Oracle9*i*® Real Application Clusters:

Oracle Applications 11i Standard Benchmark– Tuning and Best Practices

Technical White Paper
November 2001

Version 11/28/2001

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Introduction

The Oracle Applications Standard Benchmark (OASB) was used to demonstrate the scalability and performance of the Oracle database in a multi-node cluster environment. As with all benchmarks, understanding the detailed performance characteristics of a product suite that is as broad and sophisticated as Oracle Applications is far from trivial. A key to understanding the performance metrics is a well defined and reproducible workload. This workload should represent as closely as possible the work that the customer base does on a "normal" daily basis, but also it must be easy to port and run on a wide range of configurations and platforms so system vendors can run the benchmarks easily.

The Oracle Applications Standard Benchmark represents such a workload, and the overall goals are two-fold. First, to provide specific focus for Oracle developers and field organizations on how to make Oracle Applications suite perform optimally for our customers, and secondly, to increase the amount of collateral available to customers who want to understand sizing and capacity requirements for a variety of hardware platforms.

In April 2000, Oracle published the results for the first, audited OASB on a clustered system. Oracle8*i* Parallel Server 8.1.6 running on 1, 2 and 4 node Compaq ProLiant/ Windows NT achieved a 1.81 scalability with Oracle Applications 11.03, while using module partitioning to reduce the overhead of the disk pinging based protocol in order to reach higher scalability [1].

With Oracle9*i* Real Application Clusters and Cache Fusion technology, we eliminated the need for module partitioning by leveraging high-speed interconnect technologies. The Oracle Applications 11i Standard Benchmark was used to prove how well Oracle Applications can scale in a Oracle9*i* RAC environment, without the need of performing detailed workload analysis and module partitioning, or employing RAC specific configuration changes as in previous Oracle releases.

The Oracle Applications Standard Benchmark was performed on a 4-node HP L-Class cluster using Hyper Messaging Protocol (HMP) over HyperFabric II as the user-mode IPC protocol for Global Enqueue Service and Global Cache Service message traffic and block transfers. The audited numbers from the 2-node and 4-node benchmark yielded a scalability of 1.9.

Oracle Applications Standard Benchmark Overview

The Oracle Applications Standard Benchmark (OASB) is a realistic workload mix that accurately represents a common customer scenario, both with a high volume of OLTP users and a substantial batch component. The key is to ensure that the chosen workload mix is accurately reproduced as the benchmark is run on both large and small systems, with a wide range of operating systems and configurations. Over time we may introduce other related workloads and hence cover a larger set of the product suite, but initially we have focused on two of the most common subject areas across our ERP customer base: Financials and Supply Chain Management (SCM). Seven modules are utilized across these two subject areas, namely:

- Financials: Accounts Payable (AP), Accounts Receivable (AR), General Ledger (GL) and Fixed Assets (FA)
- Supply Chain Management: Purchase Orders (PO), Order Entry (OE) and Inventory (Inv)

We provide a broad set of 18 OLTP transactions across this set of modules. Given that most ERP users also have substantial batch components to their workload, we have also included seven batch jobs. Both the OLTP and batch mix are maintained to be consistent across arbitrarily large runs of the benchmark. The specific definitions of each transaction and their target rates were provided by functional consultants from Oracle who work with such ERP customers on a daily basis. The batch component consists of 4 reports, Autopost for outstanding journal entries, the postings of those journal entries themselves and an Inventory batch job that performs inserts. We have worked to ensure that the Read:Write ratio of the standard workload is quiet similar to that observed in true customers. The workload represents the normal daily activity of a mid-market sized company, with a day-time batch load of roughly 30%. The database used is a synthetic database of substantial size. Called the Vision++ database due to its connection to the Vision Demo database, it contains hundreds of thousands to millions of rows in all the tables utilized in benchmark. We do not populate the tables that are not used by this mix of transactions, and hence the overall size of the database is still manageable, being roughly 30 Gigabytes. However, if all tables were populated to the same degree then we would expect the database to be over 200 Gigabytes in size, and for comparison with customer databases this is probably a more relevant number.

Load-Driver Technology

To further simplify use of the benchmark kit we use a commercial load-driver tool, currently Mercury Interactive. This provides the simplicity of a fully GUI interface for collecting and evaluating results. Oracle has collaborated with the vendors of such tools such that we may now drive such a high-volume workload at the network layer which accurately reflects the entire three-tier nature of Oracle Applications. This technology module is now called the Oracle NCA component of Mercury Interactive' s load driver tools, and is available from them.

Run-Time Objectives

The first order results of the benchmark are user count and 90 th percentile response time for a given run. It is expect that users of the benchmark will optimize results by increasing user counts for each run of the benchmark until an unacceptable response time is observed. The audit criteria is that the 90 th percentile of response time for the run does not exceed four seconds, that the transaction rates for all users are maintained across the run, and that the specified mix of users is consistent. We also ensure that the batch component of the workload is effectively executed throughout the run.

We expect much of the benefits of using the OASB kit will come from more detailed analysis of results, results that include CPU and memory utilization on both the dataserver and the middle-tier during the steady state phase of the run, as well as Oracle database statistics such as UTLBSTAT/ULTESTAT reports. The target of a run is generally to configure your system such that CPU on the dataserver becomes the limiting resource and to evaluate what user count can be achieved with the dataserver CPU resources that are available.

Use Of Results For Sizing

The Oracle Applications Standard Benchmark is designed to provide experience and confidence that running high numbers of concurrent users is achievable both in the benchmark lab and for our customers. We believe that this workload will provide collateral that will prove useful in a sizing process, but that using benchmark numbers within the context of sizing is a somewhat subtle process, and should be performed by those with substantial experience in this area. In particular, since Oracle Applications is such a broad range of products and possible configurations, we expect that benchmark results should be used only as general guidelines for sizing, or else they should be scaled to make them more applicable to a customer with a well known but different workload mix. For example, a customer may have a different mix of modules, a larger or smaller daily batch load, and a larger or smaller database, and all these components can effect the load they see substantially.

Workload Scenario

Scenario is the term used for the detailed definition of the workload' s run-time characteristics. For example:

- How many users of what type will run?
- How often will they sleep and when will they be active?
- When will they complete?

In the Oracle Applications Standard Benchmark (as opposed to the earlier prototypes) we provide a fixed scenario for use with Mercury Interactive which should be used with little or no changes. The mix of transactions and the rate at which each user executes transactions both represent commonly seen scenarios from customers of Oracle Application, as reported by the consulting and field organizations of Oracle.

