CITI Technical Report 97–5

Performance Modeling of the PeopleSoft Multi-Tier Remote Computing Architecture

Yi-Chun Chu
ycchu@citi.umich.edu

Charles J. Antonelli
cja@citi.umich.edu

Toby J. Teorey
teorey@eecs.umich.edu

ABSTRACT

Complex client-server configurations being designed today require a new and closely coordinated approach to analytic modeling and measurement. A closed queuing network model for a two-tiered PeopleSoft 6 client-server system with an Oracle database server is demonstrated using a new performance modeling tool that applies mean value analysis. The focus of this work is on the measurement and modeling of the PeopleSoft architecture to provide useful capacity planning insights for an actual large-scale university-wide deployment. A testbed and database exerciser are then developed to measure model parameters and perform the initial validation tests. The testbed also provides preliminary test data on a proposed three-tiered deployment architecture that includes the Citrix WinFrame environment as an intermediate level between the client and the Oracle server.

December 1997

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

Performance Modeling of the PeopleSoft Multi-Tier Remote Computing Architecture

Yi-Chun Chu, Charles J. Antonelli, and Toby J. Teorey

December 1997

1. Introduction

A new approach to analytic modeling and measurement of client-server configurations being designed today is needed in order to accurately predict system performance long before such systems are implemented. Accurate modeling allows the designer to test the feasibility of many alternate configurations before nailing down the final design and implementation. In particular, PeopleSoft has developed a two-tiered architecture and application system that is now widely distributed throughout the world. However, good models of this architecture lag far behind the initial deployments of the system.

We develop a closed queuing network model for a two-tiered PeopleSoft 6 client-server system with an Oracle database server and demonstrate it using a new performance modeling tool that applies mean value analysis. The focus of this work is to provide useful capacity planning insights for an actual large-scale university-wide deployment of the PeopleSoft system. A testbed and database exerciser are then developed to measure model parameters and perform the initial validation tests. The testbed also provides preliminary test data on a proposed three-tiered deployment architecture that includes the Citrix WinFrame environment as an intermediate level between the client and the Oracle server.

In Section 2 we present the analytic modeling techniques used to model a two-tiered PeopleSoft 6 architecture. In Section 3 we describe the testbed and database exerciser we have 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 [6], which is used to build a three-tiered architecture on top of PeopleSoft 6. Finally, we summarize our results in Section 5, along with a discussion of future work.

2. Analytic Modeling of Multi-Tiered Client-Server Systems

Today many legacy applications on mainframes have been downsized to multi-tiered client-server architectures. Analytic modeling techniques developed for mainframe systems have also been applied to evaluate these distributed client-server configurations [2,13,15,18]. However, it is becoming clear that a major emerging problem is to obtain the required performance measures for estimating the model parameters. The lack of transaction-oriented monitoring facilities in a multi-tiered environment make performance measurements much more difficult than in a centralized mainframe environment [8,9]. Therefore, complex client-server configurations, such as the University of Michigan (U-M) production deployment of PeopleSoft applications, require a closely coordinated approach to analytic modeling and measurement.

2.1 Software Architecture of PPS6 in Production Environments

PeopleSoft 6 (henceforth called PPS6) is a two-tiered client-server application [17]. PPS6 clients are "thick" clients running on Windows 95/NT platforms. Each client program, called a *panel* in PeopleSoft terminology, accesses remote Oracle databases as a sequence of SQL statements when a user fills data into various fields of the panel. The underlying database connections are managed by SQL*Net, which provides network connectivity in distributed, heterogeneous computing environments [19]. This two-tiered PPS6 architecture is augmented in the U-M production environment with a third tier consisting of Citrix WinFrame servers. PPS6 clients run on these remote application servers, access an Oracle database and communicate with user desktop machines running the Citrix WinStation client to display results [16]. The deployed PPS6 production system hence resembles a generic three-tiered client-server architecture as shown in Figure 1.

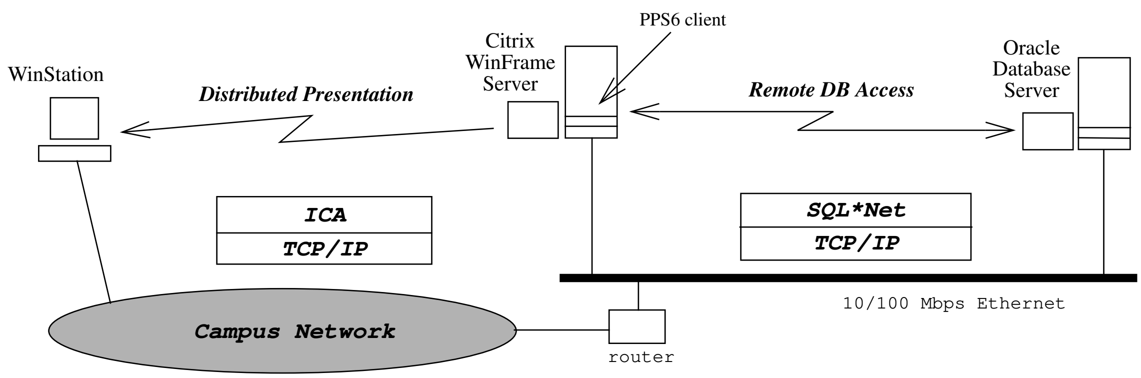


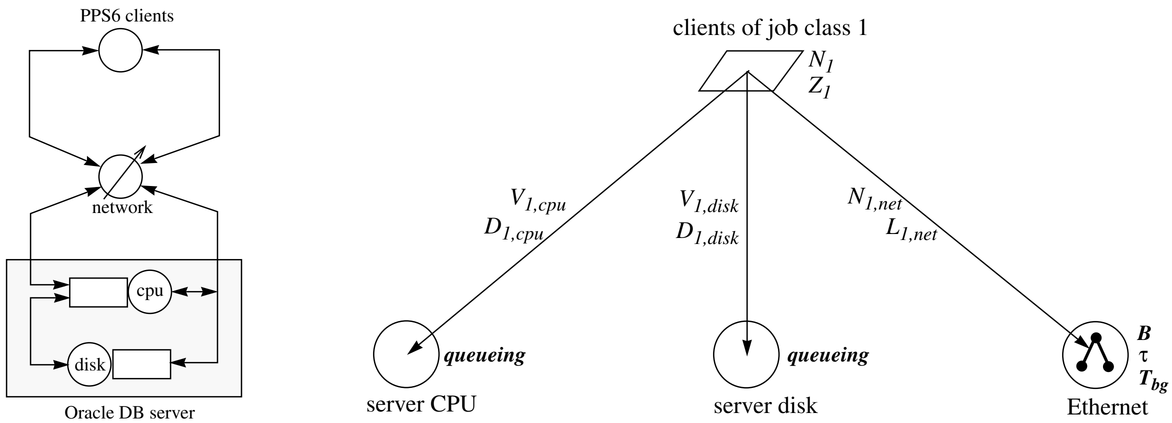
Figure 1. The production deployment of PPS6 with Citrix WinFrame extension.

Citrix WinFrame extends the Windows NT server with multi-user and distributed presentation capabilities [6]. It allows multiple users from different desktop platforms, including non-Windows platforms, to run Windows applications on a single WinFrame server. Citrix implements the distributed presentation service with its Independent Console Architecture (ICA) [5]. ICA is conceptually similar to the X-Windows protocol on UNIX platforms and utilizes highly optimized drawing primitives and a proprietary *Thinwire protocol* to achieve a distributed Windows presentation.

2.2 Analytic Modeling of PPS6 Client-Server Systems

Since PPS6 is actually a two-tiered client-server application and because we believe it is too complicated to build a three-tiered model directly, we first develop a two-tiered model to study how PPS6 clients interact with the Oracle database server. The two-tiered model is a closed queueing network model (QNM) with a fixed number of customers (PPS6 clients). This model, shown in Figure 2a, includes a fixed number of clients, a local network, and a database server.

Since the PPS6 clients (running on WinFrame servers) and the database server reside on the same Ethernet network in the production environment, the local Ethernet can be modeled as a load-dependent service center accessed by clients and the server [14,15]. PPS6 clients are modeled as delay centers with a 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, namely a server CPU and a server disk. Each class of client requests is carried out with a certain amount of server CPU time (CPU service demand) and server disk time (disk service demand), both assumed, for now, to be exponentially distributed. Therefore, client requests in the closed QNM are characterized as multi-class homogeneous workloads.



(a) Analytic model (b) MVA tool representation

Figure 2. Queueing network model of the two-tiered PPS6.

2.3 Solution Techniques for Evaluating QNMs

Mean-value analysis (MVA) is an efficient algorithm for evaluating closed QNMs with exact solutions [12,14,15]. In previous work, MVA has been 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 [14].

We have developed a performance modeling tool, called the *MVA* tool, for evaluating closed QNMs with Ethernet devices [4]. It provides a Tcl/Tk user interface for constructing analytic models and displaying model outputs after evaluation. The core evaluation technique is based on a multi-class, approximate MVA algorithm, called the *linearizer* [3], which we have enhanced to evaluate Ethernet networks (load-dependent devices). The model parameters of a closed QNM can be directly converted into equivalent model parameters in the tool. Figure 2b shows the equivalent model representation in the *MVA* tool for the analytic model introduced in Figure 2a.

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, namely its visit ratio (V) and the average service time for each visit (D). For models with multiple job classes, each job class is represented by a separate parallelogram whose service requirement is specified by links connecting to visited devices.

2.4 MVA Extension with the Ethernet Device

The modeling parameters of an Ethernet device are determined differently from CPU and disk devices. 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) [10,11]. One approach for assessing the per-packet delay in an Ethernet network is to model the network as a load-dependent (LD) device, and solve the model via the extended MVA algorithm with LD service centers [14,15]. 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.[1] 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 [1,2,11].

Assessing the per-packet delay in Ethernet with respect to the average traffic rate can be accomplished by the Ethernet delay submodel of NetMod [1]. This submodel applies Lam's formula for the per-packet delay [11], and Hammond and O'Reillys' formula for Ethernet utilization [10]. Lam's Ethernet delay model estimates per-packet delay with four input parameters: the Ethernet transmission speed, 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 with the analysis algorithm of NetMod [13]. 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 converges to the same performance measures but requires fewer iterations. 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 aggregate Ethernet traffic is estimated in a way similar to NetMod's analysis algorithm [1]. The average packet length, $\bar{L}_k$, for an Ethernet device k is estimated by the following formula:

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

1. It is difficult to estimate how many stations besides the system under evaluation will simultaneously access the Ethernet and cause contention. In our case, the problem is that multiple PPS6 clients are running on a single WinFrame server, but from the Ethernet point of view only one PPS6 client is accessing the network.

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

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

Both formulas use $X_c(\overline{N})$, the estimated throughput for a class c customer in the previous iteration [4,13], 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 [15]. Measuring details of user workloads and resource consumption is straightforward for mainframe systems such as MVS. However, this task becomes complicated and difficult in a multi-tiered environment due to the lack of centralized and transaction-oriented monitoring facilities. Therefore, developing accurate measurement methodologies is vital to and should be coordinated with analytic modeling.

3.1 Measurement Testbed

A measurement testbed is used to measure model parameters and perform the initial validation tests. 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 that also provides connectivity to the campus network, but can be configured to isolate the testbed during performance data gathering.

A database exerciser, called *Stress*, is used as the load generator against the testbed Oracle server [4]. *Stress* allows manual selection of database access functions and supports automatic request generation for synthetic database workloads. When used in the latter 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 Model Parameter Measurement and Estimation

According to the analytic model, the service requirement of a *Stress* request is described with four model parameters: the server CPU service demand (D_{cpu}), the server disk service demand (D_{disk}), the number of packets generated (N_{pkt}), and the average packet length of these packets (L_{pkt}). Although the goal is clear, there are no monitoring tools in AIX that measure these parameters. This is because most UNIX systems do not have the concept of an application transaction.[1] Therefore, the Oracle server sees only a stream of database accesses (SQL statements) from clients; there is no transaction identifier which can be used to associate server resource consumption with the end-user work unit. Although Oracle servers provide a trace facility, SQL trace and *tkprof*, the resolution is limited to one hundredth of a second, which is too coarse for the short SQL statements generated by *Stress* or PPS6 clients. As a consequence, the only feasible solution at present is to use general tools, such as *iostat*, *sar*, *tcpdump*, and *tcptrace*, available on most UNIX platforms.

1. Transaction processing monitors such as Tuxedo and Encina are relative newcomers to UNIX environments.

Iostat can report system statistics of CPU and disk utilization at specific time intervals with the granularity of seconds. Our measurement monitors the following three 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.
- *%tm_act* The percentage of time the physical disk was active (busy).

These performance metrics can be converted directly into CPU and disk utilization: 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* can display the headers and payload of packets captured by a network interface. *Tcptrace* can analyze 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 statements generating a large number of packets, *tcptrace* can produce a statistical analysis of the packet trace.

Three different *Stress* requests - *Stress insert*, S*tress update*, and *Stress delete* - have been measured for initial validation tests. 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 to derive the average completion time per request and the server throughput. The average CPU time consumed by the *Stress* program on the client machine is measured with *pview*, a Windows NT 3.51 process monitoring tool. For each type of *Stress* request, we repeat the same measurement with different numbers of concurrent clients. The measurement data are presented in Tables 3-5. Table 6 shows the number of SQL*Net packets, the average packet length, and the client CPU time consumed by *Stress* for each type of request.

Table 3. 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 4. 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 5. 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 6. 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

For each performance measurement, we set up *Stress* to generate consecutive synchronous requests. 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 3-5. The number of *Stress* clients varies from one to six except for *Stress delete*. The mean and standard deviation of the estimated service demands are listed at the end of each table. Unlike *Stress insert*, the service demands of *Stress update* and *Stress delete* are quite similar, and they even generate the same number of SQL*Net packets. This is because the *Stress* program uses a query statement to locate the record before updating or deleting it. Therefore, *Stress insert* requires less CPU time for service because each request is accomplished in a single SQL statement.

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.[1] This undesired phenomenon will cause a discrepancy between the measurement data and the model outputs.

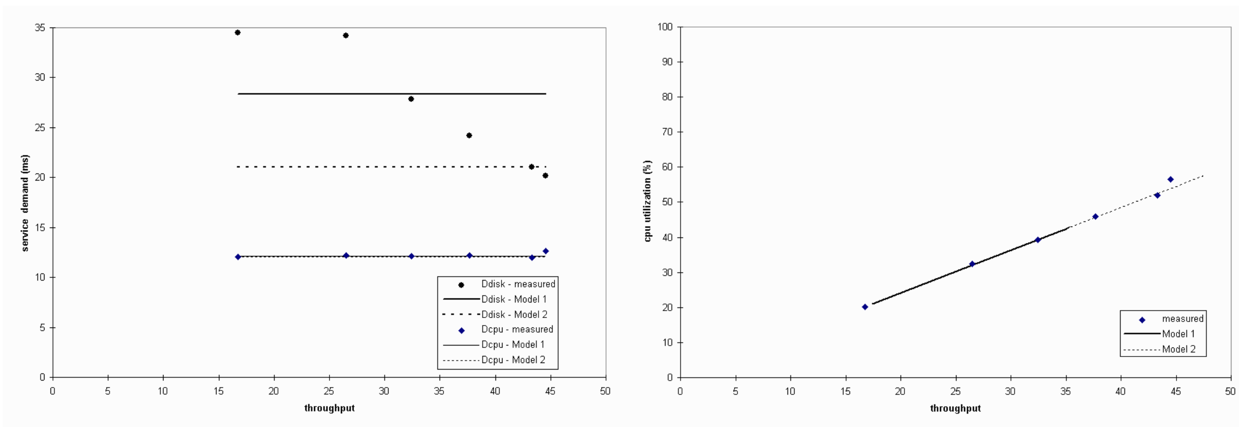
3.3 Model Validation and Performance Evaluation

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).

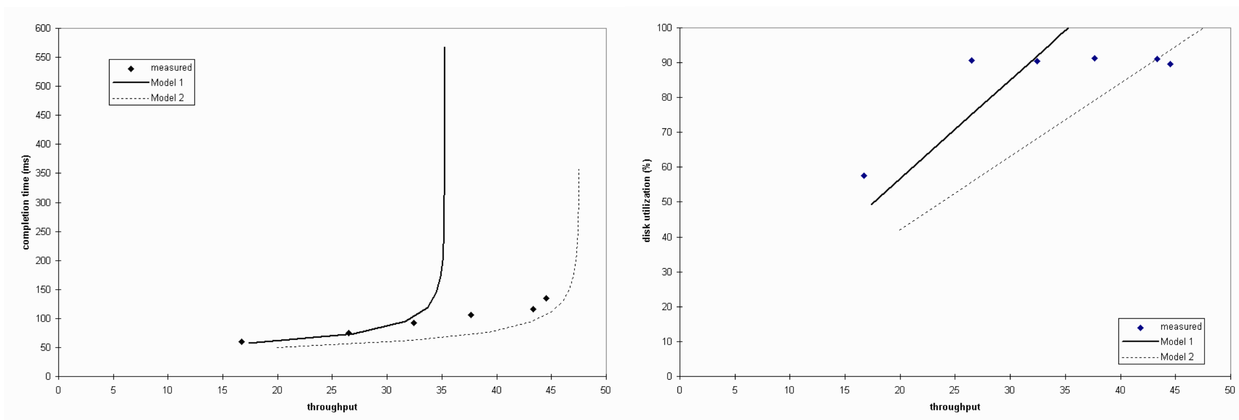
When the input parameters are applied, the *MVA* tool determines the performance measures for a fixed client population (number of concurrent clients) with zero think time. We increase the client population one at a time until the disk, which is the bottleneck device, is saturated. The model outputs and the measurement data are compared with three performance curves: completion time (the average residence time in a job class), CPU utilization, and disk utilization, all against throughput. These performance curves for

1. Similar measurement results have also been reported in [8]. We believe several reasons may cause the phenomenon. First, the reduced disk service time is most likely due to the Oracle SGA buffer caching and UNIX disk buffer caching. Second, the increasing CPU service time is probably due to the contention for shared Oracle resources such as free lists, redo log buffers, locks, and latches.

Stress insert are shown in Figure 7. Performance curves for *Stress update* and *Stress delete* are available in our technical report [4].



(a) service demands (c) CPU utilization



(b) completion time (d) disk utilization

Figure 7. Comparison of measurement data and model outputs for *Stress insert*.

Figure 7a 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 completion time (Figure 7b) and disk utilization (Figure 7d) 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 (Figure 7c).

The *MVA* tool determines that the Oracle server saturates with a throughput of 35 rps for Model 1 and 47.55 rps for Model 2. The maximum throughput can also be derived from the inverse of the disk service demands applied to the model (1/0.028 and 1/0.021) 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 for S*tress insert*. However, another undesired phenomenon arises in that the measured CPU service demand increases for *Stress update* and *Stress delete* as more clients are added. 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 overall disparity in measured service demands.

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 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.

3.4 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 [12]. The technique is applied here to estimate service demands for a larger 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 of different client populations 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 regression analysis to our measured service demands for one to five concurrent clients to estimate the service demands for six or more clients.

To discover the trend of the service demands for *Stress delete*, the measured values are plotted against the number of concurrent clients in Figure 8. We then 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 8) and curvilinear regression for the disk service demand (the bold curve). 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 9.

# of clients	measured D_{cpu}	measured D_{disk}	regression D_{cpu}	regression 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

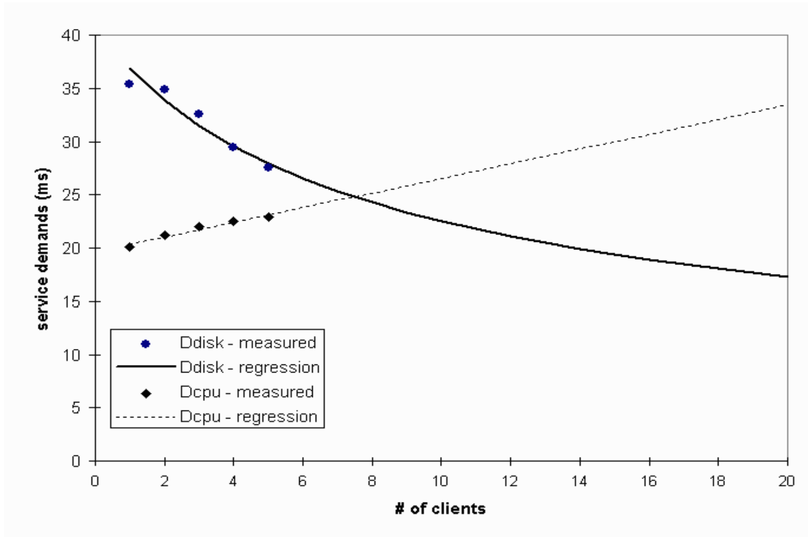
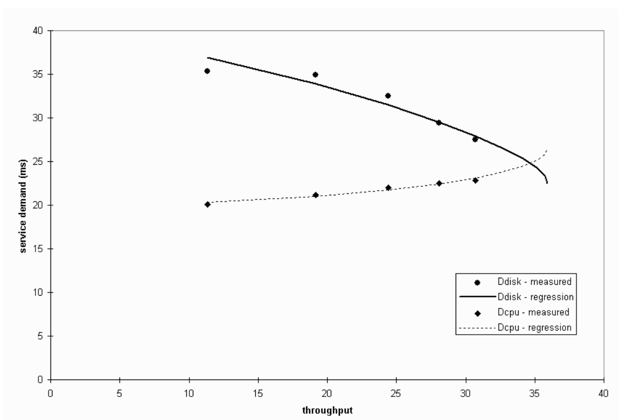
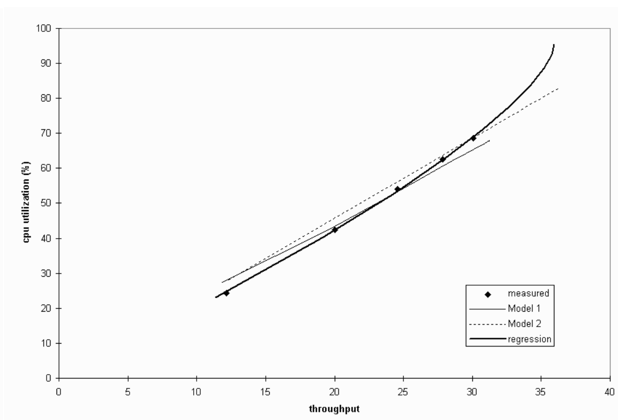


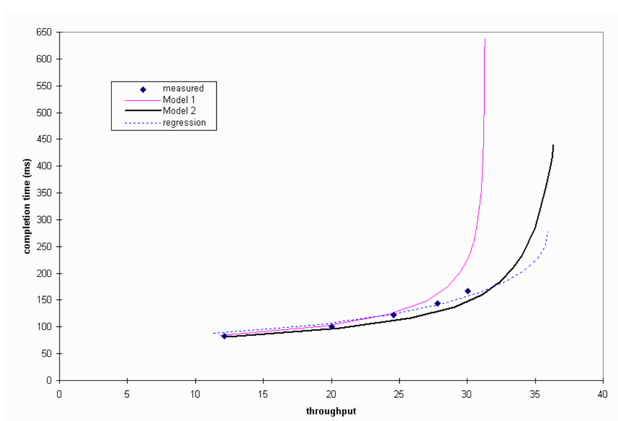
Figure 8. *Stress delete*: regression analysis for service demand estimation.



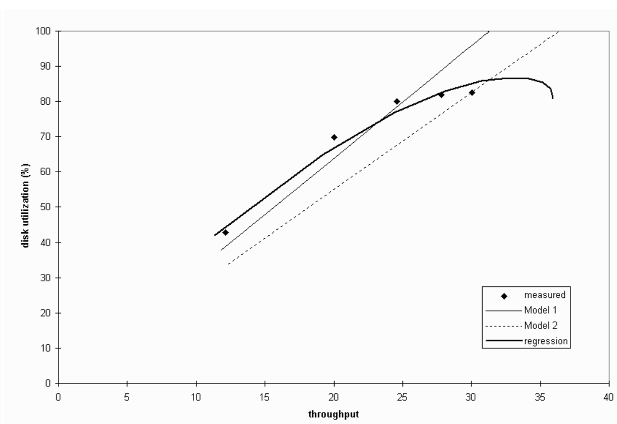
(a) service demands



(c) CPU utilization



(b) completion time



(d) disk utilization

Figure 9. Comparison of performance measures and three model outputs for *Stress delete*.

Figure 9a 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 9b-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 accurate because of limits in the regression method itself; the error in predicted value grows larger as the independent variable increases [12]. In addition, we observe that while the disk service demand curves must eventually approach the horizontal our model does not predict this behavior.

From the above experiment, we observe that regression analysis is an attractive alternative when performance measures do not provide enough information for estimating model parameters. This method nevertheless has restrictions. First, it is not very accurate at high throughput values. Second, 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. Third, 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 an LD service center requires information about how its service demand varies with its client population. These issues will be addressed in our future research plan.

4. Preliminary Investigation of Citrix WinFrame Servers

Citrix WinFrame allows Windows applications to be deployed in a network-centric, three-tiered architecture: the application execution and data storage occur on central servers, and only a “thin” piece of client software is required at the client system. The three-tiered deployment is accomplished with Citrix’s universal thin-client software, the multi-user NT server, and the Independent Console Architecture.

4.1 Citrix’s Independent Console Architecture

A distributed Windows presentation separates the graphic user interface (GUI) of an application from its execution logic. In Windows NT 3.51 and previous releases, the window manager and graphics subsystems are implemented as a separate user-mode process called the Win32 subsystem (or *csrss*) [7]. In order to redirect the Windows display to a remote machine, Citrix adds a Thinwire component into the Graphics Device Interface (GDI) and video drivers of the Win32 subsystem. The Thinwire component implements the Thinwire protocol which provides highly optimized drawing primitives for Windows presentation. The WinFrame server uses a separate instance of *csrss* to manage the Windows display for each WinStation.

ICA divides its functions into individual protocol drivers layered as a protocol stack (shown in Figure 10). The Thinwire data protocol relies on the *ICA protocol* to provide reliable, in-sequence delivery of data. Additional protocol drivers can be configured to supplement the ICA protocol with functions such as compression, encryption, framing, reliable delivery, and modem control. The bottom layer is the network transport driver (TCP/IP, IPX/SPX, NetBIOS, or PPP/SLIP) provided by Windows NT.

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

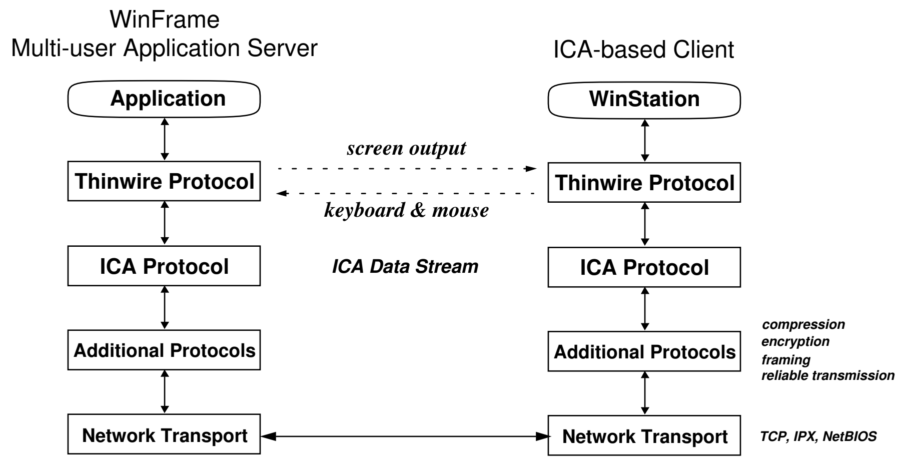


Figure 10. ICA Protocol Stack.

The Thinwire protocol uses *window frames* (WdFrames) to carry presentation information in both directions of the WinStation connection. At the WinStation, WdFrames carry key-strokes and mouse movements; at the WinFrame server, WdFrames carry Windows objects (bitmaps, brushes, glyphs, and pointers) for display at the WinStation. ICA can accommodate low network bandwidth for its presentation service because small Windows objects are cached in client memory and large bitmaps are persistently cached on the client's disk. Windows objects therefore are transported across the network on first reference; most subsequent references can be resolved through local caches. In addition, multiple WdFrames are encapsulated (or batched) in a single ICA packet to reduce the network traffic.

4.2 Characteristics of ICA Overhead

Remote computing in the WinFrame environment requires extra CPU time to manage the distributed presentation. 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 GUI is actually displayed. Therefore, we further distinguish the *ICA-client overhead* from the *ICA-server overhead* at the WinFrame server. Since ICA overhead is incurred by the distributed presentation, the GUI design of a Windows application is the key factor in determining the amount of overhead generated. However, several configuration settings of a WinStation connection can also affect ICA overhead. The two most important are the encryption and compression settings because they cause extra CPU time for each ICA packet [5]. Other configuration settings affecting the ICA overhead include the size and the color depth of the window display at the WinStation.

4.3 Performance Measurement of ICA Overhead

ICA overhead can be measured by the *task manager*, a process monitoring utility, or by *perfmon,* a general monitoring tool. Since the WinFrame server uses a separate *csrss* process to manage distributed presentation on behalf of each WinStation, WinStation-induced ICA-server overhead can be measured against the specific *csrss* process associated with each WinStation. Similarly, we can measure the ICA-client overhead against two specific processes, called *wengfN.exe* and *wfcrun32.exe*[1]. Table 11 shows the performance measures of the ICA overhead for running the *Stress* program in the three-tiered configuration. Performance measures in the shadow area are actually measured; the other measures are derived values. The measurement focuses on the ICA traffic rate and the CPU consumption of three processes - *Stress* itself, *csrss* at the

1. *Wfcrun32.exe* consumed no CPU time in our experiments, so we do not consider it further here.

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 11. Performance measurement of ICA overhead on behalf of *Stress* program.

# of DB calls	C (sec)	CPU time (sec.)			%CPU			ICA	ICA Overhead (us/byte)	
		WinServer		WinStation	WinServer		WinStation		WinServer	WinStation
		stress	*csrss*	*wengfN*	*stress*	*csrss*	*wengfN*	(bytes/sec)		
500	38.22	7.87	7.06	39	20.59	18.47	102.04	6970.53	26.49	146.39
1000	75.77	15.47	14.70	73	20.42	19.40	96.34	7214.87	26.89	133.54
1500	113.83	24.51	21.70	109	21.53	19.07	95.76	6926.25	27.53	138.25
2000	152.91	32.36	29.78	143	21.16	19.48	93.52	6926.32	28.12	135.02
2500	190.81	40.65	37.27	182	21.30	19.53	95.38	6847.54	28.52	139.30
3000	229.64	46.43	44.24	219	20.22	19.27	95.37	6684.91	28.82	142.66
3500	269.13	55.23	52.43	256	20.52	19.48	95.12	6759.26	28.82	140.73
4000	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 11. First, the cost of the distributed presentation service provided by ICA on behalf of *Stress* is very high. It consumes about 96% of the CPU time at the WinStation and 19% of the CPU time 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 have not yet found a good explanation for this five-fold overhead difference. Third, the average ICA traffic rate is low, less than 0.6% of Ethernet bandwidth, even though our test has an artificially high GUI update rate.

4.4 Comparison with Two-Tier

To further investigate the cost of remote computing, we measured the completion time and CPU consumption for running *Stress* in both two-tiered and three-tiered configurations. The measurement data are shown in Figure 12 with different numbers of database calls.

# of DB calls	2-tiered *Stress*		3-tiered *Stress*		
	CPU	C	CPU	C	wengfN
500	8.03	32.73	8.47	38.22	39
1000	15.62	63.61	16.08	75.77	73
1500	23.28	98.58	24.75	113.83	109
2000	31.09	130.59	31.58	152.91	143
2500	37.97	166.34	39.23	190.81	182
3000	45.72	200.31	47.83	229.64	219
3500	55.03	235.34	55.91	269.13	256
4000	61.33	266.92	63.75	307.81	293

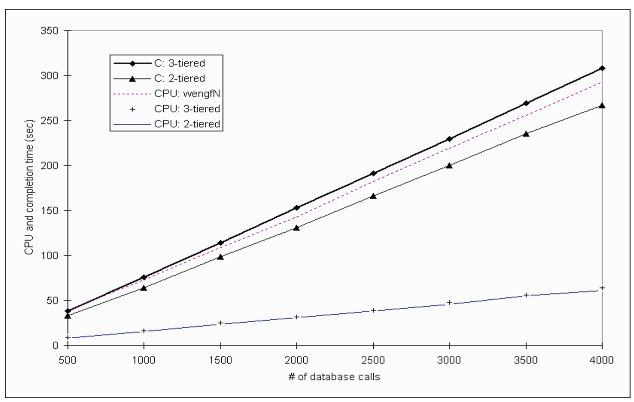
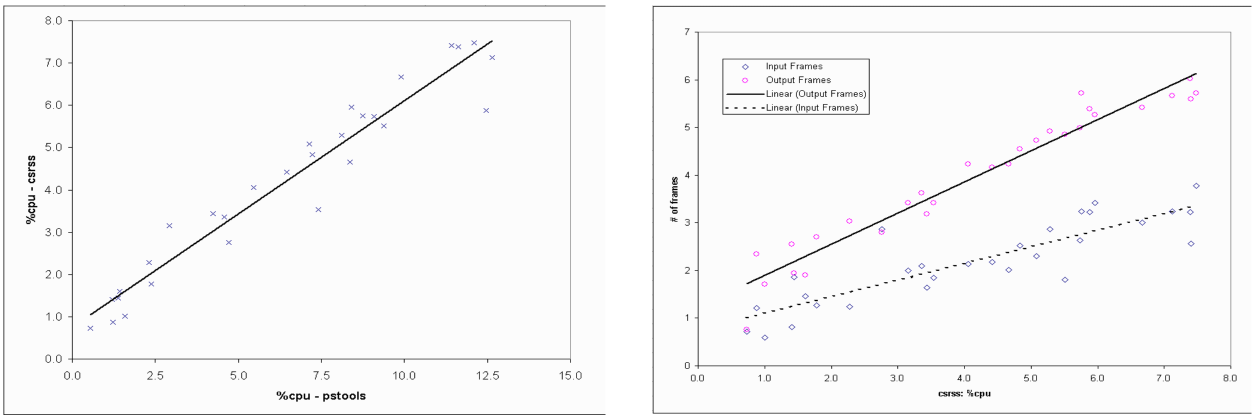


Figure 12. Measurement of application impact of Citrix WinFrame.

Comparing the measurement data in both cases, we can draw three conclusions. First, the *Stress* program consumes about an equal amount of CPU time in both configurations. 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 configuration, 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 the *Stress* program itself.

4.5 Performance Impact of Citrix in Production Environments

Modeling the WinFrame extension requires careful analysis about how ICA consumes system resources in the forms of CPU time on WinFrame servers and WinStations, and traffic rate on the Ethernet segment connecting the WinFrame server and the Oracle server. Our preliminary investigation has allowed us to develop a measurement methodology to characterize the ICA overhead. Based on the technique, we have analyzed the ICA overhead on measurement data from live stress testing of PPS6 applications in the production environment. We observe a linear relationship in CPU consumption between the PPS6 client (*pstools*) and *csrss* (Figure 13a), and between the CPU consumption of *csrss* and the number of WdFrames generated (Figure 13b). This discovery allows us to estimate resource consumption in units of PPS6 transactions and to develop an analytic model for the WinFrame extension.



(a) %cpu: *csrss* vs. *pstools*.

(b) # of frames vs. %cpu of *csrss*.

Figure 13. Resource consumption estimation of ICA overhead in the production environment.

5. Conclusions and Future Work

We presented the modeling and measurement of client-server configurations in a multi-tiered environment featured in the University of Michigan production deployment of the PeopleSoft 6 (PPS6) client/server system. We used a closed QNM to model the two-tiered PPS6, and built a performance modeling tool to evaluate the model via MVA. 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. To address this problem, we used regression analysis to estimate the varying service demands for multiple concurrent users. The model outputs closely matched the measured values after we applied the new service demands estimated with the regression analysis. However, we still have to validate the regression approach under mixed workloads when performance measurements in such an environment are feasible. We also performed some preliminary measurements for running Windows applications in a three-tiered Citrix environment, both by monitoring network traffic and by measuring CPU utilization. The study provides some valuable insights for estimating the extra resource consumption caused by the ICA overhead as required for modeling the three-tiered extension.

The major problem we have encountered is the lack of transaction-oriented monitoring facilities that charge system resource consumption to individual workloads in a multi-tiered environment. In addition, the tracing and monitoring facilities on Oracle servers do not provide detailed performance measures to study how service requirements vary with concurrent users and mixed workloads. In order to address this issue, we plan to combine several monitoring tools to estimate service demands as suggested in [8]. These tools include *iostat* and *sar* at the operating system level, Oracle dynamic performance tables (v$sysstat and v$sesstat) at the Oracle system level, and SQL trace and *tkprof* at the level of individual SQL statements.

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. Yi-Chun Chu and Charles J. Antonelli. “Modeling and Measurement of the PeopleSoft Multi-Tier Remote Computing Application.” Technical Report, CITI-TR-97-04, Center for Information Technology Integration, University of Michigan, December 1997.
5. Citrix. *ICA Technical Paper*. [http://www.citrix.com/technology/icatech.htm], 1996.
6. Citrix. *Thin-Client/Server Computing*. [http://www.citrix.com], 1997.
7. Helen Custer. *Inside Windows NT*. MicroSoft Press 1993.
8. Tim Foxon. “Performance Analysis of an Oracle-based Interactive UNIX System - a Case Study.” *Proceeding of CMG’94*, December 1994.
9. Adam Grummitt and Tim Foxon. “Performance Tuning and Capacity Planning for Oracle on UNIX - A Case Study.” *Proceeding of CMG’97*, December 1997.
10. J.L. Hammond and P. O’Reilly. *Performance Analysis of Local Computer Networks*. Reading, MA: Addison-Wesley, 1986.
11. S.S. Lam. “A Carrier Sense Multiple Access Protocol for Local Networks.” *Computer Networks*, Vol. 4, No. 1, pp. 21-32, February 1980.
12. Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
13. Masud A. Khandker. *Performance Measurement and Analytic Modeling Techniques for Client-Server Distributed Systems*. Ph.D. Thesis, University of Michigan, 1997.
14. Edward Lazowska, John Zahorjan, Scott Graham, and Kenneth Sevcik. *Quantitative System Performance*. Prentice Hall, 1984.
15. Daniel Menasce, Virgilio Almeida, and Larry Dowdy. *Capacity Planning and Performance Modeling: from Mainframes to Client-Server Systems*. Prentice Hall, 1994.
16. M-Pathways. *The M-Pathways Project*. [http://www.mpathways.umich.edu], 1997.
17. PeopleSoft. *PeopleSoft’s Distributed Architecture*. [http://www.peoplesoft.com], 1997.
18. J.A. Rolia. *Predicting the Performance of Software Systems*. Technical Report, CSRI TR260, University of Toronto, 1992.
19. Huge Toledo. *Oracle Networking*. Oracle Press, 1996.