AUTOMATE DATA GUARD BEST PRACTICES

Charles Kim, Viscosity Technology, LLC

Myriads of best practice techniques are provided from Oracle's Maximum Availability Architecture (MAA) team and industry experts that address Data Guard when it comes to performance, scalability and reliability. Compliance with best practices can help mitigate potential performance and infrastructure issues that ails many companies. One of the best ways to comply with industry best practices is with automated processes and procedures. This paper will disseminate fundamental Data Guard best practices and demonstrate how DBAs can automate setup, configuration, monitoring and maintenance of physical standby Data Guard environments with assistance from the Data Guard (DG) Toolkit with concentration on the following topics:

- Building the Physical Standby
- Monitoring and Maintaining the Physical Standby
- Configuring Data Guard Broker
- Performing Backup and Recovery with RMAN
- Setting Archive Retention
- Performing Switchovers and Failovers
- And more

THE DG TOOLKIT

The majority of this presentation is based on the DG Toolkit V2.0, formerly known as the DG Menu, which was presented and documented in the Oracle Data Guard 11g Handbook by Oracle Press. The DG Toolkit is an interactive menu driven toolkit designed to assist DBAs and architects to configure, monitor and troubleshoot Data Guard eco-systems. The DG Toolkit also provides reports to assist DBAs bulletproof their Data Guard configurations. You can leverage the DG Toolkit to check your existing Data Guard environment for compliance, create a new Data Guard environment, and/or troubleshoot a configuration with performance symptoms.

DG Toolkit V2.0 includes enhancements and new additional features not covered in the Data Guard 11g Handbook. We included a dedicated screen for physical standby setup and configuration. We have significantly updated the monitoring and setup menu options of our DG Menu. We added a screen for Broker setup, maintenance, and reporting. In addition, we also added another screen for Automatic Diagnostic Repository Command Interpreter (ADRCI) reporting and maintenance. You can simply download the tar file from the dataguardbook.com website, extract the contents, modify the main configuration file, and start executing scripts against the primary and standby database(s) in minutes. The DG Toolkit can be an arsenal of every DBA.

The DG Toolkit is continually evolving and constant being improved. With each revision, more functionality is added, new best practices are incorporated, and minor bugs are fixed. You can download the DG Toolkit from one of the following two locations:

1. http://DataGuardBook.com
2. http://DBAExpert.com/dg/

You should check the two sites for updates and new revisions from time to time.

As a side note, the DG Toolkit does not issue any DDL or destructive DML. The primary purpose for DG Toolkit is to generate command line syntax so you are equipped to become a power DBA when it comes to configuring, monitoring, maintaining and troubleshooting a Data Guard configuration. More importantly, the DG Toolkit is designed so that the DBA can review the code output and be comfortable with the syntax before executing the generated scripts. The only time when a menu option will execute SQL directly against the primary or standby database is when SELECT statements are issued against pertinent V$ and DBA_ views to exploit pertinent Data Guard related information. As menu options are chosen on the primary database or the standby database, the DG Toolkit displays the SQL script name that is currently being executed.

Of course, as more screens are added and additional features are introduced, the location of the options may change. The other great aspect of the DG Toolkit is that you can see the shell script that is being invoked and submenus are chosen. All the source code for the shell script and SQL files are completely exposed. If there's specific tweaks or enhancements that you would like to add to the DG Toolkit, you can customize the toolkit to fit your organizational requirements.

BEST PRACTICES

DBAs must be intimately knowledgeable of their companies' recovery point objective (RPO) and recovery time objective (RTO). The Data Guard configuration needs to be directly correlated to meeting the companies' RPO and RTO requirements. You can leverage the DG Toolkit to implement fundamental best practices and configure, monitor, and maintain a Data Guard environment suited for your RPO and RTO.

INSTALLATION AND CONFIGURATION

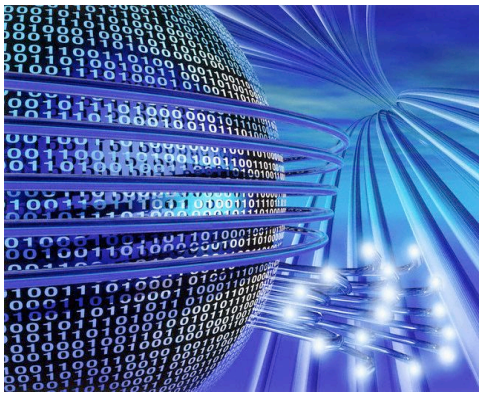
First and foremost, as part of the best practice compliance, you should apply the latest Patch Set Update (PSU) or be at a N-1 PSU release cycle specific to your database release. For the latest PSU, please check the following Metalink note:

- `Oracle Recommended Patches -- Oracle Database [ID 756671.1]`

As of end of February 2011, 11.2.0.2.1 PSU is available by downloading patch number 10248523. For Oracle Database 11g Release 1 customers, 11.1.0.7.6 Patch Set Update (Patch 10248531) is available, and the Data Guard Broker Recommended Patch Bundle #1 is also available (Patch 7628357).

SDU (SESSION DATA UNIT)

Let's start our best practice configuration with SQL*Net and focus on the concept of decreasing the number of packets injected on to the network. The default data unit transmitted in Oracle Database 11g is 8192 bytes (8k). For standby databases, data units of 8k are insufficient. Oracle redo transmission happens at data units far above 8k. For this reason, as best practices, you should set SDU to 32k.



You can setup SDU in one of two ways. You can set default SDU to be 32k in the SQLNET.ORA file, which applies to every database, or in the listener/tnsnames.ora files for each database (as shown in the BDP section). Here's the one-liner to set it in the SQLNET.ORA file:

```
Option #2:
You can add the following to:
rac561-vip: /u01/app/oracle/product/11.2.0/db/network/admin/sqlnet.ora
rac562-vip: /u01/app/oracle/product/11.2.0/db/network/admin/sqlnet.ora

# -- This will set SDU to 32k for all the databases
# --
```

```
DEFAULT_SDU_SIZE=32767
TCP.NODELAY=YES
```

In this particular portion of the DG Toolkit, we enabled SDU at 32k and also disabled Nagle Algorithm with the TCP.NODELAY option which is covered in another section of this paper. The option to generate the TNSNAMES.ORA and LISTENER.ORA settings are available in the Prepare Physical Standby Database Submenu (dg_prepare_standby_menu.ksh).

One thing to note is that DBAs and System Administrators may object to setting the SDU at the global level. Please be advised that setting this in the SQLNET.ORA will impact all databases that utilize that ORACLE_HOME.

BDP (Bandwidth-Delay Product)

In addition to SDU, you need to calculate BDP to determine the appropriate size for the receive and send buffer sizes. BDP is calculated as:

```
Bandwidth * Latency * 3
```

To calculate, BDP, you need to know what kind of network you have between the primary database and your disaster recovery site. The following matrix defines the common high-speed WAN connection bandwidths:

- T1 - 1.544 megabits per second
- T3 - 43.232 megabits per second (equivalent to 28 T1s)
- OC3 - 155 megabits per second (equivalent to 84 T1s)
- OC12 - 622 megabits per second (equivalent to 4 OC3s)
- OC48 - 2.5 gigabits per seconds (equivalent to 4 OC12s)

Once we calculate BDP, we can plug-in the SEND_BUF_SIZE and RECV_BUF_SIZE parameters in the TNSNAMES.ORA and LISTENER.ORA files. As a preliminary step, first configure the dg_bdp.conf configuration file and specify the kind of WAN that is deployed at your company and the network Round Trip Time (RTT). RTT is measured in milliseconds (ms) for a network communication to travel from the primary database to the standby database and back.

and the response time in milli-seconds from a ping command:

```
WAN=OC3

# --
#  PING RESPONSE measured in milliseconds
#  ping -c 2 -s 64000 target_host
# --
RTT_RESPONSE=44
```

The ping command, by default, only sends 56 data bytes. With sending just 56 data bytes, your RTT response will be faster and skew your results since this is not a realistic packet size transmitted by Data Guard. Instead, you should set your packet size to be something like 64k, to determine you RTT. Here's a ping command that can be leveraged to help you determine RTT:

```
ping -c 2 -s 64000 target_host
```

Once you modify the dg_bdp.conf file with the two parameters, you can execute the corresponding shell script to derive not only BDP but also auto-generate the corresponding TNSNAMES.ORA and LISTENER.ORA files:

```
#!/bin/ksh
# -- Script Name:  dg_bdp.ksh
# --

[ "$CONF" = "" ] && export CONF=$PWD/dg.conf
[ ! -f "$CONF" ] && { echo "Configuration file: $CONF is missing."; echo "Exiting."; exit 1; }
```

```
. $CONF
. $PWD/dg_bdp.conf

#
[ "$WAN" = "T1" ] && export SPEED=1.544
[ "$WAN" = "T3" ] && export SPEED=43.232
[ "$WAN" = "OC3" ] && export SPEED=155
[ "$WAN" = "OC12" ] && export SPEED=622
[ "$WAN" = "OC48" ] && export SPEED=2500

BANDWIDTH=$(echo $SPEED \* 1000000 |bc)
LATENCY=$(echo "scale=5;$RTT_RESPONSE / 1000" |bc)
BDP=$(echo "$BANDWIDTH * $LATENCY * 3 / 8" |bc)
export BDP_NODECIMAL=$(echo "$BDP" |awk -F"." {'print $1'})

echo "BANDWIDTH is:  $BANDWIDTH"
echo "LATENCY is:  $LATENCY"
echo "BDP = $BDP_NODECIMAL"


cat $PWD/dg_generate_bdp.txt |sed -e "s/###_PRIMARY_DATABASE_###/$PRIMARY_DB/g" \
                          -e "s/###_STANDBY_DATABASE_###/$STANDBY_DB/g" \
                          -e "s/###_PRIMARY_DB_###/$PRIMARY_DB/g" \
                          -e "s/###_STANDBY_DB_###/$STANDBY_DB/g" \
                          -e "s/###_PRIMARY_DB_INSTANCE_###/$PRIMARY_DB_INSTANCE/g" \
                          -e "s/###_STANDBY_DB_INSTANCE_###/$STANDBY_DB_INSTANCE/g" \
                          -e "s=###_PRIMARY_ORACLE_HOME_###=$PRIMARY_ORACLE_HOME=g" \
                          -e "s=###_STANDBY_ORACLE_HOME_###=$STANDBY_ORACLE_HOME=g" \
                          -e "s/###_PRIMARY_VIP_###/$PRIMARY_VIP/g" \
                          -e "s/###_DR_VIP_###/$DR_VIP/g" \
                          -e "s/###_PRIMARY_HOST_###/$PRIMARY_HOST/g" \
                          -e "s/###_STANDBY_HOST_###/$STANDBY_HOST/g" \
                          -e "s/###_PRIMARY_PORT_###/$PRIMARY_PORT/g" \
                          -e "s/###_BDP_###/$BDP_NODECIMAL/g" \
                          -e "s/###_STANDBY_PORT_###/$STANDBY_PORT/g"


# --
#
echo "Option #2:"
echo "You can add the following to:"
echo "${PRIMARY_HOST}: $PRIMARY_ORACLE_HOME/network/admin/sqlnet.ora "
echo "${STANDBY_HOST}: $STANDBY_ORACLE_HOME/network/admin/sqlnet.ora "
echo ""
echo "# -- This will set SDU to 32k for all the databases"
echo "# --"
echo "DEFAULT_SDU_SIZE=32767"
```

As you can see, the supplied script supports T1, T3, OC3, OC12 and OC48 configurations and are hard-coded. If you need any other WAN bandwidths to be supported, please feel free to customize the dg_bdp.ksh script. Executing the script, dg_bdp.ksh, both TNSNAMES.ORA and LISTENER.ORA entries are auto-generated for you with the appropriate SEND_BUF_SIZE and RECV_BUF_SIZE values as determined by the BDP algorithm.

```
VISK1 > ./dg_bdp.ksh
BANDWIDTH is:  155000000
LATENCY is:  .04400
BDP = 2557500

VISK_PRI =
  (DESCRIPTION =
    (SDU=32767)
    (SEND_BUF_SIZE=2557500)
    (RECV_BUF_SIZE=2557500)
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = rac561-vip)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
```

```
      (SERVICE_NAME = VISK)
    )
  )

VISK_STDBY =
  (DESCRIPTION =
    (SDU=32767)
    (SEND_BUF_SIZE=2557500)
    (RECV_BUF_SIZE=2557500)
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = rac562-vip)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = VISK_DR)
    )
  )

# --
# - Please change the SID_LIST_LISTENER LISTENER.ORA entries
# - on the primary and standby database servers include the SDU size
# - And reload or bounce the listener
# -
# For VISK
# ------------------------
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=IPC)(KEY=LISTENER)))
    (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP) (HOST = rac561-vip) (PORT = 1521) (SEND_BUF_SIZE=2557500)
(RECV_BUF_SIZE=2557500)))
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SDU=32767)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/db)
      (SID_NAME = VISK1)
    )
  )

# For VISK_DR
# -------------------------
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=IPC)(KEY=LISTENER)))
    (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP) (HOST = rac562-vip) (PORT = 1521) (SEND_BUF_SIZE=2557500)
(RECV_BUF_SIZE=2557500)))
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SDU=32767)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/db)
      (SID_NAME = VISK_DR1)
    )
  )
```

You can see from the above example that the shell script generated the TNSNAMES.ORA and LISTENER.ORA entries with a SEND_BUF_SIZE and RECV_BUF_SIZE of 2557500 bytes. You should compare the SEND_BUF_SIZE and RECV_BUF_SIZE parameters with your operating system TCP settings. If your BDP calculations turn out to be larger than your TCP maximum socket size, you may want to increase your net.ipv4.tcp_wmem and net.ipv4.tcp_rmem parameters if you happen to be in the Linux world:

```
net.ipv4.tcp_wmem = 4096      16384   4194304
net.ipv4.tcp_rmem = 4096      87380   4194304
```

The example above shows the minimum, default and maximum values. You will never have the need to change the minimum or the default value for these parameters. Instead of manually executing the dg_bdp.ksh script, you can choose this option from the Prepare Physical Standby Database Submenu (dg_prepare_standby_menu.ksh).

