Session 6

Workload Management

Advanced RAC
Auckland - May 2008

Agenda

- Database Services
- Workload Balancing
- Transparent Application Failover (TAF)
- Fast Application Notification (FAN)
- Server-Side Callouts
- Oracle Notification Service (ONS)
- Fast Connection Failover

Database Services Overview

- In Oracle 10.1 and above database services can be created and configured
- Database services
  - Define logical subsets of an application workload
  - Group of sessions performing same type of work or running same application
  - Assigned to a pool of database instances
  - Can be managed as separate entities
  - Provide the basis for high availability
  - Simplify deployment, testing, disaster recovery and administration
  - Can be configured for workload balancing

In Oracle 10.1 and above, it is possible to define database services which represent a logical subset of the workload of an application. Each database service is a group of sessions performing the same type of work or running the same application and assigned to a pool of possible database instances. Database services can be managed as separate entities, reducing the management overhead and providing the basis for high availability. Database services simplify system deployment, testing, disaster recovery, and administration.

Database services have been available in single-instance environments for many years. In Oracle 10.1 and above, they can be used in conjunction with the Resource Manager to control resource usage. In a single-instance environment, database services can also provide additional levels of granularity for statistics aggregation and tracing.

Database Services Overview

- Database services are logical groupings analogous to
  - tablