ORACLE DATABASE 12c

DBFS use case performance on Exadata configurations

ORACLE WHITE PAPER | MARCH 2016

ORACLE

Table of Contents

Introduction

Oracle Database File System (DBFS) introduced in 11g, enables database managed storage (tablespace) as a mountable file system using a combination of Operating System FUSE (File system in USEr space) driver and Oracle provided dbfs_client utility (currently available only on Linux and Solaris). This whitepaper covers basic DBFS setup and performance of dbfs_client mount[1] interface's common use cases (listed in Table 1) on a X2-2 quarter and half rack Exadata configurations (listed in Table 2).

TABLE 1. COMMON USE CASES	
1.	Calibration (cali) of dbfs_client mount interface (simple write and read using 'dd' command) to eliminate configuration issues and establish baseline performance numbers on a given hardware setup.
2.	File system operations – tar archive and extraction, ls, rm –r etc. to or from dbfs_client mount directory
3.	External table load- load database table from files stored on a dbfs_client mount directory.
4.	Export Dump – export a database table into dbfs_client mount directory.
5.	Import Dump – import into a database table from a dump file stored on a dbfs_client mount directory.

TABLE 2. EXADATA TEST CONFIGURATIONS		
1.	Quarter rack (X2-2) [2]	1 compute nodes (2 socket system) and 3 storage cells
2.	Half rack (X2-2)	4 compute nodes (2 socket system) and 7 storage cells

- [1] dbfs_client mount interface|mount point|mount - all are interchanging used throughout this document to refer to dbfs_client mount interface directory (e.g. /u01/app/dbfs_mnt1)
- [2] All Exadata configurations are with Oracle Unbreakable Enterprise kernel version 2.6.39-400.xxx.xx.el5uek.

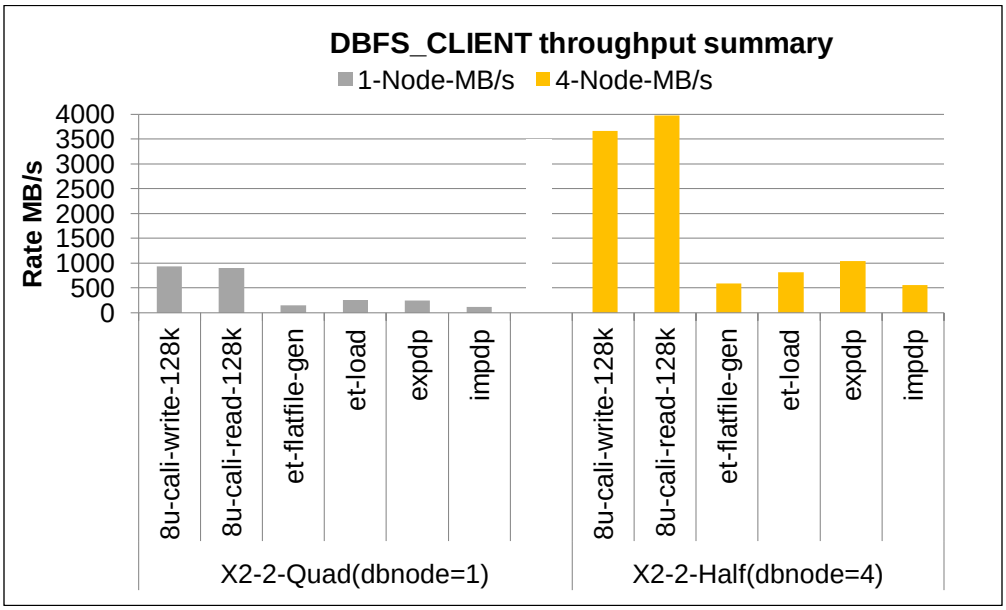
Summary

- To enable users with quick DBFS setup, a cookbook version of DBFS store creation, mounting using dbfs_client, associated RPM requirements, how to enable tracing for debug purpose and error log location are all listed in 'DBFS setup' section.