Atomic User Group And Base User Counts

A list of the transactions and their suggested completion rate is specified below in Table 1. To ensure that our workload characteristics remain constant across a variety of user counts we define the concept of an *Atomic Group* of users. The *atomic group* is the smallest number of users that can be added at any given time while still ensuring that the workload remains unchanged, specifically in terms of the percentages of each type of user in the mix. To run more users you simply add an integer number of atomic groups, and thereby the characteristics of the workload remain constant.

The atomic user group consists of the sum of all the base user counts specified in the table. The key components of a scenario are to build an atomic user group with this set of transactions with the specified base user counts.

Name of Transaction	Trans/HR/User	Base User Count
AP Enter Asset Invoice	6	2
AP Enter Inv Multi Dist	6	2
AP Invoice Dist Inquiry	7	4
AP Invoice Inquiry	7	3
AR Collections Report	4	1
AR Customer Summary	8	4
AR Insert Manual Invoice	6	2
AR View Customer Trans	6	2
FA Asset Inquiry	7	4
GL Account Inquiry	10	4
GL Journal Entry	8	4
INV Detail Report	4	1

INV Insert Misc Trans	9	4
INV View Item Attribs	8	3
INV View Item Cost	8	3
OE Detail Report	4	1
OE Insert Order	8	3
OE View Order	7	2
PO Detail Report	4	1
PO Enter Purchase Order	6	2
PO New Purchase Req	6	2
PO View Purchase Order	6	2
Total	145	56

Table 1: The Atomic Group

In the case where it is desired to add more users but there are not enough system resources to add an entire atomic group you may add individual users, but only within a specified order as specified in the documentation for the kit. Adding individual users in an order other then this will result in an invalid run.

Transaction Rates And The User Model

In the Standard Benchmark it is expected that little or no changes to the included scenario will be necessary. However, some explanation of the scenario characteristics will still prove useful. Overall one of the basic tenants of the benchmark is that the users rate of work is constant, independent of system load or response time. Therefore throughput of the benchmark is strictly a function of user count, not of response time. This is ensured primarily through the use of the iteration pacing function of the load-driver tools. The iteration pacing should correspond exactly to the minimum transaction rates specified above in Table 1. i.e., if a transaction is required to run at least 6 times per hour, then the iteration pacing is set to 10 minutes or 600 seconds. This should ensure that the minimum transaction pacing is maintained, and if for some reason that is not the case then a warning will appear in the log files of the run, stating that the iteration pacing setting has been violated. Such warnings are an indication that at least for that one transaction the minimum transaction rate was not achieved. The run may therefore be deemed invalid during an audit.

Timed Events And Response Time

Response time for a run of the benchmark is calculated as a 90th percentile of all the measured response times for the run. A *Timed Event* (TE) is anything that we have explicitly timed in the transaction scripts and for which we receive response times. The Timed Events included in the current kit represent events we deemed most significant in terms of the user experience for someone running the transaction manually. Saving an invoice or submitting a query are examples of such important Timed Events. These timed events are then summarized in the "overall_timed_event" which is shown at the end of the Transaction Performance Summary Report, one of the commonly used outputs from a benchmark run.

Runtime Usage

There are four stages to a benchmark run:

1. Ramp up - Users are logged in to the system
2. Settle Time - Wait for the resource usage of the new users to stabilize
3. Steady State - The benchmark run data is captured
4. Ramp down - The run is terminated and users log out

A complete benchmark run consisting of all four stages commonly consumes about an hour, though this time will increase as user counts get higher and the ramp up phase takes longer. An overview of this can be seen in Figure 1. For a benchmark run to be accurate, you restore the database to its original state before beginning a run. However, multiple iterations of the benchmark may be run without database restores during the initial testing phases.

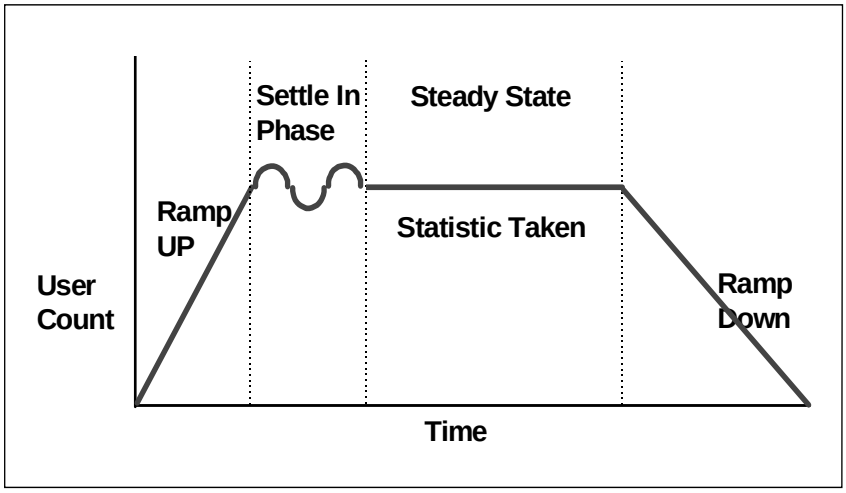

Figure 1:The four phases of a benchmark run.

During the ramp up phase, new users are started at a general rate of one atomic group every few minutes. The number of users that are initialized at any time is determined by a Mercury parameter which defaults to 5. If this initialization quota is set too high than the CPU usage during initialization is high enough that Mercury may allow some users to fail. Keeping this value near the default of 5 should allow the high overhead of log-in users to be amortized across the pauses. Using Mercury to delay the start of various groups or users within a group may also be effective. Settle time is the final opportunity to allow any spiky resource consumption to subside. Effective ramp-up and settle phases will result in a transaction load that is well distributed, therefore, allowing CPU utilization on the dataserver and middle-tier to be steady over time. These two phases should last roughly 20 minutes for user counts less than 1000, and will increase slowly as the need to start more groups increases.

The steady state phase is the true benchmark run where statistics and results are generated and collected. The first order results are the response time statistics generated for the window of time associated with the steady state phase. The only requirement for the steady state phase is that it is at least 30 minutes long. The steady state start and end time are chosen by the vendor, and this information is then provided to the auditors of a benchmark run. The "time filter" function of Mercury allows one to generate results that correspond only to this steady state phase of the benchmark, thereby ignoring the statistics generated during the ramp-up and ramp-down phases of the run.

Data Collection And Analysis

The data collected during a run should allow an observer to see the memory and CPU consumption on both the data server and the middle tier. This is commonly done through use of simple monitoring tools such as "sar", though the best tool will depend on the operating system is use. Additionally we require that utlbstat and utlestat are run - Oracle database monitoring scripts. The results of these scripts should provide direction in assessing what Oracle resources may be in contention. Finally the Mercury log file and results will allow us to verify the transaction rate of each user as well as how many, if any, users have failed during a run. These three pieces of data are then provided as input to the audit process. The report batch jobs will generate reports on disk that may take up disk space. These reports may be checked during auditing but don't need to be explicitly submitted to the auditors.

Auditing And Verification

Overall the audit process consists of the pre-audit, where the validity of the configuration and setup of the benchmark are evaluated, and the run-time audit, where the validity of the workload during execution is evaluated.

The pre-audit will address the following:

Setup and Installation:

- Versions for entire product stack, including load driver, middle-tier and data server.
- Initialization parameters for the database (init.ora)
- Database object count and schema verification

Source Code Validation:

- Validate that the Mercury scripts have not been changes, and that the provided scenario is unchanged except for think-time multipliers.

The Run-time audit will address the following:

- Verify that the overall 90th percentile of response time does not exceed the 4 second limit. Additionally the overall average should be below this 90th percentile value to ensure that the excluded 10% are not unreasonably large.
- Verify that each user process has maintained at least the minimum transaction rate during the steady state period.
- Verify that at least 90% of the batch requests submitted during the steady state of the benchmark have completed during that steady state period.
- Verify that no user threads have died during the steady state period.

Overall certification for the benchmark follows closely that of Oracle Applications in general - configuration and tuning of the benchmark must be with certified techniques and versions of the entire product stack. In this way we ensure that the tuning which is done to optimize running of the benchmark is directly applicable to improving the experience of our customer base.

Hardware and Software Configuration

Configuration Guidelines

Running the benchmark requires a full three tier setup, with a dataserver, fully configured middle tier and a load driver tier. On the following pages we show the complete hardware and software configuration used for this benchmark.

Database Tier

4 x HP L-Class, each with:

- 4 x 550 MHz
- 16 GB RAM
- 51 GB HP FC10 local disk
- 5.36TB HP FC60 shared disk storage among all four database servers
- HP UX 11.0, December 2000 Patch bundle
- Interconnect for RAC: HMP (Hyper Messaging Protocol or Lowfat) over HyperFabric II
- Interconnect to Application Tier: TCP over HyperFabric I
- Oracle9*i* Release 2 64bit, Real Application Clusters option
- Database size 40G
- ServiceGuard OPS edition A.11.09
- HyperFabric 9000 B.11.00.12

Application Tier

8 x HP N-Class, each with:

- 8 x 550 MHz CPU
- 32 GB RAM
- 34 GB HP FC10 local disk
- HP UX 11.0, June 2000 Patch bundle
- Interconnect to Database Tier: TCP over HyperFabric I
- Oracle Applications 11i Version 11.5.3
- HyperFabric 9000 B.11.00.12

Loadrunner Tier

6 x HP NT NetServer LP1000R, each with:

- 2 x Pentium III , 1GHz, 512K L2 cache per processor
- 4 GB RAM
- 2 X 36GB Ultra2 10Krpm disk
- Windows 2000 Advanced Server with Service Pack 2
- OASB Kit, Version 11.5.3
- Mercury Interactive LoadRunner 6.5

