Oracle Active Data Guard
Oracle Data Guard 11g

Oracle Maximum Availability Architecture White Paper
September 2011

Maximum
Availability
Architecture

Oracle Best Practices for High Availability

Oracle Active Data Guard
Oracle Data Guard 11*g*

Oracle Active Data Guard
Oracle Data Guard 11g

INTRODUCTION

Information Technology (IT) organizations have long sought a simple solution to improve the performance of mission-critical applications by offloading the overhead of ad hoc queries, reporting, and backups to a synchronized replica of the production database. The unpredictable nature of these long-running operations and the system resources they consume make it difficult to guarantee consistently high levels of service for business transactions.

Traditional solutions typically involve logically replicating data for reporting and using tertiary storage systems for backups. However, these approaches incur performance overhead, are often unable to support all data types and applications, and significantly increase management complexity and cost.

The most popular use of Oracle Data Guard—due to its simplicity and high performance—is to synchronize a physical standby database with its production counterpart for data protection and high availability. A physical standby database is typically operated in real-time apply mode, continuously validating and applying changes received from the primary database to ensure that a failover can complete within seconds of an outage at the primary site. Prior to Oracle Database 11*g*, it was necessary to stop Redo Apply to enable read access to a Data Guard standby database. This resulted in queries returning stale results, and would lengthen the time required to complete a failover because a backlog of data would have to be processed before the standby could assume the primary role.

The Active Data Guard Option available with Oracle Database 11*g* Enterprise Edition enables you to open a physical standby database for read-only access for reporting, for simple or complex queries, sorting, or Web-based access while Redo Apply continues to apply changes from the production database to the standby database. All queries reading from the physical standby database execute in real time, and return current results. With Active Data Guard, you can offload any operation that requires up-to-date, read-only access to the standby database, enhancing and protecting the performance of the production database without any compromise in Recovery Point or Recovery Time objectives.

These new capabilities make it possible for Active Data Guard to be deployed for a wide variety of business applications. Examples include:

- Telecommunications: Technician access to service schedules, customer inquiries to check status of service requests.
- Healthcare: Fast access to up-to-date medical records.
- Finance and Administration: Ad hoc queries and reports.
- Transportation: Package tracking queries, schedule status.
- Web businesses: Catalog browsing, order status, scale out using reader farms.

Given the widespread use of physical standby databases, all that is needed to unlock the value of the systems, software, storage and networks already deployed is to upgrade to Oracle Database 11*g* and enable the Active Data Guard Option.

Active Data Guard also includes the ability to enable RMAN block-change tracking on a physical standby database. Users have always been able to use RMAN to offload backups to a physical standby database, but with block-change tracking, it is possible to perform fast incremental backups that can be up to 20x faster than using a physical standby database without Active Data Guard.

This Maximum Availability Architecture (MAA) white paper augments the Data Guard documentation by profiling several use cases that are well suited to Active Data Guard and explaining the best practices for its use.

TYPICAL USE CASES

Read-Only Applications and Application Workloads

Read-only applications are those that do not cause any database changes, i.e. they do not generate any redo data in an Oracle database, and are commonly referred to as *reporting applications*. With Active Data Guard, you can move this type of application to the standby database, thereby freeing additional resources on the primary database to execute read-write transactions.

In addition to applications that are strictly read-only, there are applications for which there is a distinct read-only portion of the workload. For example, an online order application typically has a read-only component that allows users to browse a catalog of products. While browsing the catalog no redo is being generated. Only when a user begins populating the *Shopping Cart* are data manipulation language (DML) statements generated at the database level and redo data produced upon their execution.

Operations Allowable On a Read-Only Database

The following operations are allowed on a read-only database:

- Issue `SELECT` statements, including queries that require multiple sorts that leverage `TEMP` segments
- Use `ALTER SESSION` and `ALTER SYSTEM` statements

- Use `SET ROLE`
- Call stored procedures
- Use database links (dblinks) to write to remote databases
- Use stored procedures to call remote procedures via dblinks
- Use `SET TRANSACTION READ ONLY` for transaction level read consistency
- Issue complex queries (such as grouping `SET` queries and `WITH CLAUSE` queries)

Operations Disallowed On a Read-Only Database

An Active Data Guard standby database is subject to the same restrictions as any Oracle database that is open read-only. These include:

- Any DMLs (excluding simple `SELECT` statements) or DDLs
- Query accessing local sequences
- DMLs to local temporary tables

Please refer to Oracle documentation for complete information on restrictions:

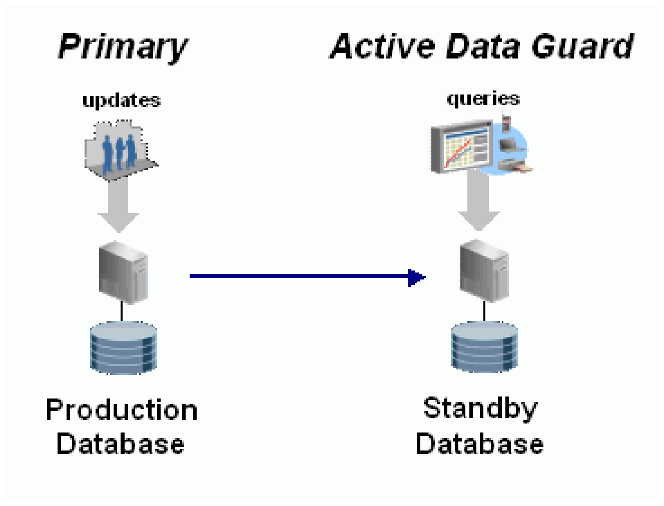
http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/start002.htm#i1006490

Typical Read-Only Configurations

Active Data Guard is typically deployed in one of the following configurations:

- A single instance physical standby
- An Oracle Real Application Clusters (Oracle RAC) primary with a single instance physical standby database
- An Oracle RAC primary and Oracle RAC physical standby database

Figure 1: Active Data Guard with a Single-Instance Primary Database



Read-only applications or read-only workloads within an application are run directly against the Active Data Guard standby. Figure 1 shows a configuration where Active Data Guard provides both read-only services and disaster recovery.

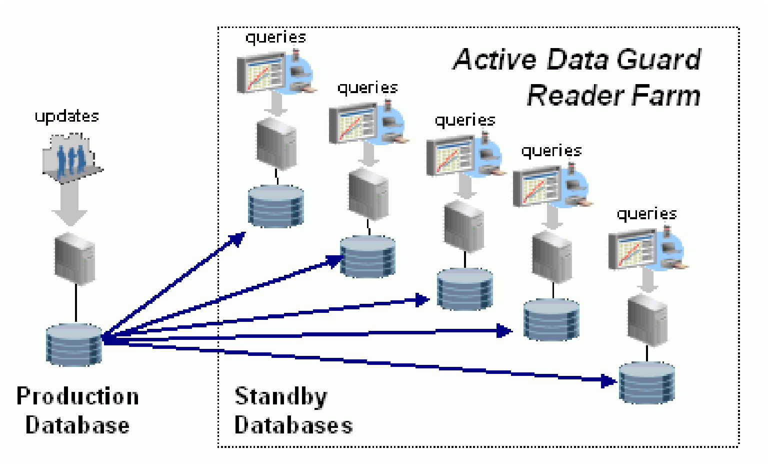
Key benefits of this simple approach include:

- Support for Oracle RAC on primary and/or standby databases
- Ability to run queries and see real-time data on the standby database
- Capacity to offload read-only workload to the standby, allowing primary database applications to scale due to more available resources
- More efficient use of disaster-recovery resources

Reader Farms: Scaling Read-Only Workload

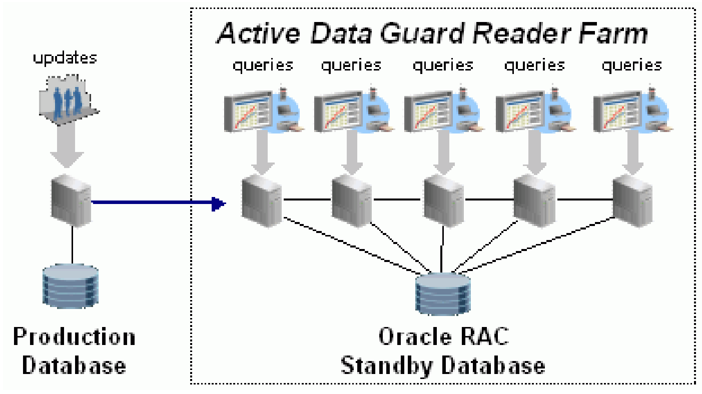
Because a Data Guard configuration can support multiple standby databases, the simple configuration shown in Figure 1 can be extended to boost the read performance of the most demanding applications beyond what the primary database alone could support. Figure 2 shows an Active Data Guard configuration with a single primary database that supports read/write transactions, and multiple standby databases that provide read-only access to users.

