CITI Technical Report 98–3

Performance Measurement of the PeopleSoft Multi-Tier Remote Computing Application

Yi-Chun Chu
ycchu@citi.umich.edu

Charles J. Antonelli
cja@citi.umich.edu

Toby J. Teorey
teorey@eecs.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.

This report describes several experiments designed to evaluate the performance of PeopleSoft 6 remote computing applications being implemented at the University of Michigan. The purpose of these experiments is to develop a proper measurement methodology for quantifying the resource requirements of PeopleSoft OLTP transactions in a three-tier architecture consisting of client WinStations, WinFrame servers, and Oracle database servers. The measurement results are applied to an analytic three-tiered performance model and can be used to generate sizing estimates for production system components.

June 1998

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

Executive Summary

In the previous 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 Oracle CPU and disk service demands remained constant; however, in several cases the Oracle service demands were found to vary with the number of clients, causing major discrepancies.

In this second phase of work, we address the problem of varying Oracle service demands by developing a new measurement methodology based on the Oracle's *dynamic performance tables*. We separate the Oracle CPU service demand into the CPU service times of Oracle server processes, Oracle background processes, and the OS overhead. More importantly, we also decompose the Oracle disk service demand into three smaller components: *data block read time*, *data block write time*, and *redo log time*. While exercising the new measurement methodology with the *Stress* program in the CITI testbed, we determined that only the CPU service demand of Oracle server processes and the redo log time cause the varying Oracle service demands.

In order to consider the decomposed Oracle service demands measured with our new measurement methodology, we have also refined the two-tiered analytic model into a Layered Queueing Model (LQM). Validating this model via the *Stress* program, we find that the model outputs predict the request completion time within a 5% error range, and the Oracle CPU and disk utilizations within 10%.

We have further extended the measurement methodology to incorporate the remote computing model of PPS6 in the U-M production environment. The measurement extension allows us to collect the CPU usages on WinStations and WinFrame servers as well as the ICA traffic generated for individual WinStation users. The performance measures collected with the extended measurement methodology can be associated with the corresponding Oracle resource usages and converted into model parameters. This allows us to extend the two-tiered LQM into a three-tiered LQM for modeling the remote computing of PPS6.

We have not been able to obtain an accurate characterization of typical PPS6 application workloads for inclusion in this study because of our inability to access real PeopleSoft applications running against real databases in the U-M production environment. To address the problem, we have installed the PPS6 application, the *PeopleSoft Financials 6 for Public Sector* suite, and a demonstration version of PPS6 application databases in the CITI testbed. In the next phase of work we plan to measure this PeopleSoft application and to record client, network, and server utilizations, as well as the number of PeopleSoft transactions generated by each PeopleSoft panel. These performance measures can be converted into model parameters very close to those of the real PPS6 in the U-M production environment.

Performance Measurement of the PeopleSoft Multi-Tier Remote Computing Application

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

June 1998

1. Introduction

The Center for Information Technology Integration has undertaken a study of the potential impact of PeopleSoft software on production computing environments at the University of Michigan. In previous work, we have constructed a closed queueing network model of a two-tiered PeopleSoft 6 distributed application (henceforth called PPS6) [4]. In evaluating this model, we determined that the model outputs closely matched the measured values in those cases where service demands remained constant with increasing numbers of clients. However, in several cases the Oracle CPU and disk service demands varied with the number of clients, causing discrepancies between the model and the system. We also did some preliminary three-tiered measurement work.

In this work, we continue our efforts to address these issues. First, we have refined the model by decomposing the disk service demand into three constituent components, isolating the disk service demand variance in one of the components. Second, we have conducted performance measurements in our three-tiered production environment, in particular validating our assumption that the component exhibiting varying service demands constitutes a small part of the overall system I/O, and gathering real data to complement our synthetic workloads. Finally, we have replicated the three-tiered Peoplesoft 6 environment on our testbed, permitting more detailed measurements with better control of workload than that possible in our production environment.

The rest of this report is organized as follows: We introduce our performance measurement framework below. Section 2 introduces the measurement methodology for collecting the Oracle resource usages on behalf of PeopleSoft transactions in a two-tiered environment. Section 3 extends the measurement methodology to three-tiered PPS6. Section 4 discusses the issues for analytic modeling of PPS6 and how to estimate the Oracle service demands for the analytic models. The conclusion and future work follow in Section 5.

1.1 The U-M Production Deployment of PPS6

PPS6 is a two-tier client-sever application with "thick" client components [25]. The production deployment of PPS6 at the University of Michigan is based on Citrix WinFrame servers [6]. This deployment provides a more secure environment for running PPS6 client programs; it also reduces the administration effort for distributing the PPS6 client software to individual user desktop machines [4,21]. The deployed PPS6 hence resembles a generic three-tier client-server architecture: the user desktop machine (called a *WinStation*) handles the presentation service; the WinFrame application server executes the PPS6 client programs; and the Oracle server hosts the PPS6 application databases. Figure 1 shows the deployed three-tiered architecture of PPS6 and the target processes for performance measurement in each tier.

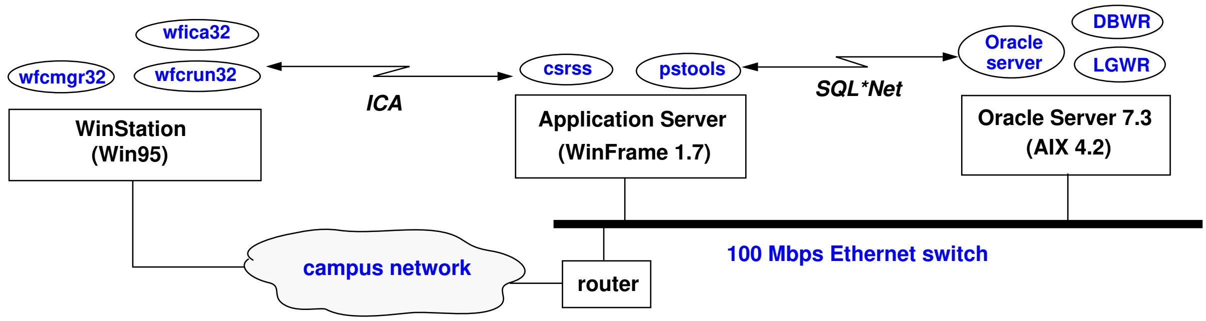


Figure 1. Target processes for performance measurement in the U-M production deployment of PPS6.

1.2 PeopleSoft Transactions

Performance measurement of database applications defines application workloads in units of transactions [14,18]. For PPS6, we can define a PeopleSoft transaction as a sequence of SQL statements generated by a PPS6 client program, called a *panel* in PeopleSoft terminology, while a user works through a sequence of dialogues and enters data into various fields in order to complete a business function such as general ledger or procurement. We can characterize the PeopleSoft transactions of a PPS6 panel by collecting the SQL statement traces generated by the panel and identifying the transaction boundary in the SQL statement traces. SQL statement traces can be collected with Oracle *SQL_TRACE* [12] or PeopleSoft *SQL Trace* [26]; both trace reports record a limited set of performance statistics for each SQL statement traced. After identifying the sequence of SQL statements comprising different PeopleSoft transactions, we can apply them to build a simulated workload with database stress testing tools.

1.3 Stress Testing Tools

Performance measurement efforts are usually hampered by the problem of setting up test workloads with enough test users. We can alleviate this problem with special stress testing tools capable of generating legitimate application workloads with minimum human intervention. Examples of such tools for benchmarking database servers are available in commercial packages, such as *Load Runner* [19] and *PreVue-C/S* [27]. These tools greatly facilitate the performance measurements of PPS6 database servers because they can be configured to generate simulated workloads with specific mixes of different PeopleSoft transactions.[1] However, these database stress testing tools cannot generate a legitimate client workload for a PPS6 panel, which is further complicated by PPS6's remote computing model.

For stress testing of three-tiered PPS6, an ideal tool would drive the PPS6 panels at WinStations with keyboard and mouse events similar to real users interacting with PPS6 panels. One type of tool requires hardware to be installed in each client, and is therefore too expensive and does not scale well. A commercial software package, called *WinBatch* [29], can generate keyboard events for Windows applications, but it does not handle mouse events well. For these reasons, we have not performed any automated testing, and have relied on live user testing.

1. We conduct several performance measurements with our own database stress testing tool, called *Stress* [4]. The *Stress* program generates simple database requests, called *Stress requests*, comprising one or two SQL statements.

1.4 Performance Measurement Environments

Most performance measurements in this paper were conducted in our testbed environment. We set up an additional measurement testbed because it provides a controlled environment without external interferences. 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, H/W: 133MHz PowerPC 604 and 256 MB RAM, S/W: AIX 4.2 and Oracle server 7.3.2). These three machines are attached to a private 10-Mbps Cisco 1900 switch which also provides connectivity to the campus network.

2. Measurement Methodology for the Oracle Database Server

Analytic modeling of Oracle-based systems uses a set of model parameters, called the Oracle service demands, to specify the CPU time, disk time(s), and SQL*Net traffic for Oracle to service a transaction. The Oracle service demands require accurate performance measures about system resources consumed by Oracle. However, accurate measurement techniques applicable to PeopleSoft applications are difficult to develop because PeopleSoft transactions, unlike general OLTP transactions, actually comprise a long sequence of SQL statements. In this section, we describe the measurement techniques for collecting Oracle resource usages based on Oracle's *dynamic performance tables*, known as the *v$tables*. Since this technique requires some knowledge about the Oracle internal architecture, we start this section with a short technical introduction to Oracle database servers.

2.1 Overview of The Oracle7 Server

An Oracle database instance comprises a set of Oracle processes, *server* and *background* processes, as well as the *System Global Area* (SGA), a group of shared memory structures [22]. Oracle creates server processes to handle requests from an Oracle application connected to the database instance. The actual number of server processes depends on the configuration of an Oracle instance as well as the number of Oracle user sessions. In a *dedicated server* configuration, a server process only handles requests from a single Oracle user session. In a *Multi-Threaded Server* (MTS) configuration, a pool of server processes are shared among all Oracle user sessions to reduce memory usage and process management. Oracle background processes are created at Oracle startup time to perform system functions as described in Table 2. We ignore the efforts of all but DBWR and LGWR in our analysis because the rest of background processes together consume a very small amount of system resources.

Table 2. Functions of Oracle background processes.

Name	Description
Process Monitor (PMON)	PMON performs process recovery when a user process fails; it cleans up the cache and frees resources that the process was using. PMON also restarts failed dispatchers and shared server processes.
Database Writer (DBWR)	DBWR manages the database buffer cache in SGA and writes modified (dirty) buffers to datafiles on disk. It manages the buffer cache with an LRU algorithm and guarantees server processes can always find free buffers. DBWR is signaled to write dirty buffers under the following four conditions: when the dirty list reaches a threshold length, when a free buffer has not been found after a threshold limit of buffer scan, when a timeout occurs (every three seconds), and when a checkpoint occurs. Whenever possible, DBWR writes dirty buffers to disk with a single *multiblock write*.
Log Writer (LGWR)	LGWR writes the redo log buffer in SGA to the online redo log file on disk. LGWR is triggered during several internal events: it writes a commit record when a user process commits a transaction; it writes one contiguous portion of the log buffer to disk every three seconds, when the redo log buffer is one-third full, and when DBWR writes modified buffers to disk. In times of high activity, LGWR may write to the redo file with *group commits* which generates less disk writes than writing each commit record individually.
System Monitor (SMON)	SMON performs instance recovery during startup, cleans up temporary segments that are no longer in use, recovers dead transactions skipped during crash and instance recovery because of file-read or off-line errors, and coalesces small chunks of contiguous free space into larger blocks of contiguous space for easier allocation by Oracle.
Checkpointer (CKPT)	The optional CKPT updates the headers of all data files during checkpoint processing. If the init.ora CHECKPOINT_PROCESS parameter is enabled, this process can improve system performance by freeing the LGWR to concentrate on the redo log buffer. The CKPT process does not write data blocks to disk; this is performed by the DBWR.
Recoverer (RECO)	RECO is started when distributed transactions are permitted and the init.ora parameter DISTRIBUTED_TRANSACTIONS is greater than zero. The Oracle distributed option uses the RECO process to automatically resolve distributed transaction failures.
Archiver (ARCH)	ARCH copies online redo log files, once they become full, to a specified storage device or location. ARCH is present only when the database is started in ARCHIVELOG mode and automatic archiving is enabled.

The Oracle SGA contains data and control information for a database instance shared by all Oracle processes [1,22]. It is implemented as a group of shared memory structures comprising three main components:

- **Database Buffer Cache**: The database buffer cache holds copies of data blocks read from datafiles on disk. The buffers are organized in two lists: the dirty list and the LRU (least recently used) list. The dirty list holds modified (*dirty*) buffers that have not yet been written to disk by DBWR. The LRU list holds *free* buffers (buffers that have not been modified), *pinned* buffers (buffers that are currently being accessed), and dirty buffers that have not yet been moved to the dirty list.
- **Redo Log Buffer**: The redo log buffer is a circular buffer that holds *redo entries* - information about changes made to the database. Redo entries are copied from the user's memory space to the redo log buffer in SGA by Oracle server processes. The background process LGWR writes the redo log buffer to the online redo log file on disk.
- **Shared Pool**: The shared pool contains three major areas: *library cache*, *dictionary cache*, and *control structures*. The library cache is further divided into *shared SQL areas* (memory areas containing the parse trees and execution plans of parsed SQL statements), *private SQL areas* (memory areas containing bind information and runtime buffers), PL/SQL procedures and packages, and additional control structures such as locks and library cache handles.

2.2 Performance Measurement Tools

Oracle's performance monitoring tools record system resource usages consumed by Oracle at the SQL statement level, the Oracle session level, and the Oracle system level. Since Oracle processes are just user processes from the OS perspective, performance measurement of Oracle servers usually requires monitoring facilities at the OS level to calibrate measurement data [9,11].

2.2.1 Collecting Oracle Resource Usages at the SQL Statement Level

The Oracle tools which allow performance monitoring of individual SQL statements are SQL_TRACE and its companion tool *TKPROF*; however, they are intended to be used for examining poorly written SQL statements that are time-consuming or resource-intensive [12,24]. SQL_TRACE records SQL statement traces generated by an Oracle user session; trace files are formatted with TKPROF into a detailed report shown in Table 3.

Table 3. A sample SQL statement in the TKPROF report of PeopleSoft traces

```
SELECT UNIT_OF_MEASURE, DESCR, DESCRSHORT
FROM
 PS_UNITS_TBL WHERE UNIT_OF_MEASURE='EA' ORDER BY UNIT_OF_MEASURE
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	15	0.01	0.02	0	0	0	0
Execute	15	0.00	0.00	0	0	0	0
Fetch	15	0.04	0.04	24	30	0	15
total	45	0.05	0.06	24	30	0	15

```
Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 244
```

In a TKPROF report, the execution of a SQL statement comprises Oracle internal calls such as *parse*, *execute*, and *fetch*. The performance statistics are categorized by these calls and their descriptions are listed below:

- *count* — Total number of times a statement was parsed, executed, or fetched.
- *cpu* — Total CPU time in seconds for all parse, execute, or fetch calls of the statement.
- *elapsed* — Total elapsed time in seconds for all parse, execute, or fetch calls of the statement.
- *disk* — Total number of data blocks physically read from datafiles on disk for all parse, execute, or fetch calls.
- *query* — Total number of buffers retrieved in *consistent* mode for all parse, execute, or fetch calls; buffers are often retrieved in consistent mode for queries, by SELECT statements.
- *current* — Total number of buffers retrieved in *current* mode; buffers are often retrieved in current mode for INSERT, UPDATE, and DELETE statements.
- *rows* — Total number of rows processed by the SQL statement. For SELECT statements, the number of rows returned appears for the fetch step. For UPDATE, DELETE, and INSERT statements, the number of rows processed appears for the execution step.

For our analysis, the value of a TKPROF report is in the column marked *disk* because it can be used to estimate how many data blocks are read from disk by the traced SQL statement. A major limitation of SQL_TRACE and TKPROF is the poor resolution in timing statistics, which is based on Oracle clock ticks in hundredths of a second. Any operation that takes time less than or equal to an Oracle clock tick may not be timed accurately. This coarse timing resolution makes it impossible to track the CPU usage of "short" SQL statements [24].

2.2.2 Collecting Oracle Resource Usages at the Oracle Session Level

V$sesstat is one of the *v$tables* that records performance statistics at the Oracle session level [1,2,23]. A session in v$sesstat can be an Oracle application connected to the Oracle instance as well as an Oracle background process such as DBWR or LGWR. V$sesstat records 161 performance statistics per session categorized into 8 different classes: user, redo, enqueue, cache, OS, parallel server, SQL, and debug. Those statistics relevant to the resource usages of individual sessions are:

- **CPU Usage**: For user sessions, v$sesstat records the CPU time for servicing user requests (*CPU used by this session*) and the portion of CPU time for parsing SQL statements (*parse time cpu*). For background processes, v$sesstat records the CPU time for performing system-wide functions (*OS User time* and *OS System time*).
- **Disk Access**: For user sessions, v$sesstat records the cumulative number of data blocks *physically* read from disk (*physical reads*). For DBWR, v$sesstat records the cumulative number of data blocks written to disk (*physical writes*) and the number of times DBWR is signalled to flush dirty buffers (*write requests*). For LGWR, v$sesstat records the number of times LGWR is signalled to write redo buffers (*redo writes*), the cumulative number of redo blocks written to disk (*redo blocks written*), and the cumulative elapsed time spent on log I/O (*redo write time*).
- **SQL*Net Traffic**: For user sessions, v$sesstat records the cumulative number of bytes in SQL*Net message sent to the client (*bytes sent via SQL*Net to client*) and received from the client (*bytes received via SQL*Net from client*) as well as the total number of round-trip packets exchanged (*SQL*Net roundtrips to/from client*).

2.2.3 Collecting Oracle Resource Usages at the Oracle System Level

Several Oracle *v$tables* record system-wide performance statistics of an Oracle instance from the time it starts. V$sysstat and v$filestat are two of these that can be used to collect Oracle resource usages at the Oracle system level. V$sysstat maintain the same 161 performance statistics as v$sesstat but it records them system-widely. V$filestat records the disk I/O statistics of individual datafiles in terms of 6 performance statistics:

PHYRDS	total number of read requests to a datafile,
PHYWRTS	total number of write requests to a datafile,
PHYBLKRD	total number of data blocks read from a datafile,
PHYBLKWRT	total number of data blocks written to a datafile,
READTIM	total disk read time in milliseconds, and
WRITETIM	total disk write time in milliseconds.

Since the redo log file is not a datafile, Oracle does not record disk I/O statistics about redo logging in v$filestat. Instead, Oracle records *redo writes*, *redo blocks written*, and *redo write time* in v$sysstat and in v$sesstat of LGWR. Note that redo blocks are a different size from data blocks and that *redo write time* is recorded in Oracle clock ticks; see Section 2.2.1.

2.2.4 Subtle Issues for Collecting Oracle Resource Usages

For most Oracle-based applications, an Oracle user session corresponds to an application instance connected to a database instance. Therefore, v$sesstat is the closest thing we can get to collecting the resource usage of individual applications, especially in a mixed workload environment. This method, however, raises some subtle issues due to the intrinsic limitation of Oracle *v$tables*. We must address these issues carefully while converting the Oracle resource usages into the service demands of application transactions.

First, Oracle consumes some resources via Oracle background processes that perform system-wide functions; Oracle does not charge these usages to individual user sessions. Second, only data block reads are recorded under individual user sessions; disk block writes and redo logs are recorded under DBWR and LGWR separately. Third, Oracle records disk I/O statistics in units of data blocks accessed instead of disk time. In addition, v$sesstat does not detail these statistics on a per tablespace basis; this causes difficulties to determine from which datafiles the data blocks are actually accessed. Last, but not least, some Oracle performance statistics in v$sesstat will change under different workload intensities; this is because of the various caching effects in the SGA such as the database buffer cache, dictionary cache, and shared pool, as well as the batching effects of multi-block writes of DBWR and group commits of LGWR. We will address these issues in the rest of this paper.

2.3 Experiments for Collecting Oracle Resource Usages

From the above discussion, we conclude that the best approach for measuring the resource usage of PeopleSoft transactions is to collect Oracle resource usages at the session level and break them down into per transaction averages. In this subsection, we first describe the important performance statistics in v$sesstat of different Oracle sessions and then assess the accuracy of resource usage data collected through the Oracle *v$tables*.

2.3.1 Important Performance Statistics in v$sesstat

Tables 4-6 list the important performance statistics recorded in v$sesstat of Oracle user sessions, DBWR, and LGWR separately. As mentioned before, only datafile reads are recorded under user sessions as *physical reads*; datafile writes are recorded under DBWR as *physical writes* and redo logs are recorded under LGWR as *redo blocks written*.

Table 4. Important performance statistics in v$sesstat of Oracle user sessions.

Performance Statistics	Description
CPU used by this session	cumulative CPU time used by this session; relevant statistics include *parse time cpu*, *OS User time*, and *OS System time*.
session logic reads	the sum of *db block gets* and *consistent gets*.
physical reads	the cumulative number of data blocks read from disk.
db blocks changes	the cumulative number of data blocks chained to the dirty list. Once the block is in the dirty list, additional changes to that data block are not counted as block changes.
user commits	the number of **commit** calls issued by this session.
redo sync writes	the number of times the redo entry is forced to disk, usually for transaction commit.
redo entries	the number of redo entries generated by this session.
redo size	the cumulative amount of redo entries generated by this session.
*bytes sent via SQL*Net to client*	the cumulative number of bytes in SQL*Net messages was sent to client.
*bytes received via SQL*Net from client*	the cumulative number of bytes in SQL*Net messages was received from client.
*SQL*Net roundtrips to/from client*	the number of times a message was sent and an acknowledgment received.

From its definition in Table 4, *db block changes* is the only user session performance statistic related to *physical writes* of DBWR; however, we find these two statistics are not correlated. This makes it impossible to distinguish *physical writes* caused by different user sessions. Therefore, we can only measure the number of data blocks written per user transaction in a homogeneous workload environment.

Table 5. Important performance statistics in v$sesstat of DBWR.

Performance Statistics	Description
physical writes	the cumulative number of data blocks written to disk.
write requests	the number of times LGWR is signalled to flush dirty buffers to disk.
OS User time used	the total amount of CPU time running in user mode.
OS System time used	the total amount of CPU time running in system mode.

Three performance statistics of user sessions are pertinent to redo logging: *redo sync writes*, *redo entries*, and *redo size*. In section 4, we will describe how to apply these statistics and LGWR *redo wastage* to estimate the *redo blocks written* on behalf of a user session.

Table 6. Important performance statistics in v$sesstat of LGWR.

Performance Statistics	Description
redo wastage	the cumulative total of unused bytes that are written to the log. Redo buffers are flushed periodically even when not completely filled.
redo writes	the number of times LGWR is signalled to write redo buffers to disk.
redo blocks written	the cumulative number of redo blocks written to disk.
redo write time	the cumulative disk time for redo logging.
OS User time used	the total amount of CPU time running in user mode.
OS System time used	the total amount of CPU time running in system mode.

Although Oracle *v$tables* provide a rich set of performance statistics; these statistics are actually used for performance tuning purposes. Before applying these statistics to estimate Oracle resource usages, we must assess the accuracy of these statistics. Therefore, we conduct several measurement experiments with the *Stress* program and compare these statistics with similar performance metrics collected with *iostat*. These experiments are described in three separate subsections as: Oracle CPU usage measurement, Oracle disk usage measurement, and SQL*Net traffic measurement.

2.3.2 Oracle CPU Usage Measurement

Oracle measures resource usages in v$sesstat by polling the system resource usages of Oracle processes via system calls such as *getrusage* and *times* [13]. This approach makes Oracle resource usage statistics fairly accurate at the OS process level; however, it does not consider system processes performing OS functions on behalf of Oracle. Oracle CPU usages in v$sesstat thus only account for portions of total CPU time actually consumed. To further study this issue, we conduct several CPU usage measurements on the testbed Oracle server and compare the Oracle CPU usage in v$sesstat with the CPU time estimated from *iostat*.

These measurement experiments are driven by the *Stress* program [4] with each *Stress* instance generating 10,000 *Stress* requests. The first set of measurements uses a single *Stress* instance and repeats for each request type. The second set of measurements uses two concurrent *Stress* instances for each request type. The third set of measurements uses three concurrent *Stress* instances with a different request type for each instance. The CPU usages of all Oracle sessions are summarized in Table 7.

Table 7. Comparison of CPU usages measured with v$sesstat and estimated from *iostat*.

#clients	**1**			**2**			**3**
request type	***insert***	***update***	***delete***	***insert***	***update***	***delete***	***mixed***
Stress	73.17	129.59	126.95	157.36	278.12	271.63	367.84
PMON	0.10	0.14	0.17	0.17	0.16	0.22	0.21
DBWR	12.30	8.76	13.90	23.09	16.89	29.08	32.43
LGWR	21.39	20.12	21.66	37.69	47.16	41.81	56.28
SMON	0.03	0.06	0.06	0.06	0.07	0.11	0.08
RECO	0.00	0.01	0.00	0.01	0.00	0.01	0.00
Oracle	106.99	158.68	162.74	218.38	342.40	342.86	456.84
iostat	136.32	223.10	222.57	273.96	470.05	475.06	619.11
OS overhead	29.33	64.42	64.83	55.58	127.65	132.24	162.27
%	78.48%	71.13%	73.12%	79.71%	72.84%	72.17%	73.79%

These measurements indicate that Oracle v$sesstat CPU usage measurements only account for 71-80% of actual CPU usage estimated from *iostat*. Since *iostat* generates negligible monitoring overhead, it is reasonable to consider the difference as the OS overhead. It is also apparent from Table 7 that different Oracle workloads cause different amounts of OS overhead. The above results coincide with other research work which also indicates CPU usage in v$sesstat only accounts for 75% of actual CPU usage on average [9,11].

2.3.3 Oracle Disk Usage Measurement

Oracle records disk access statistics in v$sesstat as *physical reads*, *physical writes*, and *redo blocks written*. Converting these statistics into individual user session disk usages raises some important issues. First, Oracle records the numbers of data/redo blocks accessed instead of disk time. Second, v$sesstat does not detail from which tablespaces these data blocks are accessed. Finally, only *physical reads* are recorded on a per user session basis; *physical writes* and *redo blocks written* are system-wide statistics.

Converting Oracle Disk Access To Disk Time

Converting Oracle disk access to disk time require us to identify the disk(s) accessed and the unit disk time per access. As a consequence, we can address the first two issues through the following four steps:

- identify the tablespaces accessed by each type of transaction,
- construct the mapping between Oracle tablespaces and physical disks,
- calculate the average disk time per block access, and
- estimate individual disk times by multiplying the disk access statistics by the average disk time per block accessed.

These steps can be facilitated by use of long-term disk I/O statistics of datafiles and redo log files in v$filestat and v$sysstat [23,24]. Table 8 lists these disk I/O statistics from the testbed Oracle server. The row marked *summary* lists the average disk statistics over all datafiles.

Table 8. Long-term disk I/O statistics for datafiles and the redo log file from v$filestat and v$sysstat.

Oracle files	PHYRDS	PHYWRTS	PHYRBLKRD	PHYBLKWRT	READTIM	WRITETIM
system01.dbf	51090	35339	84885	35339	3966	935057
rbs01.dbf	25	48886	25	48886	4	915107
summary	51115	84225	84910	84225	3970	1850164
redo log file		347290		712705		987909

Since the testbed Oracle server has only one disk, we can skip the first two steps. We then apply the disk statistics in Table 8 to estimate the average disk times per block access as shown in Table 9. We can use the summary average instead of individual averages of each datafile for the last steps because these statistics refer to the same disk.

Table 9. Average disk time per request and per block.

Oracle files	ms/phyrds	ms/ phywrts	ms/phyblkrd	ms/ phyblkwrt
system01.dbf	0.078	26.460	0.047	26.460
rbs01.dbf	0.160	18.719	0.160	18.719
summary	0.078	21.967	0.047	21.967
redo log files		28.450		13.860

To complete the last step, we ran two experiments with different workloads and collected the disk access statistics in v$sesstat. Workload I comprises 10,000 *Stress update* requests; its disk access statistics are listed in the third column of Table 10. Workload II comprises 10,000 *Stress* requests with mixed request types (10% query, 60% update, 25% insert, and 5% delete); its disk access statistics are listed in the fourth column. The average disk times per block access from Table 9 are repeated in the second column of Table 10. We can thus compute the data block read time from *physical reads*, the data block write time from *physical writes*, and the redo log time from *redo blocks written*. At bottom of Table 10, we find that the estimated disk times are very close to the *iostat disk times* for both measurements.

Table 10. Disk time estimation by combining v$sesstat and v$filestat.

disk usage statistics	disk time	10,000 update requests		10,000 mixed requests	
	ms/unit	v$sesstat	disk time (sec)	v$sesstat	disk time (sec)
physical reads	0.047	345	**0.016**	10	**0.001**
physical writes	21.976	1830	**40.200**	2997	**65.835**
write requests		267		260	
redo writes	28.450	10012	284.841	8998	255.993
redo blocks written	13.860	20006	**277.283**	18126	**251.725**
redo write time		27390	273.900	24860	248.600
estimated disk time			**317.499**		**317.561**
iostat disk time			326.860		327.840

Separating Oracle Disk Writes for Individual User Sessions

Collecting the per-session Oracle disk usages raises the third issue because Oracle does not record *physical writes* and *redo blocks written* on a per user-session basis. To the best of our best knowledge, there is no satisfactory solution for this issue because not enough information is provided in v$sesstat of Oracle user sessions. For *physical writes*, the *db block changes* of a user session does not imply the number of data blocks written on behalf of the user session. For *redo block written*, the *redo size* of a user session does not imply the *redo blocks written* on behalf of the user session because of *redo wastage*; however, *redo wastage* is only recorded for LGWR and cannot be broken down into individual user sessions. Although these two problems reveal the imperfection of the measurement methodology based on v$sesstat, we can alleviate these problems by conducting performance measurements in a homogeneous workload environment.

2.3.4 SQL*Net Traffic Measurement

For SQL*Net traffic measurement, the three performance statistics in v$sesstat (*bytes sent via SQL*Net to client*, *bytes received via SQL*Net from client*, and *SQL*Net roundtrips*) are fairly accurate compared with the SQL*Net packet traces collected with *tcpdump*. In addition, these statistics can be converted directly into model parameters, such as *#packets per request* and *average packet length*, applicable to most analytic performance models.

2.4 Performance Measures vs. Monitoring Tools

Table 11 summarizes how performance measures are associated with monitoring tools for collecting Oracle resource usages. Three OS level tools - *iostat*, *filemon*, and *tcpdump* - are also included because they can be applied to calibrate the performance statistics in Oracle *v$tables*. We further divide the v$sesstat into three columns as *USER*, *DBWR*, and *LGWR*, because some performance statistics only apply to specific Oracle sessions in v$sesstat.

There are several pitfalls while interpreting performance statistics in Oracle *v$tables* because some of these statistics are either confusing or misleading. First, Oracle disk timing statistics (such as *READTIM*, *WRITETIM*, and *redo log time*) are measured by Oracle processes at the OS user level; therefore, these statistics are actually the elapsed times of I/O system calls rather than the disk service times.

Second, some Oracle performance statistics have misleading names which easily cause confusion. For example, *physical reads*/*physical writes* in v$sesstat record the same disk access statistics as *PHYBLKRD*/*PHYBLKWRT* in v$filestat. In addition, *write requests* and *redo writes* in v$sesstat actually record the number of times DBWR and LGWR are signalled to work instead of recording actual disk write requests. The actual numbers of disk requests to Oracle datafiles are recorded as *PHYRDS* and *PHYWRTS* in v$filestat; however, no equivalent performance statistics are recorded in v$sesstat.

Third, some similar performance statistics are actually based on different units. For example, the redo blocks (512 bytes) are a different size from the data blocks (4 KB); the timing statistics in v$sesstat is based on units of 10 ms (Oracle clock ticks) but the timing statistics in v$filestat are based on units of milliseconds.

Table 11. Performance measures vs. monitoring tools for measuring Oracle resource usages.

	v$sesstat							
	USER	***DBWR***	***LGWR***	***Stress***	**v$filestat**	***iostat***	***filemon***	***tcpdump***
# transactions				X				
user commits	X							
response time				X				
CPU utilization						X		
CPU time	X	X	X			X		
disk utilization						X	X	
disk read time							X	
READTIM					X			
disk write time							X	
WRITETIM					X			
redo log time			X					
disk transfers						X		
logic reads	X							
PHYRDS					X		X	
physical reads (PHYBLKRD)	X				X		X	
write requests		X						
PHYWRTS							X	
physical writes (PHYBLKWRT)		X			X		X	
redo writes			X					
redo blocks written			X					
*bytes sent via SQL*Net to client*	X							X
*bytes received via SQL*Net from client*	X							X
*SQL*Net roundtrips*	X							X

3. Measurement Methodology for Citrix WinFrame Extension

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 user desktop machines [6]. The three-tiered deployment is accomplished with Citrix's universal thin-client software, the multi-user NT server, and the Independent Console Architecture (ICA). Performance measurement of this WinFrame extension aims at quantifying the resource consumption of the ICA presentation service while the PeopleSoft client program (henceforth called PSTOOLS) is remotely executed. The resource consumption as characterized by the ICA architecture comprises three parts: the resource usage on a user desktop machine (WinStation), the resource usage on a WinFrame application server, and the ICA traffic between the WinStation and the WinFrame server [4].

3.1 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.exe*) [8]. 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 presentations. The WinFrame server thus uses a separate instance of *csrss.exe* to manage the Windows display for each WinStation connection [5].

ICA divides its functions into individual protocol drivers layered as a protocol stack (shown in Figure 12). 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.

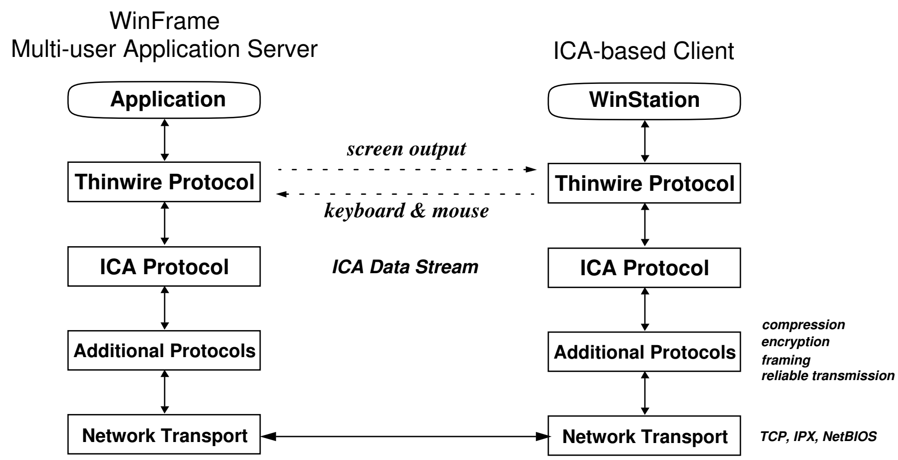


Figure 12. ICA Protocol Stack.

The Thinwire protocol uses *WdFrames* (window frames) to carry presentation information in both directions of a WinStation connection. At the WinStation, WdFrames carry key-strokes and mouse events; 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 WinStation's memory and large bitmaps are persistently cached on the WinStation'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 frame to reduce the network traffic.

3.2 Performance Measurement Tools

Performance measurement of the WinFrame extension collects resource usages consumed by ICA Windows presentation and PSTOOLS. Since the resource usages are collected at WinStations and WinFrame servers, we apply different monitoring tools available on these platforms as described in the following subsections.

3.2.1 Performance Measurement Tools on Windows 95

Most client desktop machines are personal computers running Windows 95 in the production environment. The standard monitoring tool on Windows 95 is the *system monitor*, which allows on-line monitoring of a limited set of performance statistics about file system, kernel, and memory manager. However, the *system monitor* cannot record these statistics into files; this makes it useless for resource usage measurement. Some freeware tools are available for performance monitoring purposes on Windows 95. *StatBar Console* [7] and *wintop* [20] are two of these that fit our measurement requirement.

StatBar Console reports free System/User/GDI resources and CPU utilization between two sampling points. Users can specify the delay between consecutive reports and redirect its outputs to a log file. Since each report has a timestamp, we can estimate the CPU usage in a given time period. However, it is not possible to further break down the CPU usage at the process level. *Wintop* is a Windows 95 approximation of the Unix *top* program. For each active process, *wintop* reports the percentage of CPU time used by the process in the last two seconds and the accumulated CPU time since it was started. Therefore, the CPU usage of a process can be collected at a specific measurement point.[1]

3.2.2 Performance Measurement Tools on WinFrame Servers

Unlike Windows 95, Windows NT is well-metered for performance measurement [8]. It provides a rich set of performance counters for important system objects: ***System***, ***Process***, ***Memory***, ***Physical Disk***, ***Network Interface***, etc. Windows NT also provides a standard tool, called *perfmon*, for on-line monitoring of individual performance counters or for recording all performance counters within a performance object [10]. Similar to other monitoring tools, *perfmon* generates overhead for monitoring or logging, especially for monitoring disk activities. Since the overhead grows with the number of performance counters monitored, only necessary counters should be included in any measurement plan.

3.2.3 Performance Measurement Tools for ICA Traffic

