We have discovered that the ICA overhead can be measured on Windows NT platforms by the *task manager*, a process monitoring utility, or by *perfmon,* a general monitoring and logging tool. The WinFrame server program *csrss.exe* (client-server runtime subsystem) manages all distributed presentation on behalf of a WinStation. Therefore, WinStation-induced ICA-server overhead can be measured against the specific *csrss.exe* associated with each WinStation. Similarly, we can measure the ICA-client overhead against two specific processes, called *wengfN.exe* and *wfcrun32.exe*. Table 16 shows the performance measurements of the ICA overhead in a three-tiered environment. The measurements focus on the average ICA traffic rate and the performance measures of three processes - *Stress* itself, *csrss* at the WinFrame server, and *wengfN* at the WinStation. Based on the ICA traffic rates and the CPU utilization of *csrss* and *wengfN*, we can estimate the ICA-client overhead and ICA-server overhead in CPU time cost per byte.

Table 16. Performance measurements of ICA overhead.

# of DB calls	C (sec.)	CPU time (sec.) *stress*	*csrss*	*wengfN*	%CPU *stress*	*csrss*	*wengfN*	ICA (byte/sec.)	ICA Overhead (us/byte) WinServer	WinStation
125	38.22	7.87	7.06	39	20.59	18.47	102.04	6970.53	26.49	146.39
250	75.77	15.47	14.70	73	20.42	19.40	96.34	7214.87	26.89	133.54
375	113.83	24.51	21.70	109	21.53	19.07	95.76	6926.25	27.53	138.25
500	152.91	32.36	29.78	143	21.16	19.48	93.52	6926.32	28.12	135.02
625	190.81	40.65	37.27	182	21.30	19.53	95.38	6847.54	28.52	139.30
750	229.64	46.43	44.24	219	20.22	19.27	95.37	6684.91	28.82	142.66
875	269.13	55.23	52.43	256	20.52	19.48	95.12	6759.26	28.82	140.73
1000	307.81	61.28	59.31	293	19.91	19.27	95.19	6682.80	28.83	142.44
mean					**20.71**	**19.24**	**96.09**	**6876.56**	**28.00**	**139.79**
stdev					0.57	0.35	2.53	176.11	0.93	4.21

Several interesting phenomena are revealed by Table 16. First, the cost of the distributed presentation service provided by Citrix on behalf of *Stress* is very high. It consumes about 96% of the CPU at the WinStation and 19% of the CPU at the WinFrame server; the latter figure is very close to the CPU utilization of *Stress* itself, when run in the two-tiered configuration. Second, ICA-client overhead (140 us/byte) is more costly than ICA-server overhead (28 us/byte). We cannot find a good explanation for this five-fold overhead difference. Third, the average ICA traffic rate is low, less than 0.6% of Ethernet bandwidth; although there are frequent display updates while running *Stress*. Thus we estimate that 48 concurrent connections[2] would consume 29% of a 10 Mbps Ethernet network.

1. ICA compression cannot be turned off in Citrix WinFrame server version 1.6. This restriction has been removed from version 1.7 because most high-speed LANs, unlike serial modem lines, can accommodate the higher network load. Disabling ICA compression reduces the ICA overhead at both WinStation and WinFrame server. All measurements in this section use the same WinStation configuration: compression on and encryption off.
2. This is the maximum number of users PeopleSoft recommends handling via a single WinFrame server; for the server itself PeopleSoft recommends a four-processor Pentium P6 200 and 1 GB of memory.

4.4 Application Impact of Citrix WinFrame

To further investigate the cost of running Citrix WinFrame, we measured the completion time and CPU time for running *Stress* in a two-tiered configuration (called *local*) and in a three-tiered configuration (called *remote*). The measurement data are shown in Figure 17 with different numbers of database calls in each of four repetitions per measurement.

# of DB calls	2-tier *Stress* CPU	2-tier *Stress* C	3-tier *Stress* CPU	3-tier *Stress* C	3-tier *Stress* wengfN
125	8.03	32.73	8.47	38.22	39
250	15.62	63.61	16.08	75.77	73
375	23.28	98.58	24.75	113.83	109
500	31.09	130.59	31.58	152.91	143
625	37.97	166.34	39.23	190.81	182
750	45.72	200.31	47.83	229.64	219
875	55.03	235.34	55.91	269.13	256
1000	61.33	266.92	63.75	307.81	293

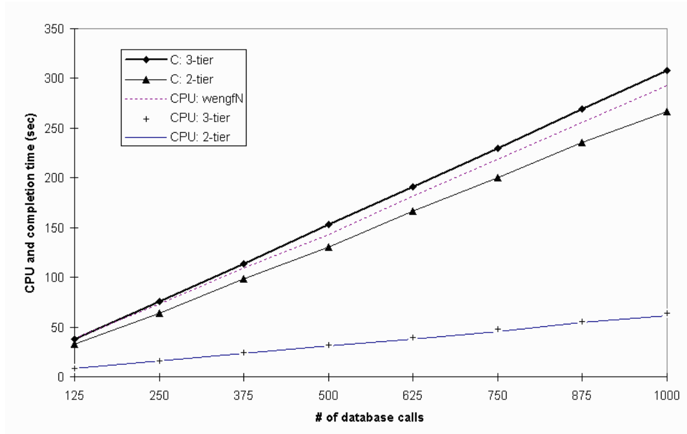


Figure 17. Measurement of application impact of Citrix WinFrame.

Comparing the measurements in both cases, we can draw three conclusions. First, in the three-tiered environment, the *Stress* program and the ICA client consume about an equal amount of CPU time. Second, there is a noticeable overhead (about 14-17%) in completion times for running *Stress* in the three-tiered as opposed to the two-tiered environment, and the difference grows linearly with the completion time. Third, the WinStation (in the form of *wengfN.exe*) consumes much more CPU time for the presentation of *Stress* results than it does running *Stress* itself.

Running programs in the three-tiered environment generates ICA-server overhead; however, our measurement tools do not account for this overhead in terms of CPU time charged to individual processes. This increases the difficulty of correlating the increased completion time with the ICA- server overhead. To date, we have been able to measure the ICA-server overhead on a per-WinStation basis (discussed in the previous subsection); analytic modeling of the increased completion time with respect to the ICA-server overhead is our next research goal.

5. Discussion and Future Work

Our project aims at providing performance measures for PPS6 capacity planning. To sustain the performance measures with a high degree of confidence, precise measurement techniques and analytic (queueing) models must be developed. We have worked toward this goal and gained experience through several performance experiments conducted on the CITI testbed, which is a minimal PPS6 system when compared to the hardware and software that will be deployed as part of the M-Pathways project. Nevertheless, these experiments have provided some interesting results, as well as some insights into conducting performance measurements in the next stage of the project. To date, three major accomplishments of this project are:

- a prototype implementation of the database exerciser, the *Stress* program,

- measurement techniques for Oracle database servers, SQL*Net traffic, Citrix WinFrame servers, and ICA traffic, and
- a general performance modeling tool for evaluating closed QNMs, the *MVA* tool.

Although the performance measurements presented in Section 3 are not entirely applicable to PPS6, our achievement and experience can be easily applied to it. In this section, we summarize and discuss our accomplishments to date and our future research plans.

5.1 Database Exerciser

A database exerciser serves two purposes in performance measurement. First, it helps benchmark a database server with different workload intensities. Due to the difficulty of setting up a testing environment with a large number of users, a database exerciser is usually the only feasible solution for conducting large-scale performance measurements. Second, it generates continuous, synchronous requests for measuring the per-request CPU and disk service demands. However, we want to develop a database exerciser which can generate synthetic SQL requests analogous to a real PPS6 panel.

While *Stress*, our prototype implementation of a database exerciser, met the testbed requirements, we believe two enhancements will help it better simulate a real PPS6 panel. First, we realize that PPS6 panels generate more complicated SQL requests than does the *Stress* program. To emulate PPS6 panels more precisely, the *Stress* program must generate SQL requests and access Oracle tables similar to those used in processing PPS6 panels. Preliminary investigation indicates that the Oracle SQL trace and the PPS6 client trace facilities can help us here.[1] For each type of PPS6 panel, both types of traces could be collected while users are using the system normally. Later, the traces can be analyzed to identify the exact SQL requests composed of the PPS6 transactions associated with each panel. After the PPS6 transactions of frequently used panels have been identified, *Stress* can be programmed to generate similar transactions for emulating PPS6 panels. Second, for emulating the user "think" time, the *Stress* program can be configured with an adjustable delay between consecutive requests. The delay can be a fixed value or a probability distribution, which is determined by measuring the real think time during normal use.

The proposed enhancements for *Stress* not only facilitate PPS6 performance measurements, but also help us characterize the database workload of PPS6 panels. A more detailed description of *Stress* is provided in Appendix A.

5.2 Measurement Methodology

Performance measurements require accurate monitoring tools and expertise in measurement techniques. In this project, performance measures must be monitored at the Oracle database server and the Citrix WinFrame server, and packet traces must be collected and analyzed for SQL*Net traffic and ICA traffic. To date, we have used several performance monitoring tools with the CITI testbed;[2] however, we find them primitive and somewhat restricted. Therefore, we remain interested in new monitoring tools to develop better measurement techniques.

5.2.1 Performance Measurement of Oracle Database Servers

The performance of the PPS6 depends on the performance of the Oracle database server, i.e. how it executes the PPS6 transactions. Several performance measures help evaluate the performance of the Oracle database server: completion time, throughput, and device utilization of the server CPU(s), disk(s), and network interface card (NIC). Since these performance measures vary with workload intensity, they are always studied with different numbers of Oracle connections (or active PPS6 panels).

1. SQL traces can be collected by the Oracle SQL trace or by the PPS6 client trace. The difference is the location where the SQL trace is collected: the Oracle SQL trace collects it at the database server, the PPS6 client trace collects it at the client machine.
2. Several tools have been proved to be useful: *iostat*, *perfmon*, *tcpdump*, and *tcptrace*.

Currently, we measure the request completion time at the client side (via *Stress* itself), and use it to further derive the server throughput. At the Oracle server, we measure the device utilization (with *iostat*), which can sustain good accuracy by taking an average value of several measurements at short intervals. This measurement technique is suitable for a controlled environment like the CITI testbed, but inconvenient for production systems. Therefore, we are looking for better monitoring tools for measuring the completion time at the Oracle server and capable of accounting for the CPU time and the disk time expended on each request. With such tools, performance measures can be collected on a regular basis from production systems for statistical analysis. These measurements can provide valuable information for performance evaluation and capacity planning.

5.2.2 Performance Measurement for Analytic Modeling

Analytic modeling requires performance measures for estimating model parameters. These performance measures are the service requirement at different devices, decomposed into individual service times at various stages of request execution. Analytic modeling of the PPS6, therefore, measures service times for the execution of SQL statements in terms of CPU time and disk time, and the number of SQL*Net packets accounting for transmission delay.

Currently, we use *iostat*, which reports CPU and disk utilization at the Oracle server, and *tcpdump*, which collects packet traces for the SQL*Net traffic, to derive the model parameters. As a consequence, the service requirement of a *Stress* request can only be decomposed into a fixed CPU service time, a fixed disk service time, and the transmission time of SQL*Net packets. We find these tools inadequate for modeling purposes because the disparities of estimated service demands under different workload intensities and the discrepancies of various performance measures between measurements and models outputs are so obvious.

Therefore, an emergent need in performance measurement is a tool capable of dissecting the service demands into smaller components. To date, we find *tkprof*, an Oracle performance diagnostic tool, a plausible solution for this task. *Tkprof* decomposes the execution of an SQL statement into three phases (parse, execute, and fetch); it also reports the elapsed time, the CPU time, and the number of disk block reads in each phase (see Table 18). These statistics help to refine the service requirement from two gross values (CPU and disk service times) into smaller components with specific purposes. We believe this extra information can help to discover the cause for the disparity in measured service demands and so to improve the accuracy of the analytic models.

Table 18. TKPROF output of an SQL statement.

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.16	0.29	3	13	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.03	0.26	2	2	4	14

5.2.3 Packet Trace Analysis of SQL*Net Traffic

The SQL*Net traffic of an Oracle connection follows an interactive, handshaking pattern. This pattern has been identified from the packet traces collected from the CITI testbed (for *Stress*) and from production systems (for PPS6 panels). An important consequence of this pattern is that it augments the completion time with a round-trip network latency and some client-side protocol latency for each handshake. Therefore, it is important to measure the number of protocol handshakes for each PPS6 transaction and the client-side latency for each handshake.

Currently, we collect SQL*Net traffic with *tcpdump* and analyze the packet traces with *tcptrace*. These tools can interpret the packet traces at the transport layer (TCP), but reveal no protocol content about the SQL*Net conversation. Although we are still able to measure the number of protocol handshakes for each *Stress* request, analytical studies of the content of SQL*Net packet traces are impossible. In addition, the current measurement techniques also prevent us from measuring the client-side latency for each handshake accurately; this requires statistical analysis from a large number of samples. Therefore, a SQL*Net protocol analyzer would be helpful for analytic and statistical studies of PPS6 packet traces.

5.2.4 Performance Measurements of Citrix WinFrame Servers

The ICA overhead and ICA traffic rate are two primary indices to evaluate the performance of the Citrix WinFrame servers. Since these two performance measures have an application-specific property and vary with some user-specific dynamic factors, the only plausible way to study them is to gather statistical data from a large number of ICA connections in production systems. Our preliminary investigation, presented in Section 4, has successfully exercised several measurement techniques for this task. Similar performance measurements will be conducted against the PPS6 panels in production systems, which use Citrix WinFrame version 1.7.

Another important factor affecting the performance of WinFrame servers is the memory requirement for running multiple Windows applications on behalf of several WinStaion connections. Two well-known properties of Windows applications are that they consume memory relentlessly and most of them are I/O bound instead of CPU-bound. Therefore, we believe the memory of a WinFrame server will probably emerge as the primary bottleneck, followed by the CPU. The average memory requirement for each WinStation connection will be investigated for capacity planning of WinFrame servers.

5.3 Performance Modeling Tool

The *MVA* tool can evaluate closed QNMs with delay service centers, load-independent service centers (called *queueing*), and Ethernet local networks. We exercise this tool by evaluating an analytic model of the *Stress* program. Although the model outputs are not very satisfactory, we attribute the cause to the model itself instead of the tool.

At the next stage of this project, we will still use the *MVA* tool, but will apply other solution techniques to supplement its insufficiency. Most importantly, what technique to use strongly depends on what kind of analytic models we want to evaluate. To date, we find the Method of Layers (MOL) an attractive technique for three-tiered client-server modeling [9,14]. MOL is designed to solve the Layered Queueing Models (LQMs). An LQM is an enhanced QNM, which models distributed client-server systems as layered software servers competing for hardware (devices) servers [14]. Therefore, we can decompose the execution of a SQL request as visiting different software servers with specific functions. This allows analytic models with a finer granularity of service requirements.

6. Summary

In this first phase of work we have developed a QNM for a two-tiered PPS6 client-server system, and built a performance modeling tool to evaluate the model via MVA. We also developed *Stress*, a load generator that exercises an Oracle database server. We used *Stress* to determine the CPU and disk service demand parameter values, after which the model outputs were compared to measured values. The model outputs closely matched the measured values in those cases where the service demands remained constant with increasing numbers of clients, as the model assumes a fixed service demand for all client populations. However, in several cases the CPU and disk service demands were found to vary with the number of clients, causing major discrepancies. In order to address this issue, we plan to refine the model by dissecting the measured service demands into smaller components, using the *tkprof* tool.

We then performed some preliminary measurements when running PPS6 in a three-tier Citrix environment, both by monitoring network traffic and by measuring CPU utilization. First, we measured the network traffic between Citrix WinStation clients and the WinFrame server, which use the ICA protocol for communication. Test loads were generated both by *Stress* and *WordPro*; the latter program was used to generate continuous display updates. We found that the average ICA traffic rate is about 0.6% of a 10 mbps Ethernet, so the network is not likely to be a bottleneck.

Second, we measured the ICA overhead in terms of CPU time expended on client and server machines. One interesting result was that the Citrix client consumes about 96% of the client machine CPU; in contrast, the Citrix server consumes about 19% of the server machine CPU -- about as much as *Stress* does when run on the same platform.

Finally, we measured completion time and CPU time when running equivalent *Stress* workloads in both the 2-tiered and 3-tiered environments. We noted a 14-17% increase in completion time when running *Stress* in the 3-tiered environment. Our measurement tools do not allow us to account for CPU time charged to individual processes, so we cannot as yet correlate the increased completion time with the ICA server overhead.

We were not able to obtain an accurate characterization of typical PPS6 application workloads for inclusion in this study. Reasons for this include the inability to access real PeopleSoft applications running against real databases. As a result we were not able to generate models that give results in terms of PeopleSoft panel accesses, but only in terms of Oracle database operations. To address this issue, we plan to measure actual PeopleSoft applications and record client, network, and server utilizations, as well as the stream of Oracle requests generated by each PeopleSoft panel. This request log will be replayed against a real database copy running on the real hardware, using a modified version of *Stress*. This work will require close coordination with M-Pathways development staff.

References

1. D.W. Bachmann, M.E. Segal, M.M. Srinivasan, and T.J. Teorey. "NetMod: A Design Tool for Large-Scale Heterogeneous Campus Networks." *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 1, pp. 15-24, January 1991.
2. D.W. Bachmann. A Methodology for Performance Modeling of Distributed Systems Based on Client-Server Protocols. Ph.D. Thesis, University of Michigan, 1992.
3. K.M. Chandy and D. Neuse. "Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems." *Communications of ACM*, Vol. 25, No. 2, pp. 126-134, February 1982.
4. Citrix. *ICA Technical Paper*. [http://www.citrix.com/technology/icatech.htm], 1996.
5. Citrix. *Thin-Client/Server Computing*. [http://www.citrix.com], 1997.
6. J.L. Hammond and P. O'Reilly. *Performance Analysis of Local Computer Networks*. Reading, MA: Addison-Wesley, 1986.
7. S.S. Lam. "A Carrier Sense Multiple Access Protocol for Local Networks." *Computer Networks*, Vol. 4, No. 1, pp. 21-32, February 1980.
8. Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
9. Masud A. Khandker. *Performance Measurement and Analytic Modeling Techniques for Client-Server Distributed Systems*. Ph.D. Thesis, University of Michigan, 1997.
10. Edward Lazowska, John Zahorjan, Scott Graham, and Kenneth Sevcik. *Quantitative System Performance*. Prentice Hall, 1984.
11. Daniel Menasce, Virgilio Almeida, and Larry Dowdy. *Capacity Planning and Performance Modeling: from Mainframes to Client-Server Systems*. Prentice Hall, 1994.
12. M-Pathways. *The M-Pathways Project*. [http://www.mpathways.umich.edu], 1997.
13. PeopleSoft. *PeopleSoft's Distributed Architecture*. [http://www.peoplesoft.com], 1997.
14. J.A. Rolia. *Predicting the Performance of Software Systems*. Technical Report, CSRI TR260, University of Toronto, 1992.
15. Huge Toledo. *Oracle Networking*. Oracle Press, 1996.

Acknowledgements

This work was supported by the Executive Director's Office of ITD. Bob Riddle built the *Stress* program and offered invaluable practical assistance and advice. Thanks to our M-Pathways project colleagues who provided some badly needed measurements; they will get the opportunity to do so again. Chuck Lever and Janani Janakiraman provided project assistance. Finally, this project benefited greatly from the weekly ministrations and behind-the-scenes efforts of Mary Jane Olsen.

Appendix A. The Database Exerciser - *Stress*

Stress is a Visual Basic program running on Windows platforms, which generates remote database calls (through the Windows ODBC API) against an Oracle database server. The graphic user interface (panel) of *Stress*, as shown in Figure 19, is divided into several areas for two main functions: automatic load generation and manual database controls.

For automatic load generation, *Stress* creates a synthetic workload from four types of database calls: insert, update, query (read), and delete a row. Each workload is specified with six parameters (see area (i) in Figure 19): total database calls, number of iterations, and the composition of workload: %inserts, %reads, %updates, and %deletes. Load generation starts when the load generator button is clicked, after which the status information area displays workload progress (g) and the processing time (h) for each type of database call.

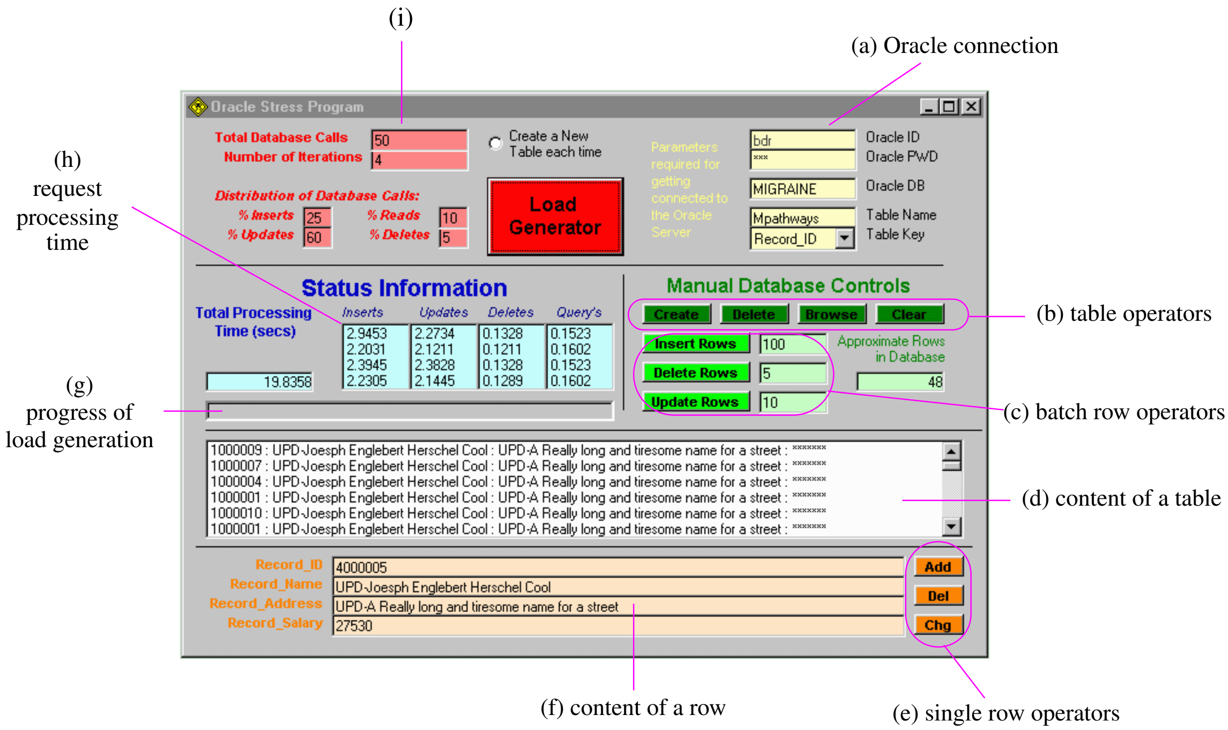


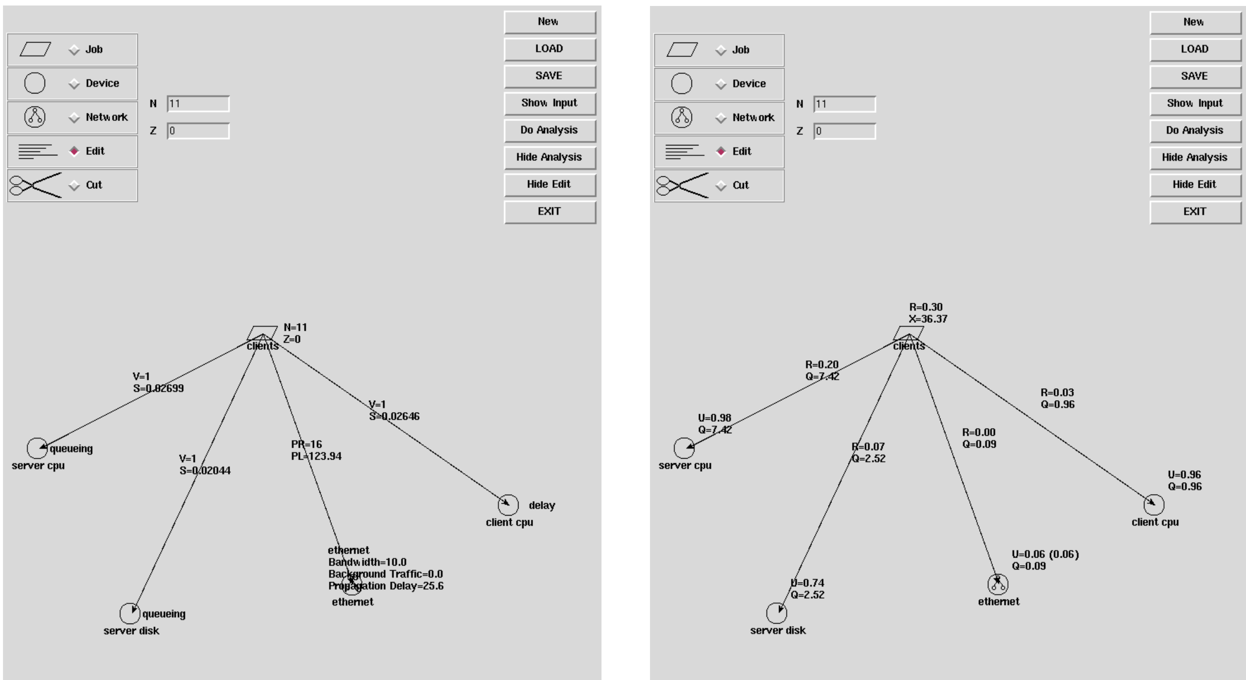
Figure 19. *Stress* panel.

The manual database controls include manual table operations and row operations. The top four horizontal buttons are table operators (b): *create*, *delete*, *browse*, and *clear* a table. The three vertical buttons provide batch operations to the rows of a table (c): *insert*, *delete*, or *update* a specified number of rows in the table. The other three vertical buttons at the lower-right corner are single row operators (e): insert (marked as *Add*), delete (marked as *Del*), and update (marked as *Chg*) a specific row of the table. The center area with a vertical scrolling bar (d) displays a cumulative log of operations to the table; the lower area (f) displays the content of the current row.

The Oracle connection area (a) has fields for Oracle user identification (Oracle ID) and password (Oracle PWD) for establishing a connection to the Oracle server. The lower three fields specify the Oracle database being accessed, the name of the table being accessed, and the field used as the primary key of the table. The table is composed of four fields: **Record_ID**, **Record_Name**, **Record_Address**, and **Record_Salary**; each field is a string with a maximum length of 255 characters.

Appendix B. The Performance Modeling Tool - *MVA*

The performance modeling tool has a graphic user interface as shown in Figure 20. The buttons in the upper-left area of the panel provide functions for model editing; the buttons in the upper-right area provide functions for file operations (save/load a model into/from a file) and model analysis.



(a) Show Input (b) Do Analysis

Figure 20. Two screen snapshots of the performance modeling tool.

A model must have at least one job class (represented as a parallelogram) and one device (represented as a circle). A link connecting the parallelogram and a circle represents the service requirement for the job class visiting the device. Modeling parameters are associated with these icons (parallelograms, devices, and links) as shown in Figure 20a. They can be set by clicking each icon; then by filling in the appropriate fields in the upper area.

Figure 20b shows the performance measures determined by the tool. For each job class, the panel displays the total residence time (marked as R) and the throughput (marked as X) of the job class. The performance measures of each device are its device utilization (marked as U) and the average queue length (marked as Q) at the device. For an Ethernet device, the utilization in parentheses is the reduced network utilization excluding the background traffic. For each link, R is the residence time (waiting time and service time) the job class spends in the device; Q is the split queue length (out of all job classes) for each job class visiting the device.

Modeling the Caching Effect of Database Access

Although it has not been exercised, the performance tool is able to model the caching effect of database access. Database access caching can be accomplished at the DBMS level, at the OS level, or both. Similar to other caching in a computer system, the disk service time of a database access is eliminated if the accessed data were cached in memory. From the modeling perspective, the only concern is how to estimate the average cache *hit ratio* (h) of a database access. If the hit ratio can be estimated, even without a thorough examination of the caching algorithm, the disk service requirement could be manipulated to reflect the caching effect in a model.

A straightforward approach is to adjust the visit ratio of a disk service with respect to the cache hit ratio of a database access. In a QNM, the disk visit ratio of a job class can be multiplied by ($1 - h$) to obtain an adjusted ratio reflecting a reduced percentage of disk visits. A more complicated approach is to change the visit ratio and the service demand together for better approximation of the real disk service requirement. This adjustment should be considered together with the effect of disk access optimization.

Modeling PPS6 Clients with Network Latency

As mentioned in Section 2, our analytic model assumes client and server machines residing at the same local network. For clients accessing the database from remote networks, the network latency along the end-to-end path is not entirely covered in the model. It is plausible to model the rest of network latency as a constant delay for two reasons. First, our modeling tool determines only steady-state, mean performance measures; the variances of the performance measures cannot be determined by the model. Second, many factors, such as the background traffic beyond the system under study also affect the end-to-end network latency. Since it is impossible to determine all these factors or to cover all of them in the model, a constant network latency is a plausible assumption. Therefore, each job class representing remote clients should visit a pseudo *delay* center to cover the rest of the end-to-end network latency.

Appendix C. Performance Measurement of Citrix WinFrame Servers

Windows NT provides a monitoring tool, called *perfmon*, for system performance measurements. Inside Windows NT, system performance measures are stored in various performance counters; a set of relevant performance counters forms a performance object. Some performance counters may have multiple instances because their performance object can refer to multiple entities in the system. Citrix WinFrame servers add two new counters - *Active WinStations* and *Total ICA Bytes/sec* - in the ***system*** object, and a new performance object called ***WinStation*** to the standard sets of performance counters. These new counters are listed in Table 21.

Performance measurements of ICA overhead are conducted at two sites: the WinStation and the WinFrame server. Each WinStation connection has two processes running at its local machine, *wfengN.exe* and *wfcrun32.exe*. The ICA-client overhead is the sum of the *processor times* of these two processes, which can be measured by *task manager* in Windows NT 4.0 or by its predecessor, *pview*, in Windows NT 3.51. Task manager can also report the instantaneous percentage of processor time allocated to a WinStation; however, only *perfmon* can report the statistical value.

Performance measurements of ICA-server overhead are more complicated for two reasons: a WinFrame server usually has multiple active WinStation connections and each WinStation has multiple processes running on the WinFrame server. The percentage of CPU time allocated to a WinStation is the specific instance of the performance counter - ***WinStation***:*%ProcessorTime* - associated with the WinStation. The total CPU time of a WinStation can be measured as the product of ***WinStation***:*%ProcessorTime* and ***WinStation***:*ElapsedTime*. The ICA-server overhead of a WinStation is the *processor time* of a specific process, *csrss.exe*, associated with the WinStation. However, it cannot be further decomposed in a per-process base.

Table 21. New performance counters added by Citrix WinFrame.