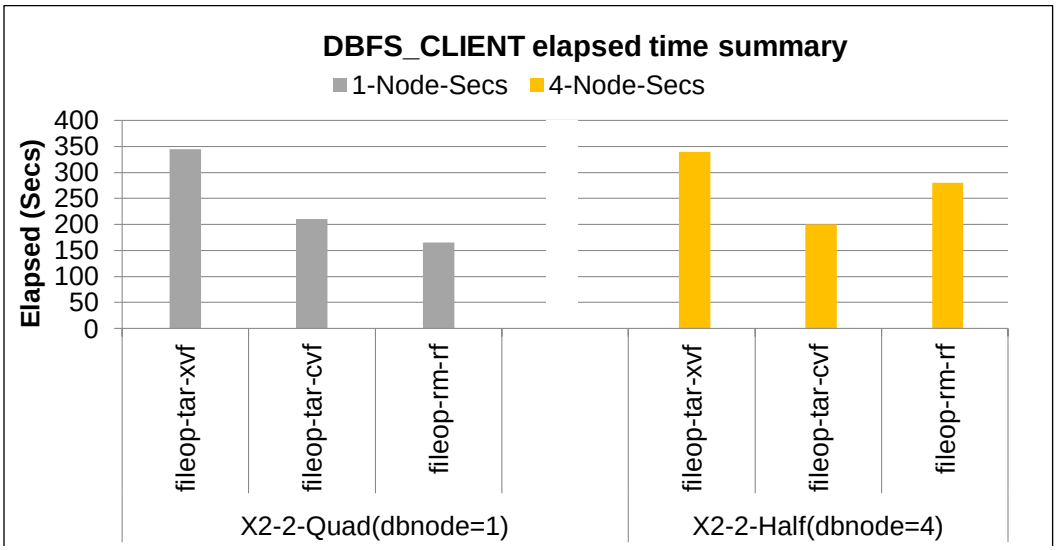
- Only –o direct_io and –o server_readahead dbfs_client mount options were experimented with and the recommendation is to use –o direct_io and –o server_readahead=0 for overall good performance. All other mount options like –o max_threads -o spool_max –o spool_min are left at default value for all the experiments listed here.

- Performance numbers reported here are reference values which can be obtained with basic setup procedure outlined in the 'DBFS setup' section and it should be noted that this numbers are not the best possible on this given hardware configurations.

Following chart gives a high level of dbfs_client throughput summary for the quarter (Quad) and half rack Exadata configurations. Here the amount of work done by each node is 1x (with N=1) and 0.25x (with N=4). Good scaling is observed with the addition of nodes, as it is seen throughput nearly quadrupling with 4 DB nodes.



Following chart gives a high level of dbfs_client elapsed time summary for the 2 Exadata configurations. Here amount of work done by each node is 1x (with N=1) and **4x** (with N=4). Again a good scaling is observed with the addition of nodes, as it is seen elapsed time remained almost the same with 4 DB nodes.



The remainder of this paper explains the basic DBFS setup, description of performance tests and results, followed by hardware and software details. List of known issues and possible workarounds and init.ora parameter values are given in the appendix section.

DBFS Setup

- CREATE TABLESPACE (name 'DBFS_TS') AND USER DBFS_DEMO

  SQL>create bigfile tablespace DBFS_TS datafile '+C_DATA' size 3T reuse autoextend on next 10g extent management local autoallocate segment space management auto;

  SQL>create user dbfs_demo identified by dbfs_demo;

  SQL>grant dba, connect, resource, **dbfs_role** to dbfs_demo;

  SQL>alter user dbfs_demo default tablespace dbfs_ts;

- DBFS STORE CREATION (store name 'DBFS_TEST')

  SQL> connect dbfs_demo/dbfs_demo

  SQL>@?/rdbms/admin/dbfs_create_filesystem.sql DBFS_TS DBFS_TEST

- RPM REQUIREMENTS AND CHECKS

  Make sure fuse driver rpm is installed and the module is loaded

  #rpm -qa |grep fuse

  fuse-2.7.4-8.0.5.el5

  fuse-devel-2.7.4-8.0.5.el5

  fuse-libs-2.7.4-8.0.5.el5

  #/sbin/lsmod |grep fuse

  fuse 78879 0

  ##to remove loaded fuse module do /sbin/rmmod fuse

  ##to reload fuse module do /sbin/modprobe fuse

  Check /bin/fusermount for right permission (setuid, owner, group, execute)

  #ls –l /bin/fusermount

  -rwsr-x--x 1 root fuse 27072 Aug 20 2009 /bin/fusermount

  Check device file /dev/fuse for right permission

  #ls –l /dev/fuse

  crw-rw-rw- 1 root root 10, 229 Apr 3 09:19 /dev/fuse

  Modify fuse configuration file (/etc/fuse.conf) for customization

  #cat /etc/fuse.conf

  ##allow other users to read and write

user_allow_other

Make mount point directory and set permission

#mkdir /u01/app/dbfs_mnt1

#chown oracle:oinstall /u01/app/dbfs_mnt1

#make a password file named passwd.f and enter password of db user dbfs_demo

#cat passwd.f

dbfs_demo

- dbfs_client MOUNT USING DEDICATED[3] CONNECTION

  #nohup $ORACLE_HOME/bin/dbfs_client dbfs_demo@ -o allow_other –o direct_io –o server_readahead=0 /u01/app/dbfs_mnt1 < passwd.f &

- FUSE ERROR LOG AND dbfs_client TRACING

  fuse driver and module errors are logged in /var/log/messages

  dbfs_client tracing can be enabled using the following mount parameters in addition to regular parameters

  #nohup $ORACLE_HOME/bin/dbfs_client dbfs_demo@ -o allow_other –o direct_io –o server_readahead=0 /u01/app/dbfs_mnt1 **–o trace_file=/tmp/dbfs.trc –o trace_size=0 –o trace_level=4** < passwd.f &

  - trace_size=0 unlimited trace file size
  - trace_level=1->DEBUG, 2->INFO, 3->WARNING, 4->ERROR, 5->CRITICAL

- [3] dbfs_client MOUNT USING SQL*NET CONNECTION(tnsnames.ora entry ‘dbfs01’)

  #nohup $ORACLE_HOME/bin/dbfs_client dbfs_demo@**dbfs01** -o allow_other –o direct_io –o server_readahead=0 /u01/app/dbfs_mnt1 < passwd.f &

Test Description and Results

A single database is used for all the tests hosting regular table and DBFS store data in a single tablespace named DBFS_TS.

Data set size is fixed. Calibration tests use 128GB fixed. Flat file generation, external table load, export and import dump tests each use 1TB fixed. File system operations test use 3GB per node. Table 3 illustrates this in detail.

TABLE 3. DATA SET USED

Exadata Configuration	cali-1u	cali-4u	cali-8u	file-op	et-gen	et-load	expdp	impdp
Quarter-X2-2 (1 db-node)	128GB	128GB	128GB	3GB	1024GB	1024GB	1024GB	1024GB
Half-X2-2 (4 db node)	32x4	32x4	32x4	**3x4**	250x4	250x4	250x4	250x4

All numbers reported in the following result tables (Table 4, 5, 6, 8 and 9) are in MB/s, except in 'file system operations' case, where it is elapsed time in seconds (Table 7).

Calibration

1u-1k-write[read] – 1 user, using 'dd' command doing write or read of 1GB file to or from dbfs_client mount point directory with a bs=1k. Sample script is given below.

write test

#dd of=/../dbfs_mnt1/DBFS_TEST/node1/file_f0_u0.txt if=/dev/zero bs=1024 count=1024000

read test

#dd if=/../dbfs_mnt1/DBFS_TEST/node1/file_f0_u0.txt of=/dev/null bs=1024

TABLE 4. CALIBRATION – 1 USER RESULTS (MB/S)

Exadata Configuration	1u-1k-write	1u-1k-read	1u-4k-write	1u-4k-read	1u-8k-write	1u-8k-read	1u-32k-write	1u-32k-read	1u-128k-write	1u-128k-read	1u-1m-write	1u-1m-read
Quarter-X2-2 (1 db-node)	35	35	120	130	205	215	390	350	405	380	400	380
Half-X2-2 (4 db node)	140	140	480	520	780	900	1560	1540	1500	1620	1540	1620

TABLE 5. CALIBRATION – 4 USER RESULTS (MB/S)

Exadata Configuration	4u-1k-write	4u-1k-read	4u-4k-write	4u-4k-read	4u-8k-write	4u-8k-read	4u-32k-write	4u-32k-read	4u-128k-write	4u-128k-read	4u-1m-write	4u-1m-read
Quarter-X2-2 (1 db-node)	115	115	385	415	650	620	900	935	880	905	880	870
Half-X2-2 (4 db node)	460	460	1460	1660	2320	2440	2640	2760	3040	3320	3040	3140

TABLE 6. CALIBRATION – 8 USER RESULTS (MB/S)												
Exadata Configuration	8u-1k-write	8u-1k-read	8u-4k-write	8u-4k-read	8u-8k-write	8u-8k-read	8u-32k-write	8u-32k-read	8u-128k-write	8u-128k-read	8u-1m-write	8u-1m-read
Quarter-X2-2 (1 db-node)	140	140	460	510	770	820	1090	935	935	900	795	880
Half-X2-2 (4 db node)	500	520	1520	1760	2520	2800	2800	3580	3660	3980	3320	3540

File system operations

Assuming a 3gb_tarball.tar file placed in a local file system under directory /u01, dbfs_client mount point directory as /u01/app/dbfs_mnt1/file_op , the following file system operations were done in this test and elapsed time in seconds is reported in Table 7.

tar xvf /u02/3gb_tarball.tar -C /u01/app/dbfs_mnt1/file_op/	>/tmp/tar_xvf.log
ls -lFR /u01/app/dbfs_mnt1/file_op	>/tmp/ls_lFR.log
ls -lFR /u01/app/dbfs_mnt1/file_op | wc -l	>/tmp/ls_lFR_wc_l.log
ls -l /u01/app/dbfs_mnt1/file_op/*/*/*/gennttab	>/tmp/ls_l_3lvl.log
ls –l /u01/app/dbfs_mnt1/file_op/a/b/c/d/lx203e4.nlb	>/tmp/ls_l_qualified.log
tar cvf /u02/dbfs.tar /u01/app/dbfs_mnt1/file_op/	>/tmp/tar_cvf.log
rm -rf /u01/app/dbfs_mnt1/file_op/*	>/tmp/rm_rf.log

A separate 3GB sized tar-ball is used by each node (approximately 93,000 files and up to 9 levels of subdirectories). In a 4 DB node test, each node works on its own directory doing tar xvf, cvf, ls and rm operations independent of other nodes using its own 3GB tar-ball, i.e. 4x more work.

TABLE 7. FILE SYSTEM OPERATION RESULTS (SECONDS)							
Exadata Configuration	tar xvf	ls –lFR	ls –lFR |wc -l	ls –l /*/*/*/a.gif	ls –l /a/b/c/d/z.txt	tar cvf	rm -rf
Quarter-X2-2 (1 db-node)	345	35	35	<1	<1	210	165
Half-X2-2 (4 db node)	340	30	30	<1	<1	200	280

External table – Flat file generation and loading

Flat file generation is done using industry standard TPC-H benchmark utility ‘dbgen’ for LINEITEM table and stored on to a dbfs_client mount point directory ‘/u01/app/dbfs_mnt1/et_gen’. Sample dbgen script is given below.

```
## dbgen is a standard TPC-H flat file generation utility
## Usage
## -f force, Overwrite exiting files
## -s scale factor -s 1 SF=1GB
## -S build the <n>th step of the data set
## -C number of processes to generate data
## -T L - generate lineitem data only
## -s 100 = 4.5 GB of lineitem data
## set Flatfile gen location using DSS_PATH environment variable
```

```
#mkdir –p /u01/app/dbfs_mnt1/et_gen
#export DSS_PATH=/u01/app/dbfs_mnt1/et_gen
#nohup dbgen -f -s 360 -S 1 -C 16 -T L >/dev/null 2>&1 &
..
#nohup dbgen -f -s 360 -S 8 -C 16 -T L >/dev/null 2>&1 &
#wait
```

An external table is created to use the above generated 8 flat files and a database table LINEITEM is loaded from this external table using a degree of parallelism (DOP) of 16 (with parallel slaves affined to local node). Sample script is given below.

```
SQL>create directory flatfiles_dir as '/u01/app/dbfs_mnt1/et_gen';

SQL>create table l_et (..)
SQL> organization external
SQL> ( type ORACLE_LOADER default directory flatfiles_dir
SQL>   access parameters(..)
SQL>   location (flatfiles_dir:'lineitem.tbl.1', ..flatfiles_dir:'lineitem.tbl.8')
SQL> ) parallel reject limit unlimited;

SQL> create table LINEITEM (..)parallel as select * from l_et;
```

In 4 DB node case, each node generates and loads 250GB of its own LINEITEM table (e.g. LINEITEM_N1, .. LINEITEM_N4).

TABLE 8. EXTERNAL TABLE – DATA GENERATION AND LOAD (MB/S)

Exadata Configuration	Flatfile generation	External table load
Quarter-X2-2 (1 db-node)	145	250
Half-X2-2 (4 db node)	585	810

Export and Import dump

Export (expdp) and import (impdp) dump - unloads, drops and reloads the table (LINEITEM_N?) created in the above external table test on to a dbfs_client mount point directory. Sample script is given below.

```
## export dump commands
#mkdir –p /u01/app/dbfs_mnt1/dp_dir

SQL>create or replace directory dp_dir as '/u01/app/dbfs_mnt1/dp_dir';

#expdp userid=dbfs_demo/dbfs_demo directory='dp_dir' tables=LINEITEM_N$n_num dumpfile=lineitem.dat
reuse_dumpfiles=Y logfile=/tmp/exp_dp_lineitem.log metrics=y

## import dump commands
SQL>drop table LINEITEM purge;
SQL>create table LINEITEM (..);

#impdp userid=dbfs_demo/dbfs_demo directory='dp_dir' tables=LINEITEM_N$n_num dumpfile=lineitem.dat
logfile=/tmp/imp_dp_lineitem.log table_exists_action=append metrics=y
```

In a 4 DB node case, each node does its own expdp/impdp of LINEITEM table (node1 doing LINEITEM_N1, node2 doing LINEITEM_N2 and so on) on to its own dbfs_client mount point directory, but using the same DBFS store.

TABLE 9. DATAPUMP RESULTS (MB/S)		
Exadata Configuration	**Export dump**	**Import Dump**
Quarter-X2-2 (1 db-node)	245	115
Half-X2-2 (4 db node)	1045	560

Import dump performance is slower than export dump (range of regression varies from 40% - 2x depending on data size).

Hardware and Software Details

Component/Spec.	X2-2 quarter rack	X2-2 half rack
CPU, Memory and others	Refer to data sheet	Refer to data sheet
DB server Liunx kernel version	2.6.39-400.128.1.el5uek	2.6.39-400.128.14.el5uek
FUSE driver version	fuse-2.7.4-8.0.4.el5	fuse-2.7.4-8.0.5.el5
FUSE libs version	fuse-libs-2.7.4-8.0.4.el5	fuse-libs-2.7.4-8.0.5.el5
RDBMS	12.1.0.1.0	12.1.0.1.0
Storage Cells Linux version	2.6.39-400.128.1.el5uek	2.6.39-400.128.14.el5uek
Cell version	cell-12.1.1.1.0_LINUX.X64_131219	cell-12.1.2.1.0_LINUX.X64_140513

Appendix

List of known issues and possible workarounds

- df space usage query resource consumption of a dbfs_client mount point interface

  SQL query to compute df space usage consumes high resources and runs frequently.

  Workaround: df space usage query can be disabled completely or run less frequently using the settings below.

```
SQL> -- assuming file system name as 'DBFS_TEST', df computing query is
SQL> -- completely disabled with 'NONE' option
SQL> declare
SQL> begin
SQL>    dbms_dbfs_sfs.addFSProperties('DBFS_TEST',
SQL>      dbms_dbfs_content_properties_t(
SQL>       dbms_dbfs_content_property_t(
SQL>        dbms_dbfs_sfs.sfs_props_df,
```

```
SQL>       'NONE',
SQL>       dbms_types.typecode_varchar2)));
SQL> commit;
SQL> end;
SQL>/

SQL> -- df computing query is set to run every 3600 seconds
SQL> declare
SQL> begin
SQL>    dbms_dbfs_sfs.addFSProperties('DBFS_TEST',
SQL>      dbms_dbfs_content_properties_t(
SQL>       dbms_dbfs_content_property_t(
SQL>        dbms_dbfs_sfs.sfs_props_df,
SQL>        3600,
SQL>        dbms_types.typecode_varchar2)));
SQL> commit;
SQL> end;
SQL>/
```

- FUSE MKNOD does not run in parallel

  Bug 9961241 - FUSE MKNOD used by dbfs_client does not run in parallel – in a single directory.

  Workaround: Use multiple directories to support higher concurrency of file creation.

- Row chaining effect on DBFS filestore base table with default PCTFREE

  The default PCTFREE value of 10% on a DBFS filestore base table can cause high number of row chaining effects in a highly concurrent file creation environment.

  For example, consider an environment where 64 users concurrently executing "mknod" on a dbfs_client mount point directory, this will create 64 rows of an empty file with an empty_blob() on the base table. These rows take up a small amount of space and get tightly packed on to a small set of blocks. Once the files are populated, LOB metadata grows to accommodate the LOBMAP for GB-sized files, and this causes chaining. If files are written sequentially (or equivalently, with low concurrency), there would be enough space on the blocks to allow row growth without forming chains.

  Workaround: Increase PCTFREE of the DBFS store base table

- High 'SYS' mode CPU usage

  File system operations on a dbfs_client mount point interface will show high SYS mode CPU usage. This is because of data transfer from the client application to dbfs_client taking place via FUSE's ipc in small pieces (default being 4096+64 bytes in non-directIO and 128K with directIO mount option) and there is very little control over it.

  dbfs_client has to invoke more number of system calls to collect the same amount of data that it eventually sends to the RDBMS. More system calls implies more system CPU usage. The total number of system calls invoked also increases for large file transfers where most of the elapsed time is spent just in the FUSE ipc channel.

  Workaround: None. It is expected that system CPU will be higher for large file transfer via non-directIO compared to directIO mounts.

- Hard links

  Hard links (eg. ln a b) does not work on a dbfs_client mount point directory.

  Workaround: None.

- Non-responding 'Control-C'

  Doing Ctl-C while the commands like ls, find, tar, etc. done on a dbfs_client mount point directory does not terminate them right away.

  Workaround: None.

- Slow response of file listing (ls –l *xyz* on a dbfs_client mount point directory with a large number of files)

  Comparing response time of ls –l *xyz* of a local Unix file system (e.g. ext3) to a dbfs_client mounted directory is unfair. This is because wildcard processing in Unix is not done by "ls", but by the shell. This process is explained below.

TABLE 15. SHELL VS DBFS_CLIENT PROCESSING OF LS –L *XYZ*

	shell	dbfs_client
1.	ls *xyz* is parsed by shell	dbfs_client is not involved at this stage
2.	All filenames in the relevant directory (current directory, for example) are generated by the shell using opendir-readdir-closedir sequence	'select * from dbfs_attributes' query on the RDBMS is invoked
3.	If there are a million files in total and only 10 match the wildcard, shell will unconditionally generate all million filenames as part of identifying the 10 that do match the patterns (*xyz*). There is no such thing as a 'predicate-push' of wildcard patterns into the POSIX readdir API	If there are a million files in DBFS store, dbfs_attributes query will process them all and hand it over to shell
4.	The shell then filters out and throws away all filenames that do not match the wildcard pattern in the 'ls' command-line leaving only those that match the pattern	dbfs_client is not involved at this stage
5.	The shell then invokes /bin/ls binary with the matching 10-100 or so filenames	dbfs_client is not involved at this stage
6.	'ls' then invokes stat() on the matching filenames	A new dbfs_attributes query is invoked or cache from step 2 is used

The slow step in all of this is step 2, where all (million, billion, etc.) filenames are generated.

Note that the above sequence of steps involving the shell and ls is orthogonal to DBFS. Exactly the same things are done when ls and wildcards are used on a local ext3 file system.

It just so happens that an algorithm like "generate all filenames and throw away those that do not match a wildcard pattern" is sufficiently fast on a local file system with fully cached metadata that no

one bothers to complain about it (unless the number of filenames is extremely large). On a network-file system like DBFS, the same thing will be slow for even moderately sized directories.

1) Workaround: None.

init.ora values

```
cluster_database=true
dbfstst1.instance_number=1
dbfstst2.instance_number=2
dbfstst3.instance_number=2
dbfstst4.instance_number=2

compatible='12.1.0.0'
control_files='+DATA/DBFSTST/control1.dbf'
db_block_size=8192
db_files=1024
db_name='DBFSTST'
log_checkpoints_to_alert=TRUE
processes=3000

sga_max_size=32G
sga_target=32G
pga_aggregate_target=10G
pga_aggregate_limit=10G

undo_management='AUTO'
dbfstst1.undo_tablespace='UNDO_TS1'
dbfstst2.undo_tablespace='UNDO_TS2'
dbfstst3.undo_tablespace='UNDO_TS3'
dbfstst4.undo_tablespace='UNDO_TS4'

_resource_manager_always_off=true
```

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US
AUTHOR: VINAYAGAM DJEGARADJANE
CONTRIBUTING AUTHOR: KRISHNA KUNCHITHAPADAM

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Hardware and Software, Engineered to Work Together

Oracle is committed to developing practices and products that help protect the environment