TPC Benchmark™ B
Full Disclosure Report for

Silicon Graphics
CHALLENGE XL Server
and ORACLE7

February 25, 1997

SiliconGraphics, Inc.
Computer Systems

Document Number 007-2221-001

TPC Benchmark™ B Full Disclosure Report for Silicon Graphics Computer Systems CHALLENGE XL Server using ORACLE7.

Abstract

Overview

This report documents the methodology and results of the TPC Benchmark™ B test conducted by Silicon Graphics Computer Systems, with the assistance of ORACLE Corporation, on the Silicon Graphics Computer Systems CHALLENGE XL Server using ORACLE7. All tests were run on a CHALLENGE XL Server used as the host computer running the IRIX (UNIX) operating system. The application code was written in C, and compiled with IRIX-ANSI C compiler.

TPC Benchmark™ B Metrics

The standard TPC Benchmark™ B metrics, tpsB (transactions per second) and price per tpsB (five year capital cost per measured tpsB) are reported as required by the benchmark specification. Throughout this report, tpsB refers to the tpsB performance metric. The next two pages contain the executive summaries of the benchmark results for the above system.

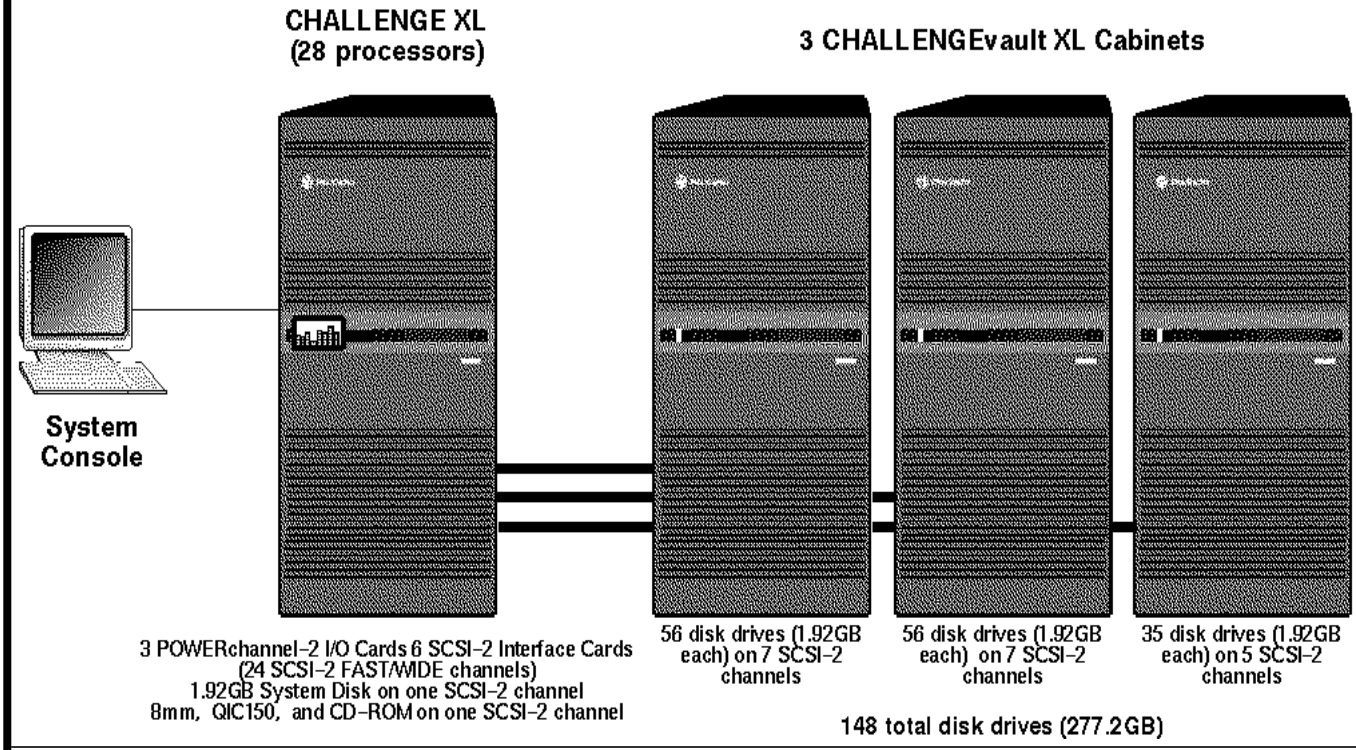
Auditor

The results of the benchmark tests, the methodology used to produce the results, and the calculations to produce the price per tpsB were independently audited by Performance Metrics, Inc. of Los Gatos, CA.

Silicon Graphics Computer Systems	CHALLENGE XL Server	TPC-B Rev 1.2
		February 25, 1997

Total System Cost	TPC-B Throughput	Price / Performance
$2,875,275.20	1786.20 tpsB	$1609.72 per tpsB

Processor	Database Manager	Operating System	Other Software	Number of Users
MIPS R4400SC	ORACLE Version 7.0.15.4.1	IRIX HEP-1.0		166



System Components	Quantity	Description
Processors	28	MIPS R4400SC with 16KB I-cache, 16KB D-cache, and 4MB combined secondary cache per processor
Memory	2GB	4 x 512MB board with 8-way interleaving
I/O Controllers	3	POWERChannel-2 I/O Controllers (with 2 each SCSI-2 FAST/WIDE channels)
SCSI-2 Cards	6	Each with 3 FAST/WIDE SCSI-2 Channels
Disk Racks	3	CHALLENGEvault XL racks with power supply
Total Disk Drives	148	Capacity - 277.24GB
Tape Drives	1	150MB QIC streaming tape
	1	5GB 8mm tape drive
Terminals	1	System console
Miscellaneous Peripherals	1	CD-ROM drive

Silicon Graphics Computer Systems	CHALLENGE XL Server	TPC-B Rev. 1.2
		February 25, 1997

Order Number	Description	Quantity	Unit Price	Extended Price	Support (5 years)
CHALLENGE XL Server					
R-45828-S4	28-cpu CHALLENGE XL Server	1	$919900.00	$919900.00	$262450.00
FTO-64UP2GB	First 2GB High Density Memory	1	212576.00	212576.00	65225.00
SD8-S-2	2GB SCSI-2 FAST/WIDE System Disk	1	8900.00	8900.00	825.00
P-S-B224	CHALLENGEvault XL 224GB Disk Bundle	1	560000.00	560000.00	46200.00
P-S-B64	CHALLENGEvault XL 64GB Disk Bundle	1	296990.00	296990.00	26400.00
P8-S-2	2GB SCSI-2 FAST/WIDE Disk	4	8900.00	35600.00	3300.00
P-S-SBX2-X	SCSIBOX2 for CHALLENGEvault XL	1	3750.00	3750.00	650.00
HU-PC2	POWERChannel-2 I/O Controller	2	12000.00	24000.00	8500.00
P-S-HIO	SCSI-2 FAST/WIDE Interface Card	6	2500.00	15000.00	5400.00
P8-QIC-CD	150MB QIC tape & CD-ROM	1	2000.00	2000.00	1700.00
P8-T4V2	5GB 8mm Internal Drive	1	7300.00	7300.00	2650.00
P-TER2	110 VAC Programming Terminal	1	1500.00	1500.00	600.00
DK-C2-001	Destination Kit for XL Series	1	0.00	0.00	0.00
DK-T2-001	Destination Kit for CHALLENGEvault XL	3	0.00	0.00	0.00
SC4-HEP-1.0	Operating System Software and Manuals	1	0.00	0.00	0.00
SC4-IDO-5.1	IRIX development option for IRIX 5.1	1	1200.00	1200.00	0.00
CS-SWCARE-DEV	Software options support (incl. IDO)	1	0.00	0.00	6000.00
	Total CHALLENGE XL Costs:			**$2088716.00**	**$429900.00**
ORACLE Software					
	ORACLE7 (XL - 192 users)	1	$230400.00	$230400.00	$138240.00
	Procedural Option (XL - 192 users)	1	46080.00	46080.00	27648.00
	Total ORACLE Costs:			**$276480.00**	**$165888.00**
TOTAL H/W and S/W COSTS				**$2365196.00**	**$595788.00**
Discounts					
	Oracle Volume Discounts			$60825.60	$24883.20
	Total Discounts			**$60825.00**	**$24883.20**
TOTAL H/W and S/W COSTS (5 years)					**$2875275.20**
tpsB					**1786.20**
$/tpsB					**$1609.72**

Notes:

Audited by Performance Metrics, Inc. of Los Gatos, CA.

Table of Contents

Preface

Document Structure

Clause 10 of the TPC Benchmark™ B specification describes the requirements for a full disclosure report. The main body of this document is organized as follows, based upon the requirements in Clause 10:

- Each portion of the main document begins with a Clause 10 requirement in an *italic* font. It is followed by normal font text that explains how each result complied with the requirement.

- Appendix A contains the source code of the application used to implement the benchmark.

- Appendix B describes the process that defines, creates, and loads the ORACLE database. Also included are sample contents from each database tables.

- Appendix C lists the tunable operating system, and database parameters used in the benchmark test configuration.

- Appendix D contains the spreadsheet calculations used to determine the storage requirements for the ACCOUNT/-BRANCH/TELLER/HISTORY tables, eight (8) hour recovery log(s) and thirty (30) days of HISTORY.

TPC Benchmark™ B Overview

TPC Benchmark™ B was developed by the Transaction Processing Performance Council (TPC). It is the intent of the TPC to develop a suite of benchmarks to measure performance of computer systems across the spectrum of simple to complex applications. Silicon Graphics Computer Systems, Inc. is a member of the TPC.

TPC Benchmark™ B exercises the system components necessary to perform tasks associated with that class of transaction processing environments emphasizing update intensive database services. Such environments are characterized by:

- Significant disk input/output
- Moderate system and application execution time
- Transaction integrity

The benchmark is not OLTP in that it does not require any terminal, networking, or think time. This benchmark uses terminology and metrics which are similar to other benchmarks, originated by the TPC and others. The only benchmark results comparable to TPC Benchmark™ B are other TPC Benchmark™ B results. In spite of similarities to TPC-A, TPC-B contains substantial differences which make TPC-B results not comparable to TPC-A.

The metrics used in TPC Benchmark™ B are throughput as measured in transactions per second (TPS), subject to a residence time constraint, and the associated price-per-tps. The metric for this benchmark is “tpsB”. All references to tpsB results must include both the tpsB rate and the price-per-tpsB to be compliant with the TPC Benchmark™ B standard. Comparison of price/performance results disclosed in one country may not be meaningful in another country because of pricing and product differences.

This benchmark uses a single, simple, update-intensive transaction to load the system under test (SUT). Thus the workload is intended to reflect the database aspects of an application, but does not reflect the entire range of OLTP requirements typically characterized by terminal and network input/output, and by multiple transaction types of varying complexities. The single transaction type provides a simple, repeatable unit of work, and is designed to exercise the basic components of a database system.

The extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC Benchmark™ B approximates the customer application. Relative system performance of systems derived from TPC Benchmark™ B do not necessarily hold for other workloads or environments. Extrapolations to unlike environments are not recommended.

A full disclosure report of the implementation details, as specified in Clause 10, must be made available along with the reported results.

Benchmark results are highly dependent upon workload, specific application requirements, and systems design and implementation. Relative system performance will vary because of these and other factors. Therefore, TPC Benchmark™ B should not be used as a substitute for a specific customer application benchmark when critical capacity planning and/or product evaluation decisions are contemplated.

All performance data contained in this report was obtained in a rigorously controlled environment, and therefore results obtained in other operating environments may vary significantly. Silicon Graphics Computer Systems, Inc. does not warrant or represent that a user can or will achieve similar performance expressed in transactions per second (tpsB) or normalized price/performance ($K/tpsB). No warranty of system performance or price/performance is expressed or implied in this report.

Clause 2 Transaction System Properties

2.1 Transaction System Properties (ACID)

Results of the ACIDity test (specified in Clause 2) must describe how the requirements were met. If a database different from that which is measured is used for durability tests, the sponsor must include a statement that durability works on the fully loaded and fully scaled database.

The TPC Benchmark™ B Standard Specification defines a set of transaction processing system properties that a System Under Test (SUT) must support during the execution of the benchmark. Those properties are Atomicity, Consistency, Isolation and Durability (ACID). This portion of the document will define each of those properties and describe the series of tests that were performed to demonstrate that the properties were met.

All of the specified ACID tests were performed on the CHALLENGE XL Server. Except for the failure of a single durable medium, each ACID test was performed on the measured database.

The test to fail a single durable medium with table/file data was run on the smaller database scaled to seven hundred (700) tpsB.

2.2 Atomicity

The system under test must guarantee that transactions are atomic; the system will either perform all individual operations on the data, or will assure that no partially-completed operations have any effects on the data.

The following tests for atomicity were successfully completed for both regular transactions and discrete transactions.

2.2.1 Completed Transaction

Perform the standard TPC Benchmark™ B transaction (see Clause 1.2) for a randomly selected account and verify that the appropriate records have been changed in the Account, Branch, Teller, and History files/tables.

A verification of a committed transaction was completed as follows:

- a random Account and Teller were selected
- the current balances for the selected Account, Teller, and the Teller's associated Branch were recorded
- the number of rows in the History table that contain the above combination of Account, Branch, and Teller was recorded
- an interactive version of the TPC Benchmark™ B application was executed that prompts a terminal for the transaction input and allows the user the option of COMMITting or ABORTing the transaction.
- the selected Account and Teller identifiers along with a random delta amount was entered for the transaction,
- the TPC Benchmark™ B application updated the appropriate Account, Branch, and Teller balances with the above delta amount, inserted an appropriate entry in the History table and prompted the user to either COMMIT or ABORT the current transaction,
- a COMMIT request was issued from the terminal
- the TPC Benchmark™ B application COMMITted the above transaction as requested.

After the transaction was COMMITted:

- the balances from the selected Account, Branch, and Teller were displayed
- it was verified that the displayed balances differed from the original balances by the delta value that was entered,
- the number of rows in the History table for the combination of the selected Account, Branch, and Teller was displayed
- it was verified that the number of History table rows was one greater than before the above transaction was executed,
- it was verified that the additional History row contained the proper values from the transaction entered.

2.2.2 Aborted Transaction

Perform the standard TPC Benchmark™ B Transaction for a randomly selected account, substituting an ABORT of the transaction for the COMMIT of the transaction. Verify that the appropriate records have not been changed in the Account, Branch, Teller, and History files/tables.

A verification of an aborted transaction was completed as follows:

- a random Account and Teller were selected
- the current balances for the selected Account, Teller, and the Teller's associated Branch were recorded
- the number of rows in the History table that contain the above combination of Account, Branch, and Teller was recorded
- an interactive version of the TPC Benchmark™ B application was executed that prompts a terminal for the transaction input and allows the user the option of COMMITting or ABORTing the transaction,
- the selected Account and Teller identifiers along with a random delta amount was entered for the transaction,
- the TPC Benchmark™ B application updated the appropriate Account, Branch, and Teller balances with the above delta amount, inserted an appropriate entry in the History table and prompted the user to either COMMIT or ABORT the current transaction,
- an ABORT request was issued from the terminal
- the TPC Benchmark™ B application ABORTed the above transaction as requested.

After the transaction was ABORTed:

- the balances from the selected Account, Branch, and Teller were displayed
- it was verified that the displayed balances were the same as before the transaction was started
- the number of rows in the History table for the combination of the selected Account, Branch, and Teller was displayed
- it was verified that the number of History table rows was no different than before the above transaction was executed,

2.3 Consistency

Consistency is the property of the application that requires any execution of a transaction to take the database from one consistent state to another.

A consistent state for the TPC Benchmark™ B database is defined to exist when:

a) the sum of the account balances is equal to the sum of the teller balances, which is equal to the sum of the branch balances;

b) for all branches, the sum of the teller balances within a branch is equal to the branch balance;

c) the history file has one logical record added for each committed transaction, none for any aborted transaction, and the sum of the deltas in the records added to the history file equals the sum of the deltas for all committed transactions.

If data is replicated, each copy must not violate these conditions.

Due to the large size of the Account file/table, no test of its consistency is specified.

The following tests were performed on the system under test (SUT) to demonstrate the property of consistency.

Prior to executing the TPC Benchmark™ B transactions:

- the balance for each Branch occurrence in the database was recorded (Initial Branch Balances),
- the sum of the above balances of all the Branches were recorded (Initial Branch Sum),
- the sum of the Teller balances within each branch were recorded (Initial Teller/Branch Balance),
- it was verified that the Initial Branch Balance equaled the sum of the Initial Teller/Branch Balances for each Branch,
- the number of History rows and the sum of the History delta values were recorded (Initial History Count and Initial History Sum),

- the TPC Benchmark™ B applications was executed and the number of committed transactions was recorded. It was verified that the number of committed transactions was not less than ten (10) times the number of Teller occurrences.

After the TPC Benchmark™ B application was executed:

- the sum of the balances of all Branch occurrences in the database was recorded (Final Branch Sum),
- the balance for each Branch occurrence in the database was recorded (Final Branch Balances),
- for each Branch, the sum of Teller balances associated with the Branch was recorded (Final Teller/Branch Balance),
- for each Branch, it was verified that the Final Branch Balance equaled the appropriate Final Teller/Branch Balance,
- the number of History rows and the sum of History row delta values amounts were recorded (Final History Count and Final History Sum),
- it was verified that the difference between the Final History Count and Initial History Count was the number of transactions recorded as committed,
- it was verified that the difference between the Final History Sum and Initial History Sum equaled the difference between the Final Branch Sum and Initial Branch Sum.

2.4 Isolation

Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order.

This property is commonly called serializability. Sufficient conditions must be enabled at either the system or application level to ensure serializability of transactions under any mix of arbitrary transactions, not just TPC Benchmark™ B transactions. The system or application must have full serializability enabled, i.e., repeated reads of the same records within any committed transactions must have returned identical data when run concurrently with any mix of arbitrary transactions.

A total of 24 isolation tests were run; that is, both COMMITted and ABORTed transactions for the Branch, Account, and Teller tables, using regular and discrete transactions.. The following two tables show the

steps used in performing the Isolation test for the Account table with a COMMITted transaction (Table 2.1) and a ABORTed transaction (Table 2.2). The same steps were used to test both the Branch and Teller tables.

2.4.1 Completed Transaction

Table 2.1: Isolation Test — Completed Transaction

Transaction 1	Transaction 2
Execute a TPC Benchmark™ B transaction to update a randomly selected Account, using the application code described in the Atomicity tests. Stop the transaction prior to COMMIT.	
	Execute a second TPC Benchmark™ B transaction that will update the same Account as Transaction 1 using a different Teller and Branch. This transaction will wait until Transaction 1 completes.
COMMIT this transaction and verify the Account balance reflects the effect of the update.	
	This transaction resumes and is COMMITted. The Account balance reflects the effect of both Transaction 1 and Transaction 2.

2.4.2 Aborted Transaction

The aborted transaction tests (Table 2.2) follows on the next page.

Table 2.2: Isolation Test — Aborted Transaction

Transaction 1	Transaction 2
Execute a TPC Benchmark™ B transaction to update a randomly selected Account, using the application code described in the Atomicity tests. Stop the transaction prior to COMMIT.	
	Execute a second TPC Benchmark™ B transaction that will update the same Account as Transaction 1 using a different Teller and Branch. This transaction will wait until Transaction 1 completes.

Table 2.2: Isolation Test — Aborted Transaction

Transaction 1	Transaction 2
ABORT this transaction and verify the Account balance remains unchanged.	
	This transaction resumes and is COMMITted. The Account balance reflects only the effect of Transaction 2.

2.5 Durability

The tested system must guarantee the ability to preserve the effects of committed transactions and insure database consistency after recovery from any one of the failures listed below:

- *Permanent irrecoverable failure of any single durable medium containing database, ABTH files/tables, or recovery log data.*
- *Instantaneous interruption (system crash/system hang) in processing which requires system reboot to recover.*
- *Failure of all or part of memory (loss of contents).*

A durable medium is a data storage medium that is either:

a) an inherently non-volatile medium, e.g., magnetic disk, magnetic tape, optical disk, etc., or

a) a volatile medium with its own self-contained power supply that will retain and permit the transfer of data, before any data is lost, to an inherently non-volatile medium after the failure of external power.

A transaction is considered committed when the transaction manager component of the system has written the commit record(s) associated with the transaction to a durable medium.

It is required that the system crash test and the loss of memory test described in Clauses 2.5.3.2 and 2.5.3.3, respectively, be performed with a full terminal load and a fully scaled database. The durable media failure tests described in Clause 2.5.3.1 may be performed on a subset of the SUT configuration and database. For that subset, all multiple hardware components, such as processors and disk/controllers in the full configura-

tion must be represented by either 10% or 2 each of the multiple hardware components, whichever is greater. The database subset must be scaled to at least 10% (minimum of 2 tps) of the fully scaled database size. The test sponsor must state that to the best of their knowledge, a fully loaded and fully scaled SUT and database configuration would also pass all durability tests.

At the time of the induced failures, it is required o have multiple home and remote transactions (see Clause5) in progress. Distributed configurations must have distributed transactions in progress as well.

All durability tests except the durable media failure test described in 2.5.3.1, were conducted on the CHALLENGE XL Server using a fully scaled database under full load. The durable media failure test was conducted successfully on a database subset scaled greater than 10% of the fully scaled database and, to the best of our knowledge, would also complete successfully with the measured database.

The fully scaled database configuration used in the ACID tests was the same configuration used for the TPC Benchmark™ B measurements. Multiple home and remote transactions were in progress during the durability tests.

The Durability tests used the following procedure:

- determined the Initial Branch Sum, Initial Branch Balances, Initial Teller/Branch Balances, Initial History Count, and Initial History Sum, as described earlier in the Consistency test,
- ran the appropriate number TPC Benchmark™ B transactions,
- induced a failure from the list of single failures,
- determined the Final Branch Sum, Final Branch Balances, Final Teller /Branch Balance, Final History Count, and Final History Sum,
- examined the above values and relationships,
- in addition, the number of occurrences in the History tables was compared with the count of completed transactions recorded by the driver in the 'success' file. This ensured all committed transactions (History tables) are also recorded as completed in the 'success' file and none were lost because of the induced failure.

All of the Durability tests listed below completed successfully. The sum of Teller balances associated with a particular Branch equaled that Branch's balance before and after the execution of the benchmark. The difference between the Final and Initial History Counts was equal to the number of recorded committed transactions. The difference between the Final and Initial History Sum equaled the difference between the Final and Initial Branch Sum, and every record in the 'success' file had a corresponding row occurrence in the History tables.

The failures listed below were induced on the system under test (SUT) to demonstrate the property of Durability.

2.5.1 Permanent Irrecoverable Failure

Permanent irrecoverable failure of any single durable medium containing database, ABTH files/tables, or recovery log data.

Two irrecoverable failures were tested, one for failure of table and catalog medium, and another for database recovery log medium.

The table and catalog medium failure was tested as follows:

- while transactions were being processed, an additional table was created “toy-table”, and a row was inserted into it. This table was for audit purposes to prove that recovery would include any table that had been cataloged.
- a failure was induced by copying bad data over the sections of the disk that stored the database catalog and another disk containing account data. This caused appropriate error messages to appear on the console and the application to stop.
- the database was shut down,
- the backup was restored, overwriting the existing contents of the disk, and the database was rolled forward using the recovery log file,
- the count of records in the success file was compared to the rows in the History table to verify that all transactions were correctly recovered,
- random rows from the success file were searched out in the History table to verify the contents were successfully recovered
- the “toy-table” was accessed to verify recovery of the catalog data.

The recovery log medium was mirrored. Failure of the recovery log was tested as follows:

- while transactions were being processed, one of the mirrored disks was physically removed from the SUT,
- processing continued unaffected and no recovery was necessary.

2.5.2 Instantaneous Interruption

Instantaneous interruption (system crash/system hang) in processing which requires system reboot to recover.

2.5.3 Loss of Memory

Failure of all or part of memory (loss of contents).

The instantaneous interruption and loss of memory tests - which were combined because the loss of power erases the contents of memory - were conducted as follows:

- a consistency check was run and the file system was synchronized to ensure the audit files were written to disk and would not be lost,
- while transactions were being processed, a failure was induced by turning off the primary power for the SUT,
- power to the SUT was restored and ORACLE was restarted,
- the database was recovered using the log file,
- the count of records in the success file was compared to the rows in the History table to verify that all committed transactions were correctly recovered.

Clause 3 Logical Database Design

3.1 Database Design

The distribution across storage media of ABTH (Accounts, Branch, Teller, and History) files/tables and all logs must be explicitly depicted.

3.1.1 Distribution and Partitioning

This benchmark was implemented as a centralized solution accessing a single logical and physical database. The account and history tables were horizontally partitioned. The partitioning was completely transparent to the application.

The benchmark and priced system configuration diagrams are shown in Figures 3.1 and 3.2, respectively. The system's configurations are essentially the same, the only difference being the fact that the SUT utilized 148 of the 166 disk drives in its configuration. The database was originally built for 2000 branches. The test was executed against an 1800 branch subset of this database. The remaining 200 branches were isolated on the remaining 18 disk drives. These drives were totally inactive during the measurement period and therefore not considered to be part of the priced configuration. A copy of the system activate report which was generated during steady state (but before opening the measurement window), as well as a copy of the ORACLE statistics file generated during the entire run, have been included in Appendix F.

The specific distribution of database tables (Account, Branch, Teller, and History) and recovery log data across storage media for both the benchmark and priced configuration are shown in Tables 3.1 and 3.2, respectively. The same allocations were used for both the benchmark and the priced configuration. The only difference is the amount of data generated for History and Log files during the benchmark did not completely fill all allocated space.

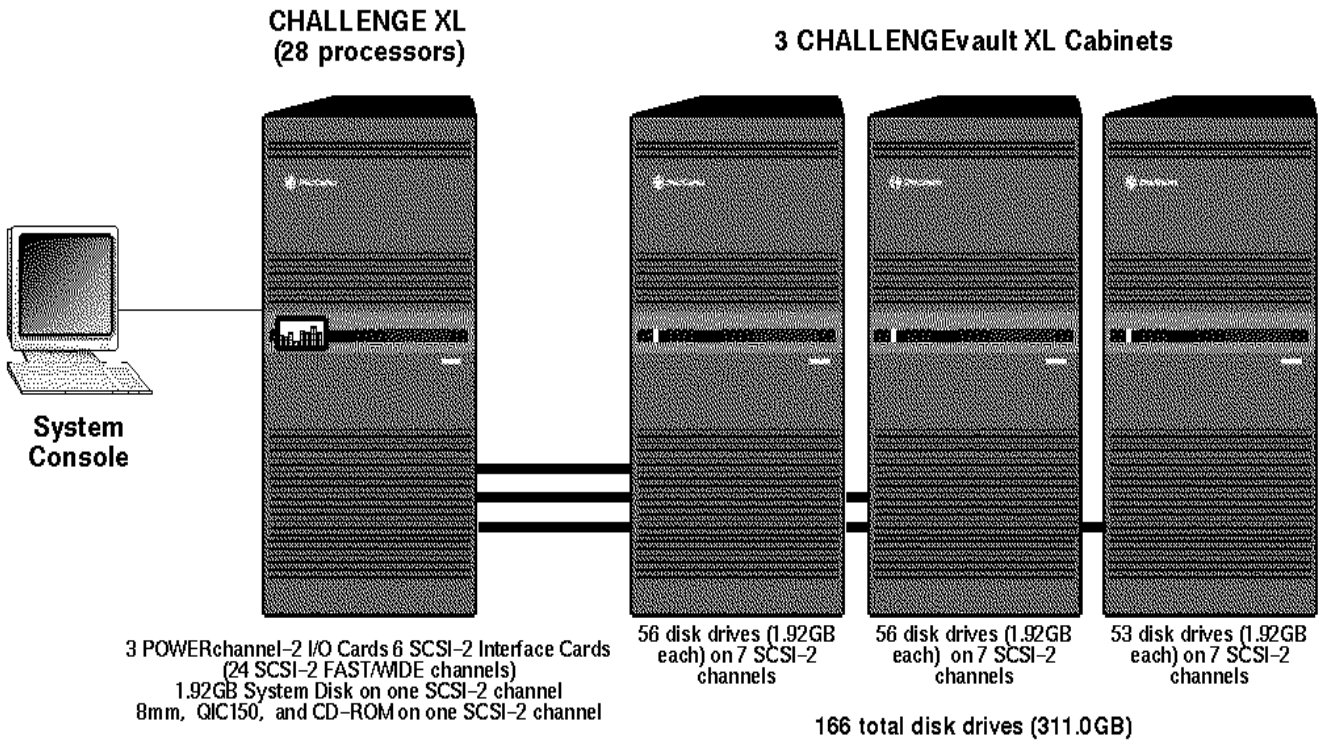


Figure 3.1 SUT System Configurations

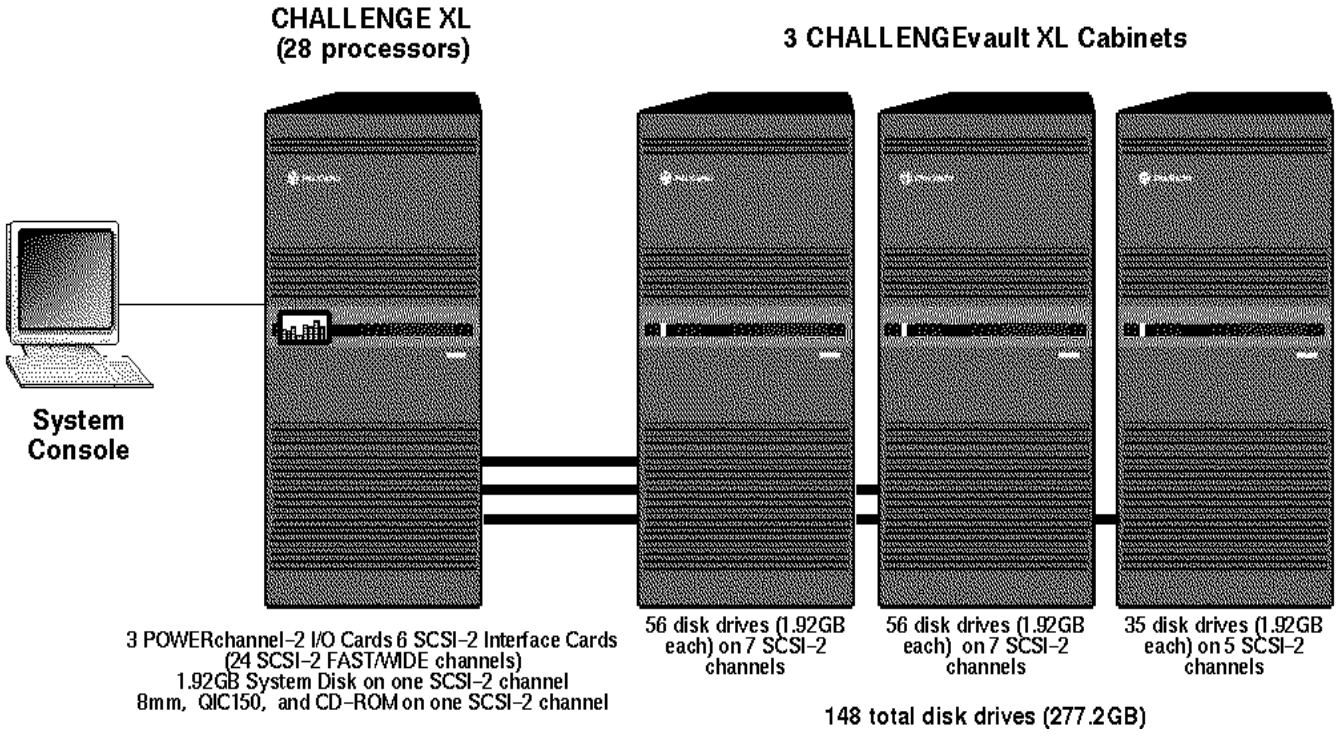


Figure 3.2 Priced System Configurations

Distribution of Data:		UNIX + swap	ORACLE System & Control Files	Branch & Teller Data	Account Data	Account Index	History Data	Log Data	Log Data (Mirror)
Disk Name(s)	Total # of Drives	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk
dks1d1	1	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
dks2d[2-6]	5	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
dks3d[134567]	6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks3d2	1	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks3d8	1	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
dks4d[1-4]	4	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
dks4d[78]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks5d[12]	2	0.00%	0.00%	0.00%	0.00%	33.33%	0.00%	0.00%	0.00%
dks5d[6-8]	3	0.00%	0.00%	0.00%	0.00%	0.00%	7.69%	0.00%	0.00%
dks6d[1-8]	8	0.00%	0.00%	0.00%	0.00%	0.00%	7.69%	0.00%	0.00%
dks7d[12]	2	0.00%	0.00%	0.00%	0.00%	0.00%	7.69%	0.00%	0.00%
dks7d[4578]	4	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks7[13]d[1-8]	16	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks72d[234578]	6	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks74d[1-7]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks74d8	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks75d[15]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks75d[234678]	6	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks76d[1245678]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks76d3	1	0.00%	0.00%	0.00%	0.00%	33.33%	0.00%	0.00%	0.00%
dks77d[1-6]	6	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks110d[1235678]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks11[14]d[1-8]	16	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks112d[1234568]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks113d[1245678]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks115d[1234578]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks116d[24578]	5	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks117d[1-5]	5	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%
dks117d[67]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
dks117d8	1	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	9.09%	0.00%
Total	148								

Table 3.1: SUT Data Distribution

Distribution of Data:		UNIX + swap	ORACLE System & Control Files	Branch & Teller Data	Account Data	Account Index	History Data	Log Data	Log Data (Mirror)
Disk Name(s)	Total # of Drives	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk	% of data/disk
dks1d1	1	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
dks2d[2-6]	5	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
dks3d[134567]	6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks3d2	1	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks3d8	1	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
dks4d[1-4]	4	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
dks4d[78]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks5d[12]	2	0.00%	0.00%	0.00%	0.00%	33.33%	0.00%	0.00%	0.00%
dks5d[6-8]	3	0.00%	0.00%	0.00%	0.00%	0.00%	0.83%	0.00%	0.00%
dks6d[1-8]	8	0.00%	0.00%	0.00%	0.00%	0.00%	0.83%	0.00%	0.00%
dks7d[12]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.83%	0.00%	0.00%
dks7d[4578]	4	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks7[13]d[1-8]	16	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks72d[234578]	6	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks74d[1-7]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks74d8	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks75d[15]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%
dks75d[234678]	6	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks76d[1245678]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks76d3	1	0.00%	0.00%	0.00%	0.00%	33.33%	0.00%	0.00%	0.00%
dks77d[1-6]	6	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks110d[1235678]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks11[14]d[1-8]	16	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks112d[1234568]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks113d[1245678]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks115d[1234578]	7	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks116d[24578]	5	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks117d[1-5]	5	0.00%	0.00%	0.00%	0.93%	0.00%	0.83%	0.00%	0.00%
dks117d[67]	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
dks117d8	1	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	9.09%	0.00%
Total	148								

Table 3.2: Priced Configuration Data Distribution

3.1.2 Population and Sample Contents

A description of how the database was populated, along with sample contents of each ABTH file/table to meet the requirements described in Clause 3.

Appendix B shows the processes that defined, created, and populated the ORACLE7 on-line database for TPC Benchmark™ B. Sample contents of each database table are included in this appendix.

3.1.3 Type of Database

A statement of the type of database utilized, e.g., relational, Codasyl, flat file, etc.

This TPC Benchmark™ B test used the ORACLE7 RDBMS relational database software.

Clause 4 Scaling Rules

4.1 Clause 4 Related Items

There are no Clause 4 Related Items required by the Full Disclosure specification. However, Clause 4 specifies scaling rules and that information is provided here as the appropriate place to describe the database size and scaling information.

4.1.1 Database Scaling, and Row Occurrences

The database was populated with the required number of row occurrences for the Account, Branch, and Teller tables to measure for 2000.00 tpsB. These numbers are listed in Table 4.1

The specific code used to create and populate these tables may be found in Appendix B. Details of the space calculated for the History table and log files may be found in Appendix D.

Table 4.1: CHALLENGE XL Server and ORACLE7 Required Row Occurrences

Table	Occurrences
Branch	2,000
Teller	20,000
Account	200,000,000

Clause 5 Distribution, Partitioning, and Transaction Generation

5.1 Random Number Generator

The method of verification of the random number generator should be described.

The UNIX functions of RAND and LRAND48 were used to generate pseudo-random numbers. The SRAND and SRAND48 functions were used to seed the random number generators using the process identification number multiplied by the result of the gettime() function.

The RAND generator was used to create the random numbers used for the Teller identifier number for both local and remote transactions.

The LRAND48 generator was used to create the Account identifiers, and the delta amounts used in the TPC Benchmark™ B transaction.

Both of these routines generate pseudo-random numbers using a well-known linear congruential algorithm.

The code from the driver program that accomplished this is shown below in Figure 5.1

In addition, the History and success files were randomly searched by the auditors for duplicates and/or patterns that would indicate the random number generator had effected any kind of discernible pattern. None were found.

```
/*
** Seed Random Number Generator.
*/
srand48(getpid() * gettime());
srand(getpid() * gettime());

/*
** Execute transactions until time is up.
*/

while(TRUE)
{

    /* Pick random amount in range -999999 to 999999: Clause 5.3.6 */
    amount = (lrand48() % 1999999) - 999999;

    /*
    ** Clause 5.3.3: For single-node systems, choose a teller at
    ** random from the entire range of tellers.
    */
    if (nhosts > 1)
       teller_no = (rand () % ((db_multiplier / nhosts) * 10)) +
     (hid - 1) * (db_multiplier / nhosts) * 10;
    else
       teller_no = (rand() % tellnum) + 1;

    /*
    ** Clause 5.3.4: "Given the randomly chosen teller...the
    ** corresponding branch is determined"
    ** There are 10 tellers per branch. First teller is 1.
    */
    branch_no = ((teller_no - 1)/10) + 1;

    /*
    ** Clause 5.3.5: Account ID Generation
    **    85% of the time, randomly choose an account from the
    **    local branch. 15% of the time, randomly choose an
    **    account from one of the other branches.
    */

    if (brannum > 1)
    {
     if (rand()%100 < 85)
     account_branch = branch_no;
     else
         do
            account_branch = (rand()%brannum) + 1;
         while (account_branch == branch_no);
    }
    else
     account_branch = branch_no;

    /* There are 100,000 accounts per branch; */
    account_no = 100000*(account_branch - 1) + (lrand48()%100000) + 1;
```

Figure 5.1: Random Number Generators

5.2 Horizontal Partitioning

Vendors must clearly disclose if horizontal partitioning is used. Specifically, vendors must satisfy the following:

1. *Describe textually the extent of transparency of the implementation.*

2. *Describe which tables / files were accessed using partitioning.*

3. *Describe how partitioned tables / files were accessed.*

The Account and History tables were horizontally partitioned. The partitioning was completely transparent to the application. The complete description of the physical positioning of the tables may be found in Chapter 3 under **Database Design,** and **Distribution and Partitioning.**

Clause 6 Residence Time

6.1 Benchmark Performance

Report all the data specified in Clause 6, including measured and reported tpsB, maximum and average residence time, as well as performance curves for number of transactions vs. residence time (see clause 6.6.1) and throughput vs. level of concurrency for three data points (see clause 6.6.5). Also, the sponsor must include the percentage of home and remote transactions, the number and percentage of in-process transactions, and the percentage of remote and foreign transactions, if applicable.

Table 6.1 contains the statistics required by the above clause.

Table 6.1: CHALLENGE XL Server and ORACLE7 Performance Statistics

Measured tpsB	1786.20	tpsB
Reported tpsB	1786.20	tpsB
90th percentile Residence time	<0.25	seconds
Maximum Residence Time	9.07	seconds
Average Residence Time	0.091	seconds
Percent of Home transactions	85.03	%
Percent of Remote transactions	14.97	%
Measured completed transactions	4501286	
Number of in flight transactions	146	
Percentage of in flight transactions	0.003	%
Concurrency at reported tpsB (C_R)	162.54	
Low concurrency (C_L)	122.61	
High concurrency (C_H)	199.73	

6.1.1 Throughput (tpsB) vs. Residence Time

The distribution of throughput (tpsB) vs. residence times for the transactions in the benchmark test are shown below in Figure 6.1. The average and 90th percentile residence times are noted on the graph.

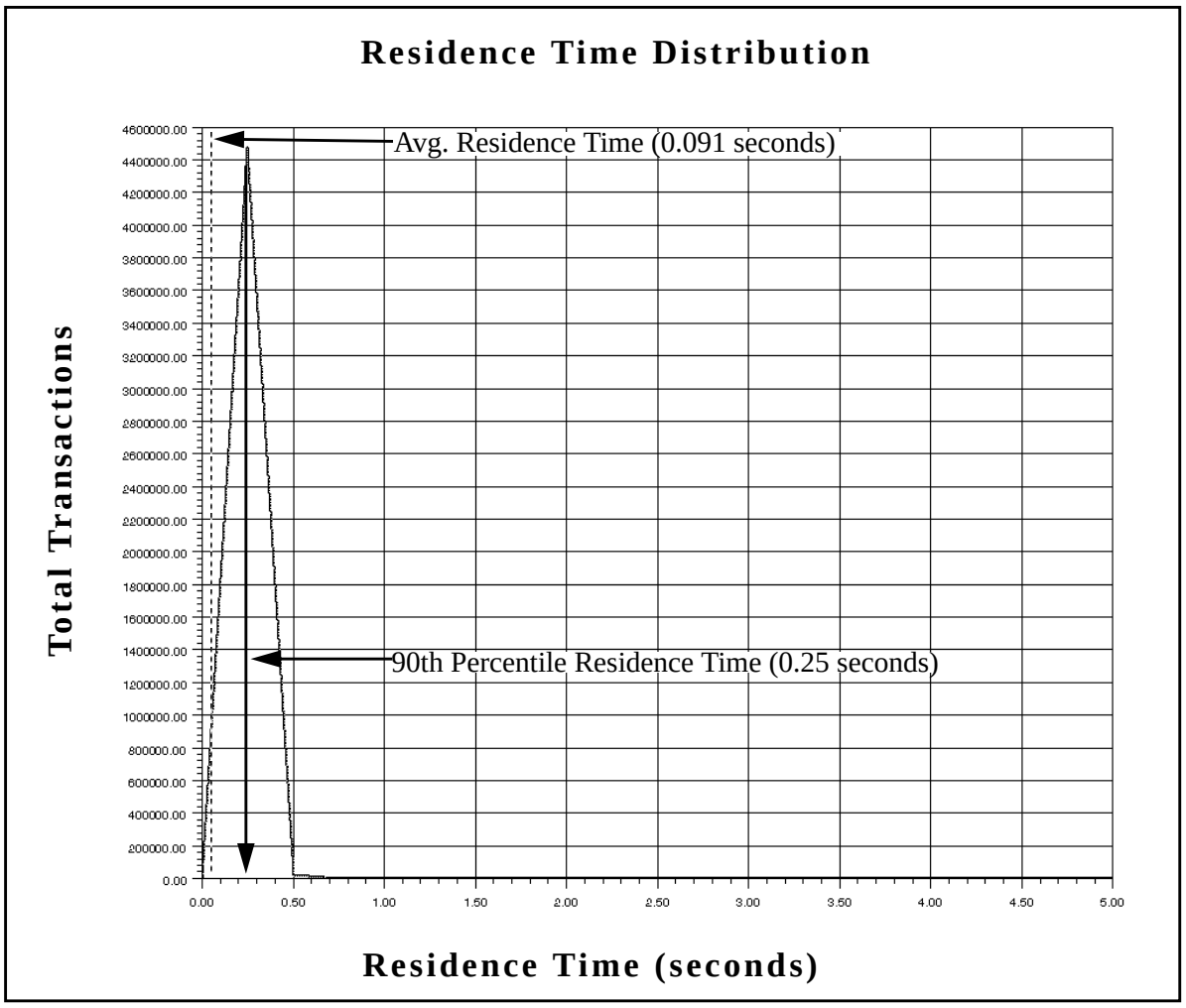


Figure 6.1: CHALLENGE XL Server and ORACLE7 Residence Times

6.1.2 Throughput (tpsB) vs. Concurrency

The tpsB vs. concurrency graph is shown in Figure 6.2. In the graph C_R denotes the concurrency at the reported rate while C_L and C_H denote low and high concurrency levels, respectively. These concurrency levels are defined as follows:

C_R = reported tpsB * average residence time

.7 C_R <= C_L <= .8 C_R and C_H >= 1.2 C_Rt

	Drivers	Average Residence Time	tpsB	Concurrency Level	Concurrency Level (% of C_R)
C_R	166	.091	1786.20	162.54	100.00
C_L	124	.074	1656.99	122.61	75.43
C_H	205	.121	1650.65	199.73	122.88

Table 6.2: Concurrency Computations

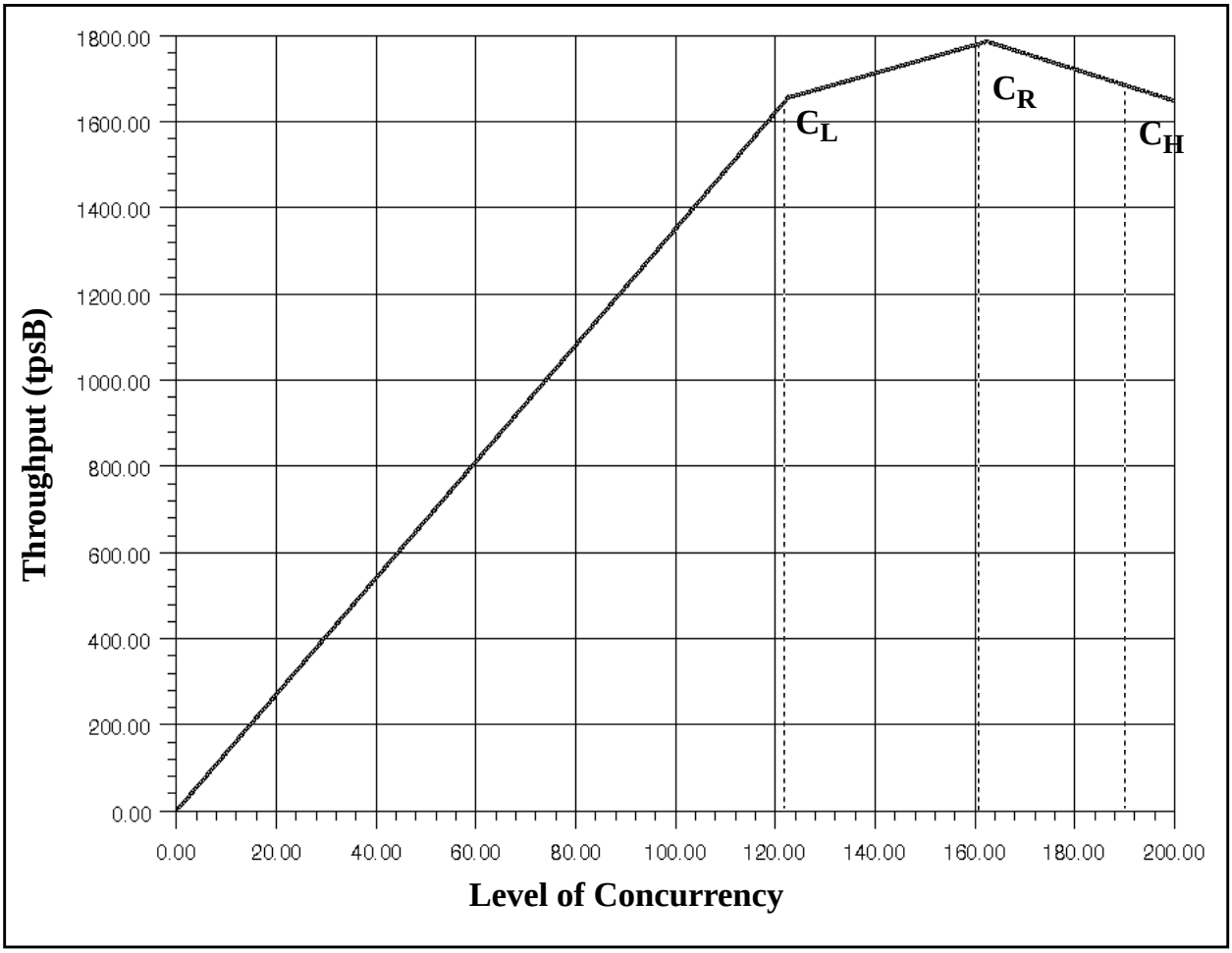


Figure 6.2: tpsB vs. Concurrency Level

Clause 7 Duration of Test

7.1 Steady State

The method used to determine that the SUT had reached a steady state prior to commencing the measure interval should be described.

The transaction throughput rate (tpsB) was measured during trial runs to determine the average time required to start all processes and begin a sustained rate of throughput. This ramp up interval was also verified by performance monitoring information. The ramp up interval of fifteen (15) minutes was sent as a parameter to the benchmark application to assure that the measured interval was started after a steady state was established.

7.2 Work Performed During Steady State

A description of how the work normally performed during a sustained test (for example checkpointing, writing redo/undo log records, etc. as required by Clause 7.2), actually occurred during the measurement interval.

During the measurement interval, the ORACLE7 RDBMS reads one account block into the buffer cache for every transaction. On average, one modified account block was written from the shared buffer cache for every transaction, but this write was only necessary to free space in the shared buffer cache, not to commit the transaction. Modified database buffers migrated to disk on a “least recently used” basis independent of transaction commits. In addition, every block modification was protected by redo log records. These redo log records were written to the redo log buffer (in memory), which were flushed to a redo log file on disk either when the transaction committed or when the redo log buffer became full.

During a checkpoint, all modified blocks in the shared buffer cache which had not been written to disk since the last checkpoint were physically written to disk. A single checkpoint was performed during the measurement interval.

The performance of the TPC Benchmark™ B transaction was improved by using the BEGIN_DISCRETE_TRANSACTION procedure (See Appendix A). This procedure streamlines transaction processing so that short, non-distributed transactions can execute more rapidly.

During a discrete transaction, all changes made to any data were deferred until the transaction committed. Redo information was generated, but was stored in a separate location in memory. When the transaction issued a commit request, the redo information was written to the redo log file (along with other group commits) and the changes to the database block were applied directly to the block. Once the commit completed, control was then returned to the application.

Notice the loop construct in the transaction profile included in Appendix A. The TPC-B transaction was implemented as a discrete transaction by calling the BEGIN_DISCRETE_TRANSACTION procedure before the first statement. Any error encountered during the processing of discrete transactions caused the pre-defined exception DISCRETE_TRANSACTION_FAILED to be raised. If this exception occurred, the TPC-B transaction was rolled back and re-executed as a normal transaction.

The discrete transaction is a fully documented performance feature in the ORACLE7 DBA Guide. An ORACLE trigger and package were used to retrieve the account balance. This is also documented in the ORACLE7 DBA Guide.

7.3 Reproducibility

A description of the method used to determine the reproducibility of the measurement results.

The benchmark was executed multiple times and the reported throughput (tpsB) and residence time varied less than 2.65 percent between the measured runs.

7.4 Measurement Period Duration

A statement of the duration of the measurement period for the reported tps (it should be at least 15 minutes and no longer than 1 hour).

Each measured run was executed for a total of 58 minutes. This included 15 minutes of ramp up time and 42 minutes of steady state. The measurement interval of 42 minutes was chosen such that a single checkpoint would occur approximately 37 minutes into the run as a result of a log file switch For the measured run the checkpoint interval was 42:45. The graph demonstrating steady state is shown below.

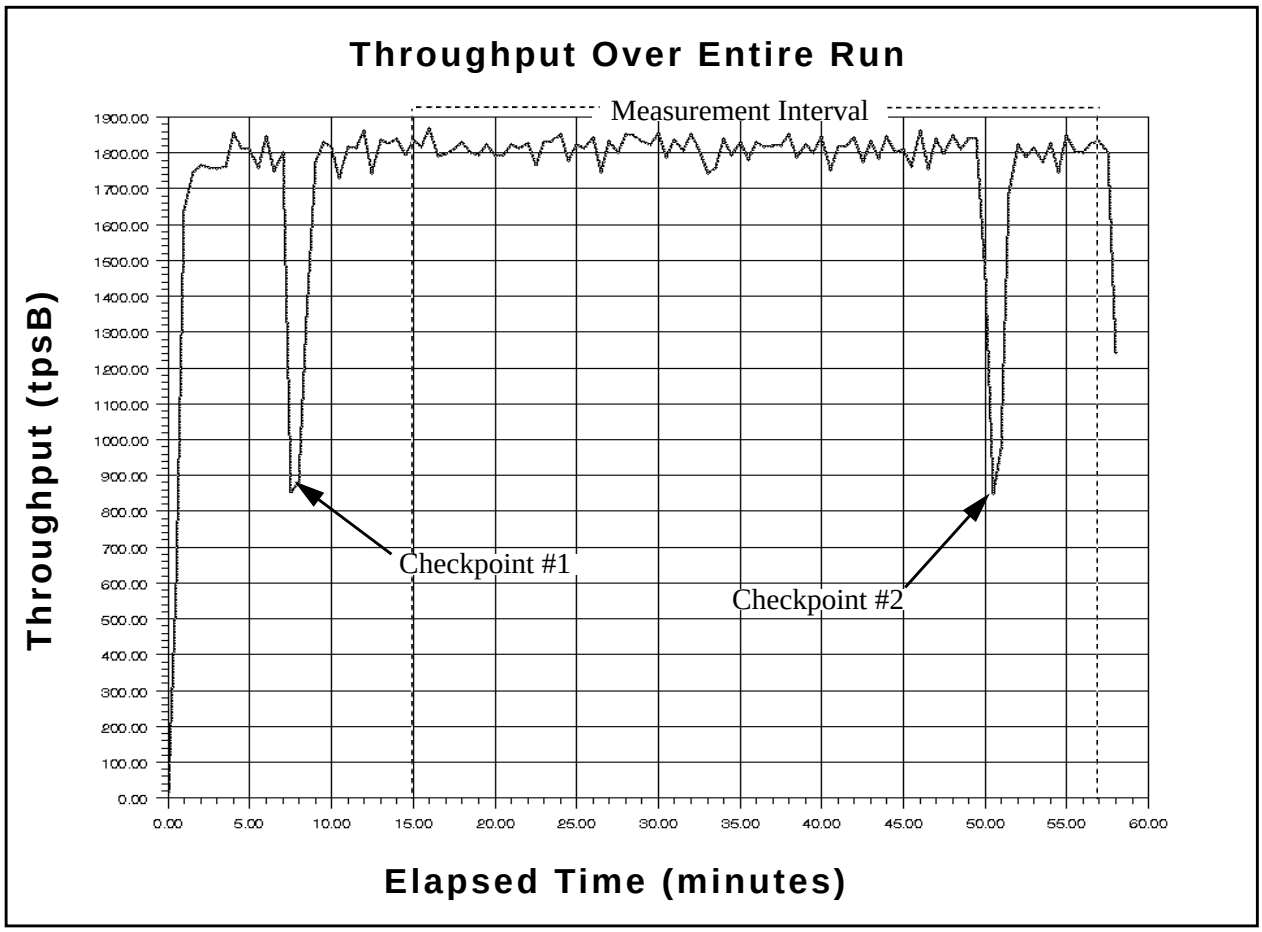


Figure 7.1: CHALLENGE XL Server and ORACLE7 Throughput tpsB

Clause 8 SUT Driver Definition

8.1 Models of the Target System

There are two forms of drivers which present transactions to the SUT: an internal driver which resides on the SUT hardware and software. An external driver which resides on a separate hardware and software complex, and typically communicates with the SUT via a communications network using a client/server remote procedure call mechanism.

The driver resided on the SUT.

8.2 Driver Components

A proof that the functionality and performance of the components being emulated in the Driver System are equivalent to that of the priced system. The sponsor must list all hardware and software functionality of the driver and its interface to the SUT.

The Driver System provides the following functionality:

- generates the input data,
- timestamps the delivery,
- receives and timestamps the response,
- generates the success file, and
- performs the necessary accounting of the residence times.

The driver initializes the required data structures and then, for each transaction executed:

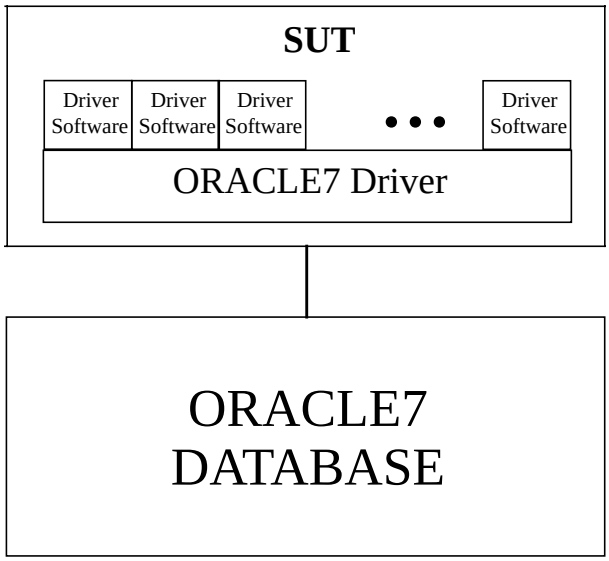
- generates the ABTH input values,

- timestamps the transaction,
- executes the TPC Benchmark™ B transaction as specified in Clause 1.2, and
- records the transaction residency time and other pertinent statistics.

The statistics report by the driver include:

- the number of transactions submitted,
- the minimum, maximum, and average residence times for all submitted transactions,
- a distribution of residence times in 0.25 second intervals, and
- total system throughput in 30 second intervals.

A functional diagram of the SUT is given below in figure 8.1.



Clause 9 Pricing

9.1 System Pricing

A detailed list of hardware and software used in the priced system. Each item must have vendor part number, description, and release/revision level, and either general availability status or committed delivery data. If package-pricing is used, contents of the package must be disclosed.

The total price of the entire configuration is required including: hardware, software, and maintenance charges. Separate component pricing is recommended. The basis of all discounts used shall be disclosed.

A statement of the measured tpsB, and the calculated price/tpsB.

9.1.1 CHALLENGE XL Server

The CHALLENGE XL Server system consists of:

- 28 R4400SC CPUs with 4MB of combined secondary cache each,
- 2GB of main memory,
- 3 POWERchannel-2 I/O boards with 6 additional SCSI-2 cards for a total of 24 FAST/WIDE SCSI-2 channels,
- 148 disk drives of 1.92GB formatted capacity each,
- 5GB 8mm tape, QIC-150 cartridge tape, and CD-ROM.

The priced configuration contains several line items which are bundled products. The specific contents of these packages are details in Table 9.2.

9.2 Support Pricing

The five year support pricing for CHALLENGE XL Server:

$2,518,616.00

The five year support pricing for ORACLE7:

$356,659.20

9.3 Availability

The CHALLENGE XL Server used in the benchmark and identified by Order Number R-45828-S2 is orderable now and will be deliverable on September 30, 1994.

The IRIX (UNIX SVR4) operating system.which was used on the SUT,, 5.1DC, is a pre-released version of IRIX HEP-1.0 which will be released on June 30, 1994.

ORACLE7 Version 7.0.15.4.1 will be available on March 30, 1994.

All other products specified in the priced configuration are current orderable and deliverable.

9.4 Priced System Configuration

The hardware, software, and support/maintenance products priced in this benchmark, are detailed below in Table 9.1. Also included in the table are the measured tpsB and calculated price/tpsB.

9.5 Priced Storage Requirements

The hardware necessary to meet the storage requirements of Clause 9.2.4 was calculated based upon the number of history records stored per page and the measured transaction rate. The log file storage required was calculated based statistics which are generated by ORACLE for each run. Details of the files that these calculations are based on are included in Appendix D.

Order Number	Description	Quantity	Unit Price	Extended Price	Support (5 years)
CHALLENGE XL Server					
R-45828-S4	28-cpu CHALLENGE XL Server	1	$919900.00	$919900.00	$262450.00
FTO-64UP2GB	First 2GB High Density Memory	1	212576.00	212576.00	65225.00
SD8-S-2	2GB SCSI-2 FAST/WIDE System Disk	1	8900.00	8900.00	825.00
P-S-B224	CHALLENGEvault XL 224GB Disk Bundle	1	560000.00	560000.00	46200.00
P-S-B64	CHALLENGEvault XL 64GB Disk Bundle	1	296990.00	296990.00	26400.00
P8-S-2	2GB SCSI-2 FAST/WIDE Disk	4	8900.00	35600.00	3300.00
P-S-SBX2-X	SCSIBOX2 for CHALLENGEvault XL	1	3750.00	3750.00	650.00
HU-PC2	POWERChannel-2 I/O Controller	2	12000.00	24000.00	8500.00
P-S-HIO	SCSI-2 FAST/WIDE Interface Card	6	2500.00	15000.00	5400.00
P8-QIC-CD	150MB QIC tape & CD-ROM	1	2000.00	2000.00	1700.00
P8-T4V2	5GB 8mm Internal Drive	1	7300.00	7300.00	2650.00
P-TER2	110 VAC Programming Terminal	1	1500.00	1500.00	600.00
DK-C2-001	Destination Kit for XL Series	1	0.00	0.00	0.00
DK-T2-001	Destination Kit for CHALLENGEvault XL	3	0.00	0.00	0.00
SC4-HEP-1.0	Operating System Software and Manuals	1	0.00	0.00	0.00
SC4-IDO-5.1	IRIX development option for IRIX 5.1	1	1200.00	1200.00	0.00
CS-SWCARE-DEV	Software options support (incl. IDO)	1	0.00	0.00	6000.00
	Total CHALLENGE XL Costs:			**$2088716.00**	**$429900.00**
ORACLE Software					
	ORACLE7 (XL - 192 users)	1	$230400.00	230400.00	$138240.00
	Procedural Option (XL - 192 users)	1	46080.00	46080.00	27648.00
	Total ORACLE Costs:			**$276480.00**	**$165888.00**
TOTAL H/W and S/W COSTS				**$2365196.00**	**$595788.00**
Discounts					
	Oracle Volume Discounts			$60825.60	$24883.20
	Total Discounts			**$60825.00**	**$24883.20**
TOTAL H/W and S/W COSTS (5 years)					**$2875275.20**
tpsB					**1786.20**
$/tpsB					**$1609.72**

Table 9.1: Priced Configuration

Order Number	Quantity	Description
R-45828-S4	1	CHALLENGE XL rack chassis
	28	150 MHz MIPS R4400SC CPUs
	4	MB combined sceondary cache per CPU
	1	64MB memory board (replaced by FTO-64UP2GB)
	1	POWERChannel-2 I/O board (includes 2 SCSI-2 FAST/WIDE channels, 1 ethernet channel, 3 RS-232C ports, 1 parallel port, and 1 RS-422 port)
	1	SCSIBOX-2 disk tray (Order Number P-S-SBX2-X)
P-S-B224	2	CHALLENGEvault XL (Order Number P-S-VXL)
	14	SCSIBOX-2 disk trays
	112	2GB SCSI-2 FAST/WIDE disks (Order Number P8-S-2)
P-S-B64	1	CHALLENGEvault XL
	4	SCSIBOX-2 disk trays
	32	2GB SCSI-2 FAST/WIDE disks

Table 9.2: Bundled Item Descriptions

Clause 10 Full Disclosure Checklist

10.1 General Item

A statement verifying the sponsor of the benchmark and any other companies who have participated.

The benchmark is being sponsored by Silicon Graphics Computer Systems, the hardware vendor, and ORACLE Corporation, the supplier of the database management system used.

Program listing of application code and definition language statements for file/tables.

Appendix A contains a listing of the application (driver) programs which were written in the "C" language. Appendix B contains the "C" source code and SQL scripts which were used to create and load the benchmark database.

Settings for all customer-tunable parameters and options which have been changed from defaults found in actual products; including but not limited to: Database options; Recovery/Commit options; Consistency/-Locking options; System parameters; application parameters, and configuration parameters. Test sponsors may optionally provide a full list of all parameters and options.

A listing of all modified operating system parameters and all database parameters configured during the benchmark is given in Appendix C.

Configuration diagrams of both the benchmark configuration and the priced system, and a description of the differences.

A diagram of the SUT is given in Chapter 8 and a diagram of the priced configuration in Chapter 9.

10.2 Clause 3 Related Items

Results of the ACIDity tests must describe how the requirements were met. If a database different from that which is measured is used for durability tests, the sponsor must include a statement that durability works on the fully loaded and fully scaled system.

The ACIDity tests performed are described in Chapter 2.

The distribution across storage media of ABTH files/tables and all logs must be explicitly depicted.

10.3 Clause 4 Related Items

Provide two functional diagrams which show CPUs, storage devices, and the interconnections between these components. The first diagram must correspond to the benchmark configuration and the second diagram must correspond to the 30-day priced configuration. A separate pair of diagrams must be provided for each reported result.

As part of each diagram, show the percentage of the total physical database which resides on each storage device for each of the ABTH files and logs. For the benchmark configuration, show the allocation during the 8-hour steady state. For the 30-day priced configuration, show database allocation including storage of 30 days of history records. Data which are duplicated on more than one device must be clearly labeled to show what is duplicated and on which device.

The distribution of the ABTH files/tables, log, and system files is depicted in Chapter 3. A diagram of the SUT is given in Chapter 8 and a diagram of the priced configuration is given in Chapter 9.

A description of how the database was populated, along with sample contents of each ABTH file/table to meet the requirements described in Clause 3.

Chapter 3 contains the details of the Logical Database Design. Samples of the ABTH file contents are shown in Appendix B.

A statement of the type of database utilized.

The benchmark was conducted using ORACLE7, a standard relational database management system which is a product of ORACLE Corporation.

10.4 Clause 5 Related Items

The method of verification of the random number generator should be described.

The random number generator used is described in Chapter 5.

10.5 Clause 6 Related Items

Report all the data specified in Clause 6.6, including maximum and average residence time, as well as performance curves for numbers of transactions versus residence time and throughput versus level of concurrency for three data points. Also, the sponsor must include the percentage of home and remote transactions, the number and percentage of in-process transactions, and the percentage of remote and foreign transactions, if applicable.

10.6 Clause 7 Related Items

Residency data, including the required graphs, are described in Chapter 6. The graph of total system throughput is given in Chapter 7.

The method used to determine that the SUT had reached steady state prior to commencing the measurement interval should be described.

That the SUT had achieved steady state was determined by observing the transaction processing rate at 30 second intervals.

A description of how the work normally performed during a sustained test actually occurred during the measurement interval.

The description of all work performed, including checkpoints, is detailed in Chapter 7.

A description of the method used to determine the reproducibility of the measurement results.

The benchmark result was reproduced with a variance of less than 2.65%.

A statement of the duration of the measurement period for the reported tpsB.

The measurement period was 42 minutes.

If the driver is commercially available, then its inputs should be specified. Otherwise, a description of the driver should be supplied.

The driver used in the benchmark is proprietary to ORACLE Corporation. It resided on the SUT and is described in Chapter 8.

A complete functional diagram of the hardware and software of the benchmark configuration including the driver must be provided. The sponsor must list all hardware and software functionality of the driver and its interface to the SUT.

Chapter 8 contains a functional diagram of the SUT.

10.7 Clause 9 Related Items

A detailed list of hardware and software used in the priced system. Each item must have vendor part number, description, and released/revision level, and either general availability status or committed delivery date. If package-pricing is used, contents of the package must be disclosed.

The total price of the entire configuration is required including: hardware, software, and maintenance changes. Separate component pricing is recommended. The basis of all discounts shall be disclosed.

All pricing information is contained in Chapter 9.

A statement of the measured tpsB, and the calculated price/tpsB.

The CHALLENGE XL Server was measured at 1786.20 tpsB at a price of $1609.72/tpsB.

Additional Clause 9 related items may be included in the full disclosure report for each country specific priced configuration.

This report contains only information specific to the United States of America.

Clause 11 Related Items

11.1 Independent Auditing

If the benchmark has been independently audited, then the auditor's name, address, phone number and a brief audit summary report indicating compliance must be included in the full disclosure report. A statement should be included, specifying when the complete audit report will become available and whom to contact in order to obtain a copy.

The Silicon Graphics Computer Systems CHALLENGE XL Server and ORACLE7 benchmark was independently audited by Performance Metrics, Inc. of Los Gatos, CA The attestation letter is included in Appendix E. A copy of the auditor's report can be obtained from Performance Metrics, Inc.

Appendix A
Application Source Code

Driver Applications

Silicon Graphics Computer Systems' implementation of the TPC Benchmark™ B consists of C programs that provide both driver and transaction functions. The following listings are those C programs used for these functions.