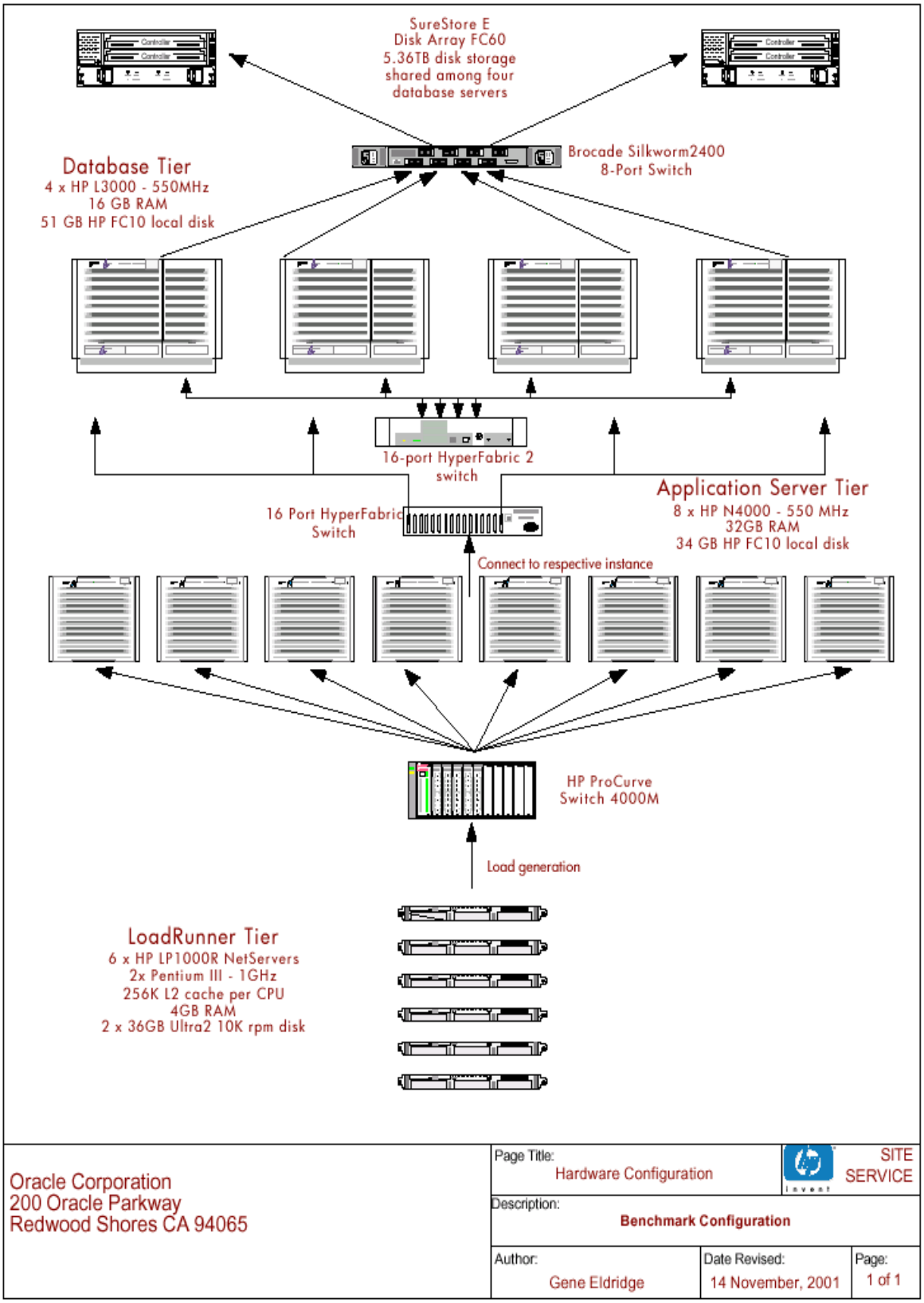

Figure 2: Benchmark Configuration

RAC specific configuration for the benchmark

Oracle Applications

With regards to configuring Oracle Applications to run on RAC, additional concurrent managers had to be created via the GUI to run on their respective Application server in the middle tier.

For each additional database server, follow the these steps [2]:

1. Configure middle-tier server node names for Concurrent Managers
    - Log onto Oracle Applications as System Administrator.
    - Navigate to Install - Nodes and enter node names for all nodes where batch processing will take place.
    - Register the nodes where the Concurrent Managers will run.

1. Create Internal Monitors for the additional nodes
    - Log onto Oracle Applications as System Administrator.
    - Navigate to Concurrent - Manager - Define
    - Create a new Concurrent Manager for each of the nodes created on step 2, but the node where the Internal Manager will run. The new manager(s) should be of type Internal Monitor (FNDIMON).

1. Update node name information for Internal Manager and Conflict Resolution Manager
    - On the same window as above, query all existing Concurrent Managers
    - Define primary and secondary nodes for the Internal Manager and for the Conflict Resolution Manager

1. Update node name for existing Concurrent Managers and create additional Managers as needed
    - Choose a primary node for all Concurrent Managers not updated previously
    - Duplicate the entry for the Standard Manager and the Inventory Manager as many times as the number of existent nodes; define exactly the same work-shift for these managers, change only node names and manager names; set cache size to 5.

1. Change shell scripts in the Application Tier
    - Edit the Oracle Applications environment variables files so that APPLDCP is set to “ON” on the node where the Internal Concurrent Manager (ICM) will run; for all other CM nodes, set APPLDCP to “OFF”.
    - Copy $FND_TOP/bin/dcpstart to the Oracle Applications executables’ owner home directory; edit that file so that TWO_TASK is properly set.
    - The node where the ICM runs should be able to execute a remote shell to the other nodes (i.e., you should configure .rhosts for that)
    - Always start the concurrent managers from the node where the ICM runs.

No other changes were made to the Oracle Applications setup or even the application.

Unlike the 11.03 OASB on OPS 8.1.6, the 11i/9i RAC OASB runs did not use any of the features below:

- DCP / PCP - Distributed concurrent processing
- MTS / CMAN - Shared servers and Connection Manager

LoadRunner

A Mercury LoadRunner scenario was set up that directs all atomic groups (one atomic group consists of 56 users / transactions) to all available database instances. No module partitioning was employed , i.e. all atomic groups were distributed evenly among all available instances. Due to a performance implication, the ARIMI transaction was routed to just one instance; ARIMI accounts for just 5% of all transactions. Also, GL Posting is designed to be a batch job that works the best with a bulk of journals. Therefore it is only running on one node / instance.

Database

The default locking scheme was used for the benchmark in order to get the full capabilities of Cache Fusion, i.e. we did *not* configure multi-block lock assignments by setting the GC_FILES_TO_LOCKS initialization parameter.

Please refer to Appendices F and G for list of all init.ora parameters used in this benchmark.

Benchmark and RAC specific tuning

Running a number of different benchmarks since Oracle9*i* Release 1 has provided us with a wealth of data which has in the end resulted in performance related enhancements and optimizations for Oracle9*i* Release 2. Thanks to those optimizations in the Oracle kernel, we did not have to perform any major benchmark or RAC related tuning.

A few parameter changes for the benchmark have been made with respect to the number of RAC related latches. These changes were necessary at the time of the benchmark, but are not required for Oracle9*i* Release 2 Production Release.

- KCL gc element parent latches
- KCL name table parent latch

Please refer to Appendices F and G for list of all init.ora parameters used in this benchmark.

Note: Underscore / Hidden parameters should only be set if instructed by Oracle.

General RAC tuning recommendations

Other RAC tuning techniques have proven to yield additional throughput by decreasing latencies and contention. If you are not experiencing the expected performance with RAC, those recommendations may help in diagnosing and addressing RAC performance related issues. Those general RAC tuning recommendations will be published in a separate Oracle technical paper. None of these techniques were applied to the OASB environment.

Oracle Applications specific tuning

The standard Applications tuning techniques below have shown to increase performance and throughput:

- Packages with a V$DB_OBJECT_CACHE.SHARABLE_MEM larger than a certain size (a good value to start with is 1 MB), can be pinned into the shared pool using the DBMS_SHARED_POOL.KEEP() procedure. This will avoid the flushing of larger packages and subsequent allocation of memory once the packages has to be reloaded. This technique drastically reduces contention on the shared pool and library cache latch.

- Increasing the cache size of Oracle Applications sequences has proven to reduce the number of SQ enqueue waits drastically. The cache size of the following sequences was increased:

OBJECT_NAME	NEW CACHE SIZE
AP.AP_BATCHES_S	200
AP.AP_INVOICES_S	200
AP.AP_INVOICE_DISTRIBUTIONS_S	200
APPLSYS.FND_CONCURRENT_PROCESSES_S	100
APPLSYS.FND_CONCURRENT_REQUESTS_S	200
APPLSYS.FND_LOGINS_S	500
APPLSYS.FND_TEMP_FILES_S	100
AR.AR_PAYMENT_SCHEDULES_S	200
AR.AR_RECEIVABLE_APPLICATIONS_S	100
AR.RA_CUSTOMER_TRX_LINES_S	500
AR.RA_CUSTOMER_TRX_S	200
AR.RA_CUST_TRX_LINE_GL_DIST_S	500
AR.RA_TRX_NUMBER_N1_204_S	100
GL.GL_JE_BATCHES_S	200
GL.GL_JE_HEADERS_S	200
INV.MTL_MATERIAL_TRANSACTIONS_S	500
PO.PO_DISTRIBUTIONS_S	200
PO.PO_HEADERS_S	200
PO.PO_LINES_S	200
PO.PO_LINE_LOCATIONS_S	200

Table 2: Sequence Cache Size adjustments

- Also, increasing the cache size of SYS.AUDSES$ reduces the number SQ enqueue waits during logon storms, and is very efficient in a multi-node environment.