Figure 2: Active Data Guard with Multiple Standby Databases



If the standby database is an Oracle RAC, then scaling read I/Os is as simple as adding additional standby instances as shown in Figure 3.

Figure 3: Active Data Guard with Oracle RAC Standby Databases and a Single Primary



Accommodating Read-Mostly Applications

Read-mostly applications perform many more read operations than writes. For example, certain reporting applications perform mainly read I/Os but must issue limited `INSERT`s, `UPDATE`s, or `DELETE`s to track user logons or report history and status. Such applications may be able to use Active Data Guard by redirecting write I/Os to a separate database.

Oracle Business Intelligence Suite Enterprise Edition Plus (Oracle BI EE Plus) is an example of a read-mostly application that is certified for use with Active Data Guard. OBI EE Plus is a comprehensive suite of enterprise BI products that delivers a full range of analysis and reporting capabilities. Configuration details for using Oracle BI EE Plus with Active Data Guard are described in the following best practice paper:

http://www.oracle.com/technetwork/database/features/availability/maa-wp-11g-biee-activedataguard-1-131999.pdf

As a general use-case for other read-mostly applications, if the updates being performed by the application must be persistent and available to all client applications accessing the database, then the updates should be redirected back to the primary database via a database link. This will make the updates available to users on both the primary and the standby database.

If the updates being performed by the application are not critical and only temporary in nature, then it is preferable to redirect the redo or updates to a *scratch* database that is located on the same host as the standby. This scratch database contains only the tablespaces and data files necessary to support the updates coming from the Active Data Guard standby database. The benefit of this approach is there is no performance impact to the primary database. In addition, the network overhead incurred by the scratch database is negligible.

Information transported to other databases via a database link can be encrypted to provide safety and security. Please refer to Appendix B for more information on using database links to accommodate read-mostly applications using an Active Data Guard standby database.

SETUP AND INSTALLATION

Active Data Guard requires a separate license, and extends basic Data Guard functionality included with Oracle Database 11*g* Enterprise Edition. If you do not have an existing Data Guard physical standby configuration, use the following links for assistance in creating one. To create a configuration with:

- Single-instance primary and a single-instance standby databases:

  http://download.oracle.com/docs/cd/B28359_01/server.111/b28294/create_ps.htm - i63561

- Oracle RAC primary database and a single-instance standby database:

  http://www.oracle.com/technetwork/database/features/availability/maa-wp-10g-racprimarysingleinstance-131970.pdf

- Oracle RAC primary database and an Oracle RAC standby database:

  http://www.oracle.com/technetwork/database/features/availability/maa-wp-10g-racprimaryracphysicalsta-131940.pdf

Note: In addition to the above methods, you can use Oracle Enterprise Manager to create a single-instance standby database from a single-instance or Oracle RAC primary database.

If you have an existing Data Guard physical standby configuration, you must meet the following prerequisites before enabling Active Data Guard:

Prerequisites:

1. Obtain a license for the Active Data Guard option
2. Set the `COMPATIBLE` parameter to at least 11.0.0 in the initialization parameter file or the SPFILE on both the primary and standby databases. In addition, the standby must be recovering redo generated after the `COMPATIBLE` parameter was set to at least 11.0.0.

BEST PRACTICES FOR ACTIVE DATA GUARD

How to Check if Active Data Guard is Already Enabled

Use the following query to confirm that Data Guard is in active mode:

```
SQL> SELECT 'Using Active Data Guard' ADG FROM V$MANAGED_STANDBY M,
V$DATABASE D WHERE M.PROCESS LIKE 'MRP%' AND D.OPEN_MODE='READ ONLY';

ADG
-----------------------
Using Active Data Guard

SQL>
```

If the query does not return the above result, and instead returns: `no rows selected`, then Active Data Guard is not enabled, follow the procedures below to enable Active Data Guard.

A future release of Enterprise Manager Grid Control will make it simple to enable Active Data Guard and report its status directly from the Data Guard Home Page.

General Best Practices

This section summarizes general best practices for deploying Active Data Guard. Each topic is discussed in greater detail either in this white paper or in the documentation referenced.

- Configure and tune Data Guard transport services:

- Use the `SYNC` redo transport mode for a high degree of synchronization between the primary and standby databases.
- Use the `ASYNC` redo transport mode for minimal impact on the primary database, but with a lower degree of synchronization.
- Optimize network throughput. See best practices for configuring Data Guard redo transport services at:

  http://download.oracle.com/docs/cd/B28359_01/server.111/b28282/configbp006.htm#CHDGIAJA

- Use Real-Time Apply on the standby database so that changes are applied as soon as the redo data is received. The Data Guard broker will automatically enable Real-Time Apply when the configuration is created. If you are using the SQL*Plus command line to create your configuration, you will need to manually enable Real-Time Apply.
- Shut down Redo Apply and all standby instances cleanly so that upon restart, you can open the standby database directly in read-only mode.
- Enable Flashback Database on the standby database to minimize downtime for logical corruptions.
- Configure clients for efficient failover (see the "Application Connection Management" section later in this white paper for more details):
  - Both the primary database and the reporting applications should connect using an Oracle Net alias that contains all hosts (both primary and standby) in the `ADDRESS_LIST`.
  - Connect to the primary database using a role specific service name.
  - Manage starting and stopping of services based on database role.
- Monitor standby performance by using Standby Statspack. See My Oracle Support Note 454848.1 for complete details on Standby Statspack.
- Use the Query SCN to monitor how far behind the standby data is from that of the primary. (See the "Query SCN" section for more details):
- Create a Data Guard broker configuration to simplify management and to enable automatic apply instance failover on an Oracle RAC standby.

Enabling Active Data Guard

The general process of enabling Active Data Guard is as simple as opening the standby database in read-only mode and starting Redo Apply.

This section describes how to enable Active Data Guard on a standby database.

1. If the standby instance and redo apply have been cleanly shut down
   a. Using SQL*Plus

i. Start the standby instance in read-only mode.

```
SQL> startup
```

ii. Once the database is open, start Redo Apply:

```
SQL> recover managed standby database disconnect
using current logfile;
```

b. If you are using the Data Guard broker to manage your Data Guard configuration

i. Start the standby instance in read-only mode. Connect to the standby instance using DGMGRL and issue the following command:

```
DGMGRL> startup
```

ii. The default state for the standby database is `APPLY ON` and Redo Apply starts automatically. If the default has been changed, then start Redo Apply by issuing the following command:

```
DGMGRL> EDIT DATABASE ‘RTQ' SET STATE=‘APPLY-ON’
```

2. If the standby database is mounted and redo apply is running using the following steps

a. Using SQL*Plus

i. Stop redo apply

```
SQL> recover managed standby database cancel;
```

ii. Open the database read only

```
SQL> alter database open read only;
```

iii. Once the database is open, start redo apply:

```
SQL> recover managed standby database disconnect
using current logfile;
```

b. If you are using the Data Guard broker to manage your Data Guard configuration

i. Stop redo apply using DGMGRL

```
DGMGRL> EDIT DATABASE ‘RTQ' SET STATE=‘APPLY-OFF’
```

ii. In SQL*Plus open the database read only

```
SQL> alter database open read only;
```

iii. Restart redo apply by issuing the following command:

```
DGMGRL> EDIT DATABASE ‘RTQ' SET STATE=‘APPLY-ON’
```

Note: When opening a standby database read only the value for the AUDIT_TRAIL parameter is evaluated to see if it is appropriate for the database open mode. As a read only database cannot support updates, an AUDIT_TRAIL value of DB will be implicitly converted to OS prior to the open read only completing.

If you are unable to open the standby in read-only mode as described above, see Appendix A for additional instructions that address rare occasions when the standby database may require additional recovery.

Evaluating if Applications are Suitable for Read-Only Database

An event can be used to validate that a read-only application can be run on an Active Data Guard standby database. The event will identify any SQL that requires read-write access to the database. See My Oracle Support Note 1206774.1 for more information on configuring and using this event. If it is discovered that a reporting application is not a true read-only application and requires some writes, (a read-mostly application), see Appendix B for additional configuration options using dblinks.

Application Connection Management

This section contains the following topics:

- Managing Initial Client Connections
- Managing Existing Connections for Unplanned Outages

Managing Initial Client Connections

In steady state, it is important that reporting applications that connect to the standby database, and primary applications that connect to the primary database, connect to the database with the correct database role. In addition, following a role transition, the applications should be able to automatically reconnect to the correct database. This can be accomplished by having separate service names for the primary application and reporting application and starting the corresponding service based on the database role.