b_drv.c:

```
/*=================================================================+
|          Copyright (c) 1991  Oracle Corp, Belmont, CA           |
|                   UNIX PERFORMANCE GROUP                        |
|                    All Rights Reserved                          |
 +=================================================================+
| FILENAME
|    b_drv.c
| DESCRIPTION
|    TPC-B benchmark process, OCI-PL/SQL version
 +=================================================================*/

#ifndef FALSE
# define FALSE 0
#endif
#ifndef TRUE
# define TRUE 1
#endif

#include <stdio.h>
#include <math.h>

void waitfortrigger();
static void usage();

/*
** Global variables.
*/
long    branch_no;  /* Branch id.  Range: 1 to (1       * database scaling) */
long    teller_no;  /* Teller id.  Range: 1 to (10      * database scaling) */
long    account_no; /* Account id. Range: 1 to (100,000 * database scaling) */

long    amount;     /* Amount added to the balance */
                    /* Clause 5.3.6: "The Delta amount field is a random */
                    /* value within [-999999, +999999]" */

double  balance;      /* New balance of the account record */
                      /* Clause 3.2.2: "Must be capable of representing */
                      /* at least 10 significant decimal digits plus sign" */

char *  uid = "tpcab/tpcab";  /* Database user name and password */
```

```
int     retries = 0;  /* Discrete mode only: Number of retries. */

/*
**  Function declarations.
*/

  /* TPC-A/B transaction functions */
extern int TPCinit();
extern int TPCexec();
extern int TPCexit();

  /* Durability test success file functions */
extern int succinit();
extern int succlog();
extern int succend();

/*
** Clause 6.3: Residence Time Constraint:
** "90% of all transactions started and completed during the measurement
** interval must have a Residence Time of less than 2 seconds."
*/
#define TPCB_FAST 2.0

main(argc, argv)
   int    argc;
   char * argv[];
{

   int i;

   /*
   **  String for Audit Re-compile.
   */
   char * audit_str = "halloween";

   /*
   **  Command-line arguments.
   */
   char    config[20];         /* Configuration identifier    */
   char    runname[15];        /* Arbitrary run name.         */
   int     timelimit;          /* Duration of run in seconds. */
   int     nproc;              /* Total number of processes.  */
   int     proc_no;            /* Process number */
   double  starttime;/* Actual Starttime with trig  */
   int     nhosts;             /* Total number of hosts/nodes */
   int     hid;               /* host ID (1 .. nhosts)       */
   int     ramp_up   = 0;      /* Ramp-up time in seconds.    */
   int     ramp_down = 0;      /* Ramp-down time in seconds.  */
   int     db_multiplier = 1;  /* Database scaling: 1 =       */
                          /*  1 branch, 10 tellers, 100,000 accounts */
   /*
   **  Number of account, teller, and branch entries.
   **  Defaults are for 1 TPS database.
   */

   long    accnum  = 100000L;    /* 100,000 accounts per tps  */
   long    tellnum = 10L;        /* 10 tellers per tps        */
   long    brannum = 1L;         /* 1 branch per tps          */

   /*
   **  Timing.
   */
   double  begin_time;         /* Start time for run. */
   double  end_time;           /* End time for run. */
   double  interval_end_time;  /* End of measurement interval. */
   double  end_stat_time;      /* End time before ramp-down. */
   int     in_timing_interval = FALSE;
   int     in_ramp_up         = TRUE;

   /*
   **  Statistics.
   */
```

```
    long    tr_count;          /* Total # of transactions.      */
    long    tr_fast;           /* Number of transactions with residence */
    double  begin_cpu;         /* Initial cpu usage.  */
    double  end_cpu;           /* Final   cpu usage.  */
                        /* time < TPCB_FAST */

    /*
    **  TPC-B Clause 6.6: Required reporting.  See TPC-B specification.
    */

    /*
    **  Clause 6.6.1: frequency distribution of residence times
    **  "The range of the X axis must be from 0 to 5 seconds residence
    **  time.  At least 20 equal non-overlapping intervals must be reported.
    **  The maximum and average residence times must also be reported.~
    */
#define NUM_TIMING_BUCKETS 20
#define MAX_SECS           5.0
#define SECONDS_PER_BUCKET ((MAX_SECS)/(NUM_TIMING_BUCKETS))
    /*
    **  Add an extra bucket to the array to hold transactions with
    **  residence time > MAX_SECS.
    */
    int     timing_buckets[NUM_TIMING_BUCKETS+1];
    double  tr_max;            /* Longest transaction time.     */
    double  tr_overhead;       /* Overhead of time function.    */
    double  tr_begin, tr_end;   /* begin and end time           */
    double  tr_time;           /* Time for 1 transaction.       */
    double  tr_min;            /* Shortest transaction time.    */
    double  tr_sum;            /* Sum of transaction times.     */


    /*
    **  Clause 6.6.2:  percentage of home and remote transactions
    */
    int     remote = 0;   /* Number of remote transactions */

    /*
    **  Clause 6.6.3:  percentage of transactions that started but did
    **  not complete during the measurement interval.  These are called
    **  in-flight transactions here.
    */
    int     in_flight_transactions = 0;

    char   filename[30];       /* Result file name     */
    FILE * fp;                 /* Result file pointer  */
    FILE * fpt;                /* Residence time distribution file pointer */
    FILE * fpr;  /* Per-process resource stats file pointer */

    int    account_branch;     /* Branch of the updated account */

    int success_file = FALSE;  /* Write success file after every transaction */

    /*
    **  Function Declarations.
    */
    FILE      * fopen();
    int         atoi();
    long        lrand48();

    double      atof();
    double      gettime();
    double      getcpu();

    /* Process command-line arguments.  */

    /*  Required Arguments.  */

    if (argc < 9)
       usage();

    /*
```

```
**  argv[1] -- Configuration name.
**  argv[2] -- Arbitrary run name.
*/

strcpy(config,  argv[1]);
strcpy(runname, argv[2]);

/*
**  argv[3] -- time limit in seconds.
*/

timelimit = atoi(argv[3]);

if (timelimit < 0)
{
   printf("Invalid time limit parameter:  '%s'\n", argv[3]);
   usage();
}

/*
**  argv[4] -- total number of processes in the test.
*/

if ((nproc = atoi(argv[4])) < 1)
{
   printf("Invalid number of processes parameter:  '%s'\n", argv[4]);
   usage();
}

/*
**  argv[5] -- process number of this process.
*/
if ((proc_no = atoi(argv[5])) < 1 || proc_no > nproc)
{
   printf("Invalid process number parameter:  '%s'\n", argv[5]);
   usage();
}

/*
**  argv[6] -- Actual starttime including trigger.
*/
if ((starttime = atof(argv[6])) < 1)
{
   printf("Invalid starttime parameter:  '%s'\n", argv[6]);
   usage();
}

/*
**  argv[7] -- total number of hosts / nodes.
*/

if ((nhosts = atoi(argv[7])) < 1)
{
   printf("Invalid number of hosts parameter:  '%s'\n", argv[7]);
   usage();
}

/*
**  argv[8] -- host ID of this host.
*/
if ((hid = atoi(argv[8])) < 1 || hid > nhosts)
{
   printf("Invalid host ID parameter:  '%s'\n", argv[8]);
   usage();
}

/*
**  Optional Arguments.
*/

argc -= 8;  argv += 8;
while(--argc)
```

```
{
   ++argv;
   switch(argv[0][0])
   {

      case 'd':
         /*
         **  Durability test.  Write to success file after every
         **  transaction.
         */
         success_file = TRUE;
         break;

      case 'e':
         /*
         **  Ramp-down time in seconds.
         */
         if ((ramp_down = atoi(++(argv[0]))) < 0)
         {
            printf("Invalid ramp down time:  '%d'\n", ramp_down);
            usage();
         }
         break;

      case 's':
         /*
         **  Ramp-up time in seconds.
         */
         if ((ramp_up = atoi(++(argv[0]))) < 0)
         {
            printf("Invalid ramp up time:  '%d'\n", ramp_up);
            usage();
         }
         break;

      case 'u':
         /*
         **  Database user id.
         */
         uid = ++(argv[0]);
         break;

      case 'x':
         /*
         **  Database multiplier.
         */
         if ((db_multiplier = atoi(++(argv[0]))) < 0)
         {
            printf("Invalid database size multiplier:  '%d'\n",
               db_multiplier);
            usage();
         }
         /*
         **  Multiply table sizes by specified multiplier.
         */
         accnum  *= db_multiplier;
         tellnum *= db_multiplier;
         brannum *= db_multiplier;
         break;

      default:
         printf("Unknown argument %s\n", argv[0]);
         usage();
         break;
   }
}

/*
**  Open output file and trigger file.  Do before connect
**  because single-task programs will cd to ORACLE_HOME/dbs
**  on connect and want output file in current directory.
*/
```

```
sprintf(filename, "tpcb_%d.log", proc_no);
if ((fp = fopen(filename, "w")) == NULL)
{
   printf("TPC-B Proc #%d:  Can't open log file: %s.\n", proc_no,
      filename);
   exit (1);
}

sprintf(filename, "tpcb_%d.int", proc_no);
if ((fpt = fopen(filename, "w")) == NULL)
{
   printf("TPC-B Proc #%d:  Can't open interval file: %s.\n", proc_no,
      filename);
   exit (1);
}

sprintf( filename, "tpcb_%d.get", proc_no );
if((fpr = fopen(filename, "w")) == NULL)
{
   fprintf(stderr, "TPC-B Proc #%d:  Can't open resource file: %s.\n",
      proc_no, filename);
   exit (1);
}

if (success_file)
{
   if (succinit(proc_no)<0)
      exit (1);
}

/*
**  Logon to Oracle. Connect to database. Initialize SQL statements.
*/
TPCinit(proc_no);

if (ramp_down == 0)
   ramp_down = 1;

/*
**  Initialize transaction timing statistics.
*/
tr_count = 0l;
tr_fast  = 0l;
tr_min   = 100000l;
tr_max   = 0l;
tr_sum   = 0l;
for (i = 0; i < NUM_TIMING_BUCKETS; i++)
   timing_buckets[i] = 0;

/*
** Determine overhead of gettime() function.
*/

gettime();
tr_overhead = 0.0;
for (i = 0; i < 1000; i++)
   tr_overhead += gettime() - gettime();
tr_overhead = tr_overhead/1000.0;


/*
** Seed Random Number Generator.
*/
srand48(getpid() * gettime());
srand(getpid() * gettime());


printf("Time %d ramp up %d ramp down %d mult %d\n",
   timelimit, ramp_up, ramp_down, db_multiplier);

/*
**  Sleep until start time.
```

```
*/
waitfortrigger (starttime + (proc_no * 0.04));


/*
**  Initialize timing.
*/

begin_time  =  gettime();
begin_time += (double)ramp_up;
interval_end_time = begin_time + (double)timelimit;
end_time    =  interval_end_time + (double) ramp_down;

/*
**  Execute transactions until time is up.
*/

while(TRUE)
{

   /* Pick random amount in range -999999 to 999999: Clause 5.3.6 */
   amount = (lrand48() % 1999999) - 999999;

   /*
   **  Clause 5.3.3: For single-node systems, choose a teller at
   **  random from the entire range of tellers.
   */
   if (nhosts > 1)
     teller_no = (rand () % ((db_multiplier / nhosts) * 10)) +
              (hid - 1) * (db_multiplier / nhosts) * 10;
   else
     teller_no = (rand() % tellnum) + 1;

   /*
   **  Clause 5.3.4: "Given the randomly chosen teller...the
   **  corresponding branch is determined"
   **  There are 10 tellers per branch.  First teller is 1.
   */
   branch_no = ((teller_no - 1)/10) + 1;

   /*
   **  Clause 5.3.5:  Account ID Generation
   **    85% of the time, randomly choose an account from the
   **    local branch.  15% of the time, randomly choose an
   **    account from one of the other branches.
   */

   if (brannum > 1)
   {
      if (rand()%100 < 85)
            account_branch = branch_no;
      else
         do
            account_branch = (rand()%brannum) + 1;
         while (account_branch == branch_no);
   }
   else
      account_branch = branch_no;

   /* There are 100,000 accounts per branch; */
   account_no = 100000*(account_branch - 1) + (lrand48()%100000) + 1;

   /* Get transaction start time */
   tr_begin = gettime();

   /* If ramp-up is over, get cpu time and start timing interval */
   if ((in_ramp_up == TRUE) && (tr_begin > begin_time))
   {
      begin_cpu  = getcpu();
      in_ramp_up = FALSE;
      in_timing_interval = TRUE;
   }
```

```
    /*
    **  Execute transaction.
    */

    if (TPCexec() < 0) {
      exit(1);
    }

    /* Get transaction end time */
    tr_end = gettime();

    if (success_file)
    {
        if (succlog(proc_no,account_no,teller_no,branch_no,amount,balance))
            exit (1);
    }

    /* Compute residence time */
    tr_time = tr_end - tr_begin - tr_overhead;

    /* Calculate statistics if in the timing interval */

    if (in_timing_interval && (tr_end <= interval_end_time))
    {
        if (account_branch != branch_no)
            remote++;

        ++tr_count;

        if (tr_time < 0)
            tr_time = 0;

        if (tr_time <= TPCB_FAST)
            ++tr_fast;

        if (tr_time < tr_min)
            tr_min = tr_time;

        if (tr_time > tr_max)
            tr_max = tr_time;

        tr_sum  += tr_time;

        if (tr_time >= MAX_SECS)
            timing_buckets[NUM_TIMING_BUCKETS]++;
        else
            timing_buckets[(int)(tr_time/SECONDS_PER_BUCKET)]++;
    }

    if (in_timing_interval && (tr_end >= interval_end_time))
    {
        end_stat_time = gettime();
        end_cpu       = getcpu();
        in_timing_interval = FALSE;
        if (tr_end > interval_end_time)
            in_flight_transactions++;
    }

    if (tr_end > end_time)
        break;
}

/*
** Print summary to result file.
*/

fprintf(fp, "%s %s %s %d ", config, runname, audit_str, proc_no);

fprintf(fp, "%d %d ",       tr_count,  tr_fast);
fprintf(fp, "%.2f %.2f %.2f ", begin_time, end_stat_time,
            tr_count ? tr_sum/(double)tr_count : 0.0);
```

```
    fprintf(fp, "%.2f %.2f %.2f ", tr_min, tr_max, end_cpu - begin_cpu);
    fprintf(fp, "%d %d %d", in_flight_transactions, remote, retries);
    fprintf(fp, " %d %d %d\n", 0, 0, 0); /* think time stats */
    fclose(fp);

    fprintf(fpt, "%s %s %s %d ", config, runname, audit_str, proc_no);
    for (i = 0; i < NUM_TIMING_BUCKETS; i++)
        fprintf(fpt, " %d", timing_buckets[i]);
    fprintf(fpt, "\n");
    fclose(fpt);

    /*
    **  Log off the database.
    */

    TPCexit();

    /*
    **  Print Getrusage result to get file.
    */
    getru(fpr, 0, config, runname, proc_no);
    getru(fpr, 1, config, runname, proc_no);
    fclose(fpr);

    if (success_file)
    {
        if (succend(proc_no))
            exit (1);
    }
    /*
    **  Exit with success.
    */
    exit(0);
}


/*
**  Print program usage message.
*/
static void usage()
{
    printf("\nUsage is:\n\n");
    printf("    tpcb config runname time_limit nproc proc_no nhosts hid [options]\n");
    printf("\nwhere:\n\n");
    printf("    config        Configuration identifier.\n");
    printf("    runname       Run name up to 10 characters.\n");
    printf("    time_limit    Elapsed time for test to run.\n");
    printf("    nproc         Total number of processes.\n");
    printf("    proc_no       This process number (1..nproc)\n");
    printf("    starttime     start time\n");
    printf("    nhosts        Total number of hosts / nodes.\n");
    printf("    hid           This host ID (1..nhosts)\n");
    printf("    options:\n");
    printf("        d         Durability tests: write success file.\n");
    printf("        ennn      Continue nnn seconds without taking stats.\n");
    printf("        snnn      Wait nnn seconds before taking stats.\n");
    printf("        uuserid   Database userid.\n");
    printf("        xnnn      Increase database size by multiplier.\n");
    printf("\n");
    exit(1);
}
```

ab_trans.c:

```
/*+================================================================+
 |          Copyright (c) 1992  Oracle Corp, Belmont, CA          |
 |                  UNIX PERFORMANCE GROUP                        |
 |                   All Rights Reserved                          |
 +================================================================*/

#include <stdio.h>

/*
** Global variables.
*/
extern long    account_no;   /* Account id to update */
extern long    branch_no;    /* Branch id to update */
extern long    teller_no;    /* Teller id to update */
extern long    amount;       /* Amount added to the balance */
extern double  balance;      /* New balance of the account record */
extern char *  uid;          /* Database user name and password */

extern int     retries;  /* Number of retries in the discrete transaction */

/*
**  Oracle variable type definitions.
*/

#define  SQLT_CHR  1                    /* (ORANET TYPE) character string */
#define  SQLT_NUM  2                    /* (ORANET TYPE) oracle numeric */
#define  SQLT_INT  3                    /* (ORANET TYPE) integer */
#define  SQLT_FLT  4                /* (ORANET TYPE) Floating point number */
#define  SQLT_RID  11                                   /* rowid */
#define  SQLT_DAT  12                   /* date in oracle format */

/*
**  Oracle cursor structure.
*/
struct csrdef
{
  short         csrrc;                  /* return code */
  unsigned short csrft;                 /* function type */
  unsigned long  csrrpc;                /* rows processed count */
  unsigned short csrpeo;                /* parse error offset */
  unsigned char  csrfc;                 /* function code */
  unsigned char  csrfil;                /* filler  */
  unsigned short csrarc;                /* reserved, private */
  unsigned char  csrwrn;                /* warning flags */
  unsigned char  csrflg;                /* error flags */
  /*              *** Operating system dependent ***              */
  unsigned int   csrcn;                 /* cursor number */
  struct {                              /* rowid structure */
    struct {
      unsigned long   tidtrba;          /* rba of first blockof table */
      unsigned short  tidpid;           /* partition id of table */
      unsigned char   tidtbl;           /* table id of table */
      }             ridtid;
    unsigned long   ridbrba;            /* rba of datablock */
    unsigned short  ridsqn;         /* sequence number of row in block */
    } csrrid;
  unsigned int   csrose;                /* os dependent error code */
  unsigned char  csrchk;                /* check byte */
  unsigned char  crsfill[26];           /* private, reserved fill */
};

typedef struct csrdef csrdef;
typedef struct csrdef ldadef;

void errrpt();

ldadef tpclda;
char   tpchda[256];
```

```
#define SQLTXT \
"\
begin\
  dbms_transaction.begin_discrete_transaction;\
  loop begin\
    update account\
      set account_balance = account_balance + :dlta\
      where account_id = :acct;\
    insert into history values\
      (:tell, :bran, :acct, :dlta, sysdate,\
       '%03d-567890123456789012345678 9012hiPETER');\
    update teller\
      set teller_balance = teller_balance + :dlta\
      where teller_id = :tell;\
    update branch\
      set branch_balance = branch_balance + :dlta\
      where branch_id = :bran;\
    commit;\
    :bala := tpcab_pack.account_bal;\
    exit;\
    exception\
      when dbms_transaction.discrete_transaction_failed or \
         dbms_transaction.consistent_read_failure then\
         rollback;\
         :retr := :retr + 1;\
    end;\
  end loop;\
end;\
"

csrdef  * csr;          /* Cursor */



/*
**  TPCinit: perform database initialization.  Log on to the database.
**      Parse the transaction.  Bind the transaction variables.
**      Return 0 on success, -1 on failure.
*/
TPCinit(proc_no)
int proc_no;
{
    char sqlbuf[1024];

    /*
    ** Log on to the database
    */
    if (orlon(&tpclda, tpchda, uid, -1, (char *) -1, -1, 0))
    {
         errrpt(&tpclda);
         return -1;
    }

    if (ocicof(&tpclda))
    {
       errrpt(&tpclda);
       return -1;
    }

    /* Allocate cursor */
    csr  = (csrdef *)malloc(sizeof(csrdef));
    if (csr == (csrdef *)0)
    {
       fprintf(stderr, "Error: TPCinit(): 0 returned by malloc\n");
       return -1;
    }

    /* Open cursor */
    if (ociope(csr, &tpclda, (char *)0, 0, -1, uid, -1))
    {
       errrpt(csr);
```

```
        return -1;
    }

    sprintf(sqlbuf, SQLTXT, proc_no);

    /* Parse sql statement */
    if (osql3(csr, sqlbuf, -1))
    {
        errrpt(csr);
        return -1;
    }

    /* Bind variables */

    if (obndrv(csr, ":ACCT", -1, &account_no, sizeof(account_no), SQLT_INT,
        -1, (short *) -1, (char *) -1, -1, -1))
    {
        errrpt(csr);
        return -1;
    }

    if (obndrv(csr, ":BALA", -1, &balance, sizeof(balance), SQLT_FLT,
        -1, (short *) -1, (char *) -1, -1, -1))
    {
        errrpt(csr);
        return -1;
    }

    if (obndrv(csr, ":BRAN", -1, &branch_no, sizeof(branch_no), SQLT_INT,
        -1, (short *) -1, (char *) -1, -1, -1))
    {
        errrpt(csr);
        return -1;
    }

    if (obndrv(csr, ":DLTA", -1, &amount, sizeof(amount), SQLT_INT,
        -1, (short *) -1, (char *) -1, -1, -1))
    {
        errrpt(csr);
        return -1;
    }

    if (obndrv(csr, ":TELL", -1, &teller_no, sizeof(teller_no), SQLT_INT,
        -1, (short *) -1, (char *) -1, -1, -1))
    {
        errrpt(csr);
        return -1;
    }

    if (obndrv(csr, ":RETR", -1, &retries, sizeof(retries), SQLT_INT,
        -1, (short *) -1, (char *) -1, -1, -1))
    {
        errrpt(csr);
        return -1;
    }
    return 0;
}

/*
** TPCexec: Execute the transaction.
**     Return 0 on success, -1 on failure.
*/
TPCexec()
{
    char msg[2048];

    if (ociexe(csr))
    {
        if (csr->csrrc)
        {
            (void) ocierr(csr, csr->csrrc, msg, 2048);
            (void) fprintf(stderr, "%s\n", msg);
```

```
        }
        orol(&tpclda);
        return -1;
    }
    return 0;
}

/*
**  TPCexit: Close cursor and log off database.
**      Return 0 on success, -1 on failure.
*/
TPCexit()
{
    /* Close cursor */
    if (ociclo(csr))
        errrpt(csr);

    /* Free cursor */
    free(csr);

    /* Log off database */
    ocilof(&tpclda);

    return 0;
}
```

Appendix B
Database Definition and Load

File Definitions for ABTH Tables

ORACLE used the following "C" code to define, create and load the Account, Teller, Branch and History tables.

ab_tab.sql:

```
CONNECT system/manager;
GRANT CONNECT,RESOURCE,UNLIMITED TABLESPACE TO tpcb IDENTIFIED BY tpcb;
CONNECT tpcb/tpcb;
DROP CLUSTER acluster INCLUDING TABLES;
CREATE CLUSTER acluster
(
account_id number(10,0)
)
HASHKEYS   200000000
HASH IS    account_id
SIZE       138
INITRANS   2
PCTFREE    0
TABLESPACE acct
STORAGE
(
INITIAL     235M
NEXT        235M
PCTINCREASE 0
MINEXTENTS  121
);
CREATE TABLE account
(
account_id        NUMBER(10,0),
branch_id         NUMBER,
account_balance   NUMBER,
filler            VARCHAR2(97)
)
CLUSTER acluster(account_id);

CONNECT tpcb/tpcb

CREATE TABLE teller (
teller_id         NUMBER(10,0),
branch_id         NUMBER(10,0),
teller_balance    NUMBER(10,0),
filler            CHAR(97)
)
PCTFREE 40
PCTUSED 4
```

```
STORAGE ( initial 210K next 210K pctincrease 0 minextents 48 );

CREATE TABLE branch
(
branch_id          NUMBER,
branch_balance     NUMBER,
filler             CHAR(98)
)
PCTFREE 90
PCTUSED 4
STORAGE (initial 40K next 40K pctincrease 0 minextents 121 );

EXIT;
ab_hist.sql:

tpcb/tpcb

rem
rem ================================================================+
rem                                          Copyright (c) 1991  Oracle Corp, Belmont, CA         |
rem                                         All Rights Reserved |
rem ================================================================+
rem  FILENAME
rem       ab_hist.sql
rem  DESCRIPTION
rem ================================================================*/
rem


   DROP TABLE history;

rem the following will fail
   create table history_coalesce (x number)
       tablespace hist
       storage (initial 2000M);


   CREATE TABLE history
   (
      teller_id        NUMBER,
      branch_id        NUMBER,
      account_id       NUMBER,
      amount                                  NUMBER,
      timestamp        DATE,
      filler                                  VARCHAR2(39)
   )
   tablespace histalloc
   storage (initial 4k
                                                 minextents 1
                                                 pctincrease 0
                                                 freelist groups 13
                                                 freelists 17
                                                 ) pctfree 0;
   alter table history allocate extent (size 78M freelist group 1);
   alter table history allocate extent (size 78M freelist group 2);
   alter table history allocate extent (size 78M freelist group 3);
   alter table history allocate extent (size 78M freelist group 4);
   alter table history allocate extent (size 78M freelist group 5);
   alter table history allocate extent (size 78M freelist group 6);
   alter table history allocate extent (size 78M freelist group 7);
   alter table history allocate extent (size 78M freelist group 8);
   alter table history allocate extent (size 78M freelist group 9);
   alter table history allocate extent (size 78M freelist group 10);
   alter table history allocate extent (size 78M freelist group 11);
   alter table history allocate extent (size 78M freelist group 12);
   alter table history allocate extent (size 78M freelist group 13);

   EXIT;
```

Code for loading ABTH files

The following code was used to generate the data that loaded the Account, Teller, and Branch tables.

ab_load.c:

```
/*=================================================================+
 |  Copyright (c) 1992  Oracle Corp, Belmont, CA          |
 | All Rights Reserved |
 +=================================================================+
 | FILENAME
 |  ab_load.c
 | DESCRIPTION
 |  load database tables for TPC-A or -B benchmark.
 |
 +=================================================================*/

typedef char  b1;
typedef short b2;
typedef int   b4;

typedef unsigned char  ub1;
typedef unsigned short ub2;
typedef unsigned int   ub4;

typedef ub1 text;

#include <stdio.h>

/* input data types */
#define  SQLT_CHR  1        /* (ORANET TYPE) character string */
#define  SQLT_INT  3        /* (ORANET TYPE) integer */

/*
**  Oracle cursor structure.
*/
struct csrdef
{
   short          csrrc;     /* return code */
   unsigned short csrft;   /* function type */
   unsigned long  csrrpc;/* rows processed count */
   unsigned short csrpeo;  /* parse error offset */
   unsigned char  csrfc;   /* function code */
   unsigned char  csrfil;        /* filler  */
   unsigned short csrarc;   /* reserved, private */
   unsigned char  csrwrn;  /* warning flags */
   unsigned char  csrflg;    /* error flags */
   /*     *** Operating system dependent ***       */
   unsigned int   csrcn;   /* cursor number */
   struct {   /* rowid structure */
     struct {
        unsigned long   tidtrba;/* rba of first blockof table */
        unsigned short  tidpid;      /* partition id of table */
        unsigned char   tidtbl;          /* table id of table */
        }    ridtid;
     unsigned long   ridbrba;/* rba of datablock */
     unsigned short  ridsqn;          /* sequence number of row in block
*/
     } csrrid;
   unsigned int   csrose;          /* os dependent error code */
   unsigned char  csrchk;     /* check byte */
   unsigned char  crsfill[26];      /* private, reserved fill */
};

typedef struct csrdef csrdef;
typedef struct csrdef ldadef;

ldadef tpclda;
char   tpchda[256];

/* SQL statements */
```

```
#define SQLTXT_ACCT \
“INSERT INTO account(account_id, account_balance, branch_id, filler)\
 VALUES (:1, :2, :3, :4)”

#define SQLTXT_TELLER \
“INSERT INTO teller(teller_id, teller_balance, branch_id, filler)\
 VALUES (:1, :2, :3, :4)”

#define SQLTXT_BRANCH \
“INSERT INTO branch(branch_id, branch_balance, filler)\
 VALUES (:1, :2, :3)”

/* SQL cursor */

csrdef * csr;

#define BRANCH  1   /* Table IDs; command line arg mapped here */
#define TELLER  2
#define ACCOUNT 3

#define LOOP  100 /* Number of rows to insert before committing. */

#define INITBAL -1111111111     /* Init balance. Use all 10 digits */

#define PAD97 \
“12345678901234567890123456789012345678901234567890\
12345678901234567890123456789012345678901234567”

#define PAD98 \
“12345678901234567890123456789012345678901234567890\
123456789012345678901234567890123456789012345678”

#define MIN(a,b) ((a) < (b) ? (a) : (b))

/*=================================================================+
 | ROUTINE NAME
 |  main
 | DESCRIPTION
 |  main routine
 | ARGUMENTS
 |  tpcbload <tablename> <#_rows_to_insert> [#_row_to_start]
 +=================================================================*/

main(argc, argv)
    int argc;
    char *argv[];
{
    char   * uid = “tpcb/tpcb”;
    char   * upasswd = “tpcb”;
    int     tellbran;  /* branch to which teller belongs */
    int     acctbran;  /* branch to which account belongs */
    int     init_bal;
    char    rowpad[128];
    int     loop;/* for array inserts    */
    char    sqlbuf[256];
    int     i, j;
    int     which_table = 0;    /* 1=acct, 2=teller, 3=branch   */
    long    nrows;          /* # of rows to insert      */
    long    row; /* row/key-value counter    */
    long    start;          /* starting key value       */
    long    end; /* ending key value     */
    int     loopcount;      /* insert loop counter      */
    int     err = 0;

    void errrpt();
    double begin_time, end_time;
    double begin_cpu, end_cpu;
    static double gettime(), getcpu();

    /*
    **  Parse command line -- look for specific table to load.
```

```
    */

    if (argc < 3) usage();

    /*
    **  argv[1]:  table name.
    */

    switch (argv[1][0])
    {
        case 'a':/* account table    */
     which_table = ACCOUNT;
     break;
        case 't':/* teller table     */
     which_table = TELLER;
     break;
        case 'b':/* branch table     */
     which_table = BRANCH;
     break;
        default:
     usage();
     break;
    }

    /*
    **  argv[2]:  # of rows to insert.
    */
    if ((nrows = atoi(argv[2])) < 1 )
    {
        fprintf(stderr, "Invalid number of rows to insert:  '%d'\n",
nrows );
        usage();
    }

    /*
    ** argv[3]:  starting row #  (optional).
    */

    if (argc > 3)
    {
        if ((start = atoi(argv[3])) < 1 )
        {
     fprintf(stderr, "Invalid start offset:  '%d'\n", start );
     exit();
        }
    }
    else start = 1;
        end = start + nrows - 1;

    /*
    ** Log on to the database
    */
    if (orlon(&tpclda, tpchda, uid, -1, (char *) -1, -1, 0))

    {
          errrpt(&tpclda);
          return -1;
    }

    if (ocicof(&tpclda))
    {
        errrpt(&tpclda);
        return -1;
    }

    csr = (csrdef *)malloc(sizeof(csrdef));

    if (csr == (csrdef *)0)
    {
        fprintf(stderr, "Error: 0 returned by malloc\n");
        exit(-1);
    }
```

```
    if (ociope(csr, &tpclda, (char *)0, 0, -1, uid, -1))
        errrpt(csr);

    /* prepare the account insert cursor */
    if (which_table == ACCOUNT)
    {
        sprintf(sqlbuf, SQLTXT_ACCT);

        init_bal = INITBAL;

        strcpy(rowpad, PAD97);

        if (osql3(csr, sqlbuf, -1))
    errrpt(csr);

        if (obndrn(csr, 1, &row, sizeof(row), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 2, &init_bal, sizeof(init_bal), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 3, &acctbran, sizeof(acctbran), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 4, rowpad, strlen(rowpad), SQLT_CHR, -1,
    (short *)NULL, -1))
    errrpt(csr);

        printf("Loading ACCOUNT table with %d rows starting with %d
...\n   ",
    nrows, start );
    }

    if (which_table == TELLER)
    {
        sprintf(sqlbuf, SQLTXT_TELLER);
        init_bal = INITBAL;
        strcpy(rowpad, PAD97);

        if (osql3(csr, sqlbuf, -1))
    errrpt(csr);

        if (obndrn(csr, 1, &row, sizeof(row), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 2, &init_bal, sizeof(init_bal), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 3, &tellbran, sizeof(tellbran), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 4, rowpad, strlen(rowpad), SQLT_CHR, -1,
    (short *)NULL, -1))
    errrpt(csr);

        printf("Loading TELLER table with %d rows starting with %d
...\n   ",
    nrows, start );
    }

    if (which_table == BRANCH)
    {
        sprintf(sqlbuf, SQLTXT_BRANCH);
        init_bal = INITBAL;
        strcpy(rowpad, PAD98);
```

```
        if (osql3(csr, sqlbuf, -1))
    errrpt(csr);

        if (obndrn(csr, 1, &row, sizeof(row), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 2, &init_bal, sizeof(init_bal), SQLT_INT, -1,
    (short *)NULL, -1))
    errrpt(csr);

        if (obndrn(csr, 3, rowpad, strlen(rowpad), SQLT_CHR, -1,
    (short *)NULL, -1))
    errrpt(csr);

        printf("Loading TELLER table with %d rows starting with %d
...\n   ",
    nrows, start );
    }

    begin_time = gettime();
    begin_cpu = getcpu();

    loopcount = 0;
    row = start;

    while (row <= end)
    {
        loop = MIN(LOOP, end - row + 1);

        for (i = 0; i < loop; i++, row++)
        {
    acctbran = ((row - 1) / 100000) + 1;
    tellbran = ((row - 1) / 10) + 1;

    if (err = oexec(csr))
    {
        orol(&tpclda);
        errrpt(csr);
    }
        }

        if (err = ocom(&tpclda))
        {
    orol(&tpclda);
    errrpt(&tpclda);
        }

        if ((++loopcount) % 50)
    printf( "." );
        else
    printf( " row %d committed.\n   ", row - 1);
    }

    end_time = gettime();
    end_cpu = getcpu();

    printf( "Load completed.  %d records processed.\n", nrows );
    printf( "      in %10.2f real, %10.2f cpu.\n",
        end_time-begin_time, end_cpu-begin_cpu );

    if (ociclo(csr))
        errrpt(csr);

    free(csr);

    ocilof(&tpclda);

    exit(0);

}
```

```
usage()
{
    printf("\n");
    printf(
        "Usage:  tpcbload <table_name> <#_rows_to_insert>
[starting_row_#]\n");
    printf("\n");
    exit(1);
}

void errrpt(cur)
    csrdef  *cur;
{
    char msg[2048];

    if (cur->csrrc)
    {
         (void) ocierr(cur, cur->csrrc, msg, 2048);
         (void) fprintf(stderr, "%s\n", msg);
    }
    exit(0);
}
```

ABTH Sample Data

Following is a script of an SQL session which gives examples of the ABTH data:

```
SQL> describe account;
 Name                           Null?    Type
 ------------------------------ -------- ----
 ACCOUNT_ID                              NUMBER(10)
 BRANCH_ID                               NUMBER
 ACCOUNT_BALANCE                         NUMBER
 FILLER                                  VARCHAR2(97)

SQL> describe branch;
 Name                           Null?    Type
 ------------------------------ -------- ----
 BRANCH_ID                               NUMBER
 BRANCH_BALANCE                          NUMBER
 FILLER                                  CHAR(98)

SQL> describe teller;
 Name                           Null?    Type
 ------------------------------ -------- ----
 TELLER_ID                               NUMBER(10)
 BRANCH_ID                               NUMBER(10)
 TELLER_BALANCE                          NUMBER(10)
 FILLER                                  CHAR(97)

SQL> describe history;
 Name                           Null?    Type
 ------------------------------ -------- ----
 TELLER_ID                               NUMBER
 BRANCH_ID                               NUMBER
 ACCOUNT_ID                              NUMBER
 AMOUNT                                  NUMBER
 TIMESTAMP                               DATE
 FILLER                                  VARCHAR2(39)

SQL> select * from account where account_id < 15;

ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------
         1          1     -1.112E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
```

```
4567890
12345678901234567

         2         1     -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567

         3         1     -1.111E+09

ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567

         4         1     -1.112E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567

         5         1     -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890

ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567

         6         1     -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567

         7         1     -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567

ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------

         8         1     -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567

         9         1     -1.110E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567


ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------
        10         1     -1.112E+09
12345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890
12345678901234567
```

```
        11         1      -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        12         1      -1.111E+09

ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        13         1      -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        14         1      -1.111E+09
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890

ACCOUNT_ID  BRANCH_ID ACCOUNT_BALANCE
---------- ---------- ---------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567


14 rows selected.

SQL> select * from branch where branch_id < 15;

 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
         1      19039773
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         2      69921693
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         3      50627709

 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         4      20050773
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         5      21558649
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
```

```
 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
123456789012345678

         6      -11428302
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

         7       17101009
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
--------------------------------------------------------------------------------
-------

         8        4056146
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

         9       23973744
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678


 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
        10       38315400
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

        11       14849198
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

        12        3702723

 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

        13       -7663864
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
123456789012345678

        14      -50856331
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890

 BRANCH_ID BRANCH_BALANCE
---------- --------------
FILLER
```

```
------------------------------------------------------------------------
-------
12345678901234567


14 rows selected.

SQL> select * from teller where teller_id < 15;

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
------------------------------------------------------------------------
-------
         1          1       15031202
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         2          1      -13669766
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         3          1       -4448416

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
------------------------------------------------------------------------
-------
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         4          1       -2508623
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         5          1       -1677821
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
------------------------------------------------------------------------
-------
12345678901234567

         6          1        7677128
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

         7          1       10427346
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
------------------------------------------------------------------------
-------

         8          1        1390624
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567
```

```
         9         1       2263375
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
        10         1       4554724
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        11         2       3613648
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        12         2      13308017

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        13         2      28927289
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890
12345678901234567

        14         2       3900913
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890

 TELLER_ID  BRANCH_ID TELLER_BALANCE
---------- ---------- --------------
FILLER
--------------------------------------------------------------------------------
-------
12345678901234567


14 rows selected.

SQL> select * from history where rownum < 10;


 TELLER_ID  BRANCH_ID ACCOUNT_ID     AMOUNT TIMESTAMP
---------- ---------- ---------- ---------- ---------
FILLER
----------------------------------------
     16408       1641  164001585    -157665 01-NOV-93
044-567890123456789012345678901 2hiPETER

      6406        641   64027858     303931 01-NOV-93
044-567890123456789012345678901 2hiPETER

     14738       1474  147364646    -336077 01-NOV-93
044-567890123456789012345678901 2hiPETER


 TELLER_ID  BRANCH_ID ACCOUNT_ID     AMOUNT TIMESTAMP
---------- ---------- ---------- ---------- ---------
FILLER
----------------------------------------
```

```
      7943        795   79457944    -267376 01-NOV-93
044-567890123456789012345678912hiPETER

      1406        141   14057548     661418 01-NOV-93
044-567890123456789012345678912hiPETER

      2520        252   25103105     713666 01-NOV-93
044-567890123456789012345678912hiPETER


 TELLER_ID  BRANCH_ID ACCOUNT_ID     AMOUNT TIMESTAMP
---------- ---------- ---------- ---------- ---------
FILLER
----------------------------------------
      6848        685   68484891    -544137 01-NOV-93
044-567890123456789012345678912hiPETER

      4816        482   48176670     984981 01-NOV-93
044-567890123456789012345678912hiPETER

     16809       1681  168008991     795424 01-NOV-93
044-567890123456789012345678912hiPETER

9 rows selected.
```

Appendix C
Tunable Parameters

Operating System Tunable Parameters

```
* This file contains local system settings for tunable parameters
* The parameter settings in this file replace the default values
* specified in mtune, if the new values are within the legal range
* for the parameter specified in mtune. The file contains one line
* for each parameter to be reset. The syntax for each line is
    <parameter name> = <value
    *parameter name: This is the external name of the tunable
    *parameter used in the mtune files
    *value: This field contains the new value for the tunable parameter
rlimit_rss_cur= 0x20000000
maxlkmem = 0x100000
posix_tty_defaulT= 1
resettable_clocal= 1
nproc= 500
nprofile= 500
maxup= 400
semmni= 100
semmns= 200
semmnu= 100
semmsl= 200
ndpri_hilIM= 30
syssEGSz 0x20000
maxpwent= 300
```

ORACLE Configuration

Init.ora Parameters:

```
PARAMETER                      VALUE
audit_file_dest                ?/rdbms/audit
audit_trail                    NONE
background_dump_dest           ?/rdbms/log
ccf_io_size                    65536
checkpoint_process             TRUE
cleanup_rollback_entries       20
commit_point_strength          1
compatible
compatible_no_recovery
control_files                  ?/dbs/cntrl@.dbf
core_dump_dest                 ?/dbs
cursor_space_for_time          TRUE
db_block_buffers               33000
db_block_checkpoint_batch      512
db_block_lru_extended_statisti 0
db_block_lru_statistics        FALSE
db_block_size                  2048
db_domain                      WORLD
db_file_multiblock_read_count  8
```

```
db_file_simultaneous_writes    4
db_files                       150
db_name                        tpcb
db_writers                     1
discrete_transactions_enabled  TRUE
distributed_lock_timeout       60
distributed_recovery_connectio 200
distributed_transactions       110
dml_locks                      800
enqueue_resources              835
event
fixed_date
gc_db_locks                    33000
gc_files_to_locks
gc_lck_procs                   1
gc_rollback_locks              20
gc_rollback_segments           20
gc_save_rollback_locks         20
gc_segments                    10
gc_tablespaces                 5
global_names                   FALSE
ifile
init_sql_files                 ?/dbs/sql.bsq
instance_number                0
license_max_sessions           0
license_max_users              0
license_sessions_warning       0
log_archive_buffer_size        127
log_archive_buffers            4
log_archive_dest               ?/dbs/arch
log_archive_format             %t_%s.dbf
log_archive_start              FALSE
log_buffer                     327680
log_checkpoint_interval        3567587327
log_checkpoint_timeout         0
log_checkpoints_to_alert       TRUE
log_files                      255
log_simultaneous_copies        28
log_small_entry_max_size       800
max_commit_propagation_delay   90000
max_dump_file_size             500
max_enabled_roles              22
max_rollback_segments          160
mts_dispatchers
mts_listener_address           (address=(protocol=ipc)(key=%s
mts_max_dispatchers            5
mts_max_servers                20
mts_servers                    0
mts_service
nls_currency
nls_date_format
nls_date_language
nls_iso_currency
nls_language                   AMERICAN
nls_numeric_characters
nls_sort
nls_territory                  AMERICA
open_cursors                   50
open_links                     4
optimizer_comp_weight          0
optimizer_mode                 CHOOSE
os_authent_prefix              ops
os_roles                       FALSE
pinned_memory_size             0
post_wait_device               /dev/postwait
pre_page_sga                   FALSE
processes                      200
remote_os_authent              FALSE
remote_os_roles                FALSE
resource_limit                 FALSE
rollback_segments              s1, s2, s3, s4, s5, s6, s7, s8
row_cache_cursors              10
```

```
row_locking                     default
sequence_cache_entries          10
sequence_cache_hash_buckets     7
serializable                    FALSE
sessions                        400
shared_pool_size                7000000
single_process                  FALSE
small_table_threshold           4
snapshot_refresh_interval       60
snapshot_refresh_keep_connecti FALSE
snapshot_refresh_processes      0
sort_area_retained_size         52428800
sort_area_size                  52428800
sort_mts_buffer_for_fetch_size 0
sort_read_fac                   5
sort_spacemap_size              512
spin_count                      6000
sql_trace                       FALSE
temporary_table_locks           400
```

Appendix D
Storage Requirements

Disk Storage Requirements

According to Clause 9.2.4.1 the priced configuration must contain sufficient disk space to store 8 hours of log data and 30 days of history data. This section documents the disk storage requirements of the priced configuration.

The total disk space requirements can be calculated as follows:

IRIX (UNIX system files +

system swap space +

ORACLE system and control files +

ORACLE database (ABT) files +

ORACLE log data (8-hours, mirrored)+

History data (30-days)

The 8-hour log data requirement was determined as follows:

8 hours * 3600 seconds/hour * 1786.2 tpsB/second =

51,442,560 transactions

The size of a log entry - 419.34 bytes - was obtained from the ORACLE statistics file. The total disk space required to hold 8 hours of mirrored log data can be computed as follows:

51,442,560 transactions * 419.34 bytes/transaction =

20572.59MB (20.09GB) of disk space - unmirrored, or

2 * 20572.59MB = 41145.18MB (40.18GB) - mirrored.

Disk drive capacity = 1918.2MB (formatted)

A total of 22 disk drives were dedicated to the 8-hour mirrored log requirement with space allocated as follows:

Total space available = 22 * 1918.2MB/drive = 42200.4MB

Log space required = 41145.18MB

Unused capacity = 1055.22MB

The size of a history file entry was determined as follows:

The 30-day history file space requirement was computed as follows:

Average number of history file entries per 2K page = 26.54

30 days * 51,442,560 transactions/day = 1,543,276,800 trans.

Number of 2K blocks required =

1,543,276,800 trans. / 26.54 bytes/page = 58,149,089 blocks

= 113,572.44MB (110.91GB)

Total disk space available for the 30-day history file requirement was computed as follows:

Total number of drives (less log drives) = 148 - 22 = 126 drives

Total disk space available = 126 drives * 1918.2MB/drive = 241693.2MB (236.03GB)

Disk space utilization:

UNIX system files:	534.6MB
Swap space:	256.0MB
ORACLE system and control files:	1738.2MB
Branch and Teller files:	900.0MB
Account files:	27178.0MB
Account index:	4350.0MB

Total space used (less log and history): 34956.8MB

Total space available for 30-day history file requirement = 241693.2MB - 34956.8MB = 206736.4MB

Appendix E
Attestation Letter

Appendix F
Supporting Documentation

The following information is being included in support of section 3.1.1 - Distribution and Partitioning - in which we stated that 18 of the drives which were configured into the SUT were complete inactive during the testing and, as a result, were not included in the priced configuration.

The following information, extracted from a report generated by the UNIX™ SAR (System Activity Reporter) facility, shows the 18 subject drives to be inactive.

System Activity Report

```
09:18:54 device %busy avque r+w/s blks/s w/s wblks/s avwait
avserv
Average dks2d5 0 0.0 0 0 0 0 0.0 0.0
 dks7d3 0 0.0 0 0 0 0 0.0 0.0
 dks7d6 0 0.0 0 0 0 0 0.0 0.0
 dks72d1 0 0.0 0 0 0 0 0.0 0.0
 dks72d6 0 0.0 0 0 0 0 0.0 0.0
 dks74d8 0 0.0 0 0 0 0 0.0 0.0
 dks75d1 0 0.0 0 0 0 0 0.0 0.0
 dks75d5 0 0.0 0 0 0 0 0.0 0.0
 dks77d7 0 0.0 0 0 0 0 0.0 0.0
 dks77d8 0 0.0 0 0 0 0 0.0 0.0
 dks110d4 0 0.0 0 0 0 0 0.0 0.0
 dks112d7 0 0.0 0 0 0 0 0.0 0.0
 dks113d3 0 0.0 0 0 0 0 0.0 0.0
 dks115d6 0 0.0 0 0 0 0 0.0 0.0
 dks116d3 0 0.0 0 0 0 0 0.0 0.0
 dks116d6 0 0.0 0 0 0 0 0.0 0.0
```

ORACLE Statistics Report

The following information was extracted from the statistics file which was generated by ORACLE during the benchmark.

```
FILE_NAME READS BLOCKS_READ READ_TIME
---------------------------------------- --------- ---------
-- ---------
/tpc_db/file10
/tpc_db/file105
/tpc_db/file110
/tpc_db/file121
/tpc_db/file124
/tpc_db/file13
/tpc_db/file17
/tpc_db/file36
/tpc_db/file40
/tpc_db/file59
/tpc_db/file63
/tpc_db/file64
/tpc_db/file82
/tpc_db/file86
/tpc_db/file87

FILE_NAME WRITES BLOCKS_WRITTEN WRITE_TIME
---------------------------------------- --------- ---------
----- ----------
/tpc_db/file10
/tpc_db/file105
/tpc_db/file110
/tpc_db/file121
/tpc_db/file124
/tpc_db/file13
/tpc_db/file17
/tpc_db/file36
/tpc_db/file40
/tpc_db/file59
/tpc_db/file63
/tpc_db/file64
/tpc_db/file82
/tpc_db/file86
/tpc_db/file87
```

The following shows the mapping between the file names used by ORACLE and the disk drive names reported by SAR. Note that the unused drive dks2d5 is mounted on /oracache which is a disk backup of ORACLE binaries.

```
FileName or FileSys Disk Drive
------------------- --------
/oracache           dks2d5
/tpc_db/file10      dks116d6
/tpc_db/file105     dks72d6
/tpc_db/file110     dks72d1
/tpc_db/file121     dks7d6
/tpc_db/file124     dks7d3
/tpc_db/file13      dks116d3
/tpc_db/file17      dks115d6
/tpc_db/file36      dks113d3
/tpc_db/file40      dks112d7
/tpc_db/file59      dks110d4
/tpc_db/file63      dks77d8
/tpc_db/file64      dks77d7
/tpc_db/file82      dks75d5
/tpc_db/file86      dks75d1
/tpc_db/file87      dks74d8
```