ICA traffic can be collected as packet traces with packet capturing tools such as a network sniffer or *tcpdump*. After the packet stream of a WinStation connection is identified in the packet traces, the *tcptrace* tool provides ICA traffic statistics of individual WinStation connections [4]. Similar performance metrics can also be collected at WinStations or at WinFrame servers. Each WinStation records ICA packets generated by itself and displays the traffic statistics in the "Client Connection Status" window as shown in Figure 13. Therefore, we can reset these statistics at the beginning of a measurement and collect the ICA traffic statistics at the end of a stress testing session.

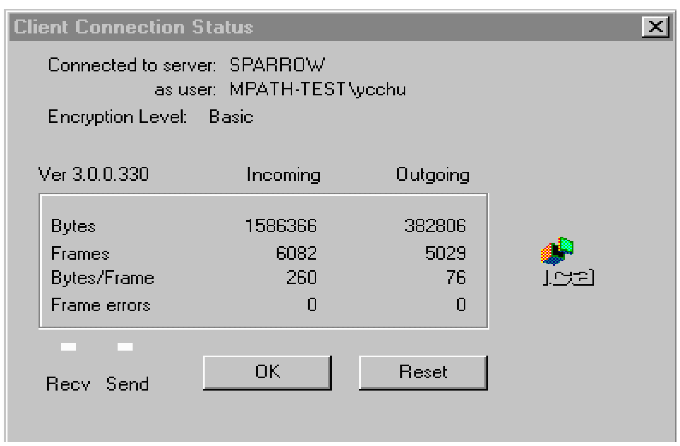


Figure 13. Client Connection Status of a WinStation.

Citrix WinFrame adds a performance object called ***WinStation*** which contains a set of performance counters associated with individual WinStation connections. The performance counters relevant to ICA traffic measurement include: *Total ICA Bytes/sec* in the ***System*** object for all WinStation connections, *Input/Output/Total Bytes/WdBytes* and *Input/Output/Total Frames/WdFrames* in the ***WinStation*** object for individual WinStation connections. These counters report ICA traffic generated between two consecutive sampling points.

1. However, *wintop* is not always accurate for CPU usage measurement because some CPU time of a destroyed thread may be lost; this results in CPU time under reporting if a process creates and destroys threads frequently. This is not an issue for us because *wfica32.exe* does not destroy threads.

3.2.4 Correlating Performance Data Across Platforms

Since performance measurement of the WinFrame extension spreads across several platforms, it creates difficulties to correlate performance data collected at different locations. In addition, we find a similar problem for correlating performance counters in different performance objects.[1] The WinStation administration tool, *winadmin*, provides important information for building cross references among performance counters in different performance objects as well as performance data gathered across different locations. As a consequence, performance measurement must also collect the WinStation connection information with *winadmin* in order to correlate the performance data.

3.3 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 this *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 frame [5]. Other configuration settings affecting the ICA overhead include the size and the color depth of the window display at the WinStation.

3.3.1 Performance Measurement of ICA Overhead

We measure ICA overhead with *perfmon* and *task manager* in our testbed environment. Since the WinFrame server uses a separate *csrss* process to manage the 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*[2]. Table 14 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.

Table 14. Performance measurement of ICA overhead on behalf of *Stress* program.

# of DB calls	C	CPU time (sec.)			%CPU			ICA traffic	ICA Overhead	
		WinServer		WinStation	WinServer		WinStation		WinServer	WinStation
	(sec)	*stress*	*csrss*	*wengfN*	*stress*	*csrss*	*wengfN*	(bytes/sec)	(us/byte)	
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

1. This problem exists because performance objects use different indices to refer multiple instances. For example, performance counters in the ***WinStation*** object are indexed by connection ID (in the form of *tcp*#), but performance counters in the ***Process*** object are indexed by process name. Correlating a process with a WinStation connection requires *winadmin* information which maps the connection ID of a WinStation to the process name of a process.
2. Citrix has changed the name of its client program from *wfengfN32.exe* to *wfica32.exe* in recent releases; *wfcrun32.exe* consumed no CPU time in our experiments, so we do not consider it further here.

The measurement focuses on the ICA traffic rate and the CPU consumption 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.

Several interesting phenomena are revealed by Table 14. 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.

3.3.2 Comparison with Two-Tier

To further investigate the cost of Citrix remote computing, we measured the completion time and CPU usages for running *Stress* in both two-tiered and three-tiered configurations. The measurement data are shown in Figure 15 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

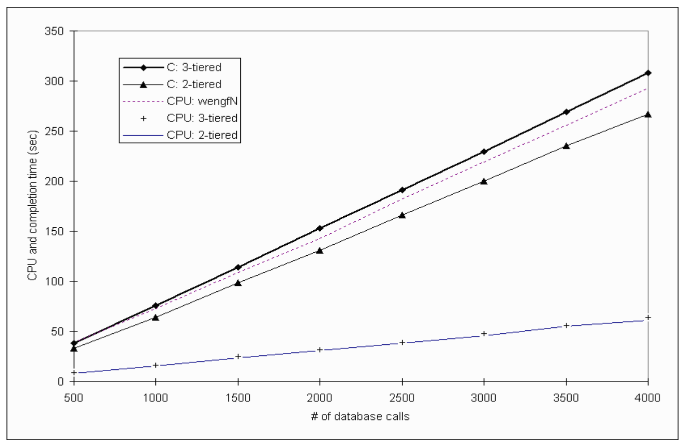


Figure 15. Measurement of application impact of Citrix WinFrame.

Comparing the measurement data in both configurations, 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.

3.4 Stress Testing of WinFrame Server in a Production Environment

Stress testing was conducted in a production environment to study the performance of a WinFrame server (a COMPAQ ProLiant 5000 running Citrix WinFrame 1.7 with 4 166MHz Pentium processors and 1 GB memory). In this testing, 35 test users connected to the WinFrame server from their local desktop machines and submitted PeopleSoft Space Management transactions. Meanwhile, *perfmon* was set up on the WinFrame server to log performance objects: ***Memory***, ***Physical Disk***, ***Process***, ***Processor***, ***System***, and ***WinStation***, every 30 seconds. The log file was played back in the testbed WinFrame server and converted into spreadsheets for statistical analysis.

3.4.1 Major Resource Usage of the WinFrame Server

The major resource usages of the WinFrame server can be observed from the ***System***, ***Memory***, and ***Physical Disk*** performance objects. Figure 16 displays the recorded values of four performance counters for the entire stress test. The *Active WinStations* counter, a dashed line peaking at 35, is also included to indicate three different phases of the stress test: connection, data entry, and disconnection. These counters correspond to the following performance metrics:

% Total Processor Time	the average processor utilization of all processors,
Total ICA Bytes/sec	ICA traffic rate for all WinStations connections in bytes/sec,
Active WinStations	the number of active WinStation connections,
% Disk Time	the disk utilization of the physical disk 0, and
Available Bytes	the size of free virtual memory in bytes.

These counters capture the essential performance characteristics of the WinFrame server in the stress test.

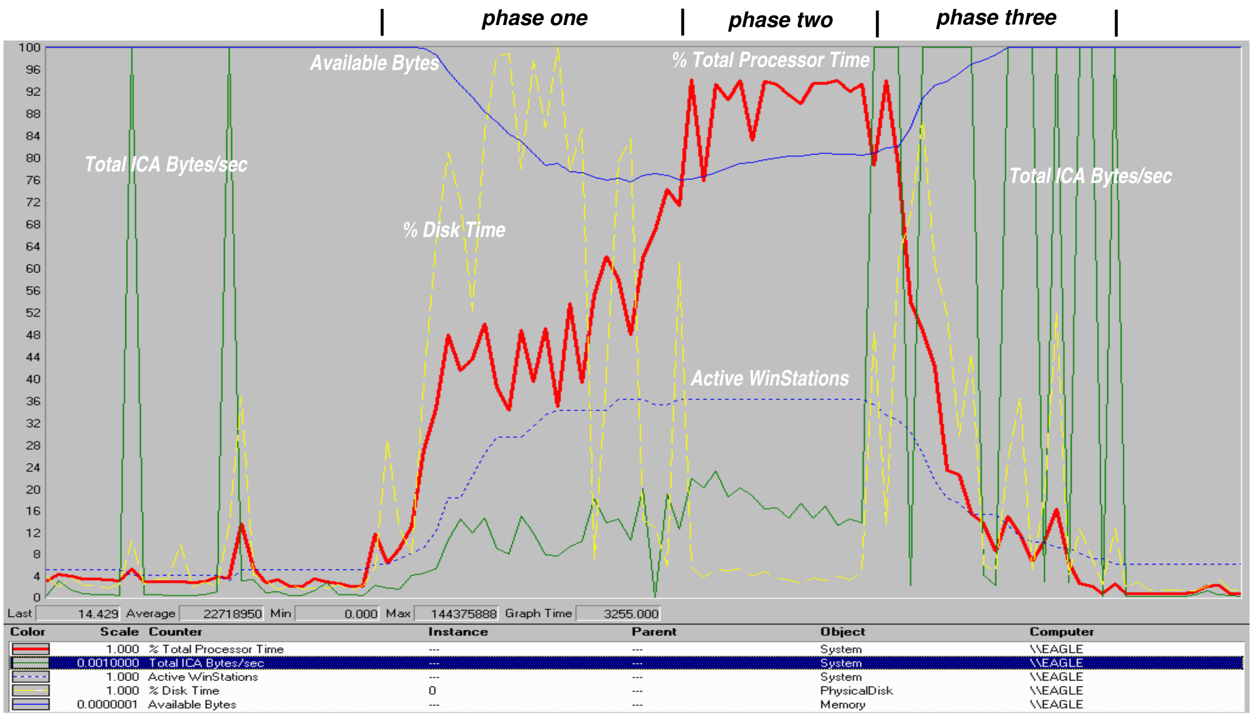


Figure 16. Major resource usages of the WinFrame server in 35-user stress testing.

First, considerable free memory - no less than 750 MB - is left unused for the entire stress testing. Considering the overall processor utilization (i.e. *% Total Processor Time*) in the data entry phase, trading some memory for faster processors can yield better performance. Second, three potential bottleneck devices can be identified in each testing phase: the physical disks in the connection phase, the processors in the data entry phase, and the network interface in the disconnection phase. Table 17 summarizes the average utilization of all processors and individual disks in each testing phase.

Table 17. Average processor and disk utilizations of the WinFrame server in different stress test phases.

Performance Counter Name		Connection	Data Entry	Disconnection
System	*% Total Processor Time*	44.62%	91.09%	34.88%
Physical Disk 0	*% Disk Time*	52.62%	4.09%	26.47%
	% Disk Read Time	1.96%	0.05%	0.62%
	% Disk Write Time	51.02%	4.05%	25.85%
Physical Disk 1	*% Disk Time*	9.88%	2.02%	1.22%
	% Disk Read Time	7.31%	0.33%	0.29%
	% Disk Write Time	2.57%	1.69%	0.93%
Physical Disk 2	*% Disk Time*	2.44%	1.49%	0.87%
	% Disk Read Time	0.02%	0.00%	0.00%
	% Disk Write Time	2.42%	1.49%	0.87%

The WinFrame server experiences moderate disk reads on disk 1 and high disk writes on disk 0 in the connection phase. Disk 1 stores the PeopleSoft client software; therefore, the disk read activities are for loading PSTOOLS. Disk 2 stores the PeopleSoft cache files of individual PeopleSoft users. The cache files contains the online elements such as panels, fields, or menus retrieved from the master copies on the database server. The disk write activities are for writing these objects into the cache files whenever PSTOOLS is started.[1] The ICA traffic is moderate in most cases except when a WinStation is disconnecting from the WinFrame server. This can be observed as several bursty peaks in *Total ICA Bytes/sec* when the value of *Active WinStations* decreases. According to Citrix, this bursty ICA traffic is caused by acknowledgment packets for closing all opened files of a WinStation connection.

3.4.2 Important Performance Metrics in the *Process* Object

The ***Process*** object records all active processes on the WinFrame server during stress testing. While remotely executing the PeopleSoft client program, each WinStation user launches seven processes: *pstools*, *csrss*, *progman*, *NTVDM*, *wfshell*, *winlogon*, and *NDDEAGNT*.

Table 18. CPU usages of WinFrame system processes and PeopleSoft client processes.

Process Name	*% Processor Time*	*% Total Processor Time*
pstools (36)	184.249	47.0%
csrss (36)	125.927	32.1%
idle	35.793	9.1%
PERFMON (2)	25.950	6.6%
system	6.382	1.6%
winadmin (2)	6.275	1.6%
icasrv	4.541	1.2%
lsass	0.949	0.2%
services	0.847	0.2%
pspmd (2)	0.456	0.1%
WINFILE	0.108	0.0%
progman (36)	0.045	0.0%
USRMGR	0.039	0.0%
NTVDM (36)	0.034	0.0%
wfshell (35)	0.028	0.0%
ibrowser	0.026	0.0%
winlogon (36)	0.025	0.0%
NDDEAGNT (36)	0.018	0.0%

1. The WinFrame/PeopleSoft Application Note suggests the PeopleSoft caches of individual users be stored on a network file server. This approach, however, causes extra network traffic each time the PeopleSoft client program references the PeopleSoft caches. The U-M approach copies these caches files from the network file server to the local disk of the WinFrame server at the startup of PeopleSoft client program. Therefore, loading the PeopleSoft caches actually causes intensive disk writes in the connection phase.

As shown in Table 18, however, most processor resources are actually consumed by two processes: *pstools* and *csrss*. *Pstools* is the PeopleSoft client program; it uses 47% of overall processor resources. *Csrss* handles remote Windows presentation for a WinStation connection; it uses 32% of overall processor resources. *Perfmon* causes the monitoring overhead at the amounts of 6.6% of overall processor resources for recording 6 performance objects every 30 seconds.

Figure 19 plots the *% Processor Time* of *csrss.exe* against the *% Process Time* of *pstools.exe* for all WinStation connections in the data entry phase. What is important here is that the ICA server overhead, in the form of *csrss* CPU usage, grows linearly with the CPU usage of *pstools* instead of highly randomly.

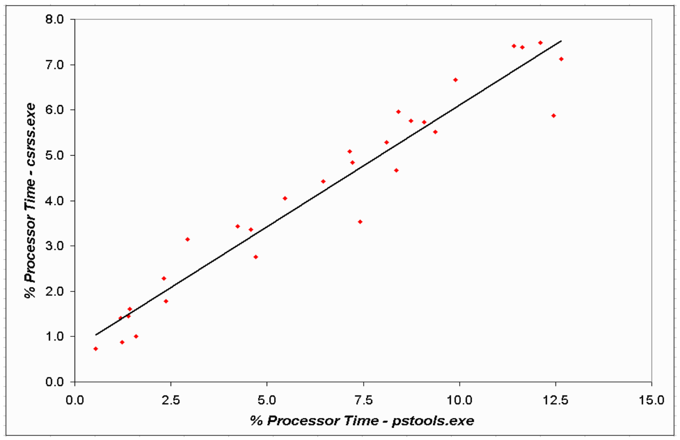


Figure 19. *% Processor Time* - *csrss.exe* vs. *pstools.exe* in the data entry phase of the stress testing.

3.4.3 Important Performance Metrics in the *WinStation* Object

The ***WinStation*** object contains performance counters relevant to WinStation connections. These counters can be divided into four major categories:

- resource usages: *% Processor Time*, *Page File Bytes*, *Virtual Bytes*, *Working Set*;
- ICA object references: reads, hits, and hit ratio of ICA object references;
- ICA traffic: input/output/total ICA bytes, WdBytes, frames, WdFrames; and
- ICA compression statistics.

Resource usage counters in the ***WinStation*** object cover processor and memory, but not disk. *% Processor Time* reports the average processor utilization consumed by the WinStation connection between two sampling points. *Page File Bytes* and *Working Set* can be used to estimate the size of system page files and the amount of physical memory required for WinFrame servers.

Table 20. Performance statistics for cumulative ICA object references.

	reads	*hits*	*hit ratio*
bitmap	82,977	81,687	98.4%
brush	36,818	36,782	99.9%
glyph	1,116,675	1,108,950	99.3%
save screen bitmap	3,384	2,228	65.8%
summary	1,239,854	1,229,647	99.2%

The ***WinStation*** object also records ICA object references for *bitmap*, *brush*, *glyph*, and *save screen bitmap*. Each ICA object has three performance counters (*reads*, *hits*, and *hit ratio*) recording cumulative statistics. Table 20 lists the content of these counters averaged from all WinStation connections. This table supports the low bandwidth requirement of ICA because 99% of ICA object references are actually cached at WinStations.

There are 12 performance counters in the ***WinStation*** object recording ICA traffic generated between two sampling points. Figure 21 plots the average numbers of ICA frames/WdFrames generated in 30 seconds against the *% Processor Time* of *csrss.exe* for individual WinStations in the data entry phase. There are two vertical scales in this figure: the scale on the left shows *Input Frames*, *Output Frames*, and *Input WdFrames*; the scale on the right shows *Output WdFrames*. Apparent from Figure 21 is that the ICA protocol puts together multiple output WdFrames into a single output frame to reduce communication overhead. In addition, we can observe a linear pattern between ICA frames generated and the *% Processor Time* of *csrss* in Figure 21. If we consider both linear patterns in Figures 19 and 21, we can conclude that the ICA overheads, in the forms of *csrss* CPU usage and ICA traffic, have regular patterns.

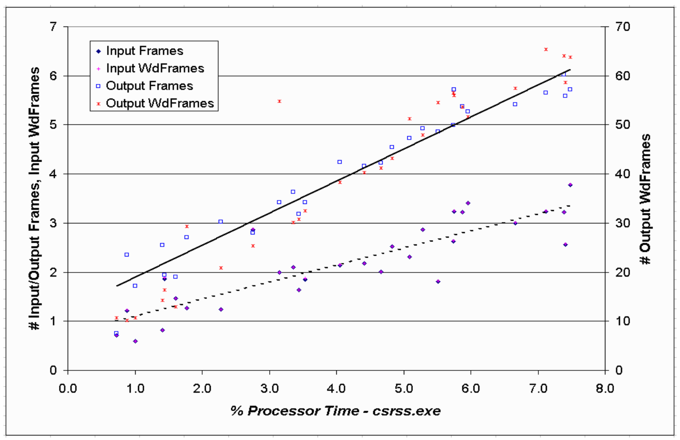


Figure 21. ICA traffic vs. *% Processor Time* of *csrss.exe* in the production environment.

3.5 Other Performance Measurement Considerations

From the discussion above, we can extend the measurement methodology to collect the resource usages consumed by the WinFrame extension on a per-WinStation basis. If we further divide these by the number of PeopleSoft transactions accomplished, we convert the resource usages into a per-transaction basis. From a performance modeling perspective, these per-transaction resource usages correspond to the service demand parameters for each type of PeopleSoft transactions as: CPU ser*vice demand at a WinStation* (the average CPU time consumed by *wfica32.exe* at a WinStation for ICA presentation service), *CPU service demand at a WinFrame server* (the average CPU time consumed by *csrss.exe* at a WinFrame server for the ICA presentation service), *#ICA frames* (the average number of ICA packets generated for the ICA presentation service), and *bytes/ICA frame* (the average packet length of ICA frames in bytes).

It should be noted that these model parameters do not include disk usages. We do not consider the disk usage of WinStations because most ICA object references are actually cached in a WinStation's memory. For WinFrame servers, the stress testing shows that the actual disk utilizations in the data entry phase (which also include disk writes of *perfmon*) are quite low - less than 4%. In addition, we can only monitor system-wide disk usages on WinFrame servers. Therefore, we believe it is reasonable to neglect the disk usage on WinFrame servers unless it can be measured on a per-process or per-WinStation basis.

4. Service Demand Estimation and Analytic Modeling

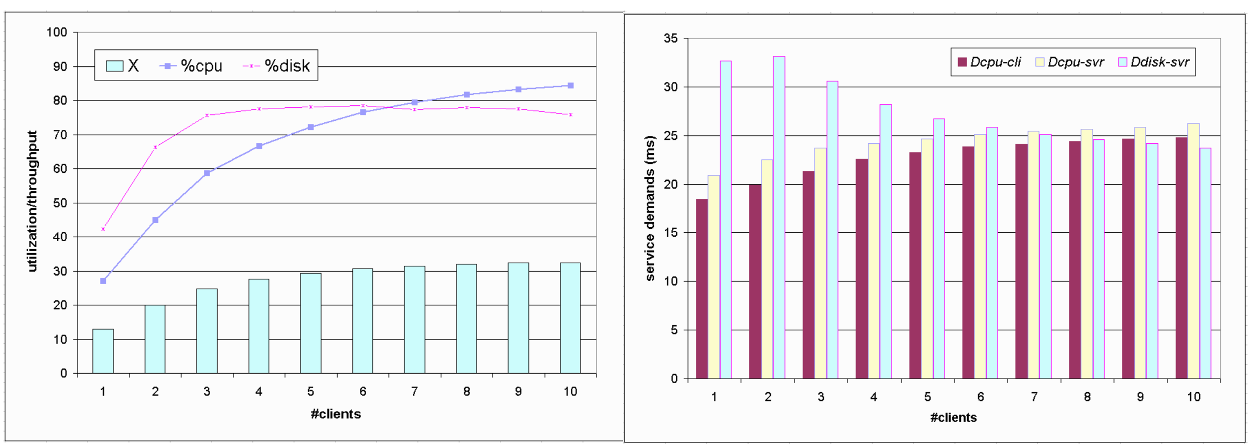
In the previous two sections, we presented the measurement methodology for measuring the system resource usages consumed by PeopleSoft applications in both two- and three-tiered environments. Since the resource usages are collected at each tier on a per-session basis, we can easily convert them into a per-transaction basis. For analytic modeling, the average resource usages per transaction correspond to the service demands of a PeopleSoft transaction. Our previous work shows that the measured Oracle service demands, especially for Oracle disk times, actually vary with the number of concurrent user sessions on the Oracle server [4]. In this section, we use the *Stress* program to investigate why Oracle service demands vary with concurrent user sessions.

4.1 Experiment Setup

We set up the experiments by running multiple instances of the *Stress* program concurrently on the testbed WinFrame server. Each *Stress* instance is configured to generate 10,000 *Stress update* requests against the testbed Oracle server; we start with a single *Stress* instance and add one instance at a time until we have 10 concurrent instances. For each experiment, we collect the Oracle resource usages by taking snapshots of v$sesstat and v$filestat before and after each stress testing, we also use *iostat* to collect the CPU and disk utilizations of the Oracle server, and we use *pview* to collect the CPU time consumed by each *Stress* instance on the WinFrame server.

4.2 Performance Metrics Derived from *Iostat*

Figure 22a plots the CPU utilization (*%cpu*), disk utilization (*%disk*), and throughput (*X*) of the Oracle server against the number of concurrent *Stress* instances (henceforth abbreviated as *#clients*). There are two interesting phenomena in this figure: first, the Oracle disk utilization saturates around 77-78%, instead of 100%, with 4 or more clients; second, the Oracle bottleneck device switches from the disk to the CPU between 6 and 7 clients.



(a) utilization and throughput. (b) service demands.

Figure 22. Performance measures of *Stress update* derived from *iostat* outputs.

These two phenomena are more explainable when we look into the measured service demands of *Stress update* in Figure 22b. It turns out that both the CPU service demand on the WinFrame server (*Dcpu-cli*) and the CPU service demand on the Oracle server (*Dcpu-svr*) increase with *#clients*; however, the Oracle disk service demand (*Ddisk-svr*) actually decreases with *#clients*. In addition, the *Dcpu-svr* begins to exceed the *Ddisk-svr* with 7 clients. To study these phenomena further, we decompose the Oracle service times with performance statistics in *v$tables*.

4.3 Oracle CPU Time Decomposition for *Stress Update*

The Oracle v$sesstat provides a better source than *iostat* to examine the Oracle CPU usage because it records the Oracle CPU times for user sessions and background processes separately. Table 23 shows the decomposed Oracle CPU times per *Stress update* request. The first five columns in Table 23 correspond to Oracle background processes. The sixth column shows the average CPU time per request consumed by *Stress* Oracle user sessions. The column marked '*Oracle*' adds up all Oracle process CPU times; the column marked '*iostat*' lists the Oracle CPU service demands from Figure 22b. From these two columns, we find out that all Oracle processes together only account for 73% of the *iostat* CPU time. The difference actually represents the per-request OS overhead because *iostat* generates negligible monitoring overheads. In addition, we find the OS overhead is quite regular because its variance against #*clients* is very small.

Table 23. Decomposition of Oracle CPU time for servicing a *Stress update* request against *#clients*.

#clients	PMON	DBWR	LGWR	SMON	RECO	*Stress*	*Oracle*	*OS overhead*	*iostat*
1	0.015	0.537	1.930	0.004	0.000	12.374	14.860	6.074	20.934
2	0.008	0.645	2.218	0.004	0.000	13.470	16.344	6.164	22.508
3	0.008	0.773	1.977	0.003	0.000	14.228	16.989	6.693	23.682
4	0.007	0.883	1.808	0.003	0.000	14.792	17.493	6.689	24.182
5	0.006	0.910	1.797	0.003	0.000	15.183	17.900	6.761	24.661
6	0.006	0.994	1.682	0.002	0.000	15.607	18.241	6.901	25.142
7	0.007	0.982	1.698	0.002	0.000	15.890	18.579	6.898	25.477
8	0.006	0.995	1.670	0.002	0.000	16.149	18.823	6.856	25.679
9	0.005	1.033	1.567	0.002	0.000	16.368	18.984	6.860	25.844
10	0.006	1.050	1.554	0.002	0.000	16.744	19.356	6.919	26.275
mean	0.007	0.875	1.791	0.003	0.000	15.080	17.757	6.681	24.438
stdev	0.003	0.172	0.204	0.001	0.000	1.384	1.382	0.309	1.672
%Oracle	0.04%	4.93%	10.09%	0.02%	0.00%	84.93%	100%		
%iostat	0.03%	3.58%	7.33%	0.01%	0.00%	61.71%	(72.66%)	27.34%	100%

After examining each Oracle process, we can pinpoint that the Oracle server process (marked as *Stress*) actually causes the Oracle CPU service demand to increase with #*clients*. Although we also observe that the DBWR CPU time increases with #clients and the LGWR CPU time decreases with #*clients*, these two processes together do not cause significant discrepancy in comparison to the *Stress* user session. The bottom two rows of table 15 show how individual Oracle processes contribute to the total Oracle CPU time. It is not surprising that the Oracle server process accounts for 85% of the share because it handles the parse, execute, and fetch of SQL statements.

4.4 Oracle Disk Time Decomposition for *Stress Update*

Figure 24 shows various measured Oracle disk times against #*clients*. The two curved lines on top represent the measured Oracle disk times (*iostat disk time* and *filestat disk time*) derived from *iostat* and *v$tables* separately. We calculate the latter values by taking the differences in disk performance statistics from v$filestat and v$sesstat of LGWR before and after each stress testing. It should be noted that two curves do not match each other closely. This discrepancy is because *iostat* records disk utilization at the OS *system* level, but Oracle records the elapsed times of disk requests at the OS *user* level. The Oracle disk statistics, therefore, include the disk waiting times whenever there are concurrent disk requests from different Oracle processes.

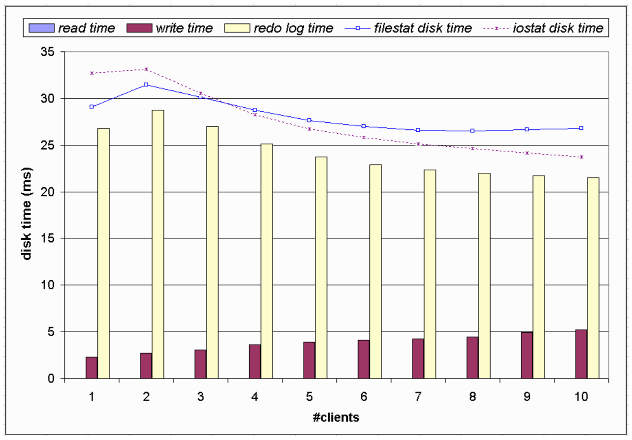


Figure 24. Oracle disk time decomposition for *Stress update*.

The barchart in Figure 24 show the decomposition of Oracle disk time as:

- *read time* — the cumulative elapsed time for Oracle server processes to read data blocks from datafiles,
- *write time* — the cumulative elapsed time for DBWR to write dirty buffers into datafiles, and
- *redo log time* — the cumulative elapsed time for LGWR to write redo entries into the online redo file.

The complete sets of Oracle disk I/O statistics from v$filestat and v$sesstat are listed in Table 25. The Oracle disk time decomposition reveals several interesting phenomena: First, for *Stress update*, Oracle disk requests are dominated by disk writes (4% reads, 15% writes, and 81% logs), and the *redo log time* accounts for 86% of the Oracle disk time. Second, the *redo log time* decreases dramatically with three or more concurrent clients, which also causes the *iostat disk time* to decrease with #*clients*.[1]

At this point, we have identified the actual cause for the discrepancy in the Oracle disk service demand. To explain this phenomenon, we have to introduce the *group commit* mechanism of LGWR. Oracle implements group commit to reduce the overhead of redo logging. According to Oracle, LGWR batches multiple redo entries in a single redo write whenever LGWR finds multiple pending commit requests. The first consequence of group commit is that it reduces the average cost of redo logging because the fixed overhead of a disk write is shared by several requests.

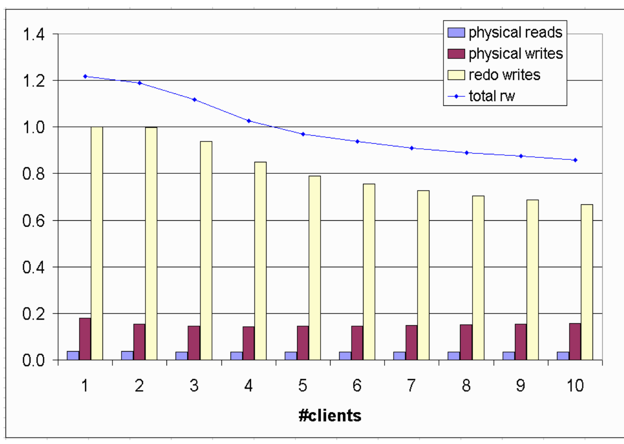
The second consequence of group commit is that it reduces the amount of "*redo wastage*." LGWR writes redo entries into the online redo file in chunks of redo blocks. Since a redo entry may not fill the entire redo block, the redo wastage will be significant if LGWR writes a single redo entry at a time. With group commit, several redo entries are written into redo file and fill contiguous redo blocks. As a consequence, the average amount of redo wastage per request is reduced, as shown in the third set of disk I/O statistics in Table 25.

1. Although we also observe that the *write time* increases with #*clients* in Figure 24, we believe this is somewhat misleading. Since Oracle disk I/O is dominated by redo logging, DBWR's disk requests are more likely waiting for LGWR's disk requests than vice versa. The *iostat disk time* in Figure 24 supports this argument because it actually decreases in accordance with the *redo log time* against #*clients*; the increasing *write time* apparently does not affect *iostat disk time* at all.

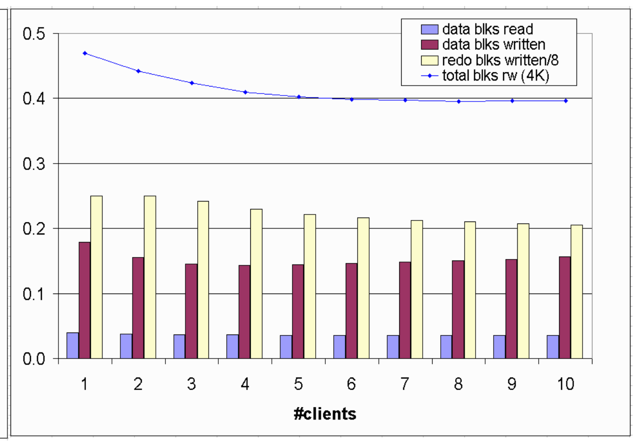
Table 25. Disk I/O statistics per *Stress update* request derived from v$filestat and v$sesstat of LGWR.

#clients	1	2	3	4	5	6	7	8	9	10	*mean*	*stdev*	%
read time	0.004	0.004	0.009	0.008	0.009	0.010	0.013	0.015	0.016	0.018	0.010	0.005	0.04%
write time	2.271	2.729	3.086	3.621	3.919	4.082	4.235	4.473	4.947	5.228	3.859	0.948	13.76%
redo log time	26.799	28.736	27.010	25.117	23.721	22.877	22.316	22.001	21.692	21.541	24.178	2.580	86.20%
filestat disk time	29.074	31.468	30.105	28.746	27.650	26.969	26.564	26.490	26.665	26.761	28.048	1.733	
iostat disk time	32.673	33.138	30.564	28.220	26.714	25.841	25.104	24.605	24.153	23.705	24.742	3.520	
physical reads	0.038	0.036	0.036	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.001	3.54%
physical writes	0.179	0.155	0.145	0.144	0.144	0.146	0.148	0.150	0.153	0.156	0.152	0.010	15.22%
redo writes	1.001	0.999	0.937	0.848	0.790	0.757	0.727	0.705	0.687	0.667	0.812	0.128	81.24%
total r/w	1.218	1.190	1.118	1.027	0.969	0.937	0.910	0.890	0.874	0.858	0.999	0.133	
data blks read	0.040	0.037	0.037	0.037	0.036	0.036	0.036	0.036	0.036	0.035	0.037	0.001	8.96%
data blks written	0.179	0.155	0.145	0.144	0.144	0.146	0.148	0.150	0.153	0.156	0.152	0.010	36.80%
redo blks written	2.000	1.989	1.931	1.837	1.773	1.735	1.703	1.679	1.660	1.639	1.796	0.138	54.24%
total blks r/w (4K)	0.469	0.443	0.423	0.410	0.402	0.398	0.397	0.395	0.396	0.397	0.413	0.025	
redo wastage	413.360	412.292	379.239	332.378	300.535	281.712	265.726	254.055	244.750	234.430			

We can also observe the consequences of group commit in Figure 26. Figure 26a plots the second set of disk I/O statistics (*physical reads*, *physical writes*, and *redo writes*) in Table 25, showing that the number of redo writes per request decreases with *#clients*. Figure 26b plots the third set of disk I/O statistics (*data blks read*, *data blks written*, and *redo blks written*) in Table 25, showing that the number of redo blocks written per request decreases with *#clients*.[1]



(a) physical reads, physical writes, & redo writes.



(b) data blocks and redo blocks read/written.

Figure 26. Disk I/O statistics of datafiles and the redo file per *Stress update*.

A good Oracle model must capture the characteristics of Oracle server processes, DBWR, and LGWR because together they account for most Oracle resource usages. Since Oracle records the resource usages of Oracle server processes on a per-session basis, these resource usages can be simply converted into the corresponding service demands. The primary issue is that Oracle does not record the resource usages of DBWR or LGWR on a per-session basis; therefore, the challenge actually lies in the analytic modeling of DBWR and LGWR.

1. The 'redo blks written' in Figure 26b is scaled by a factor of 1/8 because a redo block size (512 bytes) is one eighth of a data block size (4K bytes).

4.5 Analytic Modeling of DBWR and LGWR

Analytic modeling of DBWR and LGWR must establish their workload units, and we argue that the number of data blocks and the number of redo blocks can serve that purpose for two reasons. First, Oracle actually records disk I/O statistics in *v$tables* based on these two units; second, there are some hints in v$sesstat of a user session related to the numbers of data blocks modified and redo blocks logged on behalf of the user session. The previous experiments with respect to concurrent *Stress* instances also provide some measurement data to study this problem.

Table 27. Per-block averages of Oracle service times for DBWR and LGWR.

#clients	1	2	3	4	5	6	7	8	9	10	*mean*	*stdev*
disk_time/blk_r	0.090	0.103	0.232	0.213	0.247	0.277	0.373	0.421	0.433	0.503	0.289	0.140
cpu_time/blk_w	2.998	4.152	5.318	6.153	6.302	6.469	6.616	6.636	6.764	6.721	5.813	1.281
disk_time/blk_w	12.680	17.564	21.228	25.234	27.143	27.973	28.524	29.823	32.396	33.497	25.604	6.612
cpu_time/redo_blk	0.965	1.110	1.024	0.984	1.014	0.970	0.997	0.995	0.949	0.948	0.996	0.048
disk_time/redo_blk	13.398	14.382	13.985	13.673	13.381	13.187	13.108	13.104	13.065	13.123	13.441	0.444

Table 27 lists the per-block averages of Oracle service times for DBWR and LGWR in the form of the following:

disk_time/blk_r the estimated disk time to read a data block derived from v$filestat,
cpu_time/blk_w the average DBWR CPU time to write a data block derived from v$sesstat,
disk_time/blk_w the estimated disk time to write a data block derived from v$filestat,
cpu_time/redo_blk the average LGWR CPU time to write a redo block derived from v$sesstat, and
disk_time/redo_blk the estimated disk time to write a redo block from v$sesstat of LGWR.

Figure 28 plots the DBWR service times per data block against #*clients* in Table 27. We can observe a steep increase in *cpu_time/blk_w* from 1 to 4 clients; after that, it gradually saturates around 6.7 ms. As mentioned in the previous subsection, the *disk_time/blk_w* is somewhat misleading. Although we observe that it increases with #*clients*, the actual increased component is the disk waiting time instead of disk time. Therefore, we must calibrate this statistic with the corresponding disk statistics from the *filemon* reports.

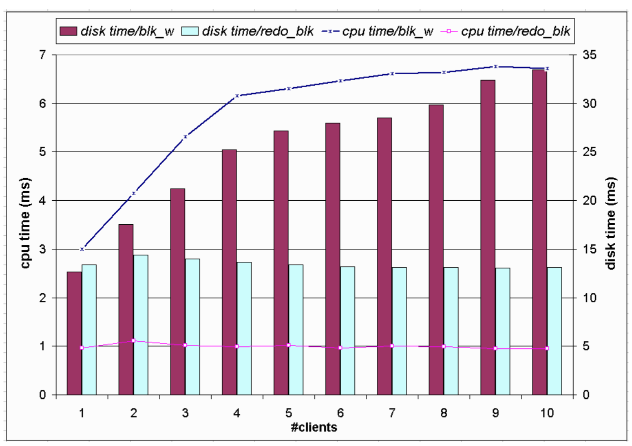


Figure 28. Decomposed Oracle service times for DBWR and LGWR in units of data/redo blocks.

Figure 28 also plots the LGWR service times per redo block against #*clients* in Table 27. Since LGWR's disk requests are less likely to be waiting for the disk requests from other Oracle processes, we can observe that *cpu_time/redo_blk* and *disk_time/redo_blk* do not vary with #*clients*.

After the unit service times of DBWR and LGWR are established, the next issue for analytic modeling is how to estimate the per-transaction averages of *data blocks written* (or *physical writes*) and *redo blocks written*. We discuss these in the following subsections.

4.5.1 Estimating Data Blocks Written per Transaction

As mentioned in Section 2.3.3, the only feasible solution to measure the number of data blocks written per user transaction is to collect the disk performance statistics of Oracle *v$tables* in a homogeneous workload environment because we cannot separate the *physical writes* in v$sesstat of DBWR for individual user sessions. Measurement data in Table 16 shows that the per-request average of data blocks written for *Stress updates* does not vary significant with #*clients* in a homogeneous workload environment.

4.5.2 Estimating Redo Blocks Written per Transaction

There are a few hints in v$sesstat for estimating *redo blocks written* per transaction. For an Oracle user session, the hints are *redo entries* and *redo size*; for LGWR, the hints are *redo wastage* and *redo blocks written*. Table 29 lists the per-request averages of these performance statistics for *Stress update*. If we divide the sum of *redo size* and *redo wastage* with *redo blocks written*, we can get an average ratio of 493.224 with a very small variance. This discovery means that we can estimate the average number of redo blocks per request if we know the *redo size* and the average amount of *redo wastage* per request.

Table 29. Estimating the number of redo blks per request for *Stress update*.

#clients	1	2	3	4	5	6	7	8	9	10	*mean*	*stdev*
redo entries	2.002	2.004	2.006	2.008	2.007	2.007	2.007	2.007	2.007	2.007		
redo blocks written	2.000	1.989	1.931	1.837	1.773	1.735	1.703	1.679	1.660	1.639		
redo size	571.475	572.790	573.699	574.372	574.239	574.207	574.161	574.113	574.050	574.032	573.741	0.833
redo wastage	413.360	412.292	379.239	332.378	300.535	281.712	265.726	254.055	244.750	234.430		
(redo size + redo wastage) / redo blks written	492.503	493.022	493.392	493.624	493.464	493.387	493.309	493.243	493.171	493.124	493.224	0.308

Although it is still difficult to obtain the average amount of redo wastage per user transaction, we can always estimate the number of redo blocks per user transaction in a worst case scenario from the average *redo size* of a user transaction according to Table 29.

4.6 Hidden OS Issues for Oracle Disk Time Measurement

According to the measurement methodology described in Section 2, we estimate Oracle disk times with the long-term disk performance statistics in v$filestat and in v$sesstat of LGWR. The merit of this approach is that it hides two complicated features about disk file IOs in AIX Journaled File Systems (JFS), i.e. caching of memory mapped files and JFS logging, that are very difficult to measure or model.

AIX uses persistent storage pages to cache data blocks of opened files retrieved from disk; the traditional disk buffer caches of UNIX are not used in AIX [15]. From the perspective of AIX file systems, the Oracle disk statistics are actually logical file I/O statistics instead of physical disk I/O statistics. Therefore, reading a data block may not cause physical disk reads because the accessed disk block is possible already cached in memory. Similarly, writing a data block merely modifies the corresponding persistent pages; the modified pages may not be written to disk immediately.

The JFS is the default file system type for local disk files in AIX. JFS logs activity associated with the file system control structures such that it can reconstruct a file system to a known state in the event of a system crash. According to Kelley [15],

> "... The goal of the journaled file system is to provide a more robust file system by logging changes made to its own structures and lists. This includes changes made to the file system super block, the inodes, directories, indirect blocks, and free lists of inodes and data blocks...

> The JFS uses a physical disk partition as a log device. Each volume group must have a JFS log device. The rootvg's log device is /dev/hd8 (8MB). The JFS also maintains a log segment (256MB) in virtual memory for each log device. Pages of this segment are written to the disk log device at regular intervals..."

In order to study how much disk writes are associated with JFS logging, we ran several experiments with multiple concurrent *stress* instances and measured disk utilizations of individual logical volumes with *filemon*. The results indicate that JFS logging actually consumes one half of the disk utilization.

From the performance modeling perspective, a good thing about JFS logging is that it uses a dedicated partition; the bad thing is that JFS logging is transparent to applications, and it is too difficult to correlate the JFS logging with Oracle disk writes. Even though we have yet to measure and correlate Oracle I/O statistics with actual physical disk I/O statistics, the Oracle disk statistics in *v$tables* are adequate for Oracle disk time measurement.

4.7 Analytic Modeling of Multi-Tiered Client/Server Distributed Applications

In our previous work [4], we have constructed a closed queueing network model for the two-tiered PPS6 distributed applications. While evaluating this model with the *Stress* program, we determined that the Oracle service demands actually vary with the number of concurrent clients and cause discrepancies between the model and the system. In this paper, we address this issue by decomposing the Oracle service demands. This decomposition requires us to improve the measurement methodology with Oracle *v$tables* and to incorporate a different analytic modeling technique called *Method of Layers* (MOL) [28]. MOL evaluates analytic performance models constructed with one or more layers of software processes, called *Layered Queueing Models* (LQMs).

4.7.1 Layered Queueing Models

LQMs have been applied to model distributed client-server systems that contain one or more layers of software servers [16,28]. We apply this technique to refine the analytic model because our measurement methodology separates the Oracle service times for different Oracle processes. Therefore, we can construct an LQM for two-tiered *Stress* as shown in Figure 30. Each parallelogram in an LQM represents a group of one or more processes and each circle is a device. Directed arcs indicate requests for service from the calling group to the serving group or device. MOL divides an LQM into two complementary models, one for software and one for devices, and combines the results to provide performance estimates for the system.

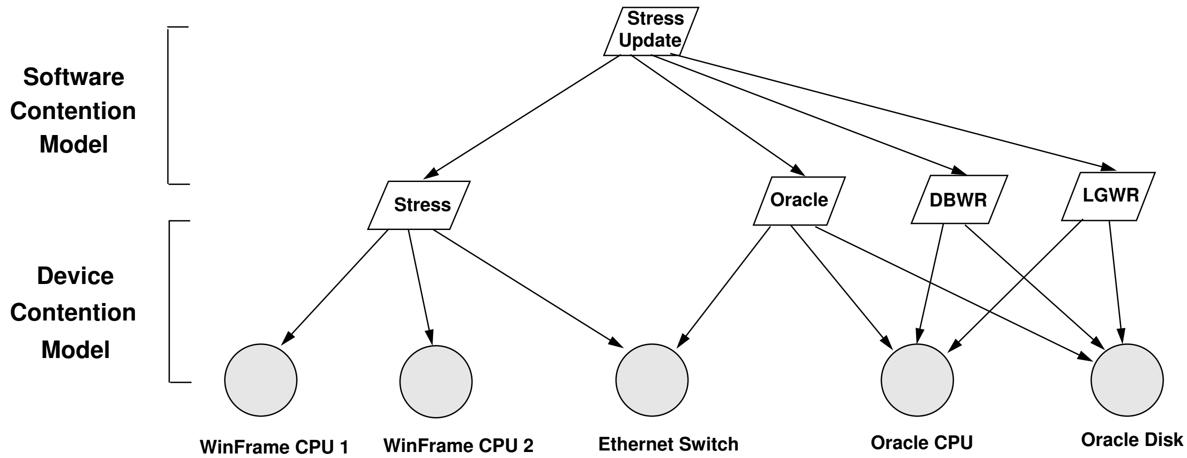


Figure 30. The two-tiered *Stress* LQM.

The two-tiered *Stress* LQM contains five hardware devices of the testbed configuration: two CPUs of the WinFrame server, the Ethernet switch, and the CPU and disk of the Oracle server. The four software groups correspond the important processes while running the *Stress* program in a two-tiered environment: *stress.exe*, Oracle server process, DBWR and LGWR background processes. The software contention model of the *Stress* LQM is depicted in the upper two layers of Figure 30; which describes the relationships in the software process architecture and is used to predict software contention delays. The device contention model of the *Stress* LQM is depicted in the lower two layers; which describes how software processes request services to hardware devices and is used to determine queueing delays at devices. The requests for service of the two-tiered *Stress* LQM are shown in Tables 31 and 32.

Table 31. Requests for service in the two-tier *Stress* LQM.

	Stress Update	Stress	Oracle	DBWR	LGWR
Stress (*FCFS*)	$V = 1$				
Oracle (*Rendezvous*)	$V = 1$				
DBWR (*FCFS*)	$V = N_{phy\text{-}blk\text{-}wrt}$				
LGWR (*FCFS*)	$V = N_{log\text{-}blk\text{-}wrt}$				
WinFrame CPU 1 (*PS*)		$V = 0.5$ $S = D_{cpu\text{-}stress}$			
WinFrame CPU 2 (*PS*)		$V = 0.5$ $S = D_{cpu\text{-}stress}$			
Ethernet Switch (*PS*)		$V = N_{SQL*Net\text{-}rt}$ $S = D_{net\text{-}SQL*Net\text{-}recv}$	$V = N_{SQL*Net\text{-}rt}$ $S = D_{net\text{-}SQL*Net\text{-}sent}$		
Oracle CPU (*PS*)			$V_1 = 1$ $S_1 = D_{cpu\text{-}oracle}$ $V_2 = 1$ $S_2 = D_{cpu\text{-}os}$	$V = 1$ $S = D_{cpu\text{-}blk\text{-}wrt}$	$V = 1$ $S = D_{cpu\text{-}blk\text{-}log}$
Oracle Disk (*PS*)			$V = N_{phy\text{-}blk\text{-}rd}$ $S = D_{disk\text{-}blk\text{-}rd}$	$V = 1$ $S = D_{disk\text{-}blk\text{-}wrt}$	$V = 1$ $S = D_{disk\text{-}blk\text{-}log}$

Note that all software processes, except **Oracle**, are modeled as FCFS servers and all hardware devices are modeled as **PS** servers (round-robin process sharing scheduling discipline). We model the Oracle server process as an Rendezvous server (i.e. an FCFS process with two phases of services); its requests for service to the Oracle CPU have the phase one service demand as the CPU time consumed by the Oracle server process and the phase two service demand as the average OS overhead while servicing a *Stress update* request. We simplify the Ethernet switch model as a PS server because the SQL*Net traffic is less than 10% of Ethernet bandwidth while we evaluate the model. We model DBWR as an FCFS server which takes requests from other software groups for writing modified data blocks to disk. A request for service to DBWR hence uses the number of data blocks written as its visit ratio. Similarly, a request for service to LGWR uses the number of redo blocks written as its visit ratio.

Table 32. Description of requests for service in the two-tiered LQM.

Legend	Description	Value
$D_{cpu\text{-}stress}$	WinFrame CPU service time for servicing a *Stress update* request	22.721 ms
$N_{SQL*Net\text{-}rt}$	*SQL*Net roundtrips to/from client*	8
$D_{net\text{-}SQL*Net\text{-}recv}$	(*bytes received via SQL*Net from client* / *SQL*Net roundtrips to/from client*) * 8 / 10Mbps	0.034 ms
$D_{cpu\text{-}oracle}$	Oracle CPU time of Oracle server process for servicing a *Stress update* request	15.080 ms
$D_{cpu\text{-}os}$	Oracle CPU time as OS overhead for servicing a *Stress update* request	6.681 ms
$N_{phy\text{-}blk\text{-}rd}$	average number of data blocks read per *Stress update* request	0.037
$D_{disk\text{-}blk\text{-}rd}$	Oracle disk service time for reading a data block	0.047 ms
$N_{SQL*Net\text{-}rt}$	*SQL*Net roundtrips to/from client*	8
$D_{net\text{-}SQL*Net\text{-}sent}$	(*bytes sent via SQL*Net to client* / *SQL*Net roundtrips to/from client*) * 8 / 10Mbps	0.056 ms
$N_{phy\text{-}blk\text{-}wrt}$	average number of data blocks written per *Stress update* request	0.152
$D_{cpu\text{-}blk\text{-}wrt}$	Oracle CPU service time for writing a data block	5.813 ms
$D_{disk\text{-}blk\text{-}wrt}$	Oracle disk service time for writing a data block	21.967 ms
$N_{log\text{-}blk\text{-}wrt}$	average number of redo blocks written per *Stress update* request	***varied***
$D_{cpu\text{-}blk\text{-}log}$	Oracle CPU service time for writing a redo block	0.996 ms
$D_{disk\text{-}blk\text{-}log}$	Oracle disk service time for writing a redo block	13.860 ms

4.7.2 Initial Validation Test of the Two-tiered *Stress* LQM

We conduct the initial validation test of the two-tiered *Stress* LQM with the performance measures from our experiment of multiple concurrent *Stress* instances.[1] The values of LQM model parameters applied to model evaluation are listed in the last column of Table 32. Note that we use constant values for all model parameters except $N_{log\text{-}blk\text{-}wrt}$ while we increase the workload population of **Stress Update** (i.e. the number of concurrent Stress instances). Since the average number of redo blocks written per *Stress update* request decreases with *#clients* and causes the major discrepancy of the Oracle disk service demand, we apply different values of $N_{log\text{-}blk\text{-}wrt}$ using redo blocks written per request recorded in Table 25.

Table 33. Comparison of Oracle service demands: measured vs. model.

#clients	**Oracle CPU service demand (ms)**			**Oracle disk service demand (ms)**		
	measured	model	%err	measured	model	%err
1	20.934	24.637	-17.7%	32.673	31.066	4.9%
2	22.508	24.635	-9.5%	33.138	31.036	6.3%
3	23.682	24.569	-3.7%	30.564	30.112	1.5%
4	24.182	24.475	-1.2%	28.220	28.803	-2.1%
5	24.661	24.411	1.0%	26.714	27.913	-4.5%
6	25.142	24.373	3.1%	25.841	27.387	-6.0%
7	25.477	24.341	4.5%	25.104	26.940	-7.3%
8	25.679	24.318	5.3%	24.605	26.614	-8.2%
9	25.844	24.299	6.0%	24.153	26.354	-9.1%
10	26.275	24.278	7.6%	23.705	26.066	-10.0%

1. Khandker has implemented the MOL with a Tcl/Tk user interface in [16]. We constructed and evaluated the two-tiered LQM with this implementation.

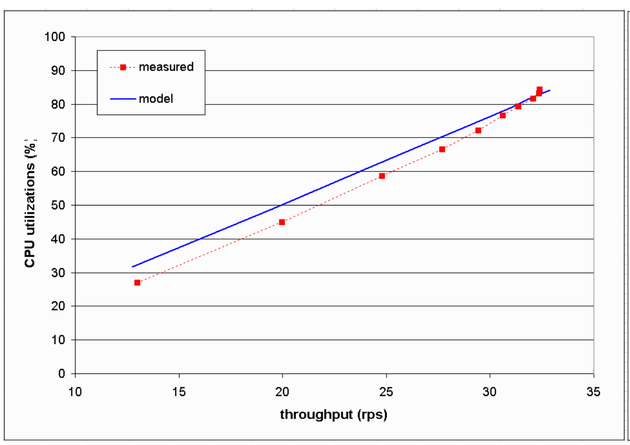
Table 33 lists the differences in Oracle service demands between the real measurements and the values applied to the LQM device contention model. The only element causing the slightly decreasing Oracle CPU service demand is due to the CPU service times for writing redo blocks varying with $N_{log\text{-}blk\text{-}wrt}$. However, the decreasing $N_{log\text{-}blk\text{-}wrt}$ actually causes the Oracle disk service demand in the LQM device contention model to decrease significantly.

The model outputs of the *Stress* LQM are shown in Table 34 and Figure 35. The LQM, in general, predicts the request completion time fairly accurate because the error range is less than 5%. For Oracle CPU and disk utilizations, there are only two cases where the error range between model outputs and measurements is greater than 10%.

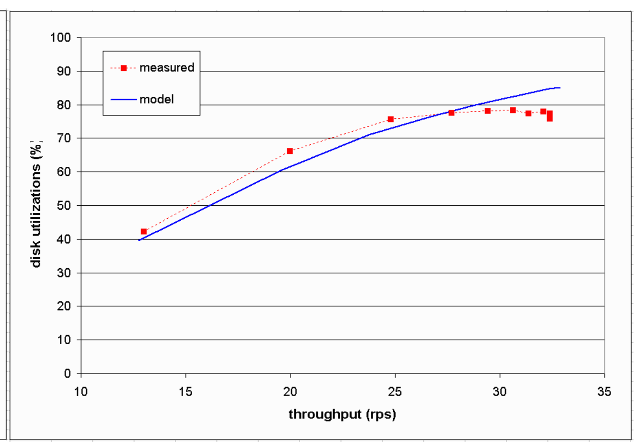
Table 34. Comparison of measurements and model outputs of 2-tier *Stress* LQM.

#clients	Oracle CPU utilization (%)			Oracle disk utilization (%)			Request completion time (ms)			$U_{wf\text{-}cpu}$	$U_{Ethernet}$
	measured	model	%err	measured	model	%err	measured	model	%err		
1	27.092	31.600	-16.6%	42.312	39.500	6.6%	77.044	78.000	-1.2%	14.500	2.700
2	44.903	49.000	-9.1%	66.277	60.400	8.9%	100.119	102.000	-1.9%	22.200	4.200
3	58.623	60.000	-2.3%	75.659	71.100	6.0%	120.995	126.000	-4.1%	26.900	5.100
4	66.615	67.700	-1.6%	77.545	76.500	1.3%	144.456	150.000	-3.8%	30.300	5.800
5	72.255	72.900	-0.9%	78.100	79.700	-2.0%	169.995	174.000	-2.4%	32.600	6.300
6	76.555	76.500	0.1%	78.425	81.700	-4.2%	196.030	200.000	-2.0%	34.100	6.600
7	79.336	79.200	0.2%	77.392	83.100	-7.4%	223.223	225.000	-0.8%	35.300	6.900
8	81.665	81.200	0.6%	77.951	84.000	-7.8%	249.528	252.000	-1.0%	36.100	7.000
9	83.170	82.800	0.4%	77.477	84.700	-9.3%	278.073	278.000	0.0%	36.700	7.200
10	84.394	84.200	0.2%	75.889	85.200	-12.3%	308.787	304.000	1.6%	37.400	7.300

Figure 35a plots the Oracle CPU utilizations against throughput. The discrepancy between measured values and model outputs is because we do not consider the increasing component in the Oracle CPU service demand caused by the Oracle server process. This increasing component in CPU service times is actually caused by the lock contention for concurrent Oracle server processes accessing the Oracle SGA. Some preliminary research works addressing this issue has been discussed in [2].



(a) Oracle CPU utilization.



(b) Oracle disk utilization.

Figure 35. Oracle server CPU and disk utilizations: measured vs. model outputs.

Figure 35b plots the Oracle disk utilization against throughput. Although we have addressed the decreasing Oracle disk service demand with decreasing $N_{log\text{-}blk\text{-}wrt}$ while evaluating the *Stress* LQM, we find there is still small discrepancy between measured values and model outputs. The actual reason causing this discrepancy is because we apply the long-term statistic value of $D_{disk\text{-}blk\text{-}log}$ while evaluating the

LQM. This value (13.860 ms) is much bigger than the average measurement value (13.441 ms) while conducting the experiment of concurrent *Stress* instances. As a consequence, this raises an important issue for applying the long-term disk statistics to derive the model parameters. We intend to address this issue in our future work.

4.7.3 A Three-Tiered LQM for PPS6 Remote Computing

In the previous subsection, we have constructed a two-tiered LQM and conducted the initial validation test for the model. The two-tiered LQM can be extended to model the three-tiered remote computing of PPS6. The three tiered LQM contains two more software servers for modeling *wfica32.exe* on a WinStation and *csrss.exe* on the WinFrame server as shown in Figure 36.

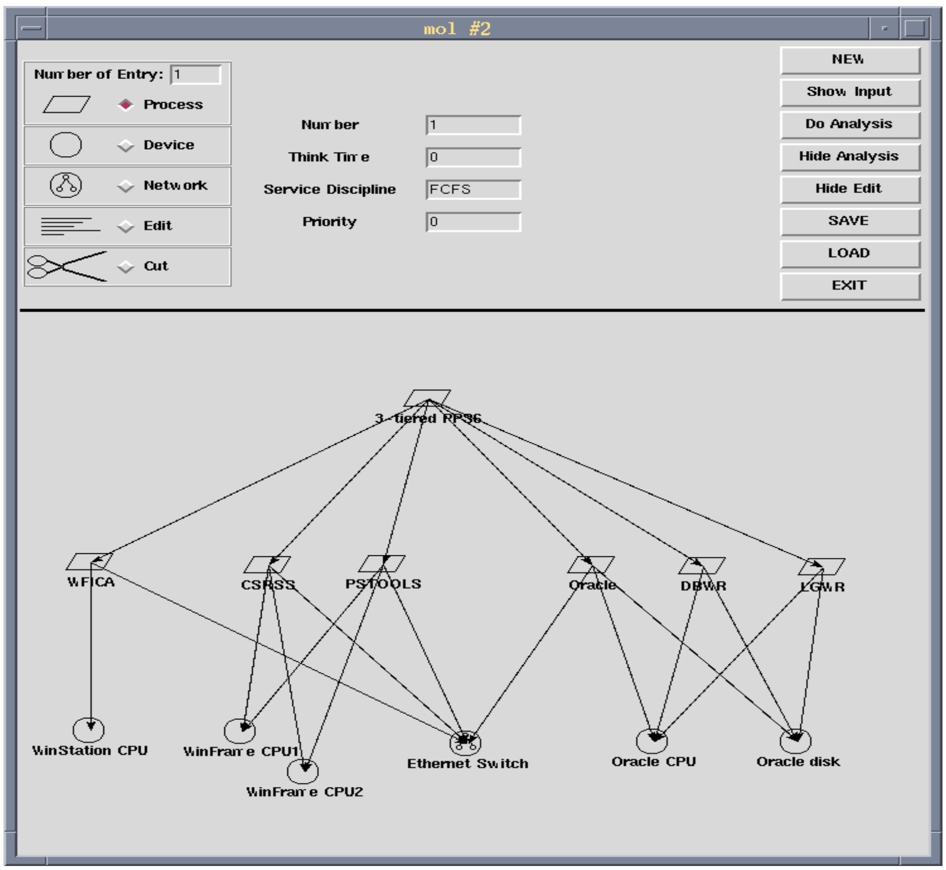


Figure 36. A three-tiered LQM for modeling the remote computing of PPS6.

5. Conclusions and Future Work

In order to address the varying Oracle disk service demands, we have refined our measurement methodology and the initial analytic model by decomposing the disk service demand into three components: data block reads, data block writes, and redo log writes. Measuring these components with Oracle *v$tables*, we discovered that only the redo log writes exhibit significant varying disk service demands due to LGWR's group commit.

Next, we performed some measurements to determine what fraction of the total disk I/O could be attributed to the redo log. We constructed a three-tier PPS6 environment testbed in the CITI lab, and measured our synthetic *Stress* workload there. This workload exhibits a highly variable disk service demand, with 54% of all disk traffic going to the redo log. This is due to *Stress'* high update rate. We then measured real PPS6 applications running against test Oracle server in the University of Michigan production environment. This workload exhibits less variation in disk service demand because the redo log accounts for only 2% of all disk traffic. We believe this is due to the PPS6 application's design, in which many database records are accessed while a new record is being assembled; this single record is then written.

For workloads similar to PeopleSoft OLTP transactions, our original model based on QNM is thus sufficient for analysis and modeling for capacity planning purposes. For workloads exhibiting behavior closer to that observed for *Stress*, it will be necessary to use an LQM model. We have started work on such a model, and have presented some preliminary results here; in general, the LQM model correlates well with our initial testbed measurements. Further work includes completing the validation of this model and extending the two-tiered LQM to three-tiered LQM.

References

1. E. Aronoff, K. Loney, and N. Sonawalla. *Advanced Oracle Tuning and Administration*. Oracle Press, 1997.
2. D.P. Atkinson. "Capacity Modeling of Oracle-based System." *Proceeding of UK-CMG'97*, 1997.
3. J.P. Buzen and A.W. Shum. "Considerations for Modeling Windows NT." *Proceeding of CMG'97*, December 1997.
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. Cygnus Production. *StatBar Console*. [http://www.mcs.net/~cygnus/freeware/freeware.htm], 1998.
8. Helen Custer. *Inside Windows NT*. MicroSoft Press 1993.
9. Tim Foxon. "Performance Analysis of an Oracle-based Interactive UNIX System - a Case Study." *Proceeding of CMG'94*, December 1994.
10. Mark Friedman. "*Windows NT Performance Monitoring: an Overview.*" [http://www.demandtech.com], 1997.
11. Adam Grummitt and Tim Foxon. "Performance Tuning and Capacity Planning for Oracle on UNIX - A Case Study." *Proceeding of CMG'97*, December 1997.
12. Guy Harrison. "Getting the Most from the SQL_TRACE Facility." *Oracle View*, Spring 1996.
13. IBM Redbooks. *RS/6000 Performance Tools in Focus*. IBM International Technical Support Organization, May 1997.
14. Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
15. D. A. Kelley. *AIX/6000 Internals and Architectures*. McGraw-Hill, 1996.
16. A.M. Khandker. *Performance Measurement and Analytic Modeling Techniques for Client-Server Distributed Systems*. Ph.D. Dissertation, The University of Michigan, 1996.

17. Edward Lazowska, John Zahorjan, Scott Graham, and Kenneth Sevcik. *Quantitative System Performance*. Prentice Hall, 1984.

18. Daniel Menasce, Virgilio Almeida, and Larry Dowdy. *Capacity Planning and Performance Modeling: from Mainframes to Client-Server Systems*. Prentice Hall, 1994.

19. Mercury Interactive. *LoadRunner*. [http://www.merc-int.com/products/loadrunguide.html], 1998.

20. MicroSoft. *Windows 95 Kernel Toys - Windows Process Watcher*. [http://www.microsoft.com/windows/download], 1997.

21. M-Pathways. *The M-Pathways Project*. [http://www.mpathways.umich.edu], 1997.

22. *Oracle7 Server Concepts Manual*. Oracle7 Documentation. Redwood Shores, CA, 1996.

23. *Oracle7 Server Reference Manual*. Oracle7 Documentation. Redwood Shores, CA, 1996.

24. *Oracle7 Server Tuning*. Oracle7 Documentation. Redwood Shores, CA, 1996.

25. PeopleSoft. *PeopleSoft's Distributed Architecture*. [http://www.peoplesoft.com], 1997.

26. PeopleSoft. *Introduction to PeopleTools*. PeopleSoft Inc., 1996.

27. Rational. *preVue-C/S*. [http://www.rational.com/products/prevue/cs/index.html], 1998.

28. J.A. Rolia and K.C. Sevcik. "The Method of Layers." *IEEE Transactions on Software Engineering*, 21(8), pp. 689-699, August 1995.

29. Willison WindowWare. *WinBatch*. [http://www.windowware.com/winware/winbatch.html], 1998.

Appendix A. Oracle v$sesstat.

Although the Oracle v$sesstat contains a rich set of performance statistics, the definitions and descriptions of these performance statistics are very poorly documented by Oracle. Table 37 provides some of these definitions or description that we can find.

Table 37. Performance Statistics in the Oracle v$sesstat table.

#	NAME	DESCRIPTION
0	*logons cumulative*	The cumulative number of connections to the database from the last warm start.
1	*logons current*	The current number of open sessions in the database.
2	*opened cursors cumulative*	This is the cumulative number of cursors opened.
3	*opened cursors current*	This is the number of the currently active SQL statements.
4	*user commits*	The number of **commit** calls issued users.
5	*user rollbacks*	The number of **rollback** calls issued by users.
6	*user calls*	The number of user calls issued by users. Calls can be any of the following: describe, parse, open fetch, close, or execute. If a SQL statement fetches more than one row at a time, each array read will count as one user call.
7	*recursive calls*	The number of recursive calls generated by the Oracle kernel rather than by users.
8	*recursive cpu usage*	The associated CPU usage for recursive calls.
9	*session logical reads*	The number of logical reads, which is the sum of *db block gets* and *consistent gets*.
10	*session stored procedure space*	The number of bytes allocated to stored procedures used in the session.
11	*CPU used when call started*	***CPU used by the current SQL statement = #11 - #12***
12	*CPU used by this session*	Amount of CPU time used in 1/100's of a second.
13	*session connect time*	The *timestamp* when this session was connected.
14	*process last non-idle time*	The *timestamp* of the last user call.
15	*session uga memory*	The amount of UGA memory allocated to this session.
16	*session uga memory max*	The maximum amount of UGA memory allocated to this session.
17	*messages sent*	These two numbers reflect the communications between the various Oracle background processes. Generally a message is sent every time a **commit** is made or a new connection the database is opened.
18	*messages received*	
19	*background timeouts*	The number of timeouts issued for Oracle background processes.
20	*session pga memory*	The amount of PGA memory allocated to this session.
21	*session pga memory max*	The maximum amount of PGA memory allocated to this session.
22	*enqueue timeouts*	The number of times an enqueue request was not granted within the allotted wait time.
23	*enqueue waits*	The number of times an enqueue request resulted in a wait.
24	*enqueue deadlocks*	the number of process deadlocks that occurred due to enqueues for DDL operations.
25	*enqueue requests*	The number of times an enqueue was requested.
26	*enqueue conversions*	The number of enqueue locks converted to a different mode.
27	*enqueue releases*	The number of times an enqueue was released.
28	*global lock gets (non async)*	
29	*global lock gets (async)*	
30	*global lock get time*	
31	*global lock converts (non async)*	
32	*global lock converts (async)*	
33	*global lock convert time*	
34	*global lock releases (non async)*	
35	*global lock releases (async)*	
36	*global lock release time*	
37	*db block gets*	This is the number of blocks accessed via single block gets (i.e., not through the consistent get mechanism). Its value increments when a block is read for update and when segment header blocks are accessed.
38	*consistent gets*	This is the number of accesses made to the block buffer to retrieve data in a consistent mode. Its value is incremented according to the operation performed: For *full table scans*: it increments once per block read. For *table access through the index*: it increments by index height (usually 2) + 2 *keys in the range. For *access inside the index only*: it increments once per block read.
39	*physical reads*	This is the cumulative number of blocks read from the disk. Its value is incremented once per read block regardless of whether the read request was for a multiblock read or single block read. ***Logical Reads / (Logical Reads + Physical Reads) = Hit Ratio***
40	*physical writes*	This is the cumulative number of blocks written to disk. Its value is incremented once per block written regardless of whether the write request was for a multiblock write or single block write. Most physical writes are done by DBWR.
41	*write requests*	This is the cumulative number of write requests. ***Physical Writes/Write Requests = # of blocks written per single write***

Table 37. Performance Statistics in the Oracle v$sesstat table.

#	NAME	DESCRIPTION
42	*summed dirty queue length*	This is the sum of the buffers left in the write queue after every write request.
43	*db block changes*	This is the cumulative number of modified blocks chained to the "dirty" list. Once the block is in the "dirty" list, additional changes to that block are not counted as block changes.
44	*change write time*	
45	*consistent changes*	This is the number of times a consistent get had to retrieve an "old" version of a block because of updates that occurred after the cursor had been opened. It does not indicate the number of updates (or changes). A more accurate statistics for number records changed is 'data blocks consistent reads - undo records applied.'
46	*redo synch writes*	The number of user commits, the number of checkpoints, and the number of log switches. Its value is incremented every time a user commits, at every checkpoint and every log switch.
47	*redo synch time*	
48	*exchange deadlocks*	
49	*free buffer requested*	The total number of free buffer requested. Free buffers in the buffer cache are requested for multiple reasons. When data is inserted into the database, a free buffer is requested every time a new block is needed. On updates, free buffers are requested to contain the rollback information.
50	*dirty buffers inspected*	This is the number of times a foreground process encounters a dirty buffer that has aged out via the LRU queue while the process is looking for the buffer for reuse.
51	*free buffer inspected*	The number of buffers skipped in the buffer cache in order to find a free buffer.
52	*commit cleanout failure: write disabled*	
53	*commit cleanout failures: hot backup in progress*	
54	*commit cleanout failures: buffer being written*	
55	*commit cleanout failures: callback failure*	
56	*total number commit cleanout calls*	
57	*commit cleanout number successfully completed*	
58	*DBWR timeouts*	
59	*DBWR make free requests*	
60	*DBWR free buffers found*	
61	*DBWR lru scans*	
62	*DBWR summed scan depth*	
63	*DBWR buffers scanned*	
64	*DBWR checkpoints*	This is the number of checkpoints messages that were sent to DBWR. During a checkpoint there is a slight decrease in performance since data blocks are being written to disk which causes I/O. If the number of checkpoints is reduced, the performance of normal database operation improves but recovery after instance failure is slower.
65	*DBWR cross instance writes*	
66	*remote instance undo block writes*	
67	*remote instance undo header writes*	
68	*remote instance undo requests*	
69	*cross instance CR read*	
70	*recovery blocks read*	
71	*recovery array reads*	
72	*recovery array read time*	
73	*CR blocks created*	
74	*Current blocks converted for CR*	
75	*calls to kcmgcs*	
76	*calls to kcmgrs*	
77	*calls to kcmgas*	
78	*next scns gotten without going to DLM*	
79	*Unnecessary process cleanup for SCN batching*	
80	*calls to get snapshot scn: kcmgss*	
81	*kcmgss waited for batching*	
82	*kcmgss read scn without going to DLM*	
83	*kcmccs called get current scn*	
84	*redo entries*	The number of entries in the redo log buffer.
85	*redo size*	The amount of redo generated in bytes.
86	*redo entries linearized*	This is the number of times a redo entry was pre-built before trying to obtain a latch in order to write into the redo buffer. ***Redo Size / Redo Entries = Average Size of Redo Entries***
87	*redo buffer allocation retries*	The number of attempts to allocate space in the redo buffer.
88	*redo small copies*	This is the total number of redo entries with fewer bytes than specified by the init.ora parameter LOG_SMALL_ENTRY_MAX_SIZE. These entries are written in the redo buffer under the protection of the redo allocation latch.

Table 37. Performance Statistics in the Oracle v$sesstat table.

#	NAME	DESCRIPTION
89	*redo wastage*	Cumulative total of unused bytes that were written to the log. The redo buffer is flushed periodically even when not completely filled, so this value may be nonzero.
90	*redo writer latching time*	This is the time needed by the process writing redo to obtain and release each copy latch. If this time is high, the timeouts for the redo allocation and copy latches should be analyzed.
91	*redo writes*	The total number of redo writes.
92	*redo blocks written*	The total number of redo blocks written.
93	*redo write time*	Cumulative elapsed time spent on log I/O.
94	*redo log space requests*	The number of times a user process waits for space in the redo log buffer of SGA.
95	*redo log space wait time*	Time spent waiting for log space in 1/100's of a second.
96	*redo log switch interrupts*	The count of times an instance log-switch was requested by another instance when running in parallel mode.
97	*redo ordering marks*	
98	*hash latch wait gets*	
99	*background checkpoints started*	The number of checkpoints started.
100	*background checkpoints completed*	The number of checkpoints completed.
101	*serializable aborts*	
102	*transaction lock foreground requests*	
103	*transaction lock foreground wait time*	
104	*transaction lock background gets*	
105	*transaction lock background get time*	
106	*transaction tables consistent reads - undo records applied*	
107	*transaction tables consistent read rollbacks*	
108	*data blocks consistent reads - undo records applied*	This number is the actual number of data records changed.
109	*no work - consistent read gets*	
110	*cleanouts only - consistent read gets*	
111	*rollbacks only - consistent read gets*	
112	*cleanouts and rollbacks - consistent read gets*	
113	*rollback changes - undo records applied*	
114	*transaction rollbacks*	
115	*immediate (CURRENT) block cleanout applications*	
116	*immediate (CR) block cleanout applications*	
117	*deferred (CURRENT) block cleanout applications*	
118	*table scans (short tables)*	The number of full table scans on tables with less than 4 db_blocks.
119	*table scans (long tables)*	The number of full table scans on tables that have more than 5 data blocks. ***Table Scans (long tables) + Table Scans (short tables) = # of full able scans***
120	*table scans (rowid ranges)*	These two statistics are used with the Parallel Query Option.
121	*table scans (cache partitions)*	
122	*table scans (direct read)*	
123	*table scan rows gotten*	The cumulative number of rows read for full table scans.
124	*table scan blocks gotten*	The cumulative number of blocks read for full table scans.
125	*table fetch by rowid*	This is the cumulative number of rows fetched from tables using a TBALE ACCESS BY ROWID operation.
126	*table fetch continued row*	This is the cumulative number of continued rows fetched.
127	*cluster key scans*	The number of requests for record reads for a given cluster key.
128	*cluster key scan block gets*	The number of database blocks accessed to retrieve a set of clustered records.
129	*parse time cpu*	Accumulative CPU time for parsing SQL statements.
130	*parse time elapsed*	Accumulative elapsed time for parsing SQL statements.
131	*parse count*	The number of parse calls made by the session/system.
132	*execute count*	The number of execution calls made by the session/system.Its value incremented for every execute request and for every time a cursor is opened.
133	*bytes sent via SQL*Net to client*	The cumulative number of bytes in SQL*Net messages was sent to client.
134	*bytes received via SQL*Net from client*	The cumulative number of bytes in SQL*Net messages was received from client.
135	*SQL*Net roundtrips to/from client*	The number of times a message was sent and an acknowledgment received.
136	*bytes sent via SQL*Net to dblink*	
137	*bytes received via SQL*Net from dblink*	
138	*SQL*Net roundtrips to/from dblink*	
139	*sorts (memory)*	This is the number of sorts small enough to be performed entirely in sort areas without using temporary segments.

Table 37. Performance Statistics in the Oracle v$sesstat table.

#	NAME	DESCRIPTION
140	*sorts (disk)*	This is the number of sorts which require the use of temporary segments for sorting.
141	*sorts (rows)*	The total number of rows sorted.
142	*session cursor cache hits*	The number o times the cursor was cached in the session and even a parse call was not made.
143	*session cursor cache count*	
144	*cursor authentications*	
145	*OS User time used*	The total amount of time running in user mode.
146	*OS System time used*	The total amount of time spent in the system executing on behalf of the processes.
147	*OS Maximum resident set size*	The maximum size, in kilobytes, of the used resident set size.
148	*OS Integral shared text size*	An integral value indicating the amount of memory used by the text segment that was also shared among other processes.
149	*OS Integral unshared data size*	An integral value of the amount of unshared memory in the data segment of a process.
150	*OS Integral unshared stack size*	
151	*OS Page reclaims*	The number of page faults serviced without any I/O activity. In this case, I/O activity is avoided by reclaiming a page frame from the list of pages awaiting reallocation.
152	*OS Page faults*	The number of page faults serviced that required I/O activity.
153	*OS Swaps*	The number of times a process was swapped out of main memory.
154	*OS Block input operations*	The number of times the file system performed input.
155	*OS Block output operations*	The number of times the file system performed output.
156	*OS Socket messages sent*	The number of IPC messages sent.
157	*OS Socket messages received*	The number of IPC messages received.
158	*OS Signals received*	The number of signals delivered.
159	*OS Voluntary context switches*	The number of times a context switch resulted because a process voluntarily gave up the processor before its time slice was completed. This usually occurs while the process waits for availability of a resource.
160	*OS Involuntary context switches*	The number of times a context switch resulted because a higher priority process ran or because the current process exceeded its time slice.