Agile Upgrades

Modern and progressive updates keep pace with change.

Features:

Migration to 12c Made Easy Using Replication Technology

DBA 201: Mitigating Risk in Database Upgrades

Using RAT Features in Oracle Database 12c to Replay Previously Captured Workloads
Regression Analyzed, Tout de Suite: Leveraging Oracle 12c Database Real Application Testing (RAT) Suite, Part 2

By Jim Czuprynski  ■  Nick Marcovecchio, Editor

The first article in this series discussed the new features of RAT in 12cR1 and illustrated how to capture an actual application workload from a pre-12c Oracle database for eventual replay and analysis against an Oracle 12c database.

To recap where Part 1 of this article series ended, we finished using Database Workload Capture to collect two different workloads from an Oracle 11.2.0.3 database — one OLTP workload, and one DSS workload — and saved the resulting shadow capture files for eventual transference to a new platform for Database Workload Replay testing and evaluation. The results of several queries against statistics for those Database Workload Capture sessions are available here.

In this second and final part of this series, we illustrate how to use the latest features of RAT in Oracle Database 12c to replay the workloads previously captured to determine if there will be any significant variance in application performance, as well as how to detect those variances and determine possible solutions.

Replaying a Workload: Preparations

To prepare for Database Replay operations against the new database system — an Oracle Database running the latest beta version of Release 12cR2 — I transferred the generated Database Workload Capture files via SFTP to two separate directories on the new destination database’s server. I also created a separate directory to hold just the results of the consolidated workloads’ execution and any Database Replay activity.

I then set up the destination database environment for replay using the code shown in Listing 1 to create corresponding directory objects for the DSS and OLTP workloads.

```
-- Create directory objects
-------
DROP DIRECTORY DBRCONS;
CREATE DIRECTORY DBRCONS AS '/home/oracle/DBRControl';
GRANT READ, WRITE ON DIRECTORY DBRCONS TO PUBLIC;

DROP DIRECTORY DBCOLTP;
CREATE DIRECTORY DBCOLTP AS '/home/oracle/DBRControl/OLTP';
GRANT READ, WRITE ON DIRECTORY DBCOLTP TO PUBLIC;

DROP DIRECTORY DBCDSS;
CREATE DIRECTORY DBCDSS AS '/home/oracle/DBRControl/DSS';
GRANT READ, WRITE ON DIRECTORY DBCDSS TO PUBLIC;

-- Create WRC user account for Workload Replay Client isolation
-------
DROP USER wrc CASCADE;
```

The code shown in Listing 1 grants PUBLIC read and write access to the directories created in Listing 2. This is because the Workload Replay client uses a user account named WRCLNT_USER to process the database workload, and this account is typically linked to a local user account on the database system. This grants the WRCLNT_USER account access to both directories for replaying the OLTP and DSS workloads.
CREATE USER wrc
  IDENTIFIED BY wrc
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp
  PROFILE DEFAULT
  QUOTA UNLIMITED ON users;
GRANT CONNECT, DBA TO wrc;

Listing 1: Preparing for Database Workload Replay

Note that I also created a new Oracle database user account named WRC that I’ll use to invoke multiple Workload Replay Client sessions. Using this separate account is a recommended best practice to make it easier to isolate any session(s) that are only involved in replaying workloads against the destination database.

Combining Database Replay Workloads

Next, I’ll combine the OLTP and DSS workloads into a single workload schedule using Oracle 12cR1’s new BEGIN_REPLAY_SCHEDULE procedure of package DBMS_WORKLOAD_REPLAY. The code to perform their combination is detailed in Listing 2.

```
DECLARE
cpt_id NUMBER;
BEGIN
  DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE(    capture_dir => 'DBCOLTP');
  DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE(    capture_dir => 'DBCDSS');
  DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY(    replay_dir_obj => 'DBRCONS'    ,schedule_name => 'TEST2DESTRUCT');
  cpt_id :=
    DBMS_WORKLOAD_REPLAY.ADD_CAPTURE(      capture_dir_name => 'DBCOLTP'      ,start_delay_seconds => 0      ,stop_replay => FALSE      ,take_begin_snapshot => TRUE      ,take_end_snapshot => TRUE      ,query_only => FALSE    );
  cpt_id :=
    DBMS_WORKLOAD_REPLAY.ADD_CAPTURE(      capture_dir_name => 'DBCDSS'      ,start_delay_seconds => 0      ,stop_replay => FALSE      ,take_begin_snapshot => TRUE      ,take_end_snapshot => TRUE      ,query_only => FALSE    );
  DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
END;
/
```

Listing 2: Combining OLTP and DSS Workloads for Database Workload Replay

Let’s break down exactly what this step accomplished:

- The PROCESS_WORKLOAD_CAPTURE procedure prepares both the DSS and OLTP workloads — which were recorded against an 11.2.0.3 database — to be replayed against the destination 12.2.0.1 database.
- The SET_REPLAY_DIRECTORY procedure defines exactly where the results of the Database Workload Replay activity will be recorded.
- The call to BEGIN_REPLAY_SCHEDULE defines the Database Workload Replay schedule named TEST2DESTRUCT which includes the DSS and OLTP workloads captured against the source system.
- The ADD_CAPTURE procedure defines several key aspects of the impending Database Workload Replay session for both workloads:
  - CAPTURE_DIR_NAME defines the location of the directories that contain the existing DSS and OLTP workload capture files.
  - Setting START_DELAY_SESSIONS to a value of zero (0) specifies that the workload sessions should begin without any delay. Note that it’s also possible to enable a replay to start after time has elapsed — say, to enable a database to attain a desired steady state.
  - Setting STOP_REPLAY to FALSE means that the replay should continue until all captured workloads have been consumed completely.
  - The calls to procedures TAKE_BEGIN_SNAPSHOT and TAKE_END_SNAPSHOT specify that an AWR snapshot should be taken just before the beginning and just after the end of consolidated workload activity.
  - Setting QUERY_ONLY to FALSE specifies that the database should replay all DML statements as well as all queries. If I had set this to TRUE, then only read-only queries would be replayed.
- Finally, the END_REPLAY_SCHEDULE procedure defines the end of the scheduled replay activity.

Initializing a Consolidated Database Workload Replay

Now that I’ve finished building my Consolidated Replay schedule, my next step is to make sure my destination database is ready to accept that workload schedule. For these tests, I simply reset my database by reloading the data into the AP schema via a series of stored procedures that generated exactly the same data that was present in the source database before I began workload capture.

Of course, in the real world and for a larger database, I could use any number of methods to reset my destination database, including:

- Rewinding the database back to a prior point in time using FLASHBACK DATABASE.
- Employing a snapshot standby version of the original source database.
- Restoring the database from earlier backups using Recovery Manager (RMAN).
- Reloading the database completely via DataPump Import.
To initialize the database for a consolidated replay, I used the code in Listing 3. Procedure INITIALIZE_CONolidATED_REPLAY assigns the task name of DBR_CONS_100 to the TEST2DESTRUCT replay schedule created earlier. Note the use of the REMAP_CONNECTION procedure to remap the database services that user sessions used to connect to the source database during the original Database Workload Capture session to just two database services (OLTP and DSS) during the consolidated Database Workload Replay to come.

--- Create . . .
---

```
BEGIN
  DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY(
    replay_name => 'DBR_CONS_100',
    schedule_name => 'TEST2DESTRUCT');
  DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION(
    connection_id => 1,
    replay_connection =>
      '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=ora12201)(PORT=1521))
        (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=DSS)))');
  . . .
  /* other REMAP_CONNECTION calls removed for sake of brevity */
  . . .
  DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION(
    connection_id => 6,
    replay_connection =>
      '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=ora12201)(PORT=1521))
        (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=OLTP)))');
  . . .
END;/
```

Listing 3: Initializing Database Workload Replay

Starting a Consolidated Database Workload Replay

Next, I’ll start up two Workload Replay Client (WRC) sessions using the script shown in Listing 4. Note that I’m using the WRC user account to log into the destination database so that any additional activity generated by WRC sessions is extraneous to the workloads being replayed.

```
####
# Start Workload Replay Client(s)
####
$> wrc wrc/wrc mode=replay replaydir=/home/oracle/DBRControl
```

Listing 4: Starting Workload Replay Client Sessions

Now that the Workload Replay Client sessions are running, I’ll start consuming the captured workloads by kicking off the Consolidated Replay with a call to procedure START_CONSOLIDATED_REPLAY, as shown in Listing 5. Note that setting START_REPLAY_TIMEOUT to FALSE turns off the default action of the WRC, which is to terminate any user call that may inadvertently cause a replay to run slowly or possibly cause it to hang.

---
-- Start the Consolidated Replay
---

```
DBMS_WORKLOAD_REPLAY.SET_REPLAY_TIMEOUT(FALSE);
DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

Listing 5: Starting a Consolidated Workload Replay

Monitoring Consolidated Database Workload Replay Progress

As the Consolidated Database Workload Replay unwound, I monitored its progress through several panels via Oracle Enterprise Manager 13c Cloud Control (EM13cCC). As its name suggests, the Database Replay panel (see Figure 1) displayed the overall progress of the ongoing Database Replay operation, including a progress bar reflecting the time it took to play back the original workloads as compared to the current consolidated workloads. It also provided tallies of any regression observed so far — for example, a variance in the number of rows queried or modified, or the number of errors observed — between the captured and replay workloads.

Figure 1: Database Workload Replay Progress Monitoring

Figure 2 shows a snapshot of the Database Performance Monitoring panel as the workload was replayed. Note that the bottom half of this panel allowed me to see how the different database services were consuming the captured workload and replaying it in the new environment.
Analyzing Consolidated Database Workload Replay Results

After the Consolidated Database Workload Replay completed its execution, I captured some statistics from the results via the queries in Listing 6; their output is shown in Listing 7.

```
SELECT     id     ,name    ,schedule_name    ,capture_id    ,user_calls    ,status    ,divergence_load_status DLS    ,TO_CHAR(start_time,'yyyy-mm-dd hh24:mi:ss') start_dtm    ,...(Omitted for brevity)...    ,think_time_ss    ,(network_time/1000000) network_time_ss    ,pause_time
FROM dba_workload_replays;
```

```
SELECT     id    ,name    ,dbname    ,dbversion    ,directory    ,capture_id    ,user_calls    ,status    ,TO_CHAR(prepare_time,'yyyy-mm-dd hh24:mi:ss') prepare_dtm    ,synchronization    ,connect_time_scale    ,think_time_scale    ,think_time_auto_correct TTAC    ,scale_up_multiplier SCM    ,default_action
FROM dba_workload_replays;
```

Finally, I kept an eye on the state of my database’s Database In-Memory (DBIM) memory utilization using one of the newest features of OEM13cCC, the In-Memory Central panel. This screenshot reflects the fact that during this particular run of Consolidated Workload Replay, I had enabled population of several key tables into In-Memory Column Store (IMCS) so that I could evaluate the impact of enabling Database In-Memory features as part of my beta-testing activities for Release 12cR2. Figure 3 shows the results of monitoring that aspect of the consolidated replay.

```
SELECT     id    ,name    ,dbname    ,dbversion    ,directory    ,capture_id    ,user_calls    ,status    ,TO_CHAR(prepare_time,'yyyy-mm-dd hh24:mi:ss') prepare_dtm    ,synchronization    ,connect_time_scale    ,think_time_scale    ,think_time_auto_correct TTAC    ,scale_up_multiplier SCM    ,default_action
FROM dba_workload_replays;
```

```
SELECT     id    ,name    ,dbname    ,dbversion    ,directory    ,capture_id    ,user_calls    ,status    ,TO_CHAR(prepare_time,'yyyy-mm-dd hh24:mi:ss') prepare_dtm    ,synchronization    ,connect_time_scale    ,think_time_scale    ,think_time_auto_correct TTAC    ,scale_up_multiplier SCM    ,default_action
FROM dba_workload_replays;
```

Database In-Memory (DBIM) memory utilization using one of the newest features of OEM13cCC, the In-Memory Central panel.
In actuality, these reports show the results from several tests I had run over a period of 24 hours with varying results, but I’ll focus in on the latest run (#167) in the following sections as we analyze the results from the consolidated replay activity. Listing 8 shows the code I used to report against that last Consolidated Database Workload Replay run. Note that the output from that report (Listing 9) shows that a consolidated replay task definitely occurred, that it ran against an Oracle 12c database, and that it was originally captured from an 11.2.0.3 database. Of course, this information is also available from the Database Workload Replay panel in HTML format.

Real Application Testing in Oracle Database 12c to replay previously-captured workloads concurrently, even when they were recorded during completely different timeframes.
Now let’s take a deeper look into whether this new configuration of my database was able to handle the consolidated workload that unwound during this Database Workload Replay task. The easiest way to tackle that topic is with tools we’ve been using since Oracle 11gR1 — running an Automatic Workload Repository (AWR) Compare Periods Report.

Figure 4.1 shows some interesting initial results: The consolidated workload appears to have completed with significantly diminished Database Time (285.2 minutes vs. 191.9 minutes), a 32.7 percent improvement. Also, it shaved almost 21 percent off of the original elapsed time for the two separate captured workloads. So far, so good — we’re not seeing any apparent performance degradation at a database level.
Figure 4.4 shows some interesting information, too. This snippet of the report summarizes database activity within Wait Classes, and this reveals some apparently significant “pushback” against the database workloads during replay. This was most likely due to a Database Resource Manager (DBRM) plan directive that was active during the replay and definitely warrants additional investigation, especially since that “pushback” wasn’t apparent during the previously captured workloads. Again, this doesn’t necessarily mean we’ve encountered performance regression, but it could be the result of an errant DBRM plan directive that was enabled during replay, or it could even be related to how 12cR2 now manages parallelism during population of the In-Memory Column Store.

Figure 4.4: AWR Results: Wait Classes Detail
Finally, Figure 4.5 shows some interesting results, too, but this time at the Database Service level: Only the workload sessions played back via the OLTP database service show any significant regression in DB and CPU time, while most of the other application sessions show slight improvement in those categories.

Figure 4.5: AWR Results: Service Statistics

**Conclusion**
In this final part of this article series, we demonstrated how to leverage the latest features of Real Application Testing in Oracle Database 12c to replay previously-captured workloads concurrently, even when they were recorded during completely different timeframes. We also discussed how to ramp up those captured workloads so that we could perform a test to destruction against the target database in its new 12c environment. Finally, we illustrated how easy it is to determine if the new destination system is able to handle the stresses of those ramped-up workloads using existing tools like AWR Compare Period Reports to quickly determine potential vectors for performance improvement or regression.

Jim Czuprynski has 35+ years of professional experience in his career in information technology. During that time, he served diverse roles at several Fortune 1000 companies — mainframe programmer, applications developer, business analyst and project manager — before becoming an Oracle DBA in 2001. Jim was awarded the status of Oracle ACE Director in March 2014 and is a sought-after public speaker on Oracle Database technology features. He has presented topics at Oracle OpenWorld, IOUG’s COLLABORATE, Hotos Symposium, Oracle Technology Network ACE Tours and Oracle User Group conferences around the world.

In August 2015, Jim joined OnX Enterprise Solutions as their Strategic Solutions Architect for Oracle Database technology, focusing on Oracle Database 12c and Oracle’s Enterprise Systems, including Oracle Database Appliance, Exadata Database Machine, SPARC Supercluster and ZFS Storage Appliance.

Jim has authored over 100 articles focused on facets of Oracle Database administration to his credit since 2003 at databasejournal.com and ioug.org. Jim’s blog, Generally … It Depends (http://jimczuprynski.wordpress.com), contains his regular observations on all things Oracle.

His most recent book, Oracle Database Problem Solving and Troubleshooting Handbook, was published in May 2016 through Pearson.