In the following example, the primary application is connecting with a service name of `sales_rw` and the reporting application is connecting with a service name of `sales_ro`. To automate starting and stopping the service, use an “on startup” trigger that checks the database role and starts the appropriate service.

```
CREATE OR REPLACE TRIGGER manage_service
after startup on database
  DECLARE
   role VARCHAR(30);
  BEGIN
   SELECT DATABASE_ROLE INTO role FROM V$DATABASE;
  IF role = 'PRIMARY' THEN
   DBMS_SERVICE.START_SERVICE('sales_rw');
  ELSE
   DBMS_SERVICE.START_SERVICE('sales_ro');
  END IF;
END;
```

Both the primary and reporting applications should connect using an Oracle Net alias that contains all hosts, both primary and standby, in the `ADDRESS_LIST`.

For example:

```
Sales_RW =
  (ADDRESS_LIST=
    (ADDRESS=(PROTOCOL=TCP)
    (HOST=hasun01)
    (PORT=1521))
    (ADDRESS=(PROTOCOL=TCP)
    (HOST=hasun02)
    (PORT=1521))
        (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = sales_rw)))

Sales_RO =
  (ADDRESS_LIST=
    (ADDRESS=(PROTOCOL=TCP)
    (HOST=hasun01)
    (PORT=1521))
    (ADDRESS=(PROTOCOL=TCP)
    (HOST=hasun02)
    (PORT=1521))
        (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = sales_ro)))
```

When the primary application is started, the initial connection attempt will cycle through the `ADDRESS_LIST` until it finds the `sales_rw` service. The reporting application will do the same until it finds the `sales_ro` service.

Note: When you create and start a new database service using srvctl, CRS will create that service in the database (service$) if it doesn't already exist. If the service creation is done on a standby RAC cluster then CRS's attempt at creating the service will fail as the standby database is opened read only (or mounted even). A simple workaround is to create and start the service on the primary cluster so that the create service redo is generated and sent to the standby. Once the redo has been recovered you will be able to start the service on the standby database. If you do not want the service on the primary site then you can remove it from CRS.

Managing Existing Connections for Unplanned Outages

During steady state, the primary application will have connections to the primary database and the reporting application will have connections to the standby. When a failover occurs, users connected to the standby will be disconnected as part of the failover process. Once the standby is transitioned to be the new primary and is opened to run in the primary database role, the primary database service will be started using the `MANAGE_SERVICE` trigger discussed earlier (in the "Managing Initial Client Connections" section). For detailed information about implementing automatic failover of the primary application consult "*Client Failover Best Practices for Highly Available Oracle Databases: Oracle Database 10g Release 2*" technical white paper on the MAA Web site at

http://www.oracle.com/technetwork/database/features/availability/maa-wp-10gr2-clientfailoverbestprac-129636.pdf

Failover of the reporting application connections and services running on the standby database are performed manually. Following a failover, you must first determine to which database the reporting application connections should be directed:

- If the new primary database has enough capacity to support both the primary application connections and the reporting application connections, then:
    1. Start the reporting database service on the primary database.
    2. Once the service is available, restart the reporting application to get connections established.
- If the primary database does not have enough capacity to support both the primary application and the reporting application, then start the reporting database service on other Active Data Guard standby databases. If you do not have multiple Active Data Guard standbys in your Data Guard configuration, then reinstate the old primary database as a new standby database. Once the new standby is operational, start the database service and bring up the reporting application.

Recovering from RAC Apply Instance Node Failure

In an Active Data Guard RAC standby, if the redo apply instance crashes, all other instances of that standby that were open read-only will be closed and returned to the MOUNT state. This will disconnect all readers of the Active Data Guard standby. This is done to prevent any possibility of queries seeing inconsistent data.

Beginning with Oracle Database 11.2.0.2, the Data Guard Broker will automatically restore the Active Data Guard state that existed prior to the crash of the apply instance. After the surviving instances have been closed, the Broker automatically returns all previously opened standby instances to their Active Data Guard state and restarts redo apply on one of the surviving instances. No special Broker configuration or restart of any of the surviving Oracle instances is necessary. The Broker chooses the standby instance designated by the Broker property `PreferredApplyInstance`, if available, to be the new apply instance.

If you are not using the Data Guard Broker, or if you are using the Broker in a release prior to Oracle Database 11.2.0.2, then a manual restore of the Active Data Guard state is required. The steps are:

1. Each of the remaining instances that were running real-time query as part of the Active Data Guard standby must be reopened with:

    ```
    ALTER DATABASE OPEN;
    ```

2. Following this, Redo Apply is restarted on one of the instances using the standard command:

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
USING CURRENT LOGFILE DISCONNECT;
```

Monitoring Performance

As changes occur on the primary database, redo is generated and sent to the standby database. The frequency of shipping redo to the standby is determined by whether the remote destination is utilizing a synchronous or asynchronous transport. Once the redo arrives at the standby, it is applied immediately if Redo Apply was started using real-time apply. With the standby database opened read-only, the changes are immediately available for users to query. The SCN at which users can query data is published on the standby database via the Query SCN.

To monitor how far behind the data lags on the standby database relative to the primary database, you should focus on the following main areas:

- Determining the transport lag
- Determining the apply lag
- Determining the query SCN or time
- Determining how far behind the standby data lags behind what has been applied on the primary database

Redo Transport

Data Guard can transmit redo either synchronously or asynchronously.

Synchronous transport: When using synchronous transport mode, transactions that commit on the primary database are not acknowledged to the application as committed until all redo generated by that transaction has been received by the standby. While this provides the highest degree of data synchronization between primary and standby databases in Active Data Guard environment, this must be weighed against the potential impact of network latency on primary database throughput. Consult best practices for configuring Data Guard redo transport services at:

http://download.oracle.com/docs/cd/B28359_01/server.111/b28282/configbp006.htm#CHDGIAJA

Asynchronous transport: When sending data asynchronously, redo is sent asynchronous with respect to a transaction commit., the primary database will not wait for standby acknowledgment before it acknowledges the commit to the application. While the standby database will not be as up to date as when using synchronous transport, asynchronous transport can offer a very high level of data protection without impacting the primary database performance.

To determine how current the redo on the standby is in respect to transport, query the `TRANSPORT LAG` metric from the `V$DATAGUARD_STATS` view:

SQL> SELECT * FROM V$DATAGUARD_STATS WHERE NAME='transport lag';

```
NAME               VALUE            TIME_COMPUTED
------------------ ---------------- ------------------------
transport lag      +00 00:00:02     15-MAY-2005 10:32:49
```

In the example, the indicates that there is 2 lag time, meaning that the latest redo received on the standby is 2 seconds behind the latest redo on the primary.

Apply Services

To maintain close synchronization with the primary database, it is recommended that you enable real-time apply on the standby database.

To determine if Redo Apply has recovered all redo that has been received from the primary, query the `V$DATAGUARD_STATS` view. For example:

```
SQL> SELECT * FROM V$DATAGUARD_STATS WHERE NAME='apply lag';


NAME               VALUE            TIME_COMPUTED
------------------ ---------------- ------------------------
apply lag          +00 0:00:04      15-MAY-2005 10:32:49
```

The query indicates that Redo Apply has not applied the last 4 seconds of redo received on the standby database. If the apply lag indicates that the standby is behind the primary by a significant amount this could indicate that a gap exist between the standby and the primary. To detect if a gap does exist, run the following query on both the primary and standby database and compare the results:

```
select 'Instance'||thread#||': Last Applied='||max(sequence#)||'
(resetlogs_change#='||resetlogs_change#||')'
from v$archived_log
where applied = (select decode(database_role, 'PRIMARY', 'NO',
'YES') from v$database)
and thread# in (select thread# from gv$instance)
and resetlogs_change# = (select resetlogs_change# from v$database)
group by thread#, resetlogs_change#
order by thread#;
```

To monitor and assess Redo Apply performance, query the `V$RECOVERY_PROGRESS` view. This view contains the following columns:

Column	Description
Average Apply Rate	Redo Applied / Elapsed Time includes time spent actively applying redo and time spent waiting for redo to arrive
Active Apply Rate	Redo Applied / Active Time is a moving average over the last 3 mins. Excludes time spent waiting for redo to arrive.
Apply Time/ Log	Active apply time averaged per redo log file.
Checkpoint Time/ Log	Log boundary checkpoint time averaged per redo log file
Last Applied Redo	SCN and timestamp of the last redo applied. This is the time as stored in the redo stream and used to compare where the standby database is relative to the primary.

The most useful statistic is the Active Apply rate because the Average Apply Rate includes idle time spent waiting for redo to arrive, so Average Apply Rate is less indicative of the apply performance.

In a Data Guard physical standby environment, it is important to determine if the standby database can recover redo as fast as, or faster than, the primary database can generate redo. The simplest way to determine application throughput in terms of redo volume is to collect Automatic Workload Repository (AWR) reports on the primary database during normal and peak workloads, and determine the number of bytes per second of redo data the production database is producing. You can then compare the speed at which redo is being generated with the Active Apply Rate columns in the `V$RECOVERY_PROGRESS` view to determine if the standby database is able to maintain the pace. For more information see Appendix C - Tuning Redo Apply, within this paper.

Query SCN

Active Data Guard 11g enables querying data as it is being applied to a standby database while guaranteeing a transactionally consistent view of the data. Oracle provides a read-consistent view of the data through the `QUERY SCN`. The `QUERY SCN` on the standby database is advanced by the recovery process after all dependent changes have been fully applied. As it is advanced, the new `QUERY SCN` is propagated to all instances in an Oracle RAC standby. Once published to all standby instances, the `QUERY SCN` is exposed to the user via the `CURRENT SCN` column of the `V$DATABASE` view on the standby database.

The `QUERY SCN` on the standby is equivalent to the `CURRENT SCN` on the primary database. This allows applications connected to the standby instances to use the `QUERY SCN` as a snapshot of where the data is in relation to the primary database. Queries on the primary and standby databases return identical results as of a particular `CURRENT SCN`.

You can determine how far behind the query results on the standby database are lagging the primary database by comparing the CURRENT SCN from the primary to the CURRENT SCN from the standby. For example, with a dblink on the primary that points to the standby database called rtq_stby, issue the following query on the primary database:

```
SQL> select scn_to_timestamp((select current_scn from v$database))-
scn_to_timestamp((select current_scn from v$database@rtq_stby)) from dual;
```

The value returned from the query indicates the number of seconds that data on the standby lags behind the current position of the primary database. To determine an approximate query lag on the standby without connecting to or using the primary database, use the `APPLY LAG` metric from the `V$DATAGUARD_STATS` view. Note that the `APPLY LAG` metric is valid only for the time that it was computed and not at the moment the query against the `V$DATAGUAD_STATS` view was issued. Use the `COMPUTED_TIME` column to determine the last computed time for the apply lag.

Avoiding ORA-01555 errors

Queries on the standby database rely on undo generated on the primary to rollback uncommitted changes. The undo_retention period set at the primary will determine how soon undo can be overwritten and thus will also determine the length of the timing window that a standby query may avoid running into an ORA-01555 error. The occurrence of ORA-01555 is rare when the active standby is operating in real-time apply mode, because the standby recovery will keep pace with the primary.

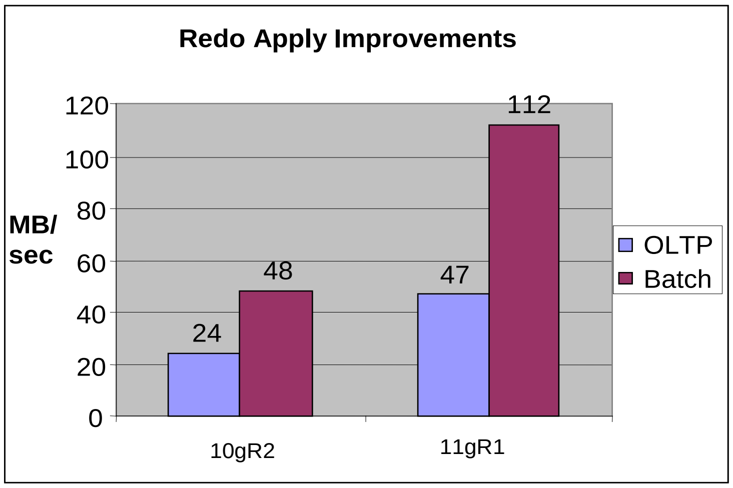
An ORA-01555 is more likely to be encountered if real-time apply is not enabled, or in situations where a network disconnect or standby outage has resulted an archive log gap. In these cases the elapsed time to complete a redo log file at the primary database can be longer than the time spent by recovery to apply it to the standby database. Lets use as an example the case where it takes the primary 10 minutes to generate 1000MB of redo and takes a standby only 30 seconds to apply (33MB/sec apply rate). If the undo_retention on the primary is set to 10 minutes, the standby has effectively reduced the retention period to 30 seconds for the standby query workload. If a standby query runs for more than 30 seconds, it is likely to run into the ORA-01555 error because recovery has already applied more redo that overwrites the rollback segment. If active standby queries experience ORA-01555 errors, first make sure the standby is in real-time apply mode, and if so, also increase the undo_retention period on the primary database.

PERFORMANCE AND SCALABILITY

Redo Apply Performance

Redo Apply performance is greatly improved in Oracle Database 11g due to several internal optimizations. Media recovery on symmetric multiprocessing (SMP) systems is even faster than previous releases. Figure 4 summarizes the results of internal tests comparing performance to previous releases.

Figure 4: Effect of Redo Apply Improvements in Release 11g



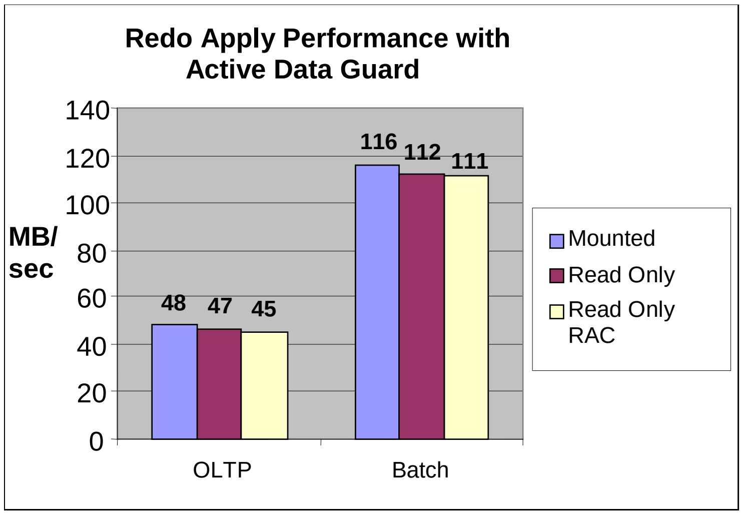
Performance improvements include:

- More parallelism
- More efficient asynchronous redo read, parse, and apply
- Fewer synchronization points in the parallel apply algorithm
- The media recovery checkpoint at a redo log boundary no longer blocks the apply of the next log

These optimizations are available by default so no new configuration is necessary. The new parallel recovery wait events can be used for tuning if the default apply rate is not satisfactory.

Figure 5 depicts the results of additional testing confirming that there was no material impact on redo apply performance when performing recovery on an open

Figure 5: Redo Apply Performance with Active Data Guard



read-only standby database, as is the case with Active Data Guard. Testing included examining the effects of having multiple standby instances open read-only while Redo Apply was being performed by a single standby instance.

Role Transition Performance

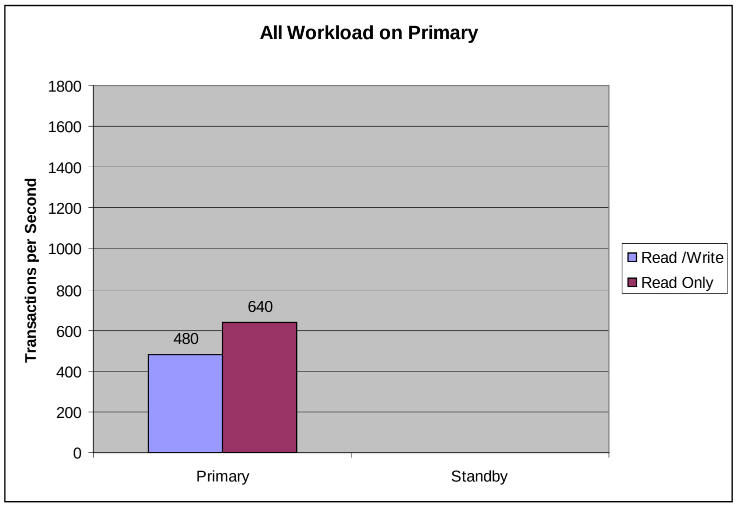
The time required to complete role transitions using an active standby database is similar to that required for a physical standby database that is not open read-only. Please see My Oracle Support Note 843803.1 for additional information.

Application Scalability

Application scalability was tested using Swingbench, an Oracle load-generation tool. Within Swingbench, the MAA tests used the Order Entry application that consists of five transaction types that are typical in any order-entry application: Creating a Customer, Browsing Products, Placing Orders, Processing Orders and Reviewing Orders. The I/O size for this workload is small (generally, it is the size of the tablespace blocksize, which in this case is 4K). Swingbench connects to the database using thin JDBC and uses all MAA client configuration best practices. The read/write workload consists of creating new customers, placing orders, and processing orders. The read-only workload consists of catalog and order browsing.

In the initial testing state, the primary database has both read/write and read-only workloads running that are connected through separate services as described under 'Managing Initial Client Connections' section. The read/write workload has 50 users connected, while the read-only workload has 100 users connected. With both workloads running on the primary database, the production host is operating at 100% capacity. Figure 6 shows the transactions per second on the primary database.

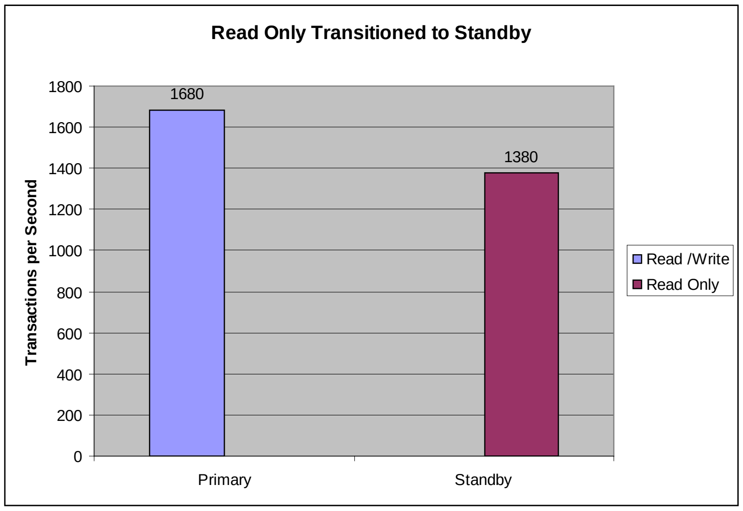
Figure 6: Graph Showing Read-Only and Read/Write Primary Database Workload



The second stage of the testing relocates the read-only workload to the Active Data Guard standby database. All 100 users performing the read-only workload are transitioned to the read-only standby by relocating the service.

Once relocated, the workload on the production host drops from running at 100% capacity to just 60% of capacity.

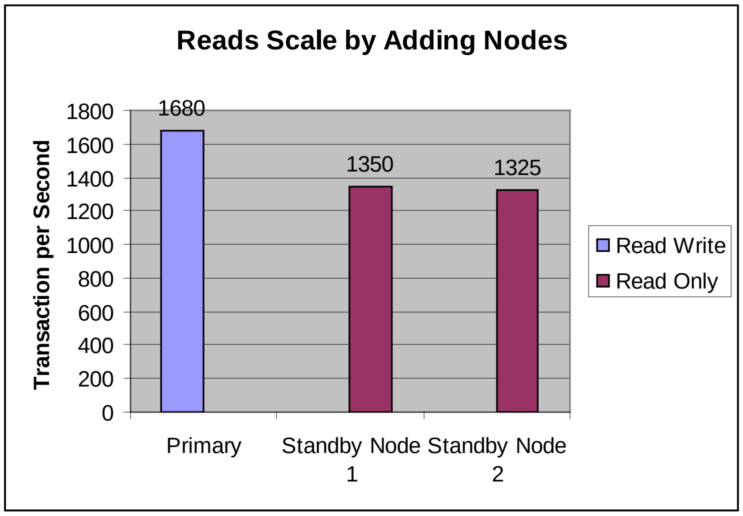
Figure 7: Graph Showing Read-Only Workload on the Active Data Guard Standby



Due to the increased resources available at the production host, the read/write users connected to the primary database can increase their transactions per second by over 250%. Similarly, the read-only users are no longer in contention for resources with the read/write users and can increase their transactions per second by 110% on the standby database (Figure 7).

The third phase of testing involved starting a second standby RAC instance and connecting an additional 100 read-only users. This test was implemented to prove that read-only workload on the standby could be scaled by adding standby instances without any impact on primary database performance (Figure 8).

Figure 8: Graph Showing Scalability When Standby Nodes Are Added



CONCLUSION

Active Data Guard provides a simple solution to improve the performance of mission-critical applications by offloading the overhead of ad hoc queries, reporting, and backups to a synchronized replica of the production database. The simplicity and performance of Active Data Guard makes it possible to be deployed for a wide variety of business applications. Active Data Guard provides a high return on investment while simultaneously protecting against data loss and downtime.

APPENDIX A: ENABLING ACTIVE DATA GUARD ON A STANDBY DATABASE THAT REQUIRES ADDITIONAL RECOVERY

The process of enabling Active Data Guard is as simple as opening the standby read-only and starting Redo Apply. This is true as long as the standby database is in a consistent state. However, if the standby is in an inconsistent state, you must make the standby consistent before opening the standby read-only. The reasons why a standby database may be in an inconsistent state include:

- This is a newly created standby that has not yet started Redo Apply
- The standby instance and Redo Apply aborted or stopped due to an error state
- The standby instance has newly restored data files
- The standby instance has a new standby controlfile

When a standby is inconsistent, you must perform a certain amount of media recovery to bring the standby database and the standby control file to a consistent state. You can enable Active Data Guard on an inconsistent standby database using SQL*Plus statements or the Data Guard broker.

To use SQL*Plus statements to enable Active Data Guard on an inconsistent standby database, perform the following steps:

1. Bring the standby database to the mount state:
   ```
   SQL> startup mount
   ```
2. Start Redo Apply:
   ```
   SQL> recover managed standby database disconnect;
   ```
3. Cancel Redo Apply after recovery has progressed:
   ```
   SQL> recover managed standby database cancel;
   ```
4. Open the standby database read-only:
   ```
   SQL> alter database open read-only;
   ```
5. Start Redo Apply:
   ```
   SQL> recover managed standby database disconnect using current
   logfile;
   ```

To use the Data Guard broker to enable Active Data Guard, use the following procedure:

1. Bring the standby database to the mount state:
   ```
   SQL> startup mount
   ```
2. With the Data Guard Broker, by default the database state for a standby is `APPLY ON` and Redo Apply starts automatically once the database is

mounted. If the default state has been modified to `APPLY OFF` then start Redo Apply. For example

- Using the Data Guard broker command-line interface:

  ```
  DGMGRL> EDIT DATABASE ‘RTQ' SET STATE=‘APPLY-ON’
  ```

  Using Oracle Enterprise Manager, then click edit to go to the Edit Properties page and select LOG APPLY ON.

3. Cancel Redo Apply after recovery has progressed. For example,

   - Using the Data Guard broker command-line interface:

     ```
     DGMGRL> EDIT DATABASE ‘RTQ' SET STATE=‘APPLY-OFF’
     ```

   - Using Enterprise Manager, click `EDIT` to go to the Edit Properties page and select `LOG APPLY OFF`.

4. Open the standby database read-only:

   - Using the Data Guard broker command-line interface:

     ```
     SQL> alter database open read only;
     ```

   - Using Enterprise Manager then use the `Advanced Options` tab of the startup dialog to select the `READ ONLY MODE`.

5. Start Redo Apply.

   - Using the Data Guard command-line interface:

     ```
     DGMGRL> EDIT DATABASE ‘RTQ' SET STATE=‘APPLY-ON’
     ```

   - Using Enterprise Manager click `EDIT` to go to the Edit Properties page and select `LOG APPLY ON`.

APPENDIX B: USING DATABASE LINKS FOR READ MOSTLY APPLICATIONS

The first step in redirecting updates from an Active Data Guard standby to a remote database is to create a database link (dblink). This section describes best practices for creating dblinks and then provides an example of the command you use to actually create a dblink.

Best Practices for Creating dblinks:

Use the following best practices to create dblinks in an Active Data Guard environment:

- Ensure all dblinks used by the configuration use Oracle Net service names that are configured for connect time failover.

  Specifically, you should configure the Oracle Net service name that the standby database uses to propagate changes to the primary database for connect time failover as such:

```
SALES =
  (DESCRIPTION =
   (ADDRESS_LIST=
    (ADDRESS = (PROTOCOL = TCP)(HOST = host1) (PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = host2) (PORT = 1521)))
   (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = SALES_RW) )
)
```

  With the Oracle Net service name (`SALES` in the example), a connection attempt is made using the first host (`host1`) in the address list. If the connection does not complete then a second attempt is made using the second host (`host2`) in the address list. While this simple modification solves the majority of the issues, you must also address several other issues as described in the following list:

- In the client side `SQLNET.ORA` file, set the `SQLNET.OUTBOUND_CONNECT_TIMEOUT` parameter.

  This parameter enables clients to quickly traverse an `ADDRESS_LIST` in the event of a failure. For example, if a client attempts to connect to a host that is unavailable, the connection attempt will be bounded to the time specified by the `SQLNET.OUTBOUND_CONNECT_TIMEOUT` parameter, after which the client attempts to connect to the next host in the `address_list`. This behavior continues for each host in the `address_list` until a connection is made. Setting the parameter to a value of 3 seconds will suffice in most environments.

- To mitigate TCP timeouts, consider setting `RECV_TIMEOUT` and `SEND_TIMEOUT` in the `SQLNET.ORA` file.

  - Set the `SQLNET.RECV_TIMEOUT` parameter to specify the number of seconds within which a connection from the primary database must wait for data from the standby database after a connection is established.
  - Set the `SEND_TIMEOUT` parameter to specify the number of seconds, within which a standby database must complete a `SEND` operation to the primary database after a connection is established.

- If you redirect updates to a scratch database that is on the same host as the standby database, consider using the IPC protocol for the Oracle Net aliases referenced by the dblink. The IPC protocol incurs less overhead than the TCP protocol and can result in better performance.
- To take advantage of database link encryption you must first install new functionality available in patch number 2647883. Please contact Oracle support to obtain the patch. In addition, you must utilize "connected user"database link. A "connected user" database link is one that connects as a specific user. For example:

  ```
  SQL> CREATE DATABASE LINK sales_prmy AUTHENTICATED BY scott
  IDENTIFIED BY tiger USING 'sales_rw';
  ```

  Encryption can be enabled for a "connected user" database link by configuring the sqlnet.ora on both the local and remote database for Kerberos Authentication. Please see Chapter 7 of the Oracle Database Advanced Security Administrator's Guide for more information.

Creating a dblink

To create a dblink in an Active Data Guard environment, issue the following SQL*Plus statement on the primary database:

```
SQL> CREATE DATABASE LINK sales_prmy AUTHENTICATD BY scott
IDENTIFIED BY tiger USING 'sales_rw';
```

In the example, `sales_prmy` is the dblink name and `sales_rw` is the Oracle Net alias that the database link will use.

Database Links and Session State

When you use a dblink to connect to a remote database, the local and remote sessions are two distinct sessions. The `NLS_%` parameters of the local session are automatically propagated to remote sessions, but almost all the rest of the local session state is not propagated. If you need to pass additional local session state to the remote session, you can implement a stored program on the remote database and pass relevant portions of local session state as parameters in a remote procedure call (RPC). This is important for applications that use the USERENV function to collect information about local sessions for connection auditing.

Redirecting Data Manipulation Language (DML)

Once the Data Definition Language (DDL) that created the dblink is applied on the standby database, you can use it to redirect writes to a remote database. For example:

```
SQL> insert into emp@sales_prmy values (999,'SMITH','SUPER
GEEK',999,sysdate,1,0);
1 row created.

SQL> commit;
```

The dblink, directs these changes to the primary database, where they are committed, and this creates redo data that Data Guard ships and applies to the standby database. Once applied, the changes are available for query by read-only users on the standby database:

```
SQL> select * from emp where empno=999;

EMPNO      ENAME      JOB        MGR        HIREDATE  SAL    COMM
---------- ---------- ---------  ---------- --------- ------ ----
999        SMITH      SUPER_GEEK 999        23-OCT-07 1      0
```

You must modify the read-mostly applications executing against the standby to include the dblink in all SQL statements that perform an UPDATE, INSERT, or DELETE. If the application cannot be modified, then consider using synonyms to make the database link names from the application. For example, the following could be performed for all objects that require support for updates:

On the primary database:

```
SQL> rename emp to emp_hidden;
Table renamed.
SQL> create synonym emp for emp_hidden@sales_prmy;
Synonym created.
```

After the redo from these changes is shipped and applied on the standby database, you can use the following DML on the standby database:

```
SQL> insert into emp values (999,'SMITH','GEEK',999,sysdate,1,0,30);
1 row created.
```

Note: You should perform the above procedure only on objects that will not be accessed routinely on the primary database. This is because a local update to the synonym will result in a network roundtrip via the database link.

When using a database link, a check is performed to determine if the database link name is the same as the GLOBAL_DBNAME of the local database:

- If the database link name and GLOBAL_DBNAME are the same then the database link is ignored and the user is able to access the object directly.

- If the database link name is different than the `GLOBAL_DBNAME` then the object is accessed through the database link.

In an Active Data Guard environment, the `GLOBAL_DBNAME` of the primary and standby are always identical because the standby is a physical copy of the primary.

- If we create a database link to be used by the standby to redirect writes back to the primary and that database link name is the same as the `GLOBAL_DBNAME` then the primary ignores the link and accesses the object directly. However, the standby also ignores the database link and attempts to modify the object directly resulting in an error as the database is opened read-only.
- If we create the link with a name different than the `GLOBAL_DBNAME` then the link will be used by both the primary and the standby. This could cause performance issues for the primary application on the primary database as it is unnecessarily performing network calls for a local update.

Redirecting Data Definition Language (DDL)

Data Definition Language (DDL) statements cannot be executed directly over a dblink. If it is necessary to redirect a DDL statement from the standby to a remote database, then you can call a remote procedure that uses dynamic SQL to perform remote DDL. However, you cannot invoke a remote procedure (even a read-only remote procedure) from a read-only database but you can put the remote procedure call in a stored procedure. For example:

On the primary database:

```
CREATE OR REPLACE PROCEDURE do_ddl(STRING IN varchar2) AS
 BEGIN
 execute immediate string;
END;

CREATE OR REPLACE PROCEDURE call_do_ddl(STRING IN varchar2) AS
 begin
  do_ddl@sales_prmy(string);
end;
```

On the standby database:

```
SQL> exec call_do_ddl('create table foo2 (col1 number)');
PL/SQL procedure successfully completed.
SQL> select * from foo2@sales_prmy;
no rows selected
```

This confirms that the table has been created – otherwise the same query would have returned an `ORA-942 "table or view does not exist"`

Sequences

If the application connected to the Active Data Guard standby needs to use a sequence, then the application must also be redirected to the remote database. For example:

```
SQL> select customers_seq.nextval from dual@sales_prmy;
   NEXTVAL
----------
      1003
```

APPENDIX C: TUNING REDO APPLY

This section describes the Oracle recommended methodology for tuning the redo apply process. The methodology uses database views as well as Standby Statspack to obtain performance related information. Standby Statspack is used on the primary database to collect data from a standby database that is opened read-only and performing recovery. See My Oracle Support Note 454848.1 for complete details about installing and using Standby Statspack.

Assessing Redo Apply Performance

The first step in tuning redo apply is to determine if there is even a need for tuning at all. If out-of-the box performance is sufficient to prevent apply lag, then no tuning is necessary.

To maintain close synchronization with the primary database, it is recommended that you enable real-time apply on the standby database. This will enable the standby to apply redo changes as they are received, and not wait for a log switch on the primary database. Real-time apply is the default If you have created your configuration using the Data Guard broker. Real-time apply will have to be enabled manually if the configuration was created using SQL*Plus.

Query the `V$DATAGUARD_STATS` view to determine if Redo Apply has recovered all redo that has been received from the primary. For example:

```
SQL> SELECT * FROM V$DATAGUARD_STATS WHERE NAME='apply lag';

NAME               VALUE             TIME_COMPUTED
------------------ ----------------  ------------------------
apply lag          +00 0:00:04       15-MAY-2005 10:32:49
```

The query indicates that Redo Apply has not applied the last 4 seconds of redo received on the standby database.

If the apply lag indicates that the standby is behind the primary by a significant amount this could indicate that a gap exist between the standby and the primary. A gap occurs when events have prevented the successful transmission of a complete redo log file. Redo Apply will not be able to proceed if this has occurred until the gap has been resolved, either automatically by Data Guard, or manually by copying the missing log files and registering them at the standby database should there be other complications preventing the automated process from completing. To detect if a gap does exist, run the following query on both the primary and standby database and compare the results:

```
select 'Instance'||thread#||': Last Applied='||max(sequence#)||'
resetlogs_change#='||resetlogs_change#||')'
from v$archived_log
where applied = (select decode(database_role, 'PRIMARY', 'NO', 'YES')
from v$database)and thread# in (select thread# from gv$instance)
and resetlogs_change# = (select resetlogs_change# from v$database)
group by thread#, resetlogs_change#
order by thread#;
```

The above query will return the latest sequence applied on the standby as well as the latest sequence archived on the primary database for each thread. There is an archive log gap if the last applied sequence on the standby is one or more sequences behind the latest archived on the primary.

Query the `V$RECOVERY_PROGRESS` view to monitor and assess Redo Apply performance. This view contains the following columns:

V$RECOVERY_PROGRESS

COLUMN	DESCRIPTION
Average Apply Rate	Redo Applied / Elapsed Time includes time spent actively applying redo and time spent waiting for redo to arrive
Active Apply Rate	Redo Applied / Active Time is a moving average over the last 3 mins. Excludes time spent waiting for redo to arrive.
Apply Time/Log	Active apply time averaged per redo log file.
Checkpoint Time/Log	Log boundary checkpoint time averaged per redo log file
Last Applied Redo	SCN and timestamp of the last redo applied. This is the time as stored in the redo stream and used to compare where the standby database is relative to the primary.

The most useful statistic is the Active Apply rate because the Average Apply Rate includes idle time spent waiting for redo to arrive, making the Average Apply Rate a poor indicator of apply performance.

In a Data Guard physical standby environment, it is important to determine if the standby database can recover redo as fast as, or faster than, the primary database can generate redo. The simplest way to determine application throughput in terms of redo volume is to collect Automatic Workload Repository (AWR) reports on the primary database during normal and peak workloads, and determine the number of bytes per second of redo data the production database is producing. You can then compare the speed at which redo is being generated with the Active Apply Rate columns in the `V$RECOVERY_PROGRESS` view to determine if the standby database is able to maintain the pace.

It is not necessary to tune redo apply If the standby database is able to keep pace with the primary database during peak periods. If `your` examination of the performance statistics described above indicate that redo apply cannot keep pace, and that the resulting apply lag is not the result of an archive log gap, then proceed with tuning as described below.

Best Practices for Tuning Redo Apply

Before tuning the Redo Apply process, it is important to understand the various wait events specific to Redo Apply.

Parallel Recovery Wait Events

The vast majority of wait events related to parallel recovery coordinators and slaves apply to the coordinator. Slaves are either applying changes (clocking on CPU) or waiting for changes to be passed from the coordinator. The key database wait events on a physical standby database are:

- Parallel Recovery Coordinator Wait Events
- Parallel Recovery Slave Wait Events

Parallel Recovery Coordinator Wait Events

- Log file sequential read - The parallel recovery coordinator is waiting on I/O from the online redo log or the archived redo log
- Parallel recovery read buffer free - All read buffers are being used by slaves, and usually indicates that the recovery slaves lag behind the coordinator.
- Parallel recovery change buffer free - The parallel recovery coordinator is waiting for a buffer to be released by one of the recovery slaves. Again, this is a sign the recovery slaves are behind the coordinator.
- Datafile init write - The parallel recovery coordinator is waiting for a file resize to finish, as would occur with file auto extend.
- Parallel recovery control message reply - The coordinator has sent a synchronous control messages to all slaves, and is waiting for all slaves to reply.

Parallel Recovery Slave Wait Events

Whenever dealing with recovery slave events, it is important to know how many slaves were started. The wait time for any recovery slave event should be divided by the number of slaves

- Recovery read - A parallel recovery slave is waiting for a batch of asynchronous data block reads to complete.
- Checkpoint completed - A parallel recovery slave is waiting for Database Writer to complete checkpoint writes and not applying redo changes.
- DB File Sequential Read - A parallel recovery slave (or serial recovery process) is waiting for a batch of synchronous data block reads to complete.
- Parallel recovery slave next change - A parallel recovery slave is waiting for a change to be shipped from the coordinator. This is in essence an idle event for the recovery slave. To determine amount of CPU a recovery slave is taking, divide the time spent in this event by number of slaves started and subtract that value from total elapsed time. This will be close, because there are some waits involved. This is in essence an idle event for the recovery slave and can be ignored.

Determining if Parallel Recovery is CPU Bound or I/O Bound

From a tuning perspective, parallel recovery is impacted primarily by being either CPU bound or I/O bound. There are no tuning knobs within the Oracle Database that are able to alleviate this because each symptom is a sign of resource issues that would require either faster or more CPU's/spindles, depending on the issue.

elapsed time = time spent working + time spent waiting

This formula applies to any tuning effort and while tuning recovery is no different than any other tuning effort, you will want to place more emphasis on reducing the higher of time-spent working and time spent waiting. To gather this information:

- From the alert log, get the number of recovery slaves started. Oracle automatically configures the number of recovery slaves (apply processes) equal to a value that is one less than the number of CPUs in the standby system. A line in the alert log similar to parallel recovery started with n processes will be logged after the recovery process has been started. If any time is recorded for a wait event related to recovery slaves, you must divide the time by the number of slaves.
- Use the `V$RECOVERY_PROGRESS` view to get the Total Elapsed Time for the recovery session. This includes all of the clock time starting from issuing the `RECOVER` command until the session has been ended, either due to hitting the value specified by using the `UNTIL` clause when starting recovery or by canceling recovery.

- Obtain a Standby Statspack report for the time period to be investigated. Ideally this report should span the elapsed recovery time.

Diagnosing CPU Bound

As stated above, the number of slaves is a function of the number of CPU's on the recovery node, so the only way to relieve CPU contention is to install more or faster CPU's. A CPU bound system displays non-I/O related wait events high on the list of top wait events from the Statspack report. If the majority of the wait events are related to the recovery coordinator, then the recovery slaves are CPU bound (that is, the coordinator is waiting for recovery slaves to complete their work). If the majority of wait events indicate idle recovery slaves, then the coordinator is CPU bound (recovery slaves are waiting for the coordinator to pass on work).

In our tuning example, assume that the Total Elapsed Time in the `V$RECOVERY_PROGRESS` view is 2649 seconds, and the alert log shows that parallel recovery used 3 parallel recovery slaves. In the Standby Statspack report, the top wait events are the following:

```
parallel recovery change buffer free, 202 seconds
log file sequential read, 440 seconds
parallel recovery slave next change, 748 seconds
parallel recovery read buffer free, 772 seconds
```

Totaling the amount of non-idle wait, the coordinator is mostly blocked waiting for the recovery slaves to complete work *(parallel recovery read buffer free 772s and parallel recovery change buffer free 202s, 974/2649 = 37 %)*. So, in this example, the recovery slaves are CPU bound.

Diagnosing I/O Bound

A system is I/O bound when there is not enough throughput to service concurrent I/O requests. In recovery, while the coordinator is reading redo blocks, the slaves are reading data blocks to apply changes and Database Writer (`DBWR`) in turn writes out the data blocks at checkpoint time, so there is a lot of concurrent I/O processing occurring. While faster disks may be helpful, the preferred method to alleviate an I/O bottleneck is to add more spindles and stripe data across those spindles.

In this example, the total runtime for this recovery session was 590 seconds. The statspack report shows the top waits are I/O related:

```
Data file init write, 120s, I/O wait
log file sequential read, 423s, I/O wait
parallel recovery slave next change, 1645s, idle slave
event
```

Log file sequential read and data file init write are coordinator I/O waits. 543/590s = 89% of the time coordinator is waiting on I/O. This immediately indicates the recovery is I/O bound. If this environment had been a synchronous I/O system, the wait event would be db file parallel read rather than recovery read.

If *parallel recovery read buffer free* is high and *recovery read* (async I/O) or *db file sequential read* (sync I/O) is high, this would be indicative of the slaves waiting for I/O.

For file systems where asynchronous I/O is not possible, then use Oracle Disk Manager (ODM), packaged with the release of Veritas Storage Foundation appropriate for the version of the Oracle Database in use. ODM has been shown to provide async-like performance. In addition to using ODM you should also set the database initialization parameters: `filesystemio_options='setall'` and `_media_recovery_read_batch=[512 or 1024]`

APPENDIX D: TUNING QUERIES ON THE STANDBY DATABASE

Tuning Queries on an Active Standby Database

This section describes the Oracle recommended tuning methodology and accompanying tool-set for:

- Optimizing more complex read-only queries executing on an Active Data Guard Standby database
- Monitoring performance in real-time
- Improving the performance of problematic queries.

While this is a different methodology than that used for an Oracle Database that is open read-write – the same goal of providing optimal performance is achieved.

Background

An Active Data Guard standby database is an exact replica of the primary database in all regards. It uses the same optimizer statistics generated by its primary database, including system statistics. It also uses the same SQL plan baselines and SQL profiles used by the primary database (SQL plan baselines are the set of accepted plans for a SQL statement that have been proven to perform well). Please see My Oracle Support note 1264838.1 for additional information on SQL Profiles in an Active Data Guard environment. All actions implemented to tune query performance are executed at the primary database and replicated to the active standby by Data Guard with the exception of any database parameter changes. These characteristics explain why the primary database plays a central role in performance tuning for read-only workload executing on an Active Data Guard standby database.

Methodology

Given that an active standby database uses the same SQL plan baselines and optimizer statistics as the primary, the following methodology will produce the best results if primary and standby databases are hosted on systems having similar performance characteristics (CPU, memory, and I/O).

- Over time, occasionally transition read-only workloads back to the primary database, especially after maintenance operations such as changes to schema and metadata definitions or changes to system settings. This enables the capture of new statistics and the evolution of SQL plan baselines.
- For primary databases that support data warehouse systems, set

  ```
  DBMS_STATS.SET_GLOBAL_PREFS('NO_INVALIDATE','FALSE')
  ```

The `NO_INVALIDATE` preference controls how cursors are invalidated when optimizer statistics are refreshed. The default behavior (`TRUE`) is to invalidate cursors over time to spread out the cost of recompilation, to avoid a spike in cursor invalidations that causes a lot of contention in OLTP systems. Oracle recommends setting it to FALSE so that new statistics cause the invalidation of cursors that are dependent on them. This is a general best practice for Data Warehouse systems as it keeps query plans in lock-step at all times whether there are one or more active standby databases in the configuration.

Note: The above is a general best practice for Data Warehouse systems. Do not alter the default behavior of TRUE *for OLTP systems since the negative performance impact outweighs the lesser incremental benefit of doing so.*

- For all workloads, both data warehouse and OLTP, utilize best practices to minimize apply lag between primary and standby databases. This is important because statistics and other performance information are propagated in the redo stream to the active standby database. Redo Apply performance is optimized using the Redo transport should be optimized using Data Guard Redo Transport and Network Configuration best practices:

  http://download.oracle.com/docs/cd/B28359_01/server.111/b28282/configbp006.htm - ABC3636807SRI1

  Redo Apply performance should also be monitored and tuned as described in the previous sections of this paper.

Monitor Performance – Ad-Hoc Queries

The above recommendations address pro-active tuning for standard queries. To the extent that periodically running read-only workloads on the primary database will collect statistics for ad-hoc queries, the recommendations also enable a degree of proactive tuning for ad-hoc workloads. To completely address requirements for ad hoc queries, however, it is necessary to monitor performance, identify problematic queries, and take corrective actions when needed that are appropriate for an active standby database.

Monitor standby performance using the following tools:

Standby Statspack and Active Session History

As a starting point, use Standby Statspack (see My Oracle Support Note 454848.1) and In-Memory ASH to identify Top SQL (by CPU, Elapsed Time, Buffer Gets, etc.).

- Standby Statspack can be used to periodically analyze where database time is being spent. Identify poorly performing queries from Standby Statspack using the Top SQL sections ('SQL ordered by CPU', 'SQL ordered by Elapsed

Time'). You may also generate a report on a specific SQL Hash ID, for more information see My Oracle Support Note 1081044.1

- Monitor real-time query performance using Active Session History (ASH). Real-time stats can be collected on an Active Data Guard 11g Release standby database using in-memory ASH. ASH reports provide analysis of transient performance problems that typically last for a few minutes. ASH also performs scoped or targeted performance analysis by various dimensions or their combinations, such as time, session, module, action, or `SQL_ID`

  For documentation on ASH see:

  http://download.oracle.com/docs/cd/E11882_01/server.112/e10821/autostat.htm - PFGRF94211

  To generate an ASH report on a physical standby instance, the standby database must be opened read-only. The ASH data on disk represents activity on the primary database and the ASH data in memory represents activity on the standby database. You must specify whether to generate the report using data sampled from the primary or standby database:

```
You are running ASH report on a Standby database.
To generate the report over data sampled on the Primary
database, enter 'P'.
Defaults to 'S' - data sampled in the Standby database.
Enter value for stdbyflag:
Using Primary (P) or Standby (S): S
```

  In the example above, the default value of Standby (S) is selected.

  Note: See My Oracle Support Note 1081055.1 for information needed to enable ASH for Active Data Guard 11g.

Real-Time SQL Monitoring *Active* Report

Beginning with Oracle Database 11g Release 2 (11.2.0.2) you may use Real-Time SQL Monitoring Active Report for a particular execution of long running SQL (accruing more than 5s of CPU or I/O) with Active Data Guard standby databases.

SQL monitor active reports can be generated directly from EM live UI while viewing a detailed SQL monitor report. There (see save/send e-mail buttons on the top right of that page), the SQL monitor detail page can be either saved or sent by e-mail as an active report. Alternatively, the active report can be directly produced using command line by invoking the PL/SQL procedure dbms_sqltune.report_sql_monitor() using "active" as the report type. For example, the following SQL*Plus script shows how to generate an active report for the statement that was monitored last by Oracle:

```
sqlmonitor (note: sql_id, sql_exec_id are available in
V$SQL_MONITOR)
set trimspool on
set trim on
set pages 0
set linesize 1000
set long 1000000
set longchunksize 1000000
spool &1\.html
select dbms_sqltune.report_sql_monitor(sql_id => '&2',
sql_exec_id => &3, type=>'active') from dual;
spool off
```

The resulting file sqlmon_active.html must be edited to remove the header (first line in the file) and the last line (the spool off). The resulting html file can then be viewed in any browser. The browser must have connectivity to OTN to load the active report code.

For additional information on SQL Monitoring Active Report, see:

http://www.oracle.com/technetwork/database/features/manageability/sqlmonitor-084401.html

Real-Time SQL Monitoring *Detail* Report

The Real-Time SQL Monitoring Detail report also supports Active Data Guard as of Oracle Database 11g Release 2 (11.2.0.2). SQL Details Active report is a new type of interactive report powered by Enterprise Manager UI technology hosted on OTN. Other active reports are SQL Monitor and SQL Performance Analyzer.

More information on SQL Details Active report for Active Data Guard will be available at the following URL:

http://www.oracle.com/technetwork/database/features/manageability/sql-detail-099420.html

SQL Performance Analyzer (SPA)

Beginning with Oracle Database 11g Release 2 (11.2.0.2) you may use SQL Performance Analyzer (SPA) to analyze the impact of primary system changes in order to tune Active Data Guard query performance. SPA is a central feature of the Real Application Testing option.

SPA is utilized with an Active Data Guard standby using the following process:

1. Enable the SQL Trace facility on the active standby database, as described at: http://download.oracle.com/docs/cd/E11882_01/server.112/e12254/spa_upgrade.htm - CIAFGCBI

2. To minimize the performance impact of running SQL Trace and still be able to fully capture a representative set of SQL statements, consider enabling SQL Trace for only a subset of the sessions, for as long as required, to capture all important SQL statements at least once. Copy the SQL trace file over to the primary database.

3. On the primary database, create a mapping table, which is used to convert the user and object identifier numbers in the SQL trace files to their string equivalents. See: http://download.oracle.com/docs/cd/E11882_01/server.112/e12254/spa_upgrade.htm - CIADAFGC

4. On the primary database construct a SQL tuning set (STS) using the SQL trace files and the `DBMS_SQLTUNE` package. The SQL tuning set will contain the SQL statements captured in the SQL trace files, along with their relevant execution context and statistics.

5. On the primary database use SPA to create a SPA task and convert the contents in the SQL tuning set into a pre-upgrade SQL trial that will be used as a baseline for comparison.

6. Implement the changes on the production system. Changes are captured in the redo and are shipped and applied to the active standby by Data Guard.

7. Then remotely test execute the SQL statements on the active standby over a database link to build a post-upgrade SQL trial. The SPA test executes the SQL statements using a public database link that you specify by connecting to the active standby database remotely and generating the execution plans and statistics for the SQL trial. The database link should exist on the primary database and connect to a remote user with privileges to execute the SQL tuning set on the active standby database.

8. SPA compares the performance of SQL statements read from the SQL tuning set during the pre-upgrade SQL trial to those captured from the remote test execution during the post-upgrade SQL trial. A report is produced to identify any changes in execution plans or performance of the SQL statements.

9. If the report reveals any regressed SQL statements, make the necessary changes to correct the regressed SQL using the SQL Tuning Advisor or SQL plan baselines as described here: http://download.oracle.com/docs/cd/E11882_01/server.112/e12254/spa_analyze.htm - CHDHGFEH

10. Repeat the process of executing the SQL tuning set and comparing its performance to a previous execution to test any changes made until you are satisfied with the outcome of the analysis.

Note: In a separate context, an Active Data Guard standby database can also be used in a limited way as a test system for its production database. Pre and post-change SPA trials for an STS can be run on an Active Data Guard standby database as outlined above, only in this case, the system changes are implemented at the Active Standby database and not the primary. Changes that can be tested on an Active Data Guard standby database include alter session, pending stats, and certain alter system changes. Changes are implemented via logon triggers. By default SPA will only execute the query portion of any DML. SPA fulldml mode (for complete execution of DMLs including insert/update/delete) cannot be used on an Active Data Guard standby database.

Oracle Active Data Guard Oracle Database 11*g*
September, 2011
Author: Michael T. Smith

Oracle USA, Inc.
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com