- It is strongly recommended to apply the following Oracle Applications performance patches

  Benchmark kit required patches:
  - 1506443
  - 1565542
  - 1550583
  - 1643710
  - 1667204
  - 1786480
  - 1806021

Mandatory patches for Oracle9*i*
- 1761421

The correct steps to eliminate the view OE_TAX_EXEMPTIONS_QP_V in Oracle9*i* are as follows:

- Apply the pre-requisite Patch 1357858
- Apply the pre-requisite Patch 1743482 (this fact is not mentioned in the README, otherwise the OMIO transaction may fail subsequently)
- Apply Patch 1761421
- Apply Patch 1699929 to fix the wrong OM defaulting packages from Patch 1761421

- The table below shows Literal SQL in the Applications Benchmark kit and the associated bug numbers:

Bug	Sub-string of SQL statement	Memory used per cursor
1731867	SELECT CUST_PO_NUMBER ,to_char(INVOICE_	18K
1731867	SELECT to_char(SOLD_TO_ORG_ID) ,LINE_CA	50K
1731867	select distinct r.document_id from oe_at	555K
1281392	Insert into CST_INQUIRY_TEMP (SESSION_I	840K
1797335	SELECT SOLD_TO,ORDER_NUMBER,VERSION_NUMB	24K
1281428	SELECT CTL_PREV_LINE_NUMBER,CTL_PREV_UNI	43K

Table 3: Literal SQL

With these non-shared SQL statements, all free space in the shared pool will eventually be consumed (mostly by the SQL area). Applying the above-mentioned patches helps reducing hard parses and `latch free` wait events caused by library cache and shared pool latches.

Note: Patches for bugs 1281428 and 1797335 were applied for this benchmark.

- Using literal replacement has shown response time improvement for certain SQL statement at the cost of about 3% additional CPU. To enable literal replacement, do

```
ALTER SYSTEM SET CURSOR_SHARING=SIMILAR
```

after the shared pool is warmed up (all Applications benchmark scripts had to run once). This way, statements with the SAME literal don't undergo literal replacement. If the parameter is set in the init.ora, the "good" soft parse SQL like

```
SELECT SUBSTR(DESCRIPTION,1,50) ..
```

would be replaced by

```
SELECT SUBSTR(DESCRIPTION,:"SYS_B_00",:"SYS_B_01") ..
```

and this would just add CPU overhead to the benchmark.

- The number of certain, critical latches can be increased, if contention is seen when running Oracle Applications:

```
cache buffer hash chain latch
cache buffer LRU latch
library cache latch
enqueue hash chain latch
undo global data latch
```

- ST enqueue waits may show up primarily because of Oracle Applications' use of temporary tables (CREATE GLOBAL TEMPORARY TABLE) by the OMIO transaction. Tempfiles use temporary segments in the TEMP tablespace. Using temporary tablespaces effectively eliminate the serialization of space management operations (allocation and deallocation of sort segment extents). ST enqueue waits dropped 98% in the benchmark just by changing the TEMP tablespace to TYPE=TEMPORARY.

- In addition, increasing the NEXT_EXTENT storage parameter of fast growing tables / indexes and rollback segments will help in reducing ST enqueue waits. The objects below had their NEXT_EXTENT changed:

OBJECT_NAME	New NEXT_EXTENT
AP.AP_BATCHES_ALL	1M
AP.AP_INVOICE_DISTRIBUTIONS_ALL	2M
AP.AP_INVOICES_ALL	1500K
AP.AP_PAYMENT_SCHEDULES_ALL	1M
APPLSYS.FND_CONC_PP_ACTIONS	1M
APPLSYS.FND_CONC_PP_ACTIONS_N1	1M
APPLSYS.FND_CONC_REQUEST_ARGUMENTS	1M
APPLSYS.FND_CONCURRENT_REQUESTS	1M
APPLSYS.FND_ENV_CONTEXT	1M
APPLSYS.FND_ENV_CONTEXT_U1	1M
AR.RA_CUSTOMER_TRX_ALL	1M
AR.RA_CUSTOMER_TRX_LINES_U1	1M
AR.RA_CUSTOMER_TRX_N11	1M
BOM.CST_INQUIRY_TEMP	1M
INV.MTL_MATERIAL_TRANSACTIONS	3500K
INV.MTL_MATERIAL_TRANSACTIONS_TEMP	1500K
INV.MTL_ONHAND_QUANTITIES	1M
PO.PO_HEADERS_ALL	500K
PO.PO_LINE_LOCATIONS_ALL	1M
PO.PO_LINES_N5	1M

Table 4: NEXT_EXTENT adjustments

- Buffer contention was observed for blocks of the objects listed below:

  The majority of the busy buffers are from rollback segments, and the contention was seen as a wait for `buffer busy global CR`.

```
         OWNER          SEGMENT_NAME                SEGMENT_TYPE
-------------------------------------------------------------
         SYS          RBS1_02                        ROLLBACK
         ONT          OE_ORDER_LINES_U1              INDEX
ONT             OE_PRICE_ADJUSTMENTS_U1       INDEX
INV             MTL_MATERIAL_TRANS_TEMP_N1    INDEX
INV             MTL_MATERIAL_TRANS_TEMP_N2    INDEX
INV             MTL_MATERIAL_TRANS_TEMP_N10   INDEX
APPLSYS         WF_ITEMS_PK                   INDEX
APPLSYS         WF_ITEM_ACTIVITY_STATUSES_N1  INDEX
APPLSYS         WF_ITEM_ACTIVITY_STATUSES_PK  INDEX
AR              RA_CUSTOMER_TRX_LINES_U1      INDEX
PO              PO_REQUISITION_HEADERS_U2     INDEX
ONT             OE_ORDER_HEADERS_U1           INDEX
APPLSYS         FND_CONCURRENT_REQUESTS       TABLE
```

  When using the new Oracle9*i* Release 2 RowCR feature, we noticed a 80% reduction in waits and wait time for `buffer busy global CR`.

- Increasing number of rollback segments can help reducing the `buffer busy global CR` wait event since most of the buffer waits occur on undo segment headers of rollback segment owned by remote instances.

- For Oracle Applications 11i, a list of init.ora parameters needs to be set. Please refer to Appendix D "OASB 11i initialization parameters" for details.

Note: Underscore / Hidden parameters should only be set if instructed by Oracle.

HP/UX specific tuning

Kernel parameter settings

Async driver

Since the database was stored on raw devices, it is strongly advised to install the Async IO driver (/dev/async) and set the configurable kernel parameter `max_async_ports` to the maximum number of asyncdsk ports that can be open at one time, i.e. `max_async_ports` limits the maximum number of processes that can concurrently use /dev/async. As a guideline, set this parameter to the sum of `processes` from init.ora + number of background processes. The background processes started at instance startup will open /dev/async twice. If `max_async_ports is reached`, subsequent processes will use synchronous I/O [9].

Other recommended, configurable parameters

swapmem_on = 1

Page size setting for the Oracle executable

By default, the Oracle executable uses a value of "L" (=LARGE) for the page size setting. The setting can be verified with the `chatr` command. In the example below, the page size is already set to 4 MB.

```
$ /usr/bin/chatr $ORACLE_HOME/bin/oracle | grep data
      6 data 8000000100000000 ---m-   4M
```

This means that we first allocate 4 pages of 1 MB, then 3 pages of 4 MB, then 2 pages of 16M, which is a very drastic page size increase. If memory becomes scarce, the page size setting of the executable can be altered with

```
/usr/bin/chatr +pd <new_size> $ORACLE_HOME/bin/oracle
```

The tradeoff of reducing the page size is higher CPU utilization. With the lowest setting for the page size (4k, "D" setting), CPU utilization can be higher as much as 20%; with the highest setting "L", memory consumption is almost 50% higher than with the "4M" setting.
We have noticed that the page size 4M seems to be the most reasonable value in terms of memory and CPU usage: it will yield a memory usage of about 10.8 MB / user.

Page size setting	PGA Memory footprint
D (4k)	unknown
64k	8.1 MB / user
256k	8.7 MB / user
4M	10.8 MB / user
L	15 MB/user

Table 5: PGA Memory footprint per Page Size setting

Using SCHED_NOAGE policy

The SCHED_NOAGE policy gives processes holding a latch a fixed priority and makes them non-preemptable during this time. This causes less latch waits and latch sleeps, hence higher throughput.

To enable the SCHED_NOAGE policy for Oracle, the following init.ora parameter needs to be added:

`hpux_sched_noage=154` (for HP/UX 11.0)
`hpux_sched_noage=178` (for HP/UX 11i)

In addition, the system privileges `RTPRIO` and `RTSCHED` need to be added to the DBA group in `/etc/privgroup`:

```
dba RTPRIO RTSCHED
```

For additional requirements, please refer to Appendix A "Using SCHED_NOAGE policy".

Using HMP vs UDP

HMP is a reliable user-mode IPC protocol, thus we can avoid sending reliable side-channel messages, which are needed for reliability in case a non-reliable IPC protocol such as UDP is used for block transfers across an interconnect or network.

When using HMP, each Oracle shadow process will require an additional 300k of memory. The memory for the processes is pre-allocated at instance startup. In order to avoid memory wastage, the HMP related configuration parameters need to be adjusted to reflect the actual user count per instance.

For HMP to work with Oracle, the DBA group (or the group the application user belongs to) has to be granted the system privilege "MLOCK". This privilege needs to be set via the `setprivgroup` command:

```
setprivgrp dba MLOCK
```

To make the changes persistent over reboots, store this privilege in `/etc/privgroup`:

```
dba MLOCK
```

When using HMP **AND** the SCHED_NOAGE policy, the DBA group entry in `/etc/privgroup` would be set to:

```
dba MLOCK RTPRIO RTSCHED
```

Note: For HMP configuration information , please refer to Appendix B "HMP configuration".

Note: For instructions on how to generate an executable with HMP, please refer to Appendix C "Linking a new executable with HMP".

Workload profile

Generic

The excerpt from one instance's Statspack report of a 4-node run should give you a guideline on what numbers are achievable:

```
STATSPACK report for

DB Name         DB Id    Instance     Inst Num Release     Cluster Host
------------ ----------- ------------ -------- ----------- ------- -------
APPS           330860882 apps1               1 9.0.2.0.0   YES     hplcls1

             Snap Id     Snap Time      Sessions Curs/Sess Comment
            ------- ------------------ -------- --------- ----------------
Begin Snap:       8 01-Nov-01 21:53:16    1,994      84.7
  End Snap:       9 01-Nov-01 22:52:13    1,837      92.0
   Elapsed:               58.95 (mins)

Cache Sizes (end)
~~~~~~~~~~~~~~~~~
               Buffer Cache:       528M      Std Block Size:         8K
           Shared Pool Size:       944M          Log Buffer:    10,240K

Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:             97,887.10             14,274.60
              Logical reads:             27,648.69              4,031.93
              Block changes:                679.94                 99.15
             Physical reads:                 39.74                  5.80
            Physical writes:                 17.20                  2.51
                 User calls:              1,066.11                155.47
                     Parses:                182.36                 26.59
                Hard parses:                  1.32                  0.19
                      Sorts:                157.91                 23.03
                     Logons:                  0.66                  0.10
                   Executes:              1,642.52                239.52
               Transactions:                  6.86

  % Blocks changed per Read:    2.46    Recursive Call %:    72.90
 Rollback per transaction %:   21.91       Rows per Sort:    11.28

Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   99.52       Redo NoWait %:  100.00
            Buffer  Hit   %:   99.86    In-memory Sort %:  100.00
            Library Hit   %:   99.94        Soft Parse %:   99.28
         Execute to Parse %:   88.90         Latch Hit %:   99.55
Parse CPU to Parse Elapsd %:   11.91     % Non-Parse CPU:   99.99
          Wait Busy Ratio %:   91.70    CPU Busy Ratio %:    8.30

 Shared Pool Statistics        Begin   End
                               ------  ------
             Memory Usage %:   63.68   67.55
    % SQL with executions>1:   61.55   49.60
  % Memory for SQL w/exec>1:   69.69   58.06
```

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~
                                                          Wait     % Total
Event                                              Waits  Time (s)   Wt Time
-------------------------------------------- ------------ ---------- -------
global cache cr request                           781,115      6,786   36.71
library cache lock[1]                             306,030      3,854   20.85
global cache null to x                            103,793      1,990   10.76
db file sequential read                           213,208      1,637    8.86
buffer busy global CR                              71,776        848    4.59
```

RAC block transfer profile

The profile indicates the increase of global cache activity when moving from 2-node to 4-node, e.g. the amount of current blocks transferred increases linearly with the increase in number of nodes.