Object/Counter	Description
Object: System	
Active WinStations	total number of active (logged on) WinStations
Total ICA Bytes/sec	total number of bytes transferred in the system as result of WinStation communications
Object: WinStation	
% Privileged Time	the percentage of elapsed time that this processes's threads have spent executing code in Privileged Mode
% Processor Time	the percentage of elapsed time that all of the threads of this process used the processor to execute instructions.
% User Time	the percentage of elapsed time that this processes's threads have spent executing code in User Mode
Bitmap Hit Ratio	bitmap hit ratio
Bitmap Hits	the number of Bitmap hits from the cache
Bitmap Reads	the number of Bitmap references of the cache
Brush Hit Ratio	brush hit ratio
Brush Hits	brush hits
Brush Reads	the number of brush references to the cache
Elapsed Time	the total elapsed time (in seconds) this process has been running
Glyph Hit Ratio	Glyph hit ratio
Glyph Hits	Glyph hits
Glyph Reads	the number of Glyph references to the cache
ID Process	the unique identifier of this process
Input Async Frame Error	number of input async framing errors
Input Async Overflow	number of input async overflow errors
Input Async Overrun	number of input async overrun errors
Input Async Parity Error	number of input async parity errors
Input Bytes	number of bytes input on this WinStation that includes all protocol overhead
Input Compress Flushes	number of input compression dictionary flushes
Input Compressed Bytes	number of bytes input after compression; this number compared with the Total Bytes input is the compression ratio
Input Compression Ratio	compression ratio of the server input data stream
Input Errors	number of input errors of all types
Input Frames	number of frames (packets) input to this WinStation
Input Timeouts	the total number of timeouts on the communication line as seen from the client side of the connection
Input WaitForOutBuf	the number of times that a wait for an available send buffer was done by the protocols on the client side of the connection
Input WdBytes	number of bytes input on this WinStation after all protocol overhead has been removed
Input WdFrames	the number of frames input after any additional protocol added frames have been removed
Output Async Frame Error	number of output async framing errors
Output Async Overflow	number of output async overflow errors
Output Async Overrun	number of output async overrun errors
Output Bytes	number of bytes output on this WinStation that includes all protocol overhead
Output Compress Flushes	number of output compression dictionary flushes
Output Compressed Bytes	number of bytes output after compression; this number compared with the Total Bytes output is the compression ratio
Output Compression Ratio	compression ratio of the server output data stream
Output Errors	number of output errors of all types
Output Frames	number of frames (packets) output on this WinStation
Output Parity Error	number of output async parity errors

Table 21. New performance counters added by Citrix WinFrame.

Object/Counter	Description
Output Timeouts	the total number of timeouts on the communication line from the host side of the connection
Output WaitForOutBuf	the number of times that a wait for an available send buffer was done by the protocols on the host side of the connection
Output WdBytes	number of bytes output on this WinStation after all protocol overhead has been removed
Output WdFrames	the number of frames output before any additional protocol frames have been added
Page Faults/sec	the rate of Page Faults by the threads executing in this process
Page File Bytes	the current number of bytes this process has used in the paging file(s)
Page File Bytes Peak	the maximum number of bytes this process has used in the paging file(s)
Pool Nonpaged Bytes	the number of bytes in the Nonpaged Pool
Pool Paged Bytes	the number of bytes in the Paged Pool
Priority Base	the current base priority of his process
Private Bytes	the current number of bytes this process has allocated that cannot be shared with other processes
Save Screen Bitmap Hit Ratio	save screen bitmap hit ratio
Save Screen Bitmap Hits	save screen bitmap hits
Save Screen Bitmap Reads	the number of Save screen bitmap reference to the cache
Thread Count	the number of threads currently active in this process
Total Async Frame Error	total number of async framing errors
Total Async Overflow	total number of async overflow errors
Total Async Overrun	total number of async overrun errors
Total Async Parity Error	total number of async parity errors
Total Bytes	total number of bytes on this WinStation that includes all protocol overhead
Total Compress Flushes	total number of compression dictionary flushes
Total Compressed Bytes	total number of bytes after compression
Total Compression Ratio	total compression ratio of the server data stream for this WinStation
Total Errors	total number of errors of all bytes
Total Frames	total number of frames (packets) on this WinStation
Total ICA Hit Ratio	the overall hit ratio of all ICA objects
Total ICA Hits	total ICA cache hits
Total ICA Hits/sec	total ICA cache hits per second
Total ICA Interval Hit Ratio	the overall hit ratio of all ICA objects in the last sample interval
Total ICA Reads	total ICA references to the cache
Total ICA Reads/sec	the total ICA references to the cache
Total WaitForOutBuf	the number of times that a wait for an available send buffer was done by the protocols on both the host and client sides of the connection
Total WdBytes	total number of bytes on this WinStation after all protocol overhead has been removed
Total WdFrames	the total number of frames input and output before any additional protocol frames have been added
Virtual Bytes	the current size in bytes of the virtual address space the process is using
Virtual Bytes Peak	the maximum number of bytes of virtual address space the process has used at any one time
Working Set	the current number of bytes in the Working Set of this process
Working Set Peak	the maximum number of bytes in the Working Set of this process at any point in time