Network Queue Size

In the world of Linux, you have two network queues, the interface transmit queue, and a network receive queue that should be adjusted for better performance. The default txqueuelen for Oracle Linux 5 and Red Hat 5 is 1000 ms. Txqueuelen of 1000 may be inadequate for long distance, high throughput pipes.

```
/sbin/ifconfig eth0 txqueuelen 10000
```

For the receiver side, there is a similar queue for incoming packets. This queue will build up in size when an interface receives packets faster than the kernel can process them. If this queue is too small (default is 1000 on Red Hat 5 and Oracle Linux 5), we will begin to lose packets at the receiver, rather than on the network. You can benefit from setting this kernel parameter to be 20000:

```
echo "sys.net.core.netdev_max_backlog=20000" >>/etc/sysctl.conf
sysctl -p
```

In Red Hat and Oracle Linux, you can reload the kernel parameters dynamically with the sysctl –p option. Importantly, you need to keep in mind that txqueuelen network interface change on the primary database server and the sys.net.core.netdev_max_backlog kernel parameter change on the standby database server need to happen in pairs and should be adjusted together.

Disable TCP Nagle Algorithm

The parameter, TCP.NODELAY, in the SQLNET.ORA file, activates and deactivates Nagle's Algorithm. TCP.NODELAY alters the way packets are delivered on the network, therefore possibly impacting performance.

Under certain conditions for some applications using TCP/IP, Oracle Net packets may not get flushed immediately to the network especially when large amounts of data are streamed. To mitigate this issue, you can specify no delays in the buffer flushing process. This is not a SQL*Net feature, but rather the ability to set the persistent buffering flag at the TCP layer.

To prevent delays in buffer flushing in the TCP protocol stack, disable the TCP Nagle algorithm by setting TCP.NODELAY to YES (the default value) in the SQLNET.ORA file on both the primary and standby systems.

```
TCP.NODELAY=YES
```

Enable Network Time Protocol (NTP)

You should synchronize your system time between your primary and standby database server by enabling the NTP daemon. You should enable NTP with the –x option to allow for gradual time changes, also referred to as slewing. This slewonly option is mandatory for Real Application Clusters (RAC), but is also recommended for Data Guard configurations. To setup NTP with the –x option, you need to modify the /etc/sysconfig/ntpd file and add the desired flag to the OPTIONS variable, and restart the service with the "service ntpd restart" command.

```
# Drop root to id 'ntp:ntp' by default.
OPTIONS="-x -u ntp:ntp -p /var/run/ntpd.pid"
```

You can check your current NTP configuration by checking the process status and filtering on the ntp daemon. You should see the –x option specified as you see here:

```
$ ps -ef |grep ntp |grep -v grep
ntp       6496     1  0 Mar10 ?        00:00:00 ntpd -x -u ntp:ntp -p /var/run/ntpd.pid
```

BCT (Block Change Tracking)

If you are performing any kind of incremental backups, you need to enable BCT. On the primary database, issue the following if your database storage is on ASM:

```
alter database enable block change tracking using file '+DATA';
```

You can now enable block change tracking on the physical standby database to quickly identify the blocks that have changed since the last incremental backup. DBAs may not be aware, but enabling BCT on the standby database requires a license of the Oracle Active Data Guard. The option to generate the BCT file syntax is available in the Data Guard RMAN Backup to Disk Submenu (dg_rman2disk_menu.ksh).

You can view the filename, status and size of the BCT file with the following query:

```
SQL> select filename, status, bytes from v$block_change_tracking;

FILENAME                                       STATUS          BYTES
---------------------------------------------- ---------- ----------
+DATA/visk/changetracking/ctf.294.744629595  ENABLED      11599872
```

From time to time, you should make sure the change tracking is being used for RMAN backups:

```
  1  SELECT count(*)
  2  FROM v$backup_datafile
  3* where USED_CHANGE_TRACKING = 'NO'
SQL> /

  COUNT(*)
----------
         0
```

FRA (Fast Recovery Area)

As a general rule, configure the FRA and define your local archiving parameters to be:

```
LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

As part of best practices, you should always enable Flashback Database on primary and standby databases for easy re-instantiation after a failover. With Flashback Database enabled, you do not have to rebuild the primary database after a database failover. You can re-instate the failed primary database. Also Flashback Database provides the mechanism to expeditiously rewind the database after an erroneous batch update, a bad data load, user error(s), or a malicious set of activities on the database.

Oracle MAA recommends that DB_FLASHBACK_RETENTION_TARGET should be set to a minimum of 60 minutes if all you are trying to achieve re-instantiation of the primary database after a failover; however, if you require the additional protection from user errors and corruptions, then you will need to extend that time. Oracle MAA best practices recommend a minimum of 6 hours retention period. You need to determine what that retention period is based on your business requirements. The longer you set the flashback retention time, more disk space you will need.

You should also enable Flashback Database on the standby database to minimize downtime resulting from logical corruptions.

Automatic Archive Switch

Oracle natively provides the mechanism to force a log switch even if the database is idle or have very little redo log activity. This capability is not known to lot of DBAs, and they often write a shell script to force a log switch if the redo logs have not

switched in a specified time frame. Instead of writing a shell script, you should set the ARCHIVE_LAG_TARGET parameter.

ARCHIVE_LAG_TARGET parameter forces a log switch after a specified time interval in seconds. By default the ARCHIVE_LAG_TARGET parameter is set to 0 indicating that the primary database does not perform a time-based redo switch. The recommended setting for this parameter is 1800 seconds (30 minutes) which informs that primary database that it must switch logfiles every 30 minutes during times of low or no activity:

```
ALTER SYSTEM SET ARCHIVE_LAG_TARGET=1800 sid='*' scope=both;
```

LOG_ARCHIVE_MAX_PROCESSES

The default is still 2, and unfortunately, not enough. Archive process on the primary database are responsible for archiving online redo log files as they become full or resolve gaps in the redo stream to a standby database. On the primary database, one of the 2 default archive processes is limited to servicing only the ORL files and not allowed to communicate with the standby database at all. Other archive processes are allowed to service both. On the standby, log archive processes archive the SRLs and forward the archive logs to the cascade standby database.

As a minimum, LOG_ARCHIVE_MAX_PROCESSES should be set to 4 (max is 30). The DG Toolkit sets this parameter to a value of 5.

(PARALLEL EXECUTION MESSAGE SIZE)

PEMs specifies the size of messages for parallel execution (formerly known as parallel query, parallel DML, parallel Recovery, replication). Best practice recommendation is to set PEMS to 8k (8192) and as high as 64k (65535). PEMS is used by all parallel query operations and retrieves memory from the shared pool. The actual value might be automatically lowered to the IPC limit for the OS.

Be aware that this larger value has a direct relationship to a larger shared pool requirement. Larger values can result in significant performance improvement of parallel recovery at the cost of additional memory use.

Starting from Oracle Database 11g Release 2, this parameter is set to 16k right out of the box:

```
SQL> select name, value, isdefault from v$parameter where name like 'parallel_execut%';

NAME                                 VALUE      ISDEFAULT
------------------------------------ ---------- ---------
parallel_execution_message_size      16384      TRUE
```

DB CACHE SIZE

On the standby database, you can set DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE to zero and allocate additional memory to DB_CACHE_SIZE to a larger value than the primary database. Additionally, you can shrink your SHARED_POOL_SIZE on the standby database since MRP does not require much shared pool memory.

Furthermore, you should incorporate two SPFILES and flip the tuned initialization parameters back and forth as database role transitions occur. You can have a tuned set of parameters for production usage and another set of parameters for media recovery.

SRL (STANDBY REDO LOGS)

Standby redo logs are recommended for all Data Guard database configurations. Even though SRLs are not required for Maximum Performance mode, we recommend that you create them as they will improve redo transport speed, data recoverability, and apply speed.

You must create at a minimum the same amount of SRL groups as online redo logs (ORL) groups on the primary database. As best practice, you should create one more SRL group than the amount of ORL groups on the primary database. In addition, the SRLs should be of the same size. Lastly, SRLs should not be multiplexed.

In anticipation of the primary database possibly becoming the standby database as a result of a database switchover or failover, you should create SRLs on the primary database as well. The same rules apply when creating SRLs on the primary database in terms of the number of SRLs to be created.

Another important tip to share is that you can create SRLs on the primary database and create the physical standby database using the DUPLICATE TARGET DATABASE FOR STANDBY FROM ACTIVE DATABASE command so that RMAN will duplicate the SRLs automatically on the physical standby database. To automate creation of SRLs with the DG Toolkit, you can execute the ./dg_generate_standby_redo.ksh script. This script will generate two SQL scripts, one for the primary database (cr_standby_redo_p.sql) and the other for the standby database (cr_standby_redo_s.sql). You can identify the difference from the _p.sql versus the _s.sql suffix.

Here's an example of the output from a 2 node RAC configuration generated from option #100 of the Launch Build Standby Database Submenu which simply executes the ./dg_generate_standby_redo.ksh script:

```
100
Max Redo Group:        4
Redo Size:             50
Redo Count:            3
Thread Count:          2
# --
# -- On the Primary Database:  VISK
# --
alter database add standby logfile thread 1 group 5 ('+FRA/VISK/onlinelog/stdby_redo_05a.rdo') size
       50m;
alter database add standby logfile thread 1 group 6 ('+FRA/VISK/onlinelog/stdby_redo_06a.rdo') size
       50m;
alter database add standby logfile thread 1 group 7 ('+FRA/VISK/onlinelog/stdby_redo_07a.rdo') size
       50m;
alter database add standby logfile thread 2 group 8 ('+FRA/VISK/onlinelog/stdby_redo_08a.rdo') size
       50m;
alter database add standby logfile thread 2 group 9 ('+FRA/VISK/onlinelog/stdby_redo_09a.rdo') size
       50m;
alter database add standby logfile thread 2 group 10 ('+FRA/VISK/onlinelog/stdby_redo_10a.rdo') size
       50m;
# --
# -- On the Standby Database:  VISK_DR
# --
alter database add standby logfile thread 1 group 11 ('+FRA/VISK_DR/onlinelog/stdby_redo_11a.rdo') size
       50m;
alter database add standby logfile thread 1 group 12 ('+FRA/VISK_DR/onlinelog/stdby_redo_12a.rdo') size
       50m;
alter database add standby logfile thread 1 group 13 ('+FRA/VISK_DR/onlinelog/stdby_redo_13a.rdo') size
       50m;
alter database add standby logfile thread 2 group 14 ('+FRA/VISK_DR/onlinelog/stdby_redo_14a.rdo') size
       50m;
alter database add standby logfile thread 2 group 15 ('+FRA/VISK_DR/onlinelog/stdby_redo_15a.rdo') size
       50m;
alter database add standby logfile thread 2 group 16 ('+FRA/VISK_DR/onlinelog/stdby_redo_16a.rdo') size
       50m;
# --
# --
# --
# --
Execute SQL Script:  cr_standby_redo_p.sql on VISK
```

```
Execute SQL Script:  cr_standby_redo_s.sql on VISK_DR
```

The shell script determined that the maximum redo group number is 4 and starts to create the standby redo group from number 5. In addition, the script knows that the redo log size is 50 MB in size and creates the standby redo logs of the same size. More, the script determines the number of threads and creates groups for each corresponding thread in the designated disk group in ASM. You can view the full script here:

```
#!/bin/ksh

# Sample Syntax
# SQL> alter database add standby logfile group 5 ('+DG_DD501/clidbadr/onlinelog/stdby_redo_05a') size 100m;
# or
#      alter database add standby logfile group 5 ('+DG_DD501') size 100m;

[ "$CONF" = "" ] && export CONF=$PWD/dg.conf
. $CONF

. $PWD/dg_remote_connect.ksh
export SYSPASSWD=$(cat .syspasswd)
export P_CONNECT="(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = ${PRIMARY_HOST})(PORT = ${PRIMARY_PORT}))
(CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = ${PRIMARY_DB_SERVICE}) (UR = A)))"

export ORACLE_RECOMMENDED_ADDTIONS=1
# --
# -- We are choosing to fully qualify our standby logfiles
#
export DG_PREFIX=$PRIMARY_FLASH_DG/$PRIMARY_DB/onlinelog

export MAX_GROUP=log/max_group_$PRIMARY_DB.log
echo "set echo off ver off pages 0 trims on head off feed off
select max(group#) from gv\$log;
exit;" | sqlplus -s "sys/${SYSPASSWD}@\"$P_CONNECT\"" as sysdba > $MAX_GROUP

export MAX_GROUP=$(tail -1 $MAX_GROUP |sed -e 's/ //g')
echo "Max Redo Group: $MAX_GROUP"

export REDO_SIZE=log/redo_size_$PRIMARY_DB.log
echo "set echo off ver off pages 0 trims on head off feed off
select max(bytes/1024/1024) from gv\$log;
exit;" | sqlplus -s "sys/${SYSPASSWD}@\"$P_CONNECT\"" as sysdba > $REDO_SIZE

export REDO_SIZE=$(tail -1 $REDO_SIZE |sed -e 's/ //g')
echo "Redo Size: $REDO_SIZE"

export REDO_COUNT=log/redo_count_$PRIMARY_DB.log
echo "set echo off ver off pages 0 trims on head off feed off
-- select count(*) from (select distinct inst_id, thread#, sequence# from gv\$log) gvlog;
select count(distinct group#)+1 from v\$log where thread#=1;
exit;" | sqlplus -s "sys/${SYSPASSWD}@\"$P_CONNECT\"" as sysdba > $REDO_COUNT
export REDO_COUNT=$(tail -1 $REDO_COUNT |sed -e 's/ //g')
echo "Redo Count: $REDO_COUNT"

export THREAD_COUNT=log/thread_count_$PRIMARY_DB.log
echo "set echo off ver off pages 0 trims on head off feed off
select count(distinct thread#) from v\$log;
exit;" | sqlplus -s "sys/${SYSPASSWD}@\"$P_CONNECT\"" as sysdba > $THREAD_COUNT
export THREAD_COUNT=$(tail -1 $THREAD_COUNT |sed -e 's/ //g')
echo "Thread Count: $THREAD_COUNT"

export STANDBY_REDO_LOG_P=cr_standby_redo_p.sql
export STANDBY_REDO_LOG_S=cr_standby_redo_s.sql
[ -f $STANDBY_REDO_LOG_P ] && rm $STANDBY_REDO_LOG_P
[ -f $STANDBY_REDO_LOG_S ] && rm $STANDBY_REDO_LOG_S

export THREAD_NUMBER=1
export REDO_NUMBER=1

echo "# --"
echo "# -- On the Primary Database:  $PRIMARY_DB "
echo "# --"
```

```
while [[ $THREAD_NUMBER -le $THREAD_COUNT ]]
do
  while [[ $REDO_NUMBER -le $REDO_COUNT ]]
  do
  (( MAX_GROUP = $MAX_GROUP + 1 ))
  if [ $MAX_GROUP -lt 10 ]; then
    echo "alter database add standby logfile thread $THREAD_NUMBER group $MAX_GROUP
('${DG_PREFIX}/stdby_redo_0${MAX_GROUP}a.rdo') size ${REDO_SIZE}m;" |tee -a $STANDBY_REDO_LOG_P
  elif [ $MAX_GROUP -ge 10 ]; then
    echo "alter database add standby logfile thread $THREAD_NUMBER group $MAX_GROUP
('${DG_PREFIX}/stdby_redo_${MAX_GROUP}a.rdo') size ${REDO_SIZE}m;" |tee -a $STANDBY_REDO_LOG_P
  fi
  (( REDO_NUMBER = $REDO_NUMBER + 1 ))
  done
  (( THREAD_NUMBER = $THREAD_NUMBER + 1 ))
  REDO_NUMBER=1
done

echo "# --"
echo "# -- On the Standby Database:  $STANDBY_DB "
echo "# --"
export THREAD_NUMBER=1
export REDO_NUMBER=1
export DG_PREFIX=$STANDBY_FLASH_DG/$STANDBY_DB/onlinelog

while [[ $THREAD_NUMBER -le $THREAD_COUNT ]]
do
  while [[ $REDO_NUMBER -le $REDO_COUNT ]]
  do
  (( MAX_GROUP = $MAX_GROUP + 1 ))
  if [ $MAX_GROUP -lt 10 ]; then
    echo "alter database add standby logfile thread $THREAD_NUMBER group $MAX_GROUP
('${DG_PREFIX}/stdby_redo_0${MAX_GROUP}a.rdo') size ${REDO_SIZE}m;" |tee -a $STANDBY_REDO_LOG_S
  elif [ $MAX_GROUP -ge 10 ]; then
    echo "alter database add standby logfile thread $THREAD_NUMBER group $MAX_GROUP
('${DG_PREFIX}/stdby_redo_${MAX_GROUP}a.rdo') size ${REDO_SIZE}m;" |tee -a $STANDBY_REDO_LOG_S
  fi
  (( REDO_NUMBER = $REDO_NUMBER + 1 ))
  done
  (( THREAD_NUMBER = $THREAD_NUMBER + 1 ))
  REDO_NUMBER=1
done


echo "# --"
echo "# --"
echo "# --"
echo "# --"
echo "Execute SQL Script:  $STANDBY_REDO_LOG_P on $PRIMARY_DB"
echo "Execute SQL Script:  $STANDBY_REDO_LOG_S on $STANDBY_DB"
```

The generated SRL script is fully qualified with database, directory names, and file name. You can change the ALTER DATABASE ADD STANDBY LOGFILE command to just include your disk group name.

You can check an existing database for SRLs from the Data Guard Preliminary Check Submenu (`dg_preliminary_check_menu.ksh`). The DG Toolkit adheres to the best practices equation for the number for SRL is:

```
# of SRL = (# of ORL + 1 ) X Threads
```

The equation simply states the number of SRLs needs to be the number of ORLs plus one per RAC instance. Even though SRLs are not required for Maximum Performance, it is recommended that you still create SRLs.

LOGGING

You should set force logging on the primary database. As best practice, you should enable force logging on the primary database at the database level with the following command:

```
SQL> alter database force logging;
```

Before making the critical decision to enable or not to enable force logging, you should be aware that:

1. Temporary tablespaces and temporary segments are never logged
2. If you enable force logging on the primary database while the database is open, force logging will not be enabled until all of the current nologging activities are complete

In certain circumstances, you may opt to enable force logging at the tablespace level instead of at the database level. For example, you may have materialized views or temporary tables used for reports that may gain additional performance if you enable no logging at the tablespace level. If you enable no logging for these tablespaces, you should also enable monitoring for the remaining tablespaces that require force logging enabled. You can create an exception table or a .conf configuration file that lists the tablespaces that can have nologging enabled. All other tablespaces that show up as nologging should have alert escalations to responsible recipients.

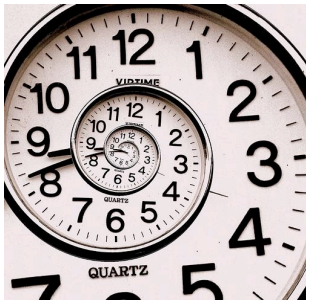
Look for a script in the DG Toolkit in the near future to monitor for tablespaces that are not being logged. Basically, you will be provided a configuration option to list the tablespaces that you wish to be excluded. These tablespaces will be your tablespaces that you explicitly wish to remain in nologging mode. All other tablespaces outside this list will throw an exception and an alert will be generated in a form of an email.

In the Preliminary Check Submenu, you can choose the option to check if the database is in force logging mode. This option also checks to see if no logging activities have occurred in the past at the datafile level and examines tablespaces to see if they are in nologging mode. Here's a sample output for the Check for forced logging and unrecoverable activities option:

```
30
Checking for forced logging at the database level
# ------------------------------------------------------------------------ #
#         Executing dg_check_force_logging.sql on DB:  VISK
# ------------------------------------------------------------------------ #
NO
# ----------------------------------------------------------#
Forced Logging is not enabled.  Please execute the following:
Syntax:  alter database force logging;
# ----------------------------------------------------------#

# ------------------------------------------------------------------------ #
#         Executing dg_check_unrecoverable.sql on DB:  VISK
# ------------------------------------------------------------------------ #
Checking for datafiles with unrecoverable activities
Checking for tablespace(s) that are not being logged
```

RTA(Real-time Apply)



Always enable RTA so that changes are applied as soon as the redo data is received! With the availability of SRLs, when the Remote File Server (RFS) process on the standby database server receives redo, the RFS process writes the redo data to the SRL. The Redo Apply will automatically apply the redo data directly from the SRL. You can utilize the dg_start.sql to enable the real-time apply feature of the physical standby:

```
alter database recover managed standby database using current logfile disconnect;
```

RTA is the default if you created your configuration with Data Guard Broker.

ARCHIVE_DEST_N Options

Always make sure that you have net_timeout and reopen options set. A net_timeout of 30 seconds is default in Oracle Database 11g, and 180 seconds in Oracle Database 10g Release 2. If you have a reliable network, you can set this to 10-15 seconds. Oracle does not recommend setting this option below 10 seconds as you will experience failed reconnects and can cause performance problems.

The reopen option controls the wait time before Data Guard will allow the primary database to attempt a reconnect. The default is 300 seconds (5 minutes). You will want to set this to 30 seconds or even 15 seconds to allow Data Guard to reconnect as soon as possible. With the setting of 5 minutes, DBAs often perceive that the Data Guard is not working right after it comes back online.

Compression is another important attribute to consider but requires the Oracle Advanced Compression license. In Oracle Database 11g Release 1, by default, Oracle will only compress archive logs when there is a gap and while using asynchronous redo transport mode. You can enable redo transport compression while in Maximum Performance protection mode by setting the undocumented the initialization parameter:

```
_REDO_TRANSPORT_COMPRESS_ALL=TRUE
```

Please review the following Metalink note: Redo Transport Compression in a Data Guard Environment [ID 729551.1] for details. You can enable compression by setting the COMPRESSION=ENABLE attribute of the LOG_ARCHIVE_DEST_x parameter or by editing the database property with the Broker as shown here:

```
DGMGRL> EDIT DATABASE 'visk' SET PROPERTY 'RedoCompression' = ENABLE;
```

In Oracle Database 11g Release 2, redo transport compression can be enabled with all protection modes.

Do not use MAX_CONNECTIONS in Oracle Database 11g. Although this parameter was introduced in Oracle Database 10g Release 2, it is not longer used in Oracle Database 11g. Using this option on Oracle Database 11g will impede redo transport performance.

DBAs often will delay apply of redo stream on the physical standby database to mitigate potential human induced errors or other hardware inflicted issues with the DELAY attribute. Instead of using the DELAY attribute, enable Flashback Database on both the primary and standby databases.

Archive Generation Rate

You need to know the amount of archivelogs that you are generating. Many DBAs calculate just the average archivelog generation per day. In addition, you will need to calculate the peak archive generation during a specified peak window to

determine how your batch jobs will behave. Knowing how to address high bursts of archive log generation will help you to determine if you can meet your RPO and RTO objectives. Companies are often behind by several hours in the morning after a big batch job in the middle of the night.

To determine your archivelog generation rate throughout the day, you can execute the following script (http://dataguardbook.com/Downloads/SQL/dg_archive_rates.sql):

```
cat dg_archive_rates.sql
set lines 255
set pages 14

SELECT
   TO_CHAR(TRUNC(FIRST_TIME),'Mon DD')                  "Date",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'00',1,0)),'9999')     "00",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'01',1,0)),'9999')     "01",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'02',1,0)),'9999')     "02",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'03',1,0)),'9999')     "03",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'04',1,0)),'9999')     "04",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'05',1,0)),'9999')     "05",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'06',1,0)),'9999')     "06",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'07',1,0)),'9999')     "07",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'08',1,0)),'9999')     "08",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'09',1,0)),'9999')     "09",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'10',1,0)),'9999')     "10",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'11',1,0)),'9999')     "11",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'12',1,0)),'9999')     "12",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'13',1,0)),'9999')     "13",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'14',1,0)),'9999')     "14",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'15',1,0)),'9999')     "15",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'16',1,0)),'9999')     "16",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'17',1,0)),'9999')     "17",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'18',1,0)),'9999')     "18",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'19',1,0)),'9999')     "19",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'20',1,0)),'9999')     "20",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'21',1,0)),'9999')     "21",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'22',1,0)),'9999')     "22",
   TO_CHAR(SUM(DECODE(TO_CHAR(FIRST_TIME,'HH24'),'23',1,0)),'9999')     "23"
FROM V$LOG_HISTORY
   GROUP BY TRUNC(FIRST_TIME)
   ORDER BY TRUNC(FIRST_TIME) DESC
/

set lines 66
```

A sample output from this query looks like this:

```
Date    00     01   02   03   04   05   06  07  08   09   10   11   12   13   14   15   16   17   18   19   20   21   22   23
------- ---- ---- ---- ---- ---- ---- ---- --- --- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
Feb 28     1    0    0    1    2    1    4   9   5    6    7    9    8    8    0    0    0    0    0    0    0    0    0    0
Feb 27     2    1    1    2    2    1    1   5   1    1    2    0    1    0    0    0    5    0    0    0    3    5    0    0
Feb 26    12    5   12    2    5    5    1   8   4    4    6    4    2    2    1    1    5    3    2    1    3    3    1    6
Feb 25     8    3    4    4    5    3    7   8   5    7    8    9    3    8   11    9   12   12    8    6    6   10   10   12
Feb 24    12   10   13    7    3    3    7   9   8    8    8    3    5    6    3    8    4    9   14   12    9    8   10   12
Feb 23    16   25   23   20   15   11    5  11   6    8    2    7    7    9    3    8    9   11   13   13    7    8    8   14
Feb 22     4    3    3    1    3    3    4   9   4    4    7    9    7    3    8    9    8    5   10    9    4    4   16   16
Feb 21     0    0    1    2    1    2    3   8   4    3    5    4    7    5    4    7   10    7    5    6    5   10    6    5
Feb 20     6   20   14    0    0    1    5   8   4    1    2    0    2    6    0    0    5    0    1    1    2    6    0    0
Feb 19     8    2    3    1    6    3    3   8   3    5    6    4    4    1    2    1    5    0    0    0    2    5    7   12
Feb 18     6    4    3    2    4    3    6   8   7    6    9   11    8   10    7   10   12    9    8    6    2    8   11    7
Feb 17     6    4    3    2    4    4    5   8   9    7   10    9    9    8    8    8   15    7   11    8    5    9    8    9
Feb 16     5    4    6    2    6    3    3   6   3    8    8   11   12   10   11   11   12    9    9    7    6    9    8    9
Feb 15     5    3    2    4    5    4    5   9   7    5    9   11    7    9    7    9   12    9    8   10    7    4   10   11
Feb 14     0    0    0    0    0    0    0   0   0    0    0    0    0    0    5   10   12    9    8    8    6    9    9    6

15 rows selected.
```

It is no longer uncommon to see 1GB or even 2GB redo logs in today's multi-terabyte databases and applications that have high transactional volumes. Ideally, a log switch should not occur more than once per 20 minutes during peak activity. You should consider sizing your redo logs accordingly to fit an archive switch within the specified window.

Providing the syntax to drop the existing redo logs and creating new redo logs is outside the scope of this paper; however, you can access pertinent scripts related to redo log management from http://dataguardbook.com/Downloads/redo/.

STANDBY FILE MANAGEMENT

Standby file management should always be set to TRUE. The DG Tooklit's ALTER SYSTEM syntax for the primary and standby database will always set the initialization parameter STANDBY_FILE_MANAGEMENT=TRUE. With the STANDBY_FILE_MANAGEMENT set to TRUE, you will not be able to perform the following commands:

1. alter database rename
2. alter database add or drop logfile
3. alter database add or drop standby logfile member
4. alter database create datafile as

If you need to issue the above commands, you will need to reset STANDBY_FILE_MANAGEMENT and perform your maintenance and re-enable it after completion of maintenance tasks.

RMAN DISK TO DISK (D2D) BACKUPS

RMAN backups are extensively covered in the DG Toolkit. DG Toolkit encompasses both RMAN backup sets and RMAN backup as copy options. You are encouraged to leverage the backup as copy option and update the baseline copy with incremental backups on a nightly basis especially if you have a VLDB situation. Conceptually, you will rarely need to perform a full level 0 image copy of the database again. Even with the backup as copy option, you should perform a full backup at least once per month or quarter basis.

Alternatively, the DG Toolkit also provides the option to perform level 0 backup sets and even compressed level 0 backup sets. The only caveat is that the DG Toolkit does not provide interfaces to the Media Management Layer (MML) so only D2D backups are supported.

DG Toolkit's D2D backups are implemented with the rman2disk.ksh script.

The `rman2disk.ksh` shell script has numerous pre-requisites and requirements. First and foremost, couple of directories need to be added to the OFA compliant directory structure. In the $ORACLE_BASE/admin/$ORACLE_SID directory, we need to add a log directory and a symbolic link called *bkups* that point to the backup destination for the database. Here's how the directory structure should look like from $ORACLE_BASE/admin/$ORACLE_SID:

```
$ ls -ltr
total 40
drwxr-xr-x   2 oracle oinstall
4096 Oct 16 16:24 exp
drwxr-xr-x   2 oracle oinstall
4096 Oct 16 16:24 adump
drwxr-xr-x  52 oracle oinstall
4096 Oct 23 12:12 cdump
```

First, the "*.sql" file are the rman scripts. You can see from our sample file listing that there are three kinds of *.sql files:

1. *_0.sql to denote level 0 backup sets
2. *_1.sql to denote incremental level 1 backups
3. *_baseline.sql script to denote rman database copy backups

The second kind of file is the log files. There are log files for each day of the backup. In addition, there is a history file to provide a high level backup statistics information of all the backups performed. You should purge *.log files on a regular basis such as anything older than 90 days except for the history log file. The history log file just stores high level information about the database backups such as when it started, when it ended and the parameters passed for the backups. The history log file will be handy to review previous backup trends. The history file is specified in the format: `rman2disk.ksh_[ORACLE_SID].log`. Here's a small excerpt of the history.log file:

```
Performing backup with arguments: -l 1 -d DBTOOLS1 -m  -n 2
Performing backup with arguments: -r
Spfile Backup:
Previous BASELINE Tag: DBTOOLS1_baseline_22Nov08_0139
#--------------------------------------------------------------------
```

```
lrwxrwxrwx   1 oracle oinstall
18 Nov 14 08:58 bkups ->
/data/oracle/DBTOOLS/bkups
drwxr-xr-x  11 oracle oinstall
20480 Nov 21 13:00 bdump
drwxr-xr-x   2 oracle oinstall
4096 Nov 22 09:48 udump
drwxr-xr-x   2 oracle oinstall
4096 Nov 22 09:57 log
```

Pay particular attention the log directory and the `bkups` symbolic link. The log directory will house all the rman backup scripts generated by the rman2disk korn shell script and the log files associated from the executed rman script. In a nutshell, rman2disk korn shell script creates an RMAN script and execute the script. The rman script is specified in the format:

- rman2disk.ksh_[date]_[time]_[level].sql

There are two basic kinds of files in the log directory:

```
-rw-r--r--  1 oracle oinstall  1107 Nov 20 21:16
rman2disk.ksh_20Nov08_2116_0.sql
-rw-r--r--  1 oracle oinstall 27654 Nov 20 22:31
rman2disk.ksh_20Nov08_2116_Level0.log
-rw-r--r--  1 oracle oinstall  1392 Nov 21 07:26
rman2disk.ksh_21Nov08_0726_1.sql
-rw-r--r--  1 oracle oinstall  6118 Nov 21 07:27
rman2disk.ksh_21Nov08_0726_Level1.log
-rw-r--r--  1 oracle oinstall   484 Nov 21 07:53
rman_copy.sql
-rw-r--r--  1 oracle oinstall   484 Nov 21 09:21
rman_copy2.sql
-rw-r--r--  1 oracle oinstall  1109 Nov 22 01:39
rman2disk.ksh_22Nov08_0139_baseline.sql
-rw-r--r--  1 oracle oinstall 29517 Nov 22 02:51
rman2disk.ksh_22Nov08_0139_Levelbaseline.log
-rw-r--r--  1 oracle oinstall  1237 Nov 22 09:43
rman2disk.ksh_22Nov08_0943_1.sql
-rw-r--r--  1 oracle oinstall 21420 Nov 22 09:44
rman2disk.ksh_22Nov08_0943_Level1.log
-rw-r--r--  1 oracle oinstall  1237 Nov 22 09:48
rman2disk.ksh_22Nov08_0948_1.sql
-rw-r--r--  1 oracle oinstall  6814 Nov 22 09:49
rman2disk.ksh_DBTOOLS1.log
-rw-r--r--  1 oracle oinstall 28976 Nov 22 09:49
rman2disk.ksh_22Nov08_0948_Level1.log
-rw-r--r--  1 oracle oinstall  1237 Nov 22 10:50
rman2disk.ksh_22Nov08_1050_1.sql
-rw-r--r--  1 oracle oinstall 30592 Nov 22 10:51
rman2disk.ksh_22Nov08_1050_Level1.log
-rw-r--r--  1 oracle oinstall   812 Nov 22 16:18
rman2disk.ksh_DBTOOLS1.history.log
```

```
# Backup Started: Sat Nov 22 10:50:19 EST 2008
#-------------------------------------------------------------------
Catalog:
#-------------------------------------------------------------------
# Backup Completed Succesfully for DBTOOLS1 on Sat Nov 22 10:51:01 EST
2008
#-------------------------------------------------------------------
DB Backup Size is:  8.80469
Archive Backup Size is:  5.02344
Performing backup with arguments: -l 1 -d DBTOOLS1 -m  -n 2
Performing backup with arguments: -r
Performing backup with arguments: -z
Spfile Backup:
Archive log mode: Y
```

Second, the DG Toolkit script starts with establishing ORACLE_BASE. ORACLE_BASE is set to the directory structure two level up from $ORACLE_HOME. In our examples, ORACLE_BASE is set to /apps/oracle

```
# --------------------
# Start of .ORACLE_BASE
export BASE_DIR=/apps/oracle
export ORACLE_BASE=/apps/oracle
export RMAN_CATALOG=RMANPROD
BACKUP_DEST=asm
```

The .ORACLE_BASE file is expected to always be located to the $HOME directory for the oracle user.

RMAN2DISK TEMPLATE FILES

Each of the template files serves a different purpose. For example, the `rman2disk.sql.cold` template will perform a cold backup. The `rman2disk.sql.compressed` will perform a compressed rman backup. It's pretty obvious what the `rman2.sql.noncompression` file would be. Based on the options specified in the command line argument of the `rman2disk.ksh` script, different template file will be sourced and modified accordingly to generate the rman script to execute.

The rman2disk.sql.cold template file simply performs a `shutdown immediate` and `startup mount` prior to issuing the backup database command. Once the backup of the database, archivelog, controlfile, pfile/spfile is complete, the script issues an `alter database open` command to open the database for general availability.

The rman scripts utilizes numerous template files as shown below:

```
-rw-r--r--  1 oracle oinstall 1232 Nov 22 01:37 rman2disk.sql.baseline
-rw-r--r--  1 oracle oinstall 1275 Nov 22 01:37 rman2disk.sql.cold
-rw-r--r--  1 oracle oinstall  963 Nov 22 15:57 rman2disk.sql.cold.noarchivelog
-rw-r--r--  1 oracle oinstall 1529 Nov 22 01:37 rman2disk.sql.compression
-rw-r--r--  1 oracle oinstall 1370 Nov 22 01:37 rman2disk.sql.for_recover
-rw-r--r--  1 oracle oinstall 1506 Nov 22 01:37 rman2disk.sql.nocompression
```

RMAN2DISK USAGE

```
#  Regular Usage with RMAN Backupsets:
#  1.  Perform a full level 0 backup
#      /apps/oracle/general/sh/rman2disk.ksh -d DATABASE -l 0 -c catalog >
/tmp/rman2disk_DATABASE.0.log 2>&1
#  2.  Perform an incremental backup
#       /apps/oracle/general/sh/rman2disk.ksh -d DATABASE -l 1 -c catalog >
/tmp/rman2disk_DATABASE.1.log 2>&1
#
#  Advanced Usage with Database Image Copies:
#  1.  Perform a baseline level 0 backup
#      /apps/oracle/general/sh/rman2disk.ksh -d DATABASE -l baseline >
/tmp/rman2disk_DATABASE.baseline.log 2>&1
#  2.  Perform an incremental backup
#       /apps/oracle/general/sh/rman2disk.ksh -d DATABASE -l 1 -r merge >
/tmp/rman2disk_DATABASE.1.log 2>&1
#  Notes:
#  1.  -r option can only be used with Image Copy Backups
#  2.  -r must find a baseline tag in the $SH directory; otherwise, the script will perform a full
level 0 copy
```

If you do not specify the –c option, the rman backup will not take advantage of the RMAN repository and will only leverage the controlfile. If you specify the –c option, you should create a .rman.pw password file in the same directory where you extracted the rman2disk.ksh shell script with the Unix permission of 600 for security reasons. The RMAN catalog can be designated in the .ORACLE_BASE file.

RMAN2 DISK PARAMETERS IN DETAIL

The left column of the table below provides the actual code example. The right side of the table provides the detailed explanation of each of the parameters:

```
while getopts :l:d:c:n:m:r:z: arguments
do
  case $arguments in
    c) CATALOG=${OPTARG}
       export CATALOG=$(echo $CATALOG |tr
'[A-Z]' '[a-z]')
       ;;
```

- `-l` option specifies the backup level. Valid options are 0 or 1
- `-d` option specifies the database name. The database must be listed in the `/etc/oratab` file.
- `-c` option specifies if you want to use the RMAN catalog to store the backup metadata. If the `-c` option is not specified, the script will not leverage the RMAN catalog and will default to the `nocatalog` syntax.
- `-n` option specifies the number of days of archive logs to retain. If the parameter is not specified, the number of days defaults to 2:

```
    d) DB=${OPTARG}
       ;;
    l) BACKUP_LEVEL=${OPTARG}
       ;;
    n) NUMBER_OF_DAYS_TO_RETAIN=${OPTARG}
       ;;
    m) BACKUP_MODE=${OPTARG}
       export BACKUP_MODE=$(echo
$BACKUP_MODE |tr '[A-Z]' '[a-z]')
       ;;
    r) RECOVERY_MODE=${OPTARG}
       export RECOVERY_MODE=$(echo
$RECOVERY_MODE |tr '[A-Z]' '[a-z]')
       ;;
    z) COMPRESSION_MODE=${OPTARG}
       export COMPRESSION_MODE=$(echo
$COMPRESSION_MODE |tr '[A-Z]' '[a-z]')
       ;;
    *) echo "${OPTARG} is not a valid
argument\n"
       echo "Usage is:  rman2disk.ksh -d
$DB -l [0 or 1 or baseline] [-m cold] [-n
# of days to retain archivelogs] [-r
merge]"
       exit -1
       ;;
  esac
done
```

```
[ "$NUMBER_OF_DAYS_TO_RETAIN" = "" ] && export
NUMBER_OF_DAYS_TO_RETAIN=2
```

- `-m` specifies the backup mode. Acceptable values for backup mode are cold or hot
- `-r` mode option can only be used with the copy backup command. The `-r` must find a baseline tag in the `$SH` directory. If the baseline tag is not found in the `$SH` directory, what is equivalent to a full level 0 copy database will be performed. As an extra feature, the baseline is established at the database level:
  `export PREVIOUS_BASELINE_TAG=$(cat $SH/rman2disk.tag.baseline.${ORACLE_SID})`

  History of all the baseline TAGs are preserved in a file called:
  `echo $TAG |tee -a $SH/rman2disk.tag.history`
  The history of all the baselines are kept at a global level. Every time you issue a rman2disk.ksh scripts with the –l baseline option, the baseline tag name is recorded in both the TAG file and the TAG history file:
  - `export TAG=${ORACLE_SID}_${BACKUP_LEVEL}_${ORADATE}`
  - `echo $BASELINE_TAG > $SH/rman2disk.tag.baseline.${ORACLE_SID}`
  - `echo $TAG |tee -a $SH/rman2disk.tag.history`
- `-z` option specifies the compression mode. Compression in normal circumstances yields incredible disk saving potential opportunities. For a specific customer who house compressed BLOBs inside the database, the compression option produced unexpected results resulting in backup times that are in orders of magnitude higher than non-compressed backupsets with compression savings of just 20-30%. For another customer, the compression option shrank their backups by 90% of the allocated database size.

Compression can produce incredible results. In this particular example, you can see a 1.91 TB database size be compressed down to 289.18 GB in size.

SESSION_KEY	INPUT_TYPE	Compression Ratio	IN_SIZE	OUT_SIZE	DAY	START_TIME	Time Taken	STATUS
…								
…								
11613	DB INCR	6.80	1.91T	289.18G	sat	19-feb 18:00	04:58:15	COMPLETED
11609	ARCHIVELOG	1.00	4.68G	4.68G	sat	19-feb 16:01	00:01:33	COMPLETED
11605	ARCHIVELOG	1.00	93.95G	93.95G	sat	19-feb 07:00	00:24:39	COMPLETED

BACKUPS FOR VLDB (VERY LARGE DATABASES)

Often DBAs think that backing up multi-terabyte databases is not a viable option because backups take too long and are worried about the amount of time to restore/recover the database in the event of a disaster. With RMAN image copy backups, full backups can be a rare event, and in the event of a SAN loss or groups of disk loss, you can "Switch to copy" and run your databases off of the image copy that's been updated on a nightly basis. If you architect a solution where the image copy backups go to another SAN, you can even survive a SAN failure.

Backup as copy plus incremental level 1 merge of the Database is a poor man's equivalent to EMC's sync and split technology with their business continuity volumes (BCVs) or Hitachi's shadow images (SI) technology. You can perform a level 0 database copy backup to the /bkups file system and from that point on perform level 1 incremental backups. Starting from Oracle Database 10g, RMAN provided the capability to take incremental backups and update the baseline level 0 image

backup. To support this functionality, we have to perform a special "backup as copy" of the database to copy the binary image of the database to the file system or ASM Fast Recovery Area (FRA). In the event of catastrophic failure to the primary disks, we can switch to copy and start the database from the image copy of the database.

The DG Toolkit provides the full capability to perform image copy backups and performs incremental updates automatically. You can choose the option to perform a backup as copy image full backup from the Data Guard RMAN Backup to Disk Submenu (dg_rman2disk_menu.ksh):

```
40
Backup Dest: fs
Performing backup with arguments: -l baseline -d VISK1 -m  -n 2
Performing backup with arguments: -r
Performing backup with arguments: -z
Spfile Backup:
Archive log mode: ARCHIVELOG
Initiating backup with database online
VISK1_baseline_01Mar11_0554
Using the template file: /home/oracle/work/dgmenu/rman2disk.sql.baseline for backup to fs

run
{
allocate channel d1 type disk;
allocate channel d2 type disk;
allocate channel d3 type disk;
allocate channel d4 type disk;

backup as copy incremental level 0 tag=VISK1_baseline_01Mar11_0554 format '/u01/app/oracle/admin/VISK1/bkups/%U'
(database) ;

#sql "create pfile=''/u01/app/oracle/admin/VISK1/bkups/init_VISK1_01Mar11_0554.ora'' from spfile";
sql "alter system archive log current";
sql "alter system switch logfile";
sql "alter system switch logfile";

#resync catalog;
change archivelog all validate;

sql "alter database backup controlfile to trace";
#sql "alter database backup controlfile to
''/u01/app/oracle/admin/VISK1/bkups/control01_VISK1_01Mar11_0554.ctl.bkup''";

backup as backupset format '/u01/app/oracle/admin/VISK1/bkups/%d.%s.%p.%t.A' skip inaccessible (archivelog all
not backed up 2 times);
backup tag=VISK1_CTL_01Mar11_0554 format '/u01/app/oracle/admin/VISK1/bkups/%d.%s.%p.%t.CTL' (current
controlfile);

delete noprompt archivelog until time 'sysdate - 2' backed up 2 times to device type disk;

release channel d1;
release channel d2;
release channel d3;
release channel d4;
}

# ---------------------------------------------------------------- #
Please look at the RMAN Backup Script:  /tmp/dba/VISK1/log/VISK1_01Mar11_0554_baseline.rman
# ---------------------------------------------------------------- #
```

To perform an incremental level 1 update to the level 0 baseline image copy, choose the option to perform a level 1 incremental backup for recover (UPDATE IMAGE COPY). This will generate the following rman backup script:

```
70
Backup Dest: fs
Performing backup with arguments: -l 1 -d VISK1 -m  -n 2
Performing backup with arguments: -r
Performing backup with arguments: -z
Spfile Backup:
Archive log mode: ARCHIVELOG
Initiating backup with database online
```

```
Previous BASELINE Tag: VISK1_baseline_01Mar11_0554
Using the template file: /home/oracle/work/dgmenu/rman2disk.sql.for_recover for backup to fs

run
{
allocate channel d1 type disk;
allocate channel d2 type disk;
allocate channel d3 type disk;
allocate channel d4 type disk;
recover copy of database with tag 'VISK1_baseline_01Mar11_0554';

BACKUP INCREMENTAL LEVEL 1 tag='VISK1_1_01Mar11_0602' FOR RECOVER OF COPY WITH TAG 'VISK1_baseline_01Mar11_0554'
format '/u01/app/oracle/admin/VISK1/bkups/%d.%s.%p.%t.L1.4R.DB' DATABASE;

#sql "create pfile=''/u01/app/oracle/admin/VISK1/bkups/init_VISK1_01Mar11_0602.ora'' from spfile";
sql "alter system archive log current";
sql "alter system switch logfile";
sql "alter system switch logfile";

#resync catalog;
change archivelog all validate;

sql "alter database backup controlfile to trace";
#sql "alter database backup controlfile to
''/u01/app/oracle/admin/VISK1/bkups/control01_VISK1_01Mar11_0602.ctl.bkup''";

backup as backupset format '/u01/app/oracle/admin/VISK1/bkups/%d.%s.%p.%t.A' skip inaccessible (archivelog all
not backed up 2 times);
backup tag=VISK1_CTL_01Mar11_0602 format '/u01/app/oracle/admin/VISK1/bkups/%d.%s.%p.%t.CTL' (current
controlfile);

delete noprompt archivelog until time 'sysdate - 2' backed up 2 times to device type disk;

release channel d1;
release channel d2;
release channel d3;
release channel d4;
}

# ------------------------------------------------------------------ #
Please look at the RMAN Backup Script:  /tmp/dba/VISK1/log/VISK1_01Mar11_0602_1.rman
# ------------------------------------------------------------------ #
```

You can take the two scripts provided and implement a backup solution for your VLDB database.

BACKUP SCHEDULE:

At a minimum, you should perform a full level 0 backup to disk on a monthly/quarterly basis. You should also perform full backup of the /bkup file system to tape on a weekly basis. Your RMAN backup schedule will resemble something like this:

Sunday: Level 1 Merge with Level 0 => it is LEVEL 0

Monday: Level 1 Merge with Level 0 => it is LEVEL 0

Tuesday Level 1 Merge with Level 0 => it is LEVEL 0

Wednesday: Level 1 Merge with Level 0 => it is LEVEL 0

Thursday: Level 1 Merge with Level 0 => it is LEVEL 0

Friday: Level 1 Merge with Level 0 => it is LEVEL 0

Saturday: Level 1 Merge with Level 0 => it is LEVEL 0

Conceptually: We do not need to perform another full backup again but we recommend that you take a full image copy at least once per month or quarter depending on your rate of change and amount of space you have.

DUPLICATE DATABASE (FROM ACTIVE DATABASE)

The DG Toolkit primarily focuses on creating a standby database with the traditional convention where DBAs:

1. Perform an RMAN backup of the primary database
2. Create the standby controlfile
3. Copy the backup of database/standby controlfile/SPFILE to the standby database server
4. Copy the orapw password file to the standy database server
5. Restore the SPFILE and standby controlfile
6. Restore the database from the RMAN backup
7. Configure both primary and standby database with Data Guard specific initialization parameters
8. Start Managed Recovery Process

You can step through each of the tasks mentioned above in the Build Standby Database Submenu screen of the DG Toolkit. For complete details on how to fully leverage the DG Toolkit to build a physical standby database, please download the DG Toolkit Create Standby Database PDF file from DBAExpert.com website:

http://dbaexpert.com/dg/DG%20Toolkit%20-%20Create%20Standby%20Database.pdf

This PDF document provides detailed scripts and automated scripts to transfer RMAN backupsets from one ASM diskgroup in a RAC environment to another ASM diskgroup on another RAC environment. If you happen to be in a 100% ASM configuration, this paper is for you!

Now, let's go back to the DG Toolkit. Another particular menu option in the DG Toolkit that you will be interested is the ability to create a physical standby database without performing an RMAN backup of the database on the primary database. As of Oracle Database 11g, Oracle provides the capability to create a physical standby database from an active copy over the network of a database without a requirement for a backup. With this technique, you can reduce the amount of work to create a physical standby database especially if your source and target database happen to be at the same data center.

You can leverage a comprehensive korn shell script called dg_duplicate_database.ksh to auto-generate the DUPLICATE TARGET DATABASE FOR STANDBY FROM ACTIVE DATABASE command:

```
#!/bin/ksh
[ "$CONF" = "" ] && export CONF=$PWD/dg.conf
. $CONF

[ "$PRIMARY_DB" = "" ] && { echo "PRIMARY_DB environment variable must be set!"; exit 1; }

#export ALTER_SYSTEM_CONF=$PWD/dg.conf
#. $ALTER_SYSTEM_CONF

export PASSWORD_FILE=$PWD/.syspasswd
[ ! -f "$PASSWORD_FILE" ] && { echo "Password file does not exist.  Please create the .syspasswd file."; exit 1;
}
export sys_pass=$(cat $PASSWORD_FILE |head -1)

export DG_TEMPLATE=dg_duplicate_database_template.txt;
[ "$DG_TEMPLATE" = "" ] && { echo "Template file not found .. exiting!"; exit 1; }

cat $DG_TEMPLATE |sed -e "s*###_PRIMARY_DATABASE_###*$PRIMARY_DB*g" \
                      -e "s*###_STANDBY_DATABASE_###*$STANDBY_DB*g" \
                      -e "s*###_PRIMARY_DB_###*$PRIMARY_DB*g" \
                      -e "s*###_STANDBY_DB_###*$STANDBY_DB*g" \
                      -e "s*###_SYS_PASSWORD_###*$sys_pass*g" \
                      -e "s*###_PRIMARY_DATA_DG_###*$PRIMARY_DATA_DG*g" \
                          -e "s*###_PRIMARY_FLASH_DG_###*$PRIMARY_FLASH_DG*g" \
                          -e "s*###_STANDBY_DATA_DG_###*$STANDBY_DATA_DG*g" \
```

```
-e "s*###_STANDBY_FLASH_DG_###*$STANDBY_FLASH_DG*g" \
-e "s*###_DG_PD102_###*$DATA_DG2*g" \
-e "s*###_DG_PD101_###*$DATA_DG*g" \
-e "s*###_FLASH_DG_###*$FLASH_DG*g" \
-e "s*###_PRIMARY_DATABASE_TNS_###*${PRIMARY_DB}_PRI*g" \
-e "s*###_STANDBY_DATABASE_TNS_###*${PRIMARY_DB}_STDBY*g" \
-e "s*###_STANDBY_DATABASE_CLONE_###*${PRIMARY_DB}_CLONE*g" \
-e "s*###_PRIMARY_DB_INSTANCE_###*${PRIMARY_DB_INSTANCE}*g" \
-e "s*###_STANDBY_DB_INSTANCE_###*${STANDBY_DB_INSTANCE}*g" \
-e "s*###_STANDBY_HOST_###*${STANDBY_HOST}*g" \
-e "s*###_STANDBY_PORT_###*${STANDBY_PORT}*g" \
-e "s*###_PRIMARY_ORACLE_HOME_###*${PRIMARY_ORACLE_HOME}*g" \
-e "s*###_STANDBY_ORACLE_HOME_###*${STANDBY_ORACLE_HOME}*g" \
-e "s*###_STANDBY_SERVER_###*${STANDBY_SERVER}*g" \
-e "/^--/d"
```

At the heart of the shell script is the sed script that parses the duplicate database template file called dg_duplicate_database_template.txt and replaces the variable holders with values derived from the dg.conf file. Executing the dg_duplicate_database.ksh script yields the following output based on your dg.conf configuration file and clear set of instructions that serves as a step-by-step process to build your Data Guard environment:

```
300
# ----------------------------------------------------------------------------------------
# -- Set ORACLE Environment
# ----------------------------------------------------------------------------------------
export ORACLE_SID=VISK_DR1
export PATH=/usr/local/bin:$PATH
. oraenv

# ----------------------------------------------------------------------------------------
# -- Add the listener list entries for both primary and standby database listener.ora files
# ----------------------------------------------------------------------------------------
#-- Primary
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
    (SDU=32767)
      (GLOBAL_DBNAME = VISK1)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/db)
      (SID_NAME = VISK1)
    )
  )

# -- Standby
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
    (SDU=32767)
      (GLOBAL_DBNAME = VISK_DR1)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/db)
      (SID_NAME = VISK_DR1)
    )
  )

# --
# 1.  Reload the listener on both the primary and standby database server
#     lsnrctl reload listener

# ----------------------------------------------------------------------------------------
# -- Add the following to your tnsnames.ora file on both the primary and standby database
# ----------------------------------------------------------------------------------------
VISK_CLONE =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = rac562-vip)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SID=VISK_DR1)
    )
```

```
  )

# ----------------------------------------------------------------------------------------
# -- Create password file on the standby database server: # rac562
# ----------------------------------------------------------------------------------------
orapwd file=/u01/app/oracle/product/11.2.0/db/dbs/orapwVISK_DR1 entries=25 password=oracle123

# ----------------------------------------------------------------------------------------
# -- Create the following initialization file for the VISK_DR1 instance:
# -- /u01/app/oracle/product/11.2.0/db/dbs/initVISK_DR1.ora
# ----------------------------------------------------------------------------------------
cat <<EOF > /u01/app/oracle/product/11.2.0/db/dbs/initVISK_DR1.ora
db_name=VISK
db_unique_name=VISK_DR
cluster_database=false
EOF

# ----------------------------------------------------------------------------------------
# -- Execute the following RMAN script on the standby database server
# --
# -- First startup nomount the database with either SQL*PLUS or RMAN>
# ----------------------------------------------------------------------------------------
echo "startup nomount;
alter system register;" |sqlplus -s / as sysdba

# ----------------------------------------------------------------------------------------
# --
# ----------------------------------------------------------------------------------------
rman <<EOF
connect target sys/oracle123@VISCO_PRI;
connect auxiliary sys/oracle123@VISCO_CLONE;
run {
allocate channel prmy1 type disk;
allocate channel prmy2 type disk;
allocate channel prmy3 type disk;
allocate channel prmy4 type disk;
allocate auxiliary channel stby type disk;
duplicate target database for standby from active database
spfile
parameter_value_convert 'VISCO','VISCO_DR'
set 'db_unique_name'='VISCO_DR'
set 'db_file_name_convert'='+DATA/VISCO','+DATA_DR/VISCO_DR'
set log_file_name_convert='+FRA/VISCO','+FRA_DR/VISCO_DR','+DATA/VISCO','+DATA_DR/VISCO_DR'
set control_files='+FRA_DR/VISCO_DR/control.ctl'
set log_archive_max_processes='5'
set fal_client='VISCO_STDBY'
set fal_server='VISCO_PRI'
set standby_file_management='AUTO'
set log_archive_config='dg_config=(VISCO,VISCO_DR)'
set log_archive_dest_1='LOCATION=+DATA_DR VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=VISCO_DR'
set log_archive_dest_2='service=VISCO_PRI LGWR ASYNC reopen=30 valid_for=(ONLINE_LOGFILES,PRIMARY_ROLE)
db_unique_name=VISCO'
set log_archive_dest_state_1='enable'
set log_archive_dest_state_2='enable'
set db_create_file_dest='+DATA_DR'
set db_create_online_log_dest_1='+DATA_DR'
set db_create_online_log_dest_2='+FRA_DR'
set cluster_database='FALSE'
set parallel_execution_message_size='8192'
```

```
set db_lost_write_protect='TYPICAL'
set db_block_checking='TRUE'
set db_block_checksum='FULL'
nofilenamecheck
;

sql channel prmy1 "alter system set log_archive_config=''dg_config=(VISCO,VISCO_DR)''";
sql channel prmy1 "alter system set log_archive_dest_2= ''service=VISCO_STDBY LGWR ASYNC reopen=30
valid_for=(online_logfiles,primary_role) db_unique_name=VISCO_DR''";
sql channel prmy1 "alter system set log_archive_max_processes=5";
sql channel prmy1 "alter system set fal_client=VISCO_PRI";
sql channel prmy1 "alter system set fal_server=VISCO_STDBY";
sql channel prmy1 "alter system set standby_file_management=auto";
sql channel prmy1 "alter system set log_archive_dest_state_2=enable";
sql channel prmy1 "alter system set parallel_execution_message_size=8192 scope=spfile sid=''*''";
sql channel prmy1 "alter system archive log current";

sql channel stby "alter database recover managed standby database using current logfile disconnect";
}
EOF
```

Before we begin our duplicate database process, we establish our Oracle environment by setting the ORACLE_SID and source the oraenv file from /usr/local/bin directory. After we source the oraenv file, we create the password file with the password that is supplied from the .syspasswd file. Next, we will set the minimal set of Oracle initialization parameters in the init${STANDBY_DB}.ora file, startup instance in nomount mode, and register the instance with the database listener.

As you can see, the LISTENER.ORA file entries are created with instructions to reload the listener afterwards. You will also notice three tnsnames.ora file entries are needed, one for the auxiliary instance, which utilizes the database SID, and two others, one for the primary and one for the standby database, which utilizes service names.

In this particular example, spfile from the primary database is copied from the primary database to the standby database ORACLE_HOME. In addition, standby redo logs that already exist on the primary database is also cloned during the duplicate database process. Unfortunately, the output of this duplicate database command is too long to display in this paper. Please visit the following URL:

http://DBAExpert.com/dg/dup.rman	To download the duplicate database script
http://DBAExpert.com/dg/dup.txt	To peruse the logfile

MONITORING

ALERT LOG MONITORING

Monitoring the database alert log is your first line of defense to identifying and troubleshooting your Data Guard issues. The alert log monitoring script should focus on mining for ORA- errors spewed in the alert_${ORACLE_SID}.log file since most of the Data Guard issues are visible in the database alert log.

You can download the latest and greatest alert log monitoring script from the Data Guard book website:
http://www.dataguardbook.com/Downloads/scripts/alert_log_monitor.ksh

Since this script is rather long, the script will not be provided in the white paper. DBAs who are new to Oracle Database 11g find it surprising that the alert log file is located in a completely different location. For those who are new to Oracle Database 11g, we recommend that you create a symbolic link from $ORACLE_BASE/admin/$ORACLE_SID/bdump directory to the trace directory in the $DIAG_DEST subdirectory to simplify alert log management in starting in Oracle Database 11g. You can also leverage the same shortcuts and aliases that you have today. Here's an example of a symbolic link that points to the alert log in the trace directory:

```
rac03:/u01/app/oracle/admin/VISK/bdump
VISK3 > ls -ltr
total 8
lrwxrwxrwx 1 oracle oinstall 62 Nov 21 13:56 alert_VISK.log -> /u01/app/oracle/diag/rdbms/visk/VISK3/trace/alert_VISK3.log
```

To quickly change your directory to the trace directory within the diagnostic destination, you can also utilize following function in your .profile or .aliases file:

```
function trace {
export ASMDB=$1
# DB = The sed command strips off the last character of ORACLE_SID off
export DB1=$ORACLE_SID
export DB=$(echo $ORACLE_SID|tr '[A-Z]' '[a-z]' |sed -e '$s/.$//')
export DBLOWER=$(echo $ORACLE_SID|tr '[A-Z]' '[a-z]')
[ "$ASMDB" = "" ] && export ASMDB=rdbms
if [ -d "$ORACLE_BASE/diag/$ASMDB/$DB/$ORACLE_SID/trace" ]; then cd
$ORACLE_BASE/diag/$ASMDB/$DB/$ORACLE_SID/trace; fi
if [ -d "$ORACLE_BASE/diag/$ASMDB/$DB1/$ORACLE_SID/trace" ]; then cd
$ORACLE_BASE/diag/$ASMDB/$DB1/$ORACLE_SID/trace; fi
if [ -d "$ORACLE_BASE/diag/$ASMDB/$DBLOWER/$ORACLE_SID/trace" ]; then cd
$ORACLE_BASE/diag/$ASMDB/$DBLOWER/$ORACLE_SID/trace; fi
}
```

Once you source this function, you can simply type the command, "trace", and your current directory will be changed to the trace directory associated with your instance in the diagnostic destination. If your instance happens to be an ASM instance, you need to type the command "trace asm" to inform the script that you instance is of type ASM.

Monitoring MRP (Managed Recovery Process)

Progress of the standby database in recovery mode is exposed in the V$MANAGED_STANDBY. This view provides information regarding current activities for Redo Apply and Redo Transport Services. You can execute the following query to look at processes relatively to MRP such as ARCH (archiver process), LGWR (log writer), RFS (Remote File Server) , and LNS (Logwriter Network Service):

```
dg_check_mrp.sql
set time on
set lines 132
set pagesize 9999
col client_pid format a12
SELECT PID, PROCESS, STATUS,
       CLIENT_PROCESS, CLIENT_PID,
       THREAD#, SEQUENCE#, BLOCK#,
       BLOCKS, DELAY_MINS
FROM V$MANAGED_STANDBY
;
```

Here's a sample output of the dg_check_mrp.sql script:

```
# ---------------------------------------------------------------------- #
#         Executing dg_check_mrp.sql on DB:  VISK_DR
# ---------------------------------------------------------------------- #
       PID PROCESS   STATUS       CLIENT_P CLIENT_PID      THREAD#  SEQUENCE#     BLOCK#     BLOCKS DELAY_MINS
---------- --------- ------------ -------- ------------ ---------- ----------
      7102 ARCH      CLOSING      ARCH     7102                  1         20      88065        186          0
      7119 ARCH      CLOSING      ARCH     7119                  1         22      90113        854          0
      7121 ARCH      CONNECTED    ARCH     7121                  0          0          0          0          0
      7123 ARCH      CLOSING      ARCH     7123                  1         21      88065        186          0
      7225 RFS       IDLE         UNKNOWN  7842                  0          0          0          0          0
     30723 RFS       IDLE         UNKNOWN  7844                  0          0          0          0          0
     30721 RFS       IDLE         UNKNOWN  7835                  0          0          0          0          0
      7223 RFS       IDLE         LGWR     7850                  1         23      10334          1          0
     30644 MRP0      APPLYING_LOG N/A      N/A                   1         23      10333     102400          0

9 rows selected.
```

Checking MRP status is an option in the Monitor Physical Standby Data Guard Submenu.

Monitoring Archive Log Gaps

Archive log gaps should be monitored in a timely manner to make sure that a gap does not exceed a large amount of archivelogs for an extended period of time. Most archive gaps are automatically resolved by the ping ARCH process or by the Fetch Archive Log (FAL) process. Typically, DBAs do not have to intervene. When extended outages occur or multiple gap sequences are present, you may have to manually copy or restore the archivelogs from the primary database to the standby database.

You can monitor for archive log gaps from the DG Toolkit's Data Guard Physical Standby Submenu (dg_physical_standby_menu.ksh):

```
40
# ---------------------------------------------------------------------- #
#         Executing dg_check_missing_arc.sql on DB:  VISK
# ---------------------------------------------------------------------- #
DB: DBATOOLS - Thread#: 1 - Last Sequence: 22
DB: DBATOOLS - Thread#: 2 - Last Sequence: 7
# ---------------------------------------------------------------------- #
#         Executing dg_check_missing_arc.sql on DB:  VISK_DR
# ---------------------------------------------------------------------- #
DB: DBATOOLS_WA - Thread#: 1 - Last Sequence: 22
DB: DBATOOLS_WA - Thread#: 2 - Last Sequence: 7

50
# ---------------------------------------------------------------------- #
#         Executing dg_check_gap.sql on DB:  VISK_DR
# ---------------------------------------------------------------------- #
# ------------------------------------------------- #
DB Unique Name: VISK_DR
# ------------------------------------------------- #
```

Monitoring Transport and Apply Lag

You can review both the transport and apply lag by looking at the statistics from the V$DATAGUARD_STATS view. By monitoring the transport lag and apply lag, you can effectively determine if you are complying with your RPO and RTO. Here's a simple script to view how far your physical standby is:

```
dg_lag.sql
col name for a13
col value for a20
col unit for a30
set lines 122
SELECT NAME, VALUE, UNIT, TIME_COMPUTED
FROM V$DATAGUARD_STATS
```

```
WHERE NAME IN ('transport lag', 'apply lag');
```

Here's a simple execution of the dg_lag.sql script from the DG Toolkit's Data Guard Physical Standby Submenu (dg_physical_standby_menu.ksh):

```
70
# ------------------------------------------------------------------------ #
#         Executing dg_lag.sql on DB:  VISK_DR
# ------------------------------------------------------------------------ #

NAME          VALUE                UNIT                          TIME_COMPUTED
------------- -------------------- ------------------------
transport lag +00 00:01:10         day(2) to second(0) interval   09/12/2010 21:41:37
apply lag     +00 00:01:21         day(2) to second(0) interval   09/12/2010 21:41:37
```

The transport lag of 01:10 indicates that the shipment of redo from the primary to the physical standby is behind by 1 minute and 10 seconds. In the event of a catastrophe, you can lose 1 minute 10 second's worth of data if you lose your primary site or have to wait 1 minute 10 seconds if you want to perform a switchover.

You will also notice that the apply lag to be 01:21 indicating that the apply lag is behind by 1 one minute and 21 seconds.

Monitoring Apply Rate

You can view recovery performance in real-time by querying the V$RECOVERY_PROGRESSS view as shown in the dg_apply_rate.sql SQL script:

```
dg_apply_rate.sql
col type for a15
set lines 122
set pages 33
col item for a20
col units for a15

select to_char(start_time, 'DD-MON-RR HH24:MI:SS') start_time,
       item,  sofar, units
from v$recovery_progress
where (item='Active Apply Rate'
       or item='Average Apply Rate'
       or item='Redo Applied')
/
```

The output of this query can be used determine how fast the standby database is able to keep up the primary database. Couples of things to note are:

1. Average apply rate includes the think time for waiting for the redo to arrive
2. Active apply rate is the calculation of redo applied over time based on a moving window during the last 3 minutes

Here's a simple execution of the dg_apply_rate.sql script from the DG Toolkit's Data Guard Physical Standby Submenu (dg_physical_standby_menu.ksh):

```
60
# ------------------------------------------------------------------------ #
#         Executing dg_apply_rate.sql on DB:  VISK_DR
# ------------------------------------------------------------------------ #

START_TIME                ITEM                     SOFAR UNITS
------------------------- -------------------- ---------- ------------
27-AUG-10 02:13:59        Active Apply Rate         22325 KB/sec
27-AUG-10 02:13:59        Average Apply Rate        11112 KB/sec
27-AUG-10 02:13:59        Redo Applied               6695 Megabytes
```

How Far Am I Really Behind

The DG Toolkit executes the dg_lag_time.ksh shell script to determine how far we are behind by comparing the SCNs on the primary and standby database with dg_current_scn.sql and converts the SCN to timestamp with dg_scn_to_timestamp.sql. Lastly, the shell script performs a subtraction algorithm of the primary database timestamp compared to the standby database timestamp and produces the delta based on weeks, days, hours, minutes and seconds. Here's an example of the output from the Data Guard Physical Standby Submenu (dg_physical_standby_menu.ksh):

```
80
Printing Standby Current SCN and Primary Current SCN
# ------------------------------------------------------------------------ #
#         Executing dg_current_scn.sql on DB:  VISK_DR
# ------------------------------------------------------------------------ #
    2226080
# ------------------------------------------------------------------------ #
#         Executing dg_current_scn.sql on DB:  VISK
# ------------------------------------------------------------------------ #
    2402183

DR_SCN:  2226080
SCN:  2402183
Printing Standby Current SCN To Timestamp and Primary Current SCN To Timestamp
# ------------------------------------------------------------------------ #
#         Executing dg_scn_to_timestamp.sql on DB:  VISK
# ------------------------------------------------------------------------ #
01-MAR-11 03.32.35.000000000 AM
27-FEB-11 11.30.59.000000000 PM

PRIMARY                        DR                                WKS DAYS HR MI SE
------------------------------ --------------------------------- ---- ----- -- -- --
01-MAR-11 03.32.32.000000000 AM  27-FEB-11 11.30.59.000000000 PM     0     1 04 01 33
```

From this output, you can see that the standby database is over 1 day 4 hours and 1 minute behind.

Maintenance

Switchover

By far the easiest way to perform a switchover is to leverage the DG Broker. Switchovers can be performed using Oracle Enterprise Manager, the Data Guard broker command-line interface, or SQL*Plus statements.

With a single command on the primary database, you can perform a graceful switchover to your standby database. There is no RAC auxiliary instance to shutdown and restart for a physical standby switchover, no restart of the old primary, or no manual startup of the apply process. The DG Toolkit provides this simple interface in the Launch the Data Guard Broker Submenu:

```
50
Execute the following on the VISK:  switchover to VISK_DR1
from DGMGRL>
```

Even though you are not executing SQL*Plus, you still need to perform all the pre-requisite checks for configuration completeness, checking for the Broker completeness, and performing preparatory steps prior to executing the switchover command. Majority of these checks can be accomplished with the DG Toolkit in the Data Guard Broker Submenu.

Here are some best practice considerations prior to performing a graceful switchover to the standby database:

- Reduce the number of ARCH processes to the minimum needed for both remote and local archiving
- Terminate all sessions on the primary database

- Online redo logs on the target physical standby need to be cleared before that standby database can become a primary database. Although this will automatically happen as part of the SWITCHOVER TO PRIMARY command, it is recommended that the logs are cleared prior to the switchover

```
SELECT DISTINCT L.GROUP#
FROM V$LOG L, V$LOGFILE LF
WHERE L.GROUP# = LF.GROUP#
AND L.STATUS NOT IN ('UNUSED', 'CLEARING','CLEARING_CURRENT');
```

  If the above query produces output, on the target physical standby issue the following statement for each GROUP# returned:

```
ALTER DATABASE CLEAR LOGFILE GROUP <ORL GROUP# returned from the query above>;
```

- For Oracle Database 11.1 and below, check to see if the standby has ever been open in read-only mode:

```
SELECT VALUE
FROM V$DATAGUARD_STATS
WHERE NAME='standby has been open';
```

- Disable jobs on primary database so it will not submit additional jobs and block further job submissions:

```
ALTER SYSTEM SET job_queue_processes=0 SCOPE=BOTH SID='*';
EXECUTE DBMS_SCHEDULER.DISABLE( <job_name> );
```

For additional details pertaining to switchover steps, please review MetaLink Note 751600.1

SWITCHOVER TRACING

You can trace the switchover (comes in handy for over-all Data Guard related troubleshooting) activities by turning on the Data Guard tracing level on primary and standby to see exactly where the switchover activities are. LOG_ARCHIVE_TRACE is an optional parameter but very useful to view details of the switchover. You can set the Data Guard trace level to 8192 on both the primary and the target physical standby databases

```
ALTER SYSTEM SET log_archive_trace=8192;
```

LOG_ARCHIVE_TRACE value of 8192 provides the greatest level of auditing and will include tracking REDO Apply activity on the physical standby. Higher the value indicates the greater level of tracing and is inclusive of the lower value trace options. LOG_ARCHIVE_TRACE of 1 tracks archiving of redo log files. LOG_ARCHIVE_TRACE setting of 2 tracks the archive status per archived redo log destination and plus LOG_ARCHIVE_TRACE of 1. Other levels that you may be interested are level 128 which tracks the FAL server process activity, 1024 which tracks the RFS process activity and 4096 which tracks real-time apply activity.

You can issue the "tail –f" command on the alert logs on both primary and standby database.

GRP (GUARANTEED RESTORE POINT)

Just like you perform a GRPs prior to database upgrade or application upgrades, you should take a GRP before a switchover. To perform a GRP, you can execute the following command:

```
CREATE RESTORE POINT <guaranteed_restore_point_name> GUARANTEE FLASHBACK DATABASE;
```

To revert the database back to it's original state, you can flashback to the <guaranteed_restore_point_name> with the following command:

```
STARTUP MOUNT FORCE;
FLASHBACK DATABASE TO RESTORE POINT <guaranteed_restore_point_name>;
```

UTILIZE DATA GUARD BROKER

For command-line DBAs, you need to take the next step and start leveraging the Data Guard Broker (if you are not already doing so). As another best practice, you should leverage the Data Guard Broker for maintaining your Data Guard environment. Data Guard Broker is the foundation for Oracle Enterprise Manager Grid Control so it only helps to know what is happening behind the scene as Grid Control monitors and maintains your Data Guard configuration. The next several sections pertain to best practices for Data Guard Broker configurations.

The DG Toolkit has a dedicated screen for the Data Guard Broker:

```
# ------------------------------------------------------------------------ #
#  Data Guard Broker SubMenu: dg_broker_menu.ksh
#  Primary Host:  rac561        Standby Host:  rac562
# ------------------------------------------------------------------------ #
#  Generate Syntax for Broker Setup and Configuration Menu                 #
# ------------------------------------------------------------------------ #
#  1.  Alter System Syntax for DG_BROKER Setup (Initialization Parameters)  #
#  2.  Generate LISTENER.ORA entries for DG Broker - uses nslookup command  #
#  3.  Create Broker Configuration                                          #
#  4.  Enable Flashback on both databases - requires DB bounce              #
#  5.  Start Observer Process on another server - should not be part of DG  #
#  6.  Change Protection Mode to Maximum Availability                       #
#  7.  Create Fast-Start Failover Configuration                             #
# ------------------------------------------------------------------------ #
#  Broker Reporting Menu                                                   #
# ------------------------------------------------------------------------ #
#  11.  Show Data Guard Configuration                                       #
#  12.  Show Database Configuration of VISK on rac561
#  13.  Show Database Configuration of VISK_DR  on rac562
#  14.  Show Status Report of VISK on rac561
#  15.  Show Status Report of VISK_DR on rac562
#  30.  View Primary Send Queue on VISK
#  31.  View Standby Receive Queue on VISK_DR
#  32.  View Top Wait Events on VISK_DR
#  33.  View Database Inconsistent Properties on VISK
#  34.  View Database Inconsistent Properties on VISK_DR
# ------------------------------------------------------------------------ #
#   Data Guard Broker Log Files                                            #
# ------------------------------------------------------------------------ #
#  40.  Generate Config file -- MUST DO STEP FOR EACH DATABASE: dg_adrci.conf
#  41.  View drcVISK1.log on VISK
#  42.  View drcVISK_DR1.log on VISK_DR
# ------------------------------------------------------------------------ #
#  50.  Syntax to perform a graceful switchover to VISK_DR1
# ------------------------------------------------------------------------ #
#  60.  Show Fast-Start Failover
#  61.  SQL Query FSFO State                                                #
# ------------------------------------------------------------------------ #
#  x.  Exit                                                                #
# ------------------------------------------------------------------------ #
```

The Data Guard Broker submenu is broken down into several sections:

1. Setup and Configuration
2. Reporting
3. Status Review
4. Log File Review

Pertinent options from the Data Guard Broker submenu are setup of initialization parameters, setting up the appropriate LISTENER.ORA entries, creating the broker configuration and creating the Fast-Start Failover configuration. Let's start with the initialization parameters. The first option from the Data Guard Broker submenu will generate the required Alter System Syntax for DG_BROKER Setup (Initialization Parameters):

```
1
# --
# -- Execute the following on the Primary Database:  VISK
alter system set dg_broker_config_file1='+DATA/VISK/broker1.dat' scope=both sid='*';
alter system set dg_broker_config_file2='+FRA/VISK/broker2.dat' scope=both sid='*';
alter system set dg_broker_start=true scope=both sid='*';

# --
# -- Execute the following on the Standby Database:  VISK_DR
alter system set dg_broker_config_file1='+DATA2/VISK_DR/broker1.dat' scope=both sid='*';
alter system set dg_broker_config_file2='+FRA/VISK_DR/broker2.dat' scope=both sid='*';
alter system set dg_broker_start=true scope=both sid='*';
```

Second, generate LISTENER.ORA entries for DG Broker option will generate the necessary LISTENER.ORA entries. The only reason why this step is important is because the database listener has a special attribute for the Broker for the GLOBAL_DBNAME parameter:

```
2
# -- Primary Database Server
# --
LISTENER_VISK =
.
..

SID_LIST_LISTENER_VISK =
  (SID_LIST =
    (SID_DESC =
    (SDU=32767)
      (GLOBAL_DBNAME = VISK_DGMGRL)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/db)
      (SID_NAME = VISK1)
    )
  )

# -- Standby Database Server
# --
LISTENER_VISK_DR =
.
..

SID_LIST_LISTENER_VISK_DR =
  (SID_LIST =
    (SID_DESC =
    (SDU=32767)
      (GLOBAL_DBNAME = VISK_DR_DGMGRL)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/db)
      (SID_NAME = VISK_DR1)
    )
  )
```

Now, let's proceed to create the Broker configuration.

```
3
create configuration VISK_DGCONFIG as primary database is "VISK" connect identifier is VISK_PRI;
add database "VISK_DR" as connect identifier is VISK_STDBY maintained as physical;
enable configuration;
```

Once the DG Toolkit generates the script, you can execute the script within dgmgrl. The last important option to demonstrate is how the DG Toolkit can be used to create a FSFO configuration:

```
7
edit database 'VISK' set property 'LogXptMode'='SYNC';
edit database 'VISK_DR' set property 'LogXptMode'='SYNC';
edit database 'VISK' set property FastStartFailoverTarget='VISK_DR';
edit database 'VISK_DR' set property FastStartFailoverTarget='VISK';
edit configuration set protection mode as maxavailability;
enable fast_start failover;
show fast_start failover;
```

With DG Toolkit, setting up the Broker is straight forward, and DBAs who like to know what is happening behind the scene will appreciate tools like this. In summary, we've demonstrated how we can leverage the DG Toolkit to generate the complete syntax to setup FSFO based on the dg.conf configuration file.

OBSERVER

The Observer is a third-member quorum that ensures that a failover only occurs when everything meets the rules that were defined. The Observer should ideally be placed in another data center away from both the primary and standby databases. The Observer can be installed on any system and does not have to match your OS or bit level. The only requirement is that the Oracle Client is of the same release or higher as the database version. You can only have one Observer per a Data Guard FSFO configuration.

The DG Toolkit provides the shell script to launch the observer in the Data Guard Broker SubMenu. You will see an option to start the Observer process on another server. In addition to create the TNSNAMES and LISTENER.ORA entries, the shell script will also generate the script to start the Observer process on another server preferably on a different data center from where the primary and standby data:

```
# --
# -- You can use the following script to start the observer
#!/usr/bin/ksh
dgmgrl <<___EOF >/tmp/observer_`hostname`.log
connect sys/oracle123@VISK_PRI
start observer
___EOF
```

FSFO (FAST-START FAILOVER) CONSIDERATIONS

A fast-start failover happens when the Observer and the standby database both lose connectivity with the production database for a specified time interval that surpasses the value designated in the FastStartFailoverThreshold and the Obsever and the standby database are still communicating. Best practice recommended settings for FSFO thresholds differ between RAC and single instance non-RAC databases. For single instance databases with low latency, reliable network, the FSFO threshold is 10-15 seconds. For a single instance database on a high latency network over a WAN, the FSFO recommended threshold is 30-45 seconds. For RAC databases, this logic changes a bit. We need to factor in misscount + reconfiguration time and then add another 20 seconds or so to the equation. CSS misscount in Oracle Database 11g Release 2 continues to be 30 seconds. The reconfiguration time should be based on the user experience.

RECOVER IN PARALLEL

At a minimum, you should set the initialization parameter PARALLEL_MAX_SERVER to be the same as the number of CPUs available on the standby database server. You can also set the parameter RECOVERY_PARALLELISM to the number of CPUs to start as well. Optionally, you can perform RECOVER MANAGED STANDBY DATABASE with the PARALLEL clause.

DB_BLOCK_CHECKING AND DB_BLOCK_CHECKSUM

The following table lists the best practice configuration settings on the primary and the physical standby database for corruption detection, prevention, and Automatic block repair:

Primary Database	Physical Standby Database
DB_BLOCK_CHECKSUM=FULL ** Best practice is to set DB_BLOCK_CHECKSUM=FULL on both the primary and standby databases. Setting the parameter to FULL will incur 4% to 5% overhead on the system while setting the parameter to TYPICAL will incur 1% to 5% overhead on the system.	DB_BLOCK_CHECKSUM=FULL
DB_BLOCK_CHECKING=FULL ** Performance overhead is incurred on every block change; thus, you need to performance test this parameter. Block checking can cause 1% to 10% performance overhead depending on workload. With higher rates of insertions or updates, this parameter becomes more expensive for block checking. Oracle recommends setting DB_BLOCK_CHECKING to FULL on the primary database but you need to assess if the performance overhead is acceptable and adjust to your business needs.	DB_BLOCK_CHECKING=OFF
DB_LOST_WRITE_PROTECT=TYPICAL Primary database performance impact is negligible (see next section)	DB_LOST_WRITE_PROTECT=TYPICAL
	Starting from 11.2, You can use Active Data Guard to enable Automatic Block Repair

As part of best practice, you can set DB_BLOCK_CHECKSUM to FULL, so both disk corruption and in-memory corruption are detected and the block is not written to disk; thus preserving the integrity of the physical standby database. This parameter has minimal effect on Redo Apply performance.

For Oracle Database 11g, you should consider setting the DB_ULTRA_SAFE parameter on the primary database since it is the comprehensive data corruption prevention and detection. DB_ULTRA_SAFE automatically configures data protection block checking levels for DB_BLOCK_CHECKING, DB_BLOCK_CHECKSUM, and DB_LOST_WRITE_PROTECT parameters.

** For additional details, please refer to Metalink Note 1302539.1: Best Practices for Corruption Detection, Prevention, and Automatic Repair - in a Data Guard Configuration

LOST_WRITE_PROTECT

You should set LOST_WRITE_PROTECT to TYPICAL to prevent corruptions due to stray or lost writes on the primary database from being propagated and applied to the standby database. By setting LOST_WRITE_PROTECT, database server can record buffer cache block reads in the redo log so that lost writes can be detected. From the Prepare Physical Standby Database Submenu (dg_prepare_standby_menu.ksh), if you choose the option to Generate Alter System Commands for the primary and standby databases, you will notice that LOST_WRITE_PROTECT is set with best practices in mind to TYPICAL.

Setting this parameter has a negligible effect on the standby database.

ARCHIVELOG RETENTION POLICY

You should set your RMAN archivelog retention policy so that archivelogs are not deleted on the primary database until confirmation is either made that they are received or applied on the standby database. From RMAN on the primary database instances, issue one of the following commands

```
configure archivelog deletion policy to shipped to all standby;
 ------------------------   OR   ----------------------------
configure archivelog deletion policy to applied on all standby;
```

On the standby database, from RMAN, issue the following command

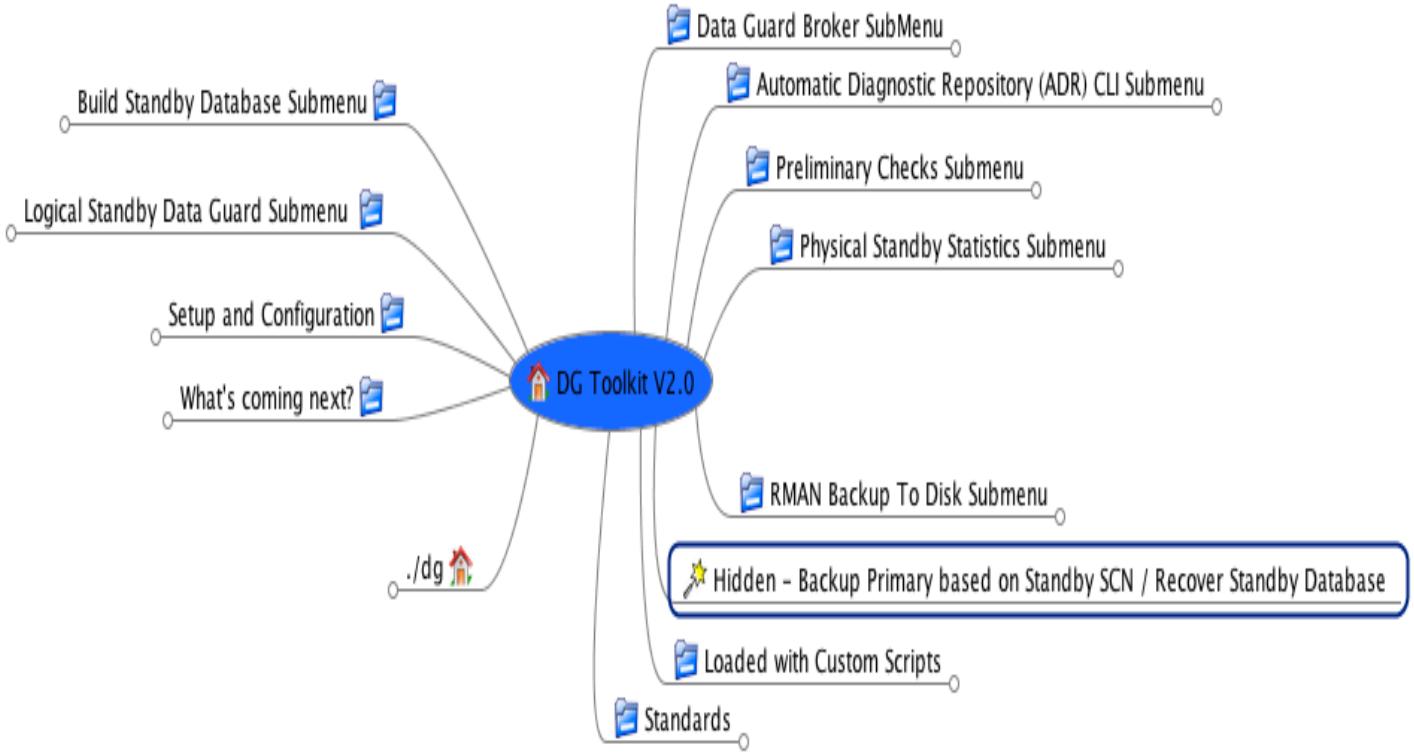
```
configure archivelog deletion policy to applied on standby;
```

We want to make sure that we do not purge any archivelogs until they are applied. The optios to generate the archivelog retention policies is available in the Data Guard RMAN Backup to Disk Submenu (dg_rman2disk_menu.ksh).

For performance implications, archival history count should be less than 10000. You should set control_file_record_keep_time parameter accordingly.

DG MENU / TOOLKIT V2.0

Here's the hierarchy of the DG Toolkit menu system.



The master driver script is the ./dg shell script. The dg script launches the comprehensive menu system. Each of the menu's option launches a submenu, and each of the submenu options executes a shell script that executes a dependent shell script. For the scope of this presentation, several of the topics are not covered such as the Logical Standby Submenu or the ADR Submenu. You are strongly encouraged to peruse through the menu options that are not covered in this presentation.

As submenu options are chosen, the DG Toolkit will either execute a shell script or execute a simple command. For simple logic, we simply leverage echo statements and send output to the screen or pipe it to SQL*Plus for the menu options. For more complex tasks, we invoke other shell scripts.

Launching the DG Tooklit is simple. You can invoke the ./dg shell script to see the following driving menu structure:

```
# ------------------------------------------------------------------------ #
#                       Data Guard Menu System                             #
#   Primary Host:      rac561          Standby Host:  rac562
#   Primary DB:        VISK            Standby DB:  VISK_DR
# ------------------------------------------------------------------------ #
#  10.  Launch Preliminary Check Submenu                                   #
# ------------------------------------------------------------------------ #
#  20.  Launch Build Standby Database Submenu                              #
# ------------------------------------------------------------------------ #
#  30.  Launch the Data Guard Broker Submenu                               #
# ------------------------------------------------------------------------ #
#  40.  Launch Monitor Physical Standby Data Guard Submenu                 #
# ------------------------------------------------------------------------ #
#  50.  Launch Monitor Logical Standby Data Guard Submenu                  #
# ------------------------------------------------------------------------ #
#  60.  Launch Automatic Diagnostic Repository (ADR) CLI Submenu           #
# ------------------------------------------------------------------------ #
# 100.  Launch RMAN Backup to Disk Submenu                                 #
# ------------------------------------------------------------------------ #
```

```
#   x.  Exit                                                                  #
# ------------------------------------------------------------------------- #
#   Enter Task Number:
```

At the heart of the DG Toolkit is a shell script called `dg_remote_connect.ksh`. The primary purpose of this script is to execute scripts on the primary and/or the standby database. As scripts are executed on the primary database, you can pass the parameter "P" as the first argument and the SQL script to execute as the second argument. Likewise, if you want to execute a script on the standby database server, you can pass the parameter "S" as the first argument and the SQL script to execute as the second argument. Here's the full content of the dg_remote_connect.ksh shell script:

```
#!/bin/ksh

function check_sys_pass
{
export PWFILE=.syspasswd
if [ ! -f $PWFILE ]; then
  echo "Please enter the SYS password on the primary database:  $PRIMARY_DB on $PRIMARY_HOST :";
  read sys_pass
else
  sys_pass=$(cat $PWFILE |head -1)
fi
}

function remote_connect
{
export TARGET=$1
export FILE=$2

export PWFILE=.syspasswd
if [ ! -f $PWFILE ]; then
  echo "Please enter the SYS password on the primary database:  $PRIMARY_DB on $PRIMARY_HOST :";
  read sys_pass
else
  sys_pass=$(cat $PWFILE |head -1)
fi
export SYS_PASSWORD=$sys_pass

#  Check to see if service name is different than DB name
#  Some DBAs may have weird service names
[ "$PRIMARY_DB_SERVICE" = "" ] && export PRIMARY_DB_SERVICE=$PRIMARY_DB
[ "$STANDBY_DB_SERVICE" = "" ] && export STANDBY_DB_SERVICE=$STANDBY_DB

if [ "$TARGET" = "P" ]; then
echo "# ------------------------------------------------------------------------- #"
echo "#         Executing $FILE on DB:  $PRIMARY_DB "
echo "# ------------------------------------------------------------------------- #"
# Uncomment for debugging purposes only
#echo "sqlplus -s sys/${SYS_PASSWORD}@\"$PRIMARY_HOST:$PRIMARY_PORT/$PRIMARY_DB\" as sysdba"

# Uncomment for debugging purposes only
#echo "(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = ${PRIMARY_HOST})(PORT = ${PRIMARY_PORT})) (CONNECT_DATA
= (SERVER = DEDICATED) (SERVICE_NAME = ${PRIMARY_DB}) (UR = A)))"

sqlplus -s sys/${SYS_PASSWORD}@"(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = ${PRIMARY_HOST})(PORT =
${PRIMARY_PORT})) (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = ${PRIMARY_DB_SERVICE}) (UR = A)))" as
sysdba <<+
@$FILE
+

elif [ "$TARGET" = "S" ]; then
echo "# ------------------------------------------------------------------------- #"
echo "#         Executing $FILE on DB:  $STANDBY_DB "
echo "# ------------------------------------------------------------------------- #"

sqlplus -s sys/${SYS_PASSWORD}@"(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = ${STANDBY_HOST})(PORT =
${STANDBY_PORT})) (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = ${STANDBY_DB_SERVICE}) (UR = A)))" as
sysdba <<+
@$FILE
```

```
+
fi
}
```

As you can see, the full TNS string is wrapped in double quotes as part of the SQL*PLUS connect string. By leveraging this technique, we do not need to update the TNSNAMES.ORA file for the primary or standby database. Majority of the scripts are executed with the following syntax within DG Toolkit:

```
remote_connect P $SQL | tee $LOG
remote_connect S $SQL | tee $LOG
```

Because the connections are made to both primary and standby databases, the connections must be made with SYSDBA privileges.

DG CONFIGURATION FILE

With the DG Toolkit, DBAs do need to spend time modifying settings in the dg.conf file. Once all the parameters are defined appropriately, you can launch the DG Toolkit and proceed to create, configure, monitor, maintain and troubleshoot your Data Guard environment. Here's a sample dg.conf file:

```
cat dg.conf.DBATOOLS
# - For RAC configuration, use the VIP for PRIMARY_HOST and STANDBY_HOST
# 1. You may have to use the fully qualified hostname.domain_name for PRIMARY_HOST and STANDBY_HOST
# 2. Typically the PRIMARY_DB_SERVICE will be same as the PRIMARY_DB
#    PRIMARY_DB_SERVICE can be blank
#
PRIMARY_HOST=rac5501-vip
PRIMARY_DB=DBATOOLS
PRIMARY_DB_SERVICE=DBATOOLS
PRIMARY_PORT=1521
STANDBY_HOST=rac5502-vip
STANDBY_DB=DBATOOLS_WA
STANDBY_DB_SERVICE=DBATOOLS_WA
STANDBY_PORT=1521
PRIMARY_DOMAIN=dbaexpert.com
PRIMARY_DB_INSTANCE=DBATOOLS1
STANDBY_DB_INSTANCE=DBATOOLS_WA1
#
# - Bystander Physical Standby Database Configuration
# -
SECOND_STANDBY_HOST=srac5501
SECOND_STANDBY_DB=DBATOOLS_DR
SECOND_STANDBY_DB_SERVICE=DBATOOLS_DR
SECOND_STANDBY_PORT=1521
SECOND_STANDBY_DB_INSTANCE=DBATOOLS_DR1
#
# - Disk Group information
# -
PRIMARY_DATA_DG=+DATA_PR
STANDBY_DATA_DG=+DATA_WA
PRIMARY_FLASH_DG=+FRA
STANDBY_FLASH_DG=+FRA
#
# - Valid entries for FS=FS for file system or ASM for automated storage management
FS=ASM
#
# - Primarily used for the DG Broker but used at multiple places
# 1.  If you are not RAC, please make PRIMARY_SERVER and STANDDBY_SERVER
#     equal to PRIMARY_HOST and STANDBY_HOST respectfully
PRIMARY_ORACLE_HOME=/u01/app/oracle/product/11.2.0/db
STANDBY_ORACLE_HOME=/u01/app/oracle/product/11.2.0/db
# --
# -  The server names are non-vip server names
# -  For non-RAC database environments, the PRIMARY_HOST and PRIMARY_SERVER will be the same
# -  For non-RAC database environments, the STANDBY_HOST and STANDBY_SERVER will be the same
# -
PRIMARY_SERVER=rac5501
```

```
STANDBY_SERVER=rac5502
#
# -
# - The following section is used for ADR including viewing of ALERT LOGs
export DG_ADRCI_CONF=dg_adrci.conf
export DG_ADRCI_COMMAND_FILE=dg_last_adrci_command.log
export DG_ADRCI_COMMAND_LOGFILE=dg_adrci_command.log
export DG_ADRCI_ALERT_LINES=40
#

# -
# -  The following section is only used by the dg_asm_cp.ksh script
# -    as part of the backupset copy from one ASM instance to another
# -  If you are cloning a database from one ASM instance to another, you do not
# -    need to modify this section
# ------------------------------------------------------------------------------- #
#  For RAC, change the LOCAL_ASM_INSTANCE=+ASM1
#  For non-RAC standalone databases, change the LOCAL_ASM_INSTANCE=+ASM (without the 1)
#
#
export LOCAL_ASM_INSTANCE=+ASM1
export REMOTE_ASM_INSTANCE=+ASM2
#
#
# Both LOCAL_DIR and REMOTE_DIR must NOT have a / at the end.
# The scripts will append a / at the very end
#
export LOCAL_DIR=+fra/dbatools/BACKUPSET/2010_09_09
# --
# --
# You MUST Create the following REMOTE_DIR on the target ASM instance
export REMOTE_DIR=+FRA/DBATOOLS/backupset/2010_09_09
export ASM_SYS_PASSWORD=oracle123
#  What is different between this gen_asm_cp.ksh with dg_asm_cp.ksh script:
#  A.  export TARGET_HOST=rac5502
#  Instead of TARGET_HOST, we will use STANDBY_HOST from dg.conf
#  B.  export REMOTE_ASM_PORT=1521
#  Instad of REMOTE_ASM_PORT, we will STANDBY_PORT from dg.conf
#
# -- For debugging purposes
#export DBI_TRACE=1
#
# -- The following section is used by the dg_cp_database.ksh script
export REMOTE_DB_COPY_SAME_DIR=TRUE
export REMOTE_DB_COPY_DIR=+DATA_WA/dbatools/datafile
```

We tried to provide comments in the appropriate portions of the configuration file. Several features of the dg.conf file is outside the scope of this presentation. For example, the DG Toolkit provides a mechanism to copy a database backup from one ASM instance (source) to another ASM instance on another RAC or non-RAC ASM instance (target) with the full asmcmd cp syntax.

STANDARDS

The DG Toolkit adheres to standards to naming conventions and coding guidelines. First, all the shell scripts start with dg_ prefix. If the shell script is a submenu, then the shell script has the format of dg_${MENU_NAME}_menu.ksh where the $MENU_NAME can be any of the following:

- ADR
- BROKER
- LOGICAL
- PHYSICAL STANDBY
- PRELIMINARY CHECK

- PREPARE STANDBY
- RMAN TO DISK

Importantly, DG Toolkit is written in 100% korn shell, anonymous PL/SQL blocks, and SQL scripts so all the source is 100% exposed. On top of each submenu, you will see the name of the submenu shell script to the right. Overall, the DG Toolkit is either a shell script invoking another shell script or a shell script executing SQL*Plus executing a .SQL script. Below you will see other pertinent information about DG Toolkit:

- All scripts that relate to physical standby start with dg_ prefix
- All scripts that relate to logical standby start with logical_ prefix
- All menu screens have the _menu.ksh suffix
- Lot of dg_ scripts can be executed standalone
- Several configuration files but the heart of the tool is dg.conf. If the Unix variable $CONF is specified prior to execution of the ./dg executable, it will override the dg.conf configuration file. For example, you can set CONF=dg.conf.ERPPROD and the DG Toolkit will look at the Data Guard environment for another database. This way we can support multiple databases in a single server by simply changing the $CONF variable

SUMMARY

With the DG Toolkit DBAs can quickly setup configure, monitor, manage and maintain one or more Data Guard environment(s). The DG Toolkit is not designed to replace Grid Control, but merely empower the DBAs who seek the command-line options. Understanding the command-line syntax will help you understand the architecture under the covers and ultimately lead you in the direction to becoming a Data Guard expert.

Just about every shell script and SQL script that this paper references is included in the DG Toolkit and most likely a part of a menu option in a submenu. You will notice that each of the submenu's have a shell script name at the top of the menu. You can add your own options if you have scripts that are specific to your organization. Because DG Toolkit is 100% shell script and SQL script driven, you can deploy a Data Guard environment without any additional software installation. As best practices are redefined or changed by Oracle Corporation, so will the DG Toolkit to comply with the latest techniques.

The scripts are provided in a single archived Unix tar format. You can download the dg.tar file and extract it using the command: `tar -xvf dg.tar`. As long as you have access to the korn or bourne shell and SQL*Plus, you can execute the DG Toolkit. Most of the scripts are designed to be executable as a standalone shell script. All the SQL scripts can be executed independently.

There's quite a bit of the DG Toolkit that's not even mentioned or covered in this paper. Also, the DG Toolkit is constantly being improved up on and new features are added as Oracle evolves. At the same time, if you want to request certain features to be added, please feel free to contact me at ckim@dbaexpert.com with your enhancement requests.

UPDATES

This document will also continue to evolve. Please check for updates from the dataguardbook.com website.

REFERENCES

Oracle Data Guard 11g Handbook, Oracle Press 2009

Oracle Active Data Guard, Oracle Data Guard 11g Release 1, Oracle MAA White Paper – November 2010

Fast-Start Failover Best Practices, Oracle Data Guard 10g Release 2, Oracle MAA White Paper – July 2010

AUTHOR BIOGRAPHY

Charles Kim is an Oracle ACE Director, an Oracle Certified Professional, and an Oracle Certified RAC Expert.

His experience spans variety of Unix platforms in both SMB and Fortune 500 companies with specialization in high volume, mission-critical, real-time OLTP databases. Charles works predominately in the Maximum Availability Architecture (MAA) space (RAC, ASM, Data Guard, and other High Availability solutions). Charles released his first book titled Oracle Database 11g New Features for DBA and Developers on November 2007. Charles also co-authored the Linux Recipes for Oracle DBAs with APress to be released in November 2008. Unleashing his expertise in the maximum availability sector, Charles is also the co-author of the Oracle Data Guard 11g Handbook with Oracle Press.

Charles is an active participant in the Oracle Database Beta programs, including participating in the invitation-only Independent Oracle User Council (IOUC) Beta migration tests at the Oracle Headquarters in Redwood Shores, CA. Charles has presented advanced topics for IOUG and Oracle OpenWorld on such topics as RAC/ASM, Linux, Data Guard, and 7x24 High Availability Considerations.

Charles is also the author of the MAA case study at Oracle's Website (http://www.oracle.com/technetwork/database/features/availability/fnf-casestudy-082608.html). Charles holds certifications in Oracle, Red Hat Linux, and Microsoft, has over 20 years of IT experience and has worked with Oracle since 1991. Charles blogs regularly at http://blog.dbaexpert.com and provides technical solutions to Oracle DBAs and developers.