2-node run

```
Statistic                           Total     per Second    per Trans
--------------------------- ------------------ -------------- ------------
global cache cr blocks received       204,806           55.7          7.8
global cache cr blocks served         204,501           55.7          7.8
global cache current blocks recei      90,724           24.7          3.5
global cache current blocks serve      86,855           23.6          3.3
```

4-node run

```
Statistic                           Total     per Second    per Trans
--------------------------- ------------------ -------------- ------------
global cache cr blocks received       560,707          158.5         24.9
global cache cr blocks served         550,022          155.5         24.4
global cache current blocks recei     189,411           53.6          8.4
global cache current blocks serve     176,773           50.0          7.8
```

[1] The library cache lock optimization has not been implemented at the time of the benchmark audit.

Benchmark results and Scalability numbers

The 2-node and 4-node benchmark numbers were obtained during an audit in November 2001 at Oracle HQ. A single-node number was not published at the date and time of this writing.

For more information, please go to http:/www.oracle.com/apps_benchmark, and click on *Benchmark Results*.

Scale	Scalability factor
2 nodes -> 4 nodes	1.90

Table 6: Scalability number

Note: For all runs, memory was the constraint, i.e. we ran our of available RAM before maxing out available CPU resources or allowable response time.

Appendix A: Using SCHED_NOAGE policy on HP/UX

Overview

On HP-UX, most processes run under a timesharing scheduling policy. This policy can have detrimental effects on Oracle performance by de-scheduling an Oracle process during critical operations (e.g., holding a latch). HP has created a modified scheduling policy (referred to as "SCHED_NOAGE") specifically to address this issue. Unlike the normal timesharing policy, a process scheduled with SCHED_NOAGE will not increase or decrease in priority, nor will it be preempted.

Greatest performance gains are likely to be seen in on-line transaction processing (OLTP) environments (lab tests have shown improvements of 5 to 10%; customer results will vary). OLTP environments tend to cause competition for critical resources. Little or no gains are likely in decision support (DSS) environments since there is little resource competition. As each application and server environment is different, customers should test and verify that their environment will benefit from the SCHED_NOAGE policy.

Environments Supported

Hardware

all systems

HP-UX versions

11.0 (patches required: June 2000 patch bundle), 11i, and beyond

Oracle versions

8.1.7.2 patchset, Oracle9*i* (9.0.1), and beyond

Enabling SCHED_NOAGE for Oracle

System administrators must grant Oracle the ability to change its scheduling policy with the appropriate privileges and tell Oracle what priority level it should use when setting the policy. The steps to do this are:

1. The system administrator must grant the privileges RTSCHED and RTPRIO to the group that owns the Oracle executable. At most locations that group is DBA. The system administrator must give the following command as root:

```
setprivgrp dba RTSCHED RTPRIO
```

To make those privileges persistent over reboots, the system administrator must enter the following into the file /etc/privgroup:

```
dba RTSCHED RTPRIO
```

If the file /etc/privgroup does not exist, it should be created.

2. The Oracle DBA must add the parameter HPUX_SCHED_NOAGE to the INIT.ORA file of each instance. HPUX_SCHED_NOAGE takes an integer parameter. For HPUX 11.0 the valid ranges are 154 to 255. For HPUX 11i, the valid ranges are 178 to 255. Higher priorities are represented by lower values. For a discussion of priority policies and priority ranges, see the man pages for rtsched(1) and rtsched(2). Also see the documents at the HP documentation web site http://docs.hp.com [5].

Appendix B: HMP configuration

The file `/opt/clic/lib/skgxp/skclic.conf` contains the Hyper Messaging Protocol (HMP) configuration parameters that are relevant for Oracle [6].

```
attr CLIC_ATTR_APPL_MAX_PROCS                1300
attr CLIC_ATTR_APPL_MAX_NQS                  1300
attr CLIC_ATTR_APPL_MAX_MEM_EPTS             1500
attr CLIC_ATTR_APPL_MAX_RECV_EPTS            1300
attr CLIC_ATTR_APPL_DEFLT_PROC_SENDS         1024
attr CLIC_ATTR_APPL_DEFLT_NQ_RECVS           1024
```

Parameter description:

Parameter	Description
CLIC_ATTR_APPL_MAX_PROCS	Maximum number of Oracle processes. This includes the background and shadow processes. It does not include non-IPC processes like SQL client processes.
CLIC_ATTR_APPL_MAX_NQS	This is a derivative of the first parameter, it will be removed in the next release. For the time being, this should be set to the value of CLIC_ATTR_APPL_MAX_PROCS.
CLIC_ATTR_APPL_MAX_MEM_EPTS	Maximum number of Buffer descriptors. Oracle seems to require about 1500-5000 of them depending on the block size (8K or 2K). You can choose the maximum value indicated above.
CLIC_ATTR_APPL_MAX_RECV_EPTS	Maximum number of Oracle Ports. Typically, Oracle requires as many ports as there are processes. Thus it should be identical to CLIC_ATTR_APPL_MAX_PROCS.
CLIC_ATTR_APPL_DEFLT_PROC_SENDS	Maximum number of outstanding sends. You can leave it at the default value of 1024.
CLIC_ATTR_APPL_DEFLT_NQ_RECVS	Maximum number of outstanding receives on a port or buffer. You can leave it at the default value of 1024.

