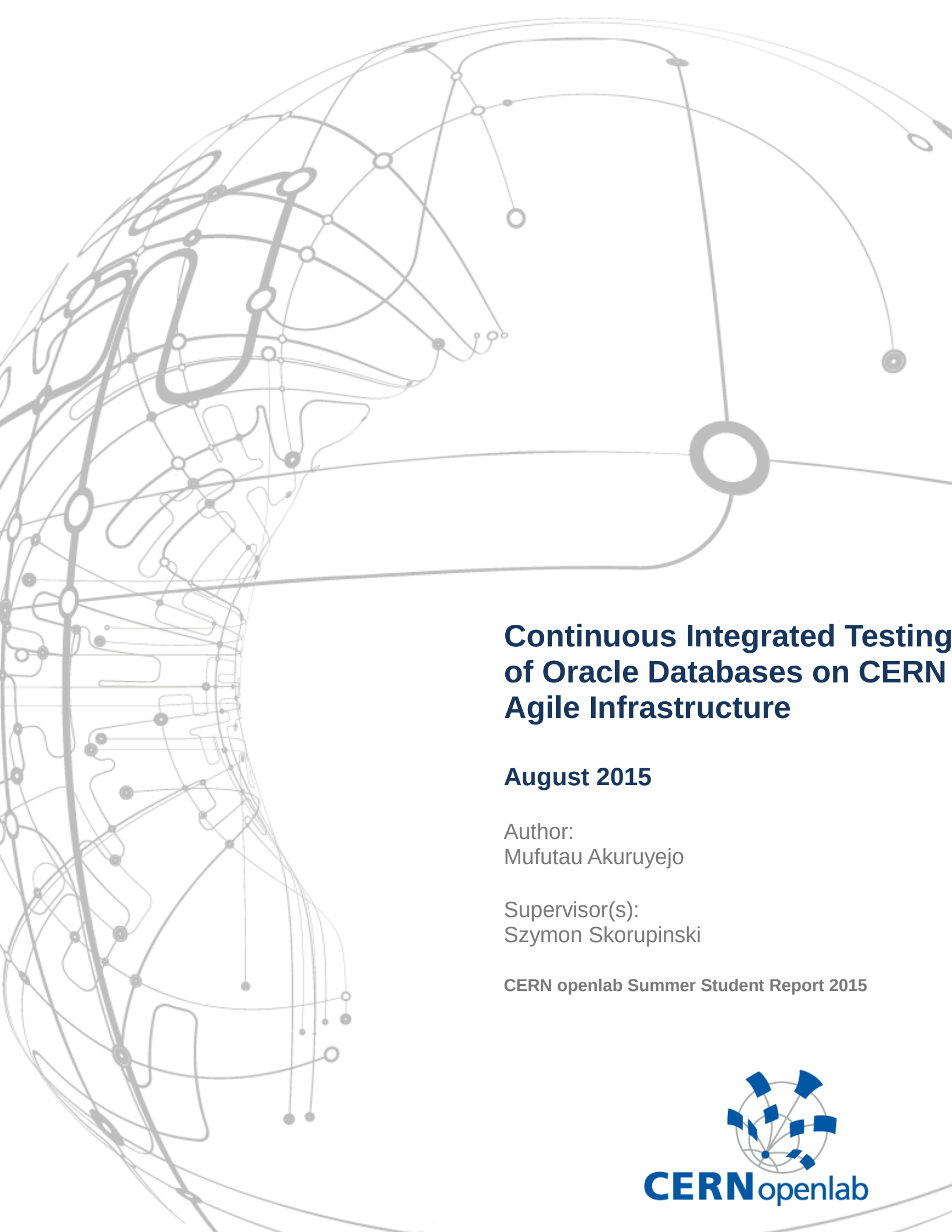


Continuous Integrated Testing of Oracle Databases on CERN Agile Infrastructure

August 2015

Author:
Mufutau Akuruyejo

Supervisor(s):
Szymon Skorupinski

CERN openlab Summer Student Report 2015

CERN openlab

Project Specification

With the introduction of Puppet configuring machines hosting Oracle databases managed by IT-DB Group of the European Organization for Nuclear Research (CERN), it is now much easier to apply any changes and perform the upgrades. Doing so on production environments is always a risk, which has to be addressed, by creating proper test environment. With Oracle Real Application Testing and related features it is currently possible to thoroughly test impact of any change, before implementing it on production and avoid virtually all problems.

The aim of the project is to add a new kind of testing i.e. the Oracle Real Application Testing (RAT) to an existing framework – DBTest. With the addition of this module, the DBTest framework would be more robust and be more able to detect performance issues in databases due to modifications. The author within the time frame didn't complete the RAT module but has written a significant part of the module. Also, the author made some improvements to the existing framework. Some of the improvements are starting a database before connecting to it, making the reporting of the framework easier to read and also enhancing the error messages.

Abstract

Changes occur to production environments regularly and they pose a risk to production. This could be patches to the underlying kernel, migrating to another distribution or an upgrade to the Operating System version. It could also be parameter changes to the database or a change in the vendor. Sometimes, this leads to degradation in the performance of the database. The author extended a framework for database testing by including a module that utilizes a testing tool provided by Oracle – Real Application Testing. The Real Application Testing (RAT) allows workload of a database to be captured and replayed on a test environment. This document explains the Oracle Real Application Testing then gives a brief overview of DBTest – a framework that automates testing of databases. After this, the author presents how to use this tool to help detect performance changes in databases.

The result is a tool that makes databases performance testing easy. The author made improvements to the original framework and also wrote the rudimentary code for the Real Application Testing framework. Hence, the original framework, DBTest can now also start databases before connecting to them amongst other improvements. It is the aim of the author that subsequently, the Real Application Testing module can be completed.

Table of Contents

1 Introduction

Making changes to existing production system has always been an issue. This is because many problems can arise when changes are made. A possibility is that the database performance can get degraded. This happens in a lot of cases where some aspects of the system would be affected due to applying a patch.

There are many performance simulation tools available, but there is a significant cost involved implementing them, especially licensing, procuring and configuration of servers, and time. In order to execute these tools users need to create scripts, develop queries and provide a range of parameters to run the load, but this may only provide a small part of the production work load, which will not simulate your actual production load. Moreover, despite such testing many issues often go undetected until deployed into production. Database Replay of Real Application Testing offers a comprehensive solution to the problem. It allows a complete capture of the workload of the production system. This can then be transferred to a test system and replayed as many times as wished.

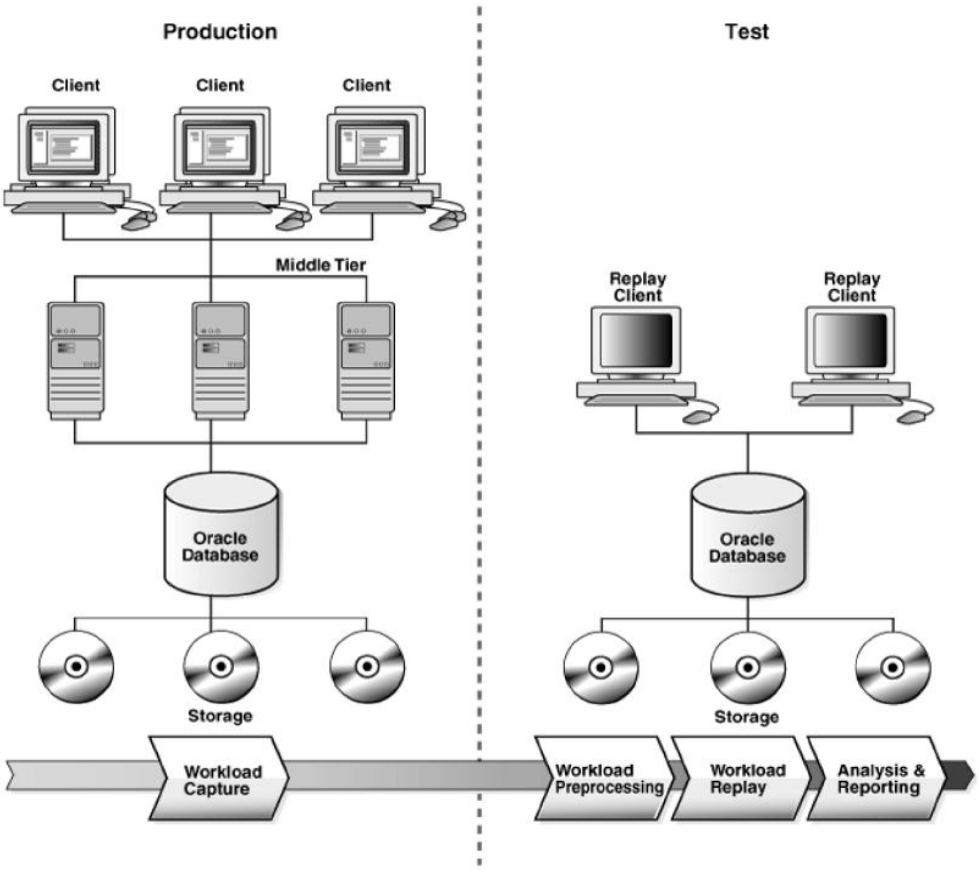
Some individuals attempt to perform tests using some third-party load generation tool that can run automatically to simulate real user activity. Many times, this is not a representative workload of your production system but rather merely the running of a small part of your production workload executed several times. A framework named DBTest has been created that enables automatic testing of databases using three tools:

1) Silly Little Oracle Benchmark (SLOB)
2) Database Benchmark Test
3) Real Application Testing (RAT)

There are some templates provided in which the user inputs what is needed. The templates provide customization allowing the framework to be versatile and easily re-usable in different environments and contexts. Another key feature of the tool is that it allows the testing of databases that reside on a remote machine. This makes the tool very useful for organizations that have a large number of databases and clusters running on many hosts – such as CERN – as there wouldn't be a need to install it on all the instances. This saves time and cost for the user.

2 Oracle Real Application Testing (RAT)

Database replay was introduced in Oracle 11g R1, and it allows users to capture workloads on a Production system and replay them in your Development or test environments while maintaining the unique characteristics of the workload. The workload consists of the SQL queries, the datafiles, control files and other parameters that make up the database. It is hence a faithful reproduction of the production system. This enables the user to test a variety of system changes such as hardware/software migration, operating system/ database upgrade, patches, database configuration changes etc. and eliminate the risks before implementing into production.



The database replay component of Oracle Real Application Testing has four main steps:

1) Workload Capture
2) Workload Pre-processing
3) Workload Replay
4) Analysis and Reporting

The aim of the project is to automate the process of replay and analysis and reporting (i.e. the last two steps). Each step is described next.

2.1 Workload Capture

In this phase users will capture the production workload from the database. Enabling the workload capture involves tracking and recording all external requests from users, application etc. and changes will be stored in binary files called capture files. These files will have information such as System Change Number (SCN), SQL Text, bind variables, etc. The SCN is an internal number maintained by the database management system (DBMS) to log changes made to a database. It increases over time as changes are made to the database. It thus enables recovery to a point of interest by specifying the SCN.

The steps involved in Workload Capture are described below.

On all nodes (machines) as root:
Change to root and make directories for the RAT. This is where all the files involved in the capture and replay would be stored. Change the user to oracle and mount /ORA/dbs77/RAT/
Then set environment properly, e.g. set_pdbr_rac50 mkdir -p
Check if RAT is possible on the machine
Create directory object in the database to point to our directory in the system create directory
Double-check if there is anything else running
Delete any old capture found
Check existing filters
Create filters to exclude capture of backup and monitoring operations
Start capture
Monitor the process
#Generate capture report.
Export the AWR (Automatic Workload Repository) snapshots associated with the ID

After these steps have succeeded without errors, you then restore the database and do some preparation for replay. The steps are listed below:
Generate information about existing configuration - on production
Create audit directory on a node
Mount NAS volumes with backups - on both nodes:
Restore control files
Unmount backup volumes on both nodes
Open the database in restricted mode and disable scheduler sqlsys shutdown immediate;
Remove all database links
Drop all network ACLs to prevent impacting production services during replay. Note that you should remove both connect and resolve privileges to each user that was listed by the query.
Check invalid objects and recompile them if there is difference between test and production
Add database to CRS (Cluster Readyware Service)
Create snapshots
List if there are already any snapshots for specified volume
Create the snapshots for all volumes containing: datafiles, redolog files and controlfiles
Prepare users. This could involve changing the password if necessary

2.2 Workload Pre-Processing

At this stage, the workload has been captured and some pre-processing is needed for it to be replayable. The pre-processing could be done on the production system itself or on the test but it's preferred for the pre-processing to be in the test to limit the impact on the production system.

Pack the replay files into a tar ball. This step is performed so that if need be just in case the replay needs to be processed and replayed multiple times
If needed - generate subset of captured workload. The tool is very flexible. The user can decide to only process a subset of the work.

The user then needs to determine answers to some questions here to find the optimum. Questions such as: how long pre-processing should take, names to be used for the files, how to map entries and strings, etc.

After this is done, the user needs to carry out some cleaning up
#Then, initialize replay
-- Check information about replay
-- Recompile all invalid objects
Create testing baseline snapshots
Create final baseline snapshots - only after testing is finished

2.3 Workload Replay

Everything in this section should be performed as user oracle. It's also important to set the necessary environmental variables.

There are two ways in which the workload can be replayed: as benchmark or not. The difference is that in the former, there is no benchmark to compare the replay to and hence, the particular replay serves as a benchmark for subsequent replays. In the not-benchmark mode, the particular test is compared to a previous benchmark.

Consequently, the steps needed to be executed are listed:

Mount the volume

Initialize all needed variables. The variables are listed below:

1) Path to the capture files. This should be the absolute path
2) Name of the database
3) Name of the snapshot (if the database is to be restored to an earlier time)
4) Mount point
5) The TNS (Transport Network Substrate). This contains the parameters such as the address of the database, the port, protocol amongst others
6) The path where the diagnostic files.
7) The hosts where the database resides, etc.

If not –benchmark mode, then get the base replay directory number
Entity name of the machine where test database is located

Check if database is running and stop it if it is the case
Clear all previous logs
If -benchmark - remove all replays directories and files
Else if not -benchmark - remove all previous replays dirs, apart from baseline. The baseline directory number is usually stored in the database.
Remove old wrc (Workload Replace Capture) clients output. The wrc is the capture files of a capture session and previous sessions can be safely deleted
If the previous steps complete successfully, start the database
Before replay, restart all services to ensure that they are running on preferred instances:
Check if no replay is running
Prepare replay with the query below:

```
exec    dbms_workload_replay.prepare_replay(synchronization    =>    'SCN',
connect_time_scale     =>     100,          think_time_scale     =>     100,
think_time_auto_correct => TRUE);
```

# Check if replay correctly prepared
# Calibrate replay to determine how many clients are required:
select dbms_workload_replay.calibrate('CAPTURE_DIR') from dual;
This results in a clob(character large object) file with XML content. Here is a sample output:

```
  <output>
     <max_concurrent_sessions>102</max_concurrent_sessions>
     <total_sessions>2414</total_sessions>
-->     <replay_clients>3</replay_clients>
     <replay_cpus>1</replay_cpus>
     <mem_per_client>127.5</mem_per_client>
   </output>
```

The needed data to be taken from here is the number of replay clients (which is 3 in this case). This is needed for the next step.
Start replay clients as many in total as reported above
Check if correct number of workload clients connected, if not exit with error
Start replay with this query:

```
exec dbms_workload_replay.start_replay();
```

Wait until replay is finished
After this completes successfully, new replay subdirectory created. Its name contain the replay directory number

```
select id, replay_dir_number from dba_workload_replays;
```

ID	REPLAY_DIR_NUMBER
1	767587594

Save the id and replay_dir_num as replay_id and curr_dir_num respectively

#If –benchmark, save the curr_dir_num as also base_rep_dir and store it in the database.

2.4 Reporting Phase

At this phase, we generate a report of the replays that have been undertaken. It allows many customizations to fit the need of the user. This is because the report generated is comprehensive giving the user a wide range of choice of the parameters to monitor

-- Generate replay report

```
select dbms_workload_replay.report(1, 'HTML') from dual;

select dbms_workload_replay.report(1, 'TEXT') from dual;
```

If not -benchmark

Populate capture and replay database views and store its output as capture_id

- Generate compare replay reports and store the resulting output in 2 files: one in an XML and the other in an HTML format

```
exec    dbms_workload_replay.compare_period_report(replay_id1    =>    11,
replay_id2 => 1, format
=> DBMS_WORKLOAD_CAPTURE.TYPE_HTML, result => :v_clob);
=> DBMS_WORKLOAD_CAPTURE.TYPE_XML, result => :v_clob);
```

#stop database

If -benchmark

Copy diag directory and /tmp/wrc* files for further reference to rep directory

Grep and count alert.log errors and wrc files errors - save them in the database as wrc_err and alert_err

Else if not -benchmark

Parse the XML output of compare period report to get numbers - e.g. Database Time of 1st and 2nd replay and whatever is of interest

Count alert log errors and wrc files errors - compare them with stored values

Each phase of Real Application Testing involves many steps. Hence, the aim of the project is to automate the last two phases: Workload Replay and Reporting Phase. The tool for achieving that is a framework called DBTest. DBTest is a framework for automated testing of databases. The two tests already implemented in it are the Silly Little Oracle Benchmark (SLOB) by Kevin Closson and Database Benchmark Test by Oracle.

3 DBTest

DBTest is a framework for testing of databases. It is modular and consists of check classes that can be used to detect change in performance after there is modification to the underlying environment of the database. The script is written in Python.

3.1 Architecture

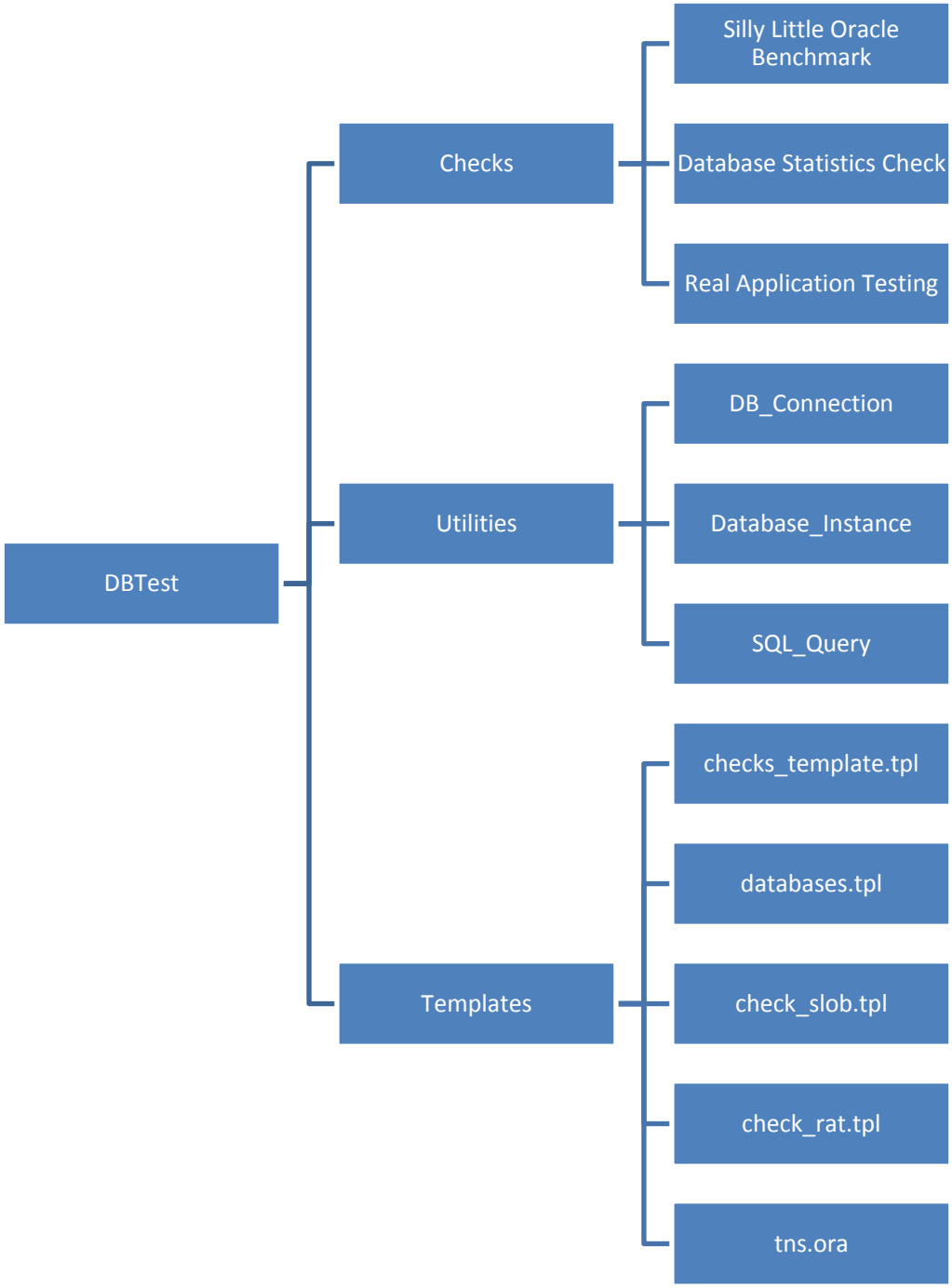


Figure 1: DBTest architecture as an horizontal hierarchy

The checks, utilities and templates would hereby be described briefly.

3.2 Checks

The checks are the performance tests that can be performed on databases. There previously were only two tests implemented – Silly Little Oracle Benchmark (SLOB) and Database Benchmark Test. An aim of the project was to add a new module i.e. the Oracle Real Application Testing.

3.2.1 Silly Little Oracle Benchmark

The check runs a SLOB test with parameters specified by the user. The check updates some parameters - and then it executes a SLOB test. The results are taken from the awr.txt report file in the SLOB folder and then compared the baseline in the database. The values that will be considered from the awr.txt file are specified in the slob_check.tpl in the templates folder

3.2.2 Database Statistics Check

The check runs a database internal test and then collects the output of these tests and compares them to the benchmark. A test repository contains a table. In the table are stored the input and benchmark parameters.

3.2.3 Real Application Testing

This is the new check implemented and it runs the last two phases of the Oracle Real Application Test i.e. the workload replay and reporting phase. The values that would be considered are specified in the rat_check.tpl file in the templates folder.

3.3 Utilities

These are the modules that aid the checks to perform their functions. They are python classes that are easily reusable and hence put on a class of their own so that they can be re-used by the different modules.

3.3.1 DB_Instance

DB_Instance is a wrapper over the cx_oracle connector for connecting to database. It provides a cleaner syntax compared to the original API. cx_oracle.so is provided by Oracle to enable

3.3.2 Database_Connection

This module uses as a basis, the DB_Instance class. It provides the ability to connect to the database trying with multiple host names i.e. when there are multiple database names

and, this module tries connecting to them at a time and creates connection to the first successful database.

3.3.3 SQL_Query

This module adds more functions to the SQL Query connector of Python making the code more readable

3.4 Templates

They are a key part of DBTest and they allow the user to customize the tool to the need. They are hereby described further.

3.4.1 checks_template.tpl

The checks_template.tpl is a template file that contains the names of the tests to use in a key-value format. The name of the check is the key while an integer is the value. Any value greater than zero means the particular check should be used.

3.4.2 databases.tpl

The databases.tpl is a configuration file that contains the names of the databases to run the tests against. The databases listed here are tested.

3.4.3 tns.ora

For the databases in the databases.tpl, the tns.ora contains their TNS values. TNS is Transport Network Substrate and it contains the details of how to connect to a database.

3.4.4 check_slob.tpl

In the check_slob.tpl, the user specifies the parameters that are wanted when running the Silly Little Oracle Benchmark (SLOB).

3.4.5 check_rat.tpl

The template check_rat.tpl contains the parameters that the user is interested in when running the Real Application Testing.

3.5 Improvements made to the existing framework

Some of the improvements the author made to the existing framework include:

1. Making the code better conform to PEP 8 specifications
2. Starting a database before trying to connect to it.
3. Improved the documentation of the code
4. Improved the error handling

5. Added some methods to existing classes to increase modularity and reusability. This includes the SQL query class and database connection class.

4 Usage of the tool

There are two modes which the tool can be used:

1) Benchmark: This is when the particular test to be performed is to serve as a benchmark for subsequent tests. This means that the values of the parameters that are gotten from the parameters are stored as benchmark in the database.
2) Not benchmark: In this mode, the tool would assume that there is a benchmark. The tool when making a report would compare the performance characteristics of the current run to the stored benchmark.

Before usage, set the DBTEST_HOME environment variable to the folder containing DBTest.

```
export DBTEST_HOME = /home/user/DBTest
```

Also, the Oracle environment must be properly set in order to be able to connect to databases. The command must also be run from user oracle
Invoke command:

```
python main.py [benchmark]
```

Outline of how DBTest (RAT module) works

- Get the username from a file named login_data. Also, the corresponding password is gotten by sending the username as a parameter to a script. This is the login that would be used in connecting to the databases which are to be tested.
- The TNS of the databases are gotten from tns.ora file. The TNS contains information such as name of host, protocol to be used, address where the database resides and other connector descriptors needed to connect to the database.
- The test repository is the database where the input and benchmark parameters for each of the checks are located. A connection is made to the test repository.
- The databases to be tested are located in the template databases.tpl. The template is parsed and the database names are gotten there. A connection is then made to each of the databases. The database is first started if it is not already started. After this, a connection is then made to the database and it is added to a list containing the list of databases to be scanned.
- To perform the actual checks, a method is invoked (run_checks). A parameter is passed to the function to indicate whether the test is a benchmark run or not.
- In the file checks_template are located the implementd checks and a number. If the number is greater than zero, the check is then performed on the databases list that has been populated earlier.
- After all the tests have been performed, a report is generated that gives a summary of the checks that were performed on the databases and its analysis.

5 Conclusion

Risks that occur due to change in production environment can be mitigated by capturing the exact activities of end users into Database Replay tool and then replaying them on a test system. The author in the time span given made significant improvements to the existing framework and also wrote the rudimentary classes and methods for the check_rat module. It is hoped that this would be completed. Nevertheless, with this tool, performance degradation in databases due to changes in the environment can be significantly reduced.

6 Bibliography

[1] Oracle Corporation (2014, June). Database Manageability [Online]. Available http://www.oracle.com/technetwork/database/manageability/database-manageability-wp-12c-1964677.pdf

[2] Oracle Corporation (2008, August). Real Application Testing User's Guide on Oracle Database 11g [Online]. Available http://www.oracle.com/technetwork/oem/grid-control/overview/owp-real-application-testing-11g-1-129463.pdf

[3] Oracle Corporation (2013). Real Application Testing on 12c [Online]. Available: http://www.oracle.com/technetwork/database/manageability/real-application-testing-wp-12c-1896131.pdf

[4] John O'Donahue "Java Database Programming Bible", 1st ed. West Sussex: John Wiley & Sons 2002

[5] Server overview [Online]. Available: https://docs.oracle.com/cd/E11882_01/server.112/e41481/toc.htm

[6] https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

[7] Silly Little Oracle Benchmark: http://kevinclosson.net/2012/'02/06/introducing-slob-the-silly-little-oracle-benchmark.

[8] Ian Abramson, Michael Abbey, Michael Corey, "Oracle Database 11g, a beginner's guide" Oracle Press, New York

[9] http://www.oracle.com/technetwork/articles/sql/11g-replay-099279.html

[10] http://allthingsoracle.com/oracle-database-replay-for-your-workload-test/

[11] https://twiki.cern.ch/twiki/bin/viewauth/DB/Private/DBTestDocumentation

[12] James O Knowlton, "Python: Create, Modify, Reuse", 1st. ed. Wrox Publishers 2008