Two Stress Test Tools for Oracle DBMS:

1) Orabm: an Open Source suite to test CPU performance under a CPU-intensive Oracle workload.

2) Orastress!: a tool to generate a multi-session, multi-instance, multi-mode workload from a single command line.

by Geoff Ingram (geoff.ingram@linxcel.co.uk)

About the Author

Geoff Ingram began his IT career in the 1980s after gaining a Physics degree at the University of Birmingham, UK. He has operated as a freelance Oracle consultant for several years, preceded by several years in product development at Oracle Corp, latterly as a principal software engineer. He is the author of "High Performance Oracle" (ISBN: 0471224367) published by John Wiley Inc. in August 2002:

http://www.amazon.com/exec/obidos/search-handle-url/index=books&field-author=Ingram,Geoff

Background Reading: the TPC-C benchmark

All DBAs should have at least some familiarity with the TPC-C benchmark, as Orabm and Orastress! are based on some of its facilities. The full TPC-C specification - including examples of the 5 TPC transactions and an input data generator (written in embedded SQL) - can be found at www.tpc.org.

A TPC-C compliant data-loader (Orabmload) is included, that implements all the functions in the specification. This data is the basis for both Orabm and Orastress!.

Introduction

Oracle DBAs and developers often find themselves in situations where they need to measure database performance against a baseline, typically to ensure that operational improvements deliver the expected benefits or to ensure that no regression in performance has occurred as a result of changes.

The changes in question may be due to a server upgrade, an Oracle upgrade, an operating system upgrade, or something more radical like a change in operating system or server hardware vendor.

This paper describes three command line tools to facilitate basic baseline performance information capture with minimal effort:

- Orabmload: loads TPC-C compliant data into a user-chosen number of Warehouses
- Orabm: stresses the CPU and memory of an Oracle DBMS server
- Orastress!: stresses the DBMS server with a multi-session, multi-instance load

Orabmload generates the test data for both Orabm and Orastress!. It's Open Source, and available as a Pro*C program. Appendix A includes instructions on how to build the executable from source.

These tools are not fully-fledged, all encompassing, or totally configurable. Instead they are designed to run out-of-the box and deliver results quickly, to provide a stake in the ground for the performance of a server running several different types of Oracle DBMS workload:

- CPU intensive workload - Orabm
- Mixed-workload Online Transaction Processing (OLTP) - Orastress!
- I/O intensive INSERT workload - Orastress!
- I/O intensive direct-path INSERT workload - Orastress!

Ready built executables for Windows are provided in all cases: all you need is a low end PC with Oracle9*i* client installed, and one or more Oracle databases (any 8*i* or 9*i* server platform) accessible by Oracle Net to get started.

Orabm transactions are based on the Stock-Level and Order-Status read-only transactions in TPC-C. By ensuring that the data fits totally in memory (through an appropriate Oracle configuration), Orabm can generate a CPU and memory intensive workload with almost zero physical I/O.

Orastress! runs in one of four possible modes, where the mode is provided through a command line parameter. These modes are described in the following table:

Command Line Arg	Description
READ	Runs the identical read-only workload as Orabm to stress CPU and memory.
OLTP	Uses all five TPC-C transactions: New-Order, Order-Status, Payment, Stock-Level and Delivery, in order to generate a mixed-transaction (SQL INSERT/UPDATE/DELETE) OLTP workload which stresses CPU, memory, and I/O.
INS	Runs an I/O intensive SQL INSERT workload by copying rows into the ITEM_INS table from the TPC-C ITEM table.
DIO	Runs an I/O intensive direct-path INSERT workload by copying rows into the ITEM_DIO table from the TPC-C ITEM table.

All tests run entirely on the database server without client-server traffic generation, and consecutive transactions run without any delay period: that's because the goal of the tests is to stress the DBMS server. In these respects the tests differ fundamentally from TPC-C, which simulates the behavior of a real-world application with real end users who introduce thinking and data-entry time delays between transactions.

Orabm

Orabm is a set of SQL scripts and command line program (Orabm) designed to help answer the question:

"how does my server perform under a CPU/memory intensive Oracle database workload"

The code is downloadable from:

http://www.linxcel.co.uk/orabm/orabm.tar

The Orabm data loader program (Orabmload) generates a TPC-C compliant set of data against which the test runs, and orabm runs the stress test itself.

Source code for both Orabm (orabm.c) and Orabmload (orabmload.pc) is available.

> **Note: Ready-built executables for Oracle9*i* on Sun Solaris, Linux, and Windows are provided – the Windows version means you can run the entire suite from any PC, even a low-end one, where Oracle client software is installed. The database can be any Oracle database accessible by Oracle Net.**

If you don't run Oracle on those platforms you can easily build your own versions from source, using instructions in Appendix A. POSIX threads support is required to build orabm, which runs multiple concurrent database sessions at once from separate threads in the same program.

The chief characteristics of a workload to meet the goal of stressing the CPUs are:

1. it runs entirely within the DBMS server thereby avoiding delays from client/server roundtrips.
2. there is no wait time between transactions
3. all transactions are read-only to avoid file-write operations
4. the set of tables which the stress test runs against fit entirely within a 200MB buffer cache

Provided that you allocate an Oracle System Global Area (SGA) with a 200MB buffer cache, the Oracle workload imposed by orabm should result in no physical I/O at all, once the data is cached - just logical I/O from the Oracle buffer cache, resulting in intensive use of CPU and memory.

Installation

All objects used by the stress test are owned by ORABM. Follow the steps below to install the objects and load the test data, making sure that ORACLE_SID is set to the database that you want to run the test against:

#	Operation	Command
1	create the ORABM user (assumes TOOLS tablespace, TEMP temporary tablespace)	sqlplus system/pwd @orabm_user
2	create the tables	sqlplus system/pwd @orabm_tab
3	load the data	$ orabmload Warehouses 1
4	create the indexes	sqlplus system/pwd @orabm_ind
5	analyze the tables and indexes	sqlplus system/pwd @orabm_analyze
6	create the stress-test PL/SQL procedures	sqlplus system/pwd @orabm_serverside_stress
7	cache the table and index data in the SGA	sqlplus system/pwd @orabm_cache

Note: you can optionally run Orabmload against a remote database by first setting the environment symbol LOCAL (Windows) or TWO_TASK (UNIX/Linux) to contain an Oracle Net alias where you installed the objects. Keep in mind this will be a lot slower than running the load from the server where the database is located.

Atfer completing the list, you can use the orabm_query_cache.sql script to display the approximate percent of each table's data and index blocks present in the block buffer cache. This should be close to 100%.

How Orabm Works

Orabm works by running a user-specified number of database transactions in each of a user-specified number of concurrent database sessions. The transactions are executed by the ORABM_SERVERSIDE_STRESS stored procedure, under the schema ORABM.

For each concurrent session, ORABM_SERVERSIDE_STRESS runs the number of transactions specified on the orabm command line, and returns the transactions per second (TPS) value for that session during the sampling interval on completion. To ensure that all concurrent sessions are processing transactions during the sampling interval, the TPS value only includes results from the middle 80% of transactions: the first 10% and last 10% are ignored.

The transactions are loosely based on the TPC-C Order-Status and Stock-Level transactions, using a predefined distribution of transactions. The transaction split, which is based on data returned by the DBMS_RANDOM package, should be:

Stock-Level:Order-by-Customer-Name:Order-by-Customer-Id
50%:30%:20%

The string returned by ORABM_SERVERSIDE_STRESS includes the transaction split during the test, to ensure that the transaction distribution is correct, subject to random fluctuations e.g.:

...sl=4042(50.5%) on=2384(29.8%) oi=1573(19.7%)...

Orabm Command Line

Once you have set up the test tables, data, and indexes, you're ready to run orabm. The following command shows orabm running 20000 transactions in a single session against the Oracle database identified by ORACLE_SID in the UNIX environment:

$ orabm 1 20000

This command line runs the same workload against a remote database identified by the Oracle Net alias linxceld1.co.uk from a Windows command box:

C:\> orabm 1 20000 linxceld1.co.uk

> **Note: running against a remote database has little (if any), affect on the transaction throughput, because all processing takes place on the DBMS server.**

Execution of a single Orabm session should show a single CPU at close to 100% utilization, provided that all table and index data is present in the Oracle block buffer cache and no other workload is running on the database server. On UNIX or Linux, you can use the "top" command to confirm this, or check that no "db file sequential read" event waits are taking place for the Oracle session using info in the V$SESSION_EVENT view - these indicate waits for physical I/O.

Alternatively, if your Oracle DBMS is running on Linux, you can use the gkrellm performance monitor to show that CPU utilization of a single CPU is at ~100% and no physical I/O is taking place. Gkrellm can be downloaded from:

http://web.wt.net/~billw/gkrellm/gkrellm.html

Here's an example of the command line you would use to run 10000 transactions against a local Oracle database for three iterations. In the first iteration, one session runs, in the second iteration two concurrent sessions run, and in the third iteration, six concurrent sessions run:

$ orabm 1,2,6 10000

Keep in mind that the specified number of transactions is run in **each** concurrent session.

> **Note: you should specify sufficient transactions such that the TPS results produced don't fluctuate significantly between runs for a given number of sessions; 100000 is a good value to choose.**

Results Output

Output is appended to a log file orabm.*database*.log, where *database* is either the ORACLE_SID or TNS alias that identifies the database where the test was run e.g. orabm.t92.log. For each iteration, the TPS value for each concurrent session appears between *begin* and *end* markers. For example, the following shows the contents of the log for two concurrent sessions - in this case the second iteration for the previous command line example - where txn(all) displays the total transaction count, and xn(sam) and t(sam) show the total transactions and time for the middle 80% of transactions for which sampling took place:

```
---begin sess=2 txn=10000 ORACLE_SID=t92 Fri Nov  8 20:31:48 2002
T92.WORLD txn(all)=10000 xn(sam)=7999 t(sam)=44 tps=182 ...
T92.WORLD txn(all)=10000 xn(sam)=7999 t(sam)=45 tps=178 ...
---end - Fri Nov  8 20:32:46 2002
```

The total TPS for this iteration is the sum of the TPS for the two concurrent sessions (182+178=360). A shell script (orabm_tps.sh) can be used to process output from the log on UNIX and Linux. The script aggregates the TPS values for concurrent sessions in a single iteration into a total TPS value for that iteration. The output based on the log info from the previous command line (3 iterations with 1, then 2, then 6 concurrent sessions) shows:

```
$ orabm_tps.sh orabm.t92.log
ORACLE_SID=t92 sess=1 tps=182
ORACLE_SID=t92 sess=2 tps=360
ORACLE_SID=t92 sess=6 tps=364
```

In this example the server was a 2 CPU model - as a result, 2 concurrent sessions running in orabm are enough to completely utilize all available CPU capacity. Additional sessions should result in the total TPS remaining unchanged, or even falling slightly as the operating system performs context switches to share the overloaded CPU resource between more ready-to-run sessions than available CPUs.

Orastress!

Orastress! is a Windows command line utility and Oracle package procedure designed to help answer the question:

"how does my server perform under a wide range of Oracle DBMS workloads"

The code is downloadable from:

http://www.linxcel.co.uk/orabm/orastress.zip

Note: to run Orastress! you need a Windows PC with Oracle9*i* client software, and one or more Oracle instances accessible via Oracle net.

Installation

Orastress! requires the same TPC-C schema used by the Orabm CPU stress test. The schema and objects should be installed as shown in the following table.

It's important to be aware that, while Orabm uses a TPC-C schema with a single Warehouse to ensure all data can be cached in memory, with Orastress! you can choose the number of Warehouses to be a higher value.

> **Note: the size of the data loaded scales with the number of Warehouses. A 10 Warehouse schema requires approximately 1GB of database space. If you already loaded data for Orabm, run the SQL "DROP USER ORABM CASCADE" first to prepare for Orastress!.**

Create the ORABM schema objects, then load and analyze the data as follows:

#	Operation	Command
1	create the ORABM user (assumes TOOLS tablespace, TEMP temporary tablespace)	sqlplus system/pwd @orabm_user
2	create the tables	sqlplus system/pwd @orabm_tab
3	load the data	$ orabmload Warehouses *n*
4	create the indexes	sqlplus system/pwd @orabm_ind
5	analyze the tables and indexes	sqlplus system/pwd @orabm_analyze
6	Ensure ORABM can run the DBMS_LOCK package	As SYS run: GRANT EXECUTE ON DBMS_LOCK TO ORABM

The Orastress! workload for each session is generated by the ORABM.ORASTRESS.STRESS package procedure. To create the tables required for the Orastress! insert tests, and install the package against the database identified by <your-TNS-alias> (where you loaded the test data), run:

```
sqlplus orabm/orabm@<your-TNS-alias> @orastress_tab.sql
sqlplus orabm/orabm@<your-TNS-alias> @orastress.plh
sqlplus orabm/orabm@<your-TNS-alias> @orastress.plb
```

Obtaining A Key

By default you can run one session per database at any time. This in itself provides useful information on server performance.

You need to register and get a key to create more than one concurrent load session per database.

To get a trial key free of charge, please complete the following online form. Your trial key details will be emailed to you shortly thereafter.

http://www.linxcel.co.uk/7505/10218_trialkey_orastress!.html

Your Serial number is displayed by running Orastress! without command line arguments from a DOS command window e.g.:

C:\orabm>orastress!
Serial: D5F8-C0A0

After receiving the key, install as follows by entering the Name and Key into the dialog box that displays when you run:

C:\orabm>orastress! register

Orastress! Command Line

Orastress! contains significant enhancements over Orabm:

- it provides 4 different types of workloads.
- it can execute against multiple instances in each iteration, from the same command line.

Command line usage:

orastress! –s *sess-iterations-list* –t *transacts-per-sess* –c *tns-alias-list* –m *mode* [-s *batchsize*]

where **transactions-per-session** in each iteration is run concurrently against all aliases in tns-alias-list
mode is one of **READ OLTP INS DIO**
batchsize is optional number of rows inserted per transaction for INS and DIO mode (default 10000)

Note: a COMMIT takes place after each batchsize insert in INS or DIO mode.

Note: when Orastress! runs in READ mode against a single Warehouse, the code executed is identical to Orabm.

Example 1:

orastress! –s 1 –t 50000 –c inst1.world –m oltp

Run one iteration - comprising 50000 transactions in 1 session - against inst1.world.

Example 2:

orastress! –s 2 –t 50000 –c inst1.world,inst2.world –m oltp

Run one iteration comprising 2 concurrent sessions of 50000 transactions on each of inst1.world,inst2.world giving a total of 4 concurrent sessions. The location of the instances specified by inst1.world,inst2.world is entirely down to the user. They could be:

- 2 instances of a RAC cluster
- different non RAC instances on the same server
- different non RAC instances on different servers (e.g. one on Linux, one on Solaris)

Example 3:

orastress! –s 1,2,4 –t 50000 –c inst1,inst2,inst3,inst4 –m read

Run 3 iterations each of 50000 transactions on each of four instances: inst1, inst2, inst3 and inst4, where the number of concurrent sessions in each iteration is:

1st iteration: 1 concurrent session on each instance, giving 4 sessions total
2nd iteration: 2 concurrent sessions on each instance, giving 8 sessions total
3rd iteration: 4 concurrent sessions on each instance, giving 16 sessions

Results Output

Output is very similar to Orabm, showing the transactions per second for each session in each iteration and the transaction split between the five TPC-C transactions for OLTP mode:

```
---begin mode=oltp sess=1 txn=5000 TNS=l Fri Sep 19 22:11:06 2003
LXD1.WORLD #1 w=3 txn(all)=5000 xn(sam)=3999 t(sam)=431 tps=92.8 end=190903-12:25:39
---end - Fri Sep 19 22:20:19 2003
```

You need to add the tps= values in each iteration session to give the total tps for that iteration.

Note: the instance id is given by #1.

Monitoring Performance During Execution

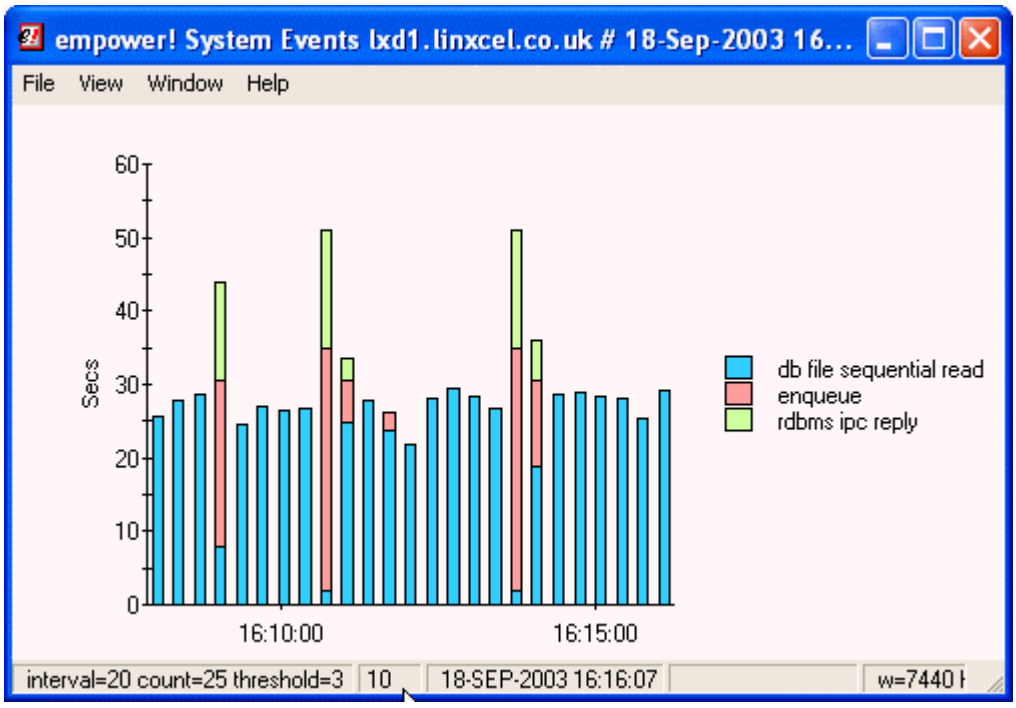
Orastress! in OLTP mode enables you to monitor performance via SQL during execution as follows:

```
-- use this SQL to show total Transactions/Sec (TPS) for a single instance
select sum(substr(module,instr(module,'#')+1)) tps from
v$session where module like 'oltp%#%';

-- use this SQL to show total TPS per instance for RAC
select inst_id,sum(substr(module,instr(module,'#')+1)) tps from
gv$session where module like 'oltp%#%'
group by inst_id

-- use this SQL to show total TPS cluster-wide for RAC
select sum(substr(module,instr(module,'#')+1)) tps from
gv$session where module like 'oltp%#%'
```

To convey the maximum information about performance during execution, a graphical presentation of wait events and statistics is required. For example, you can use the Event Profiles and Statistic Profiles options in the Enterprise Edition free trial of *empower!* to show wait events and statistics for all nodes in a RAC cluster on a single display. You can download *empower!* from:

http://www.linxcel.co.uk/empower/software/2.1/empower!.msi

The following screenshot shows significant amounts of waits for I/O ("db file sequential read") and row level locks ("enqueue") during a three-session iteration of Orastress! on Red Hat Linux ES 2.1, running on a single instance:



Reference Result

The following hardware and Oracle configuration produced ~100 TPS for a single session in repeated tests:

- Oracle 9.2.0.4
- Red Hat Linux ES 2.1 kernel 2.4.9-e.12
- 1 x Pentium P4 1.4GHz processor
- 1 GB ECC RAM
- database located on 10000rpm Ultra160 18GB SCSI drive
- 10 Warehouse schema (Orabmload – 50 mins load time)
- 8K blocksize database
- 3 x 100MB redo logs
- 500K redo log buffer

TOOLS tablespace definition as follows:

```
CREATE TABLESPACE TOOLS BLOCKSIZE 8192
DATAFILE
  '/u03/oradata/LXD1/tools01.dbf' SIZE 2000M AUTOEXTEND ON
  NEXT 1280K MAXSIZE UNLIMITED
  ONLINE PERMANENT EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K SEGMENT SPACE MANAGEMENT AUTO
/
```

Tip: set the TOOLS tablespace file size to ensure no autoextend takes place during the monitored interval.

Significant Oracle initialization parameters:

```
db_cache_size                  268435456
db_file_multiblock_read_count  16
log_buffer                     524288
log_checkpoint_interval        0
pga_aggregate_target           25165824
shared_pool_size               83886080
sga_max_size                   403772716
workarea_size_policy           AUTO
```

Appendix A – Building Orabm and Orabmload from source

If the supplied executables of orabm and orabmload don't work
for your OS or version of Oracle, you can create both from the source code.

- orabmload.pc is a Pro*C program.
- orabm.c is an OCI program.

Pre-requisites

For orabmload you need Pro*C installed (check for $ORACLE_HOME/bin/proc).
For orabmload and orabm, you need a C compiler. On Linux, gcc is available.

You can also download ready-built binaries of gcc for many other OS, including Solaris.

Note: The table and index creation scripts assume a TOOLS tablespace. For 10*g* this doesn't exist by default, so you need to create it.

To build orabmload

```
$ cd $ORACLE_HOME/rdbms/demo
```

Copy orabmload.pc to this directory.

Edit demo_rdbms.mk and add orabmload to the DEMOS symbol e.g.:

DEMOS=orabmload cdemo1 …

```
$ make -f demo_rdbms.mk orabmload PROC=$ORACLE_HOME/bin/proc
```

To build orabm

```
$ cd $ORACLE_HOME/rdbms/demo
```

Copy orabm to this directory.

Edit demo_rdbms.mk and add orabm to the DEMOS symbol e.g.:

DEMOS=orabm cdemo1 …

```
$ make -f demo_rdbms.mk orabm
```