Table 7: List of HMP parameters

Appendix C: Linking a new executable with HMP libraries

Step 1: Generate a new Makefile:

```
$ cd $ORACLE_HOME/rdbms/lib
$ make -f ins_rdbms.mk rac_on part_on
$ make -f ins_rdbms.mk -n ioracle > make_HMP.sh
$ vi make_HMP.sh
```

Step 2: Search for `-lskgxp9` in `make_HMP.sh` and remove the string `-lskgxp9` from the link line
Step 3: Add the following strings in the place where `-lskgxp9` was present:

```
/opt/clic/lib/skgxp/libskgxph.a /opt/clic/lib/pa20_64/libclic_csi.a
```

The library `/opt/clic/lib/skgxp/libskgxph.a` contains the following object files:

- `skcsi.o`
- `skgxpu.o`
- `sskgxpu.o`

Example of the Makefile change:

- **OLD LINK LINE**

```
[…] -lodm9 -lskgxp9 -lskgxn9 -lclient9 […]
```

- **MODIFIED LINE**

```
[…] -lodm9 /opt/clic/lib/skgxp/libskgxph.a
/opt/clic/lib/pa20_64/libclic_csi.a -lskgxn9 -lclient9 […]
```

Step 4: Relink a new executable

```
$ sh make_HMP.sh
```

The Oracle executable in `$ORACLE_HOME/bin` is now linked with the HMP libraries [7].

Appendix D: OASB 11i Initialization parameters

```
#
# OASB 11i
#
nls_date_format                 = DD-MON-RR
nls_numeric_characters          = ".,"
nls_sort                        = binary
nls_language                    = american
nls_territory                   = america
utl_file_dir                    = /usr/tmp
o7_dictionary_accessibility     = true
#
# PL/SQL compatibility
#
event="10841 trace name context forever"
event="10932 trace name context forever, level 32768"
event="10943 trace name context forever, level 16384"
# the following event is a workaround for bug 1723171
# event="10933 trace name context forever, level 512"
#
# CBO parameters
#
optimizer_features_enable       = 8.1.7
optimizer_mode                  = choose
_optimizer_undo_changes         = false
_optimizer_mode_force           = true
optimizer_max_permutations      = 2000
_complex_view_merging           = TRUE
_push_join_predicate            = TRUE
_sort_elimination_cost_ratio    = 5
_use_column_stats_for_function  = TRUE
_like_with_bind_as_equality     = TRUE
_or_expand_nvl_predicate        = TRUE
_push_join_union_view           = TRUE
_table_scan_cost_plus_one       = TRUE
_fast_full_scan_enabled         = FALSE
_ordered_nested_loop            = TRUE
_optimizer_percent_parallel     = 0
query_rewrite_enabled           = true
_always_anti_join               = OFF
_always_semi_join               = OFF
_sqlexec_progression_cost       = 0
```

Appendix E: initapps1.ora

```
ifile=/home/apps/pfile/commonapps.ora
instance_number=1
instance_name=apps1
thread=1
ifile=/home/apps/pfile/initappsrbs1.ora
```

Appendix F: initappsrbs1.ora

```
rollback_segments=(rbs1_01,rbs1_02,rbs1_03,rbs1_04,rbs1_05,rbs1_06,
rbs1_07,rbs1_08,rbs1_09,rbs1_10,rbs1_11,rbs1_12,rbs1_13,rbs1_14,rbs1_15,
rbs1_16,rbs1_17,rbs1_18,rbs1_19,rbs1_20,rbs1_21,rbs1_22,rbs1_23,rbs1_24,
rbs1_25,rbs1_26,rbs1_27,rbs1_28,rbs1_29,rbs1_30,rbs1_31,rbs1_32,rbs1_33,
rbs1_34,rbs1_35,rbs1_36,rbs1_37,rbs1_38,rbs1_39,rbs1_40,rbs1_41,rbs1_42,
rbs1_43,rbs1_44,rbs1_45,rbs1_46,rbs1_47,rbs1_48,rbs1_49,rbs1_50,rbs1_51,
rbs1_52,rbs1_53,rbs1_54,rbs1_55,rbs1_56,rbs1_57,rbs1_58,rbs1_59,rbs1_60)
```

Note: The parameter files for instances 2,3 and 4 just differ by the instance number.

Appendix G: commonapps.ora

```
# Oracle9i RAC / OASB 11i benchmark
#
ifile=/home/apps/pfile/ifileapps11i.ora
# ------- RAC
#
cluster_database=true
# ------- RAC - END
#
# ------- MISC
control_files=(/dev/vg_apps_rbs/rapps_ctrl_1,
               /dev/vg_apps_rbs/rapps_ctrl_2)
compatible= 9.2.0.0.0
db_name = apps
db_block_size=8192
processes=1900
sessions=3800
enqueue_resources= 6000
open_cursors= 300
transactions= 1000
transactions_per_rollback_segment=20
dml_locks= 4000
db_file_multiblock_read_count = 4
db_block_checksum=false         # set to FALSE for benchmark ONLY
log_checkpoint_timeout = 900    # every 15 minutes
log_checkpoints_to_alert = true # log checkpoints to the alert.log
max_enabled_roles=100           # needed for Oracle Applications
max_rollback_segments=100
# ------- MISC - END
#
# ------- SGA
shared_pool_size= 800000000
shared_pool_reserved_size= 80000000
java_pool_size=20000000
sort_area_size=512000
sort_area_retained_size=512000
db_cache_size=480M
log_buffer=10485760
session_cached_cursors=100      # actually, 20 would be enough
cursor_space_for_time=true
# ------- SGA - END
#
```

(continued on next page)

(continued from previous page)

```
# ------- Latches
hpux_sched_noage=154             # HP/UX SCHED_NOAGE policy

# ------- Latches - END
#
# ------- TRACING
timed_statistics= false          # disable for audit
#-----------------------------------------------------------
# end of file
```

References

[1] www.oracle.com/apps_benchmark
[2] John Rimoli, Oracle, March 2001, E-Mail
[4] Kotaro Ono, Oracle, August 2001, RAC Performance Bulletin Board, Oracle Internal Web site
[5] System Platform Division, Oracle, Summer of 2001, E-Mail
[6] Velur Eunni Ramesh, HP, July 2001, E-Mail
[7] Vijay Sridharan, Oracle, and Mai Cutler, HP, May 2001, E-Mail
[8] Cecilia Gervasio, Oracle, July 2001, E-Mail
[9] HP: IT Resource Center Forums, http://forums.itrc.hp.com/

ORACLE®

November 2001

Technical Authors: **Stefan Pommerenk, Cluster and Parallel Storage Technology**
Nancy Lan, Data Server Applied Technology

Contributing Author: Bill Kehoe, Cluster and Parallel Storage Technology

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
650.506.7000
Fax 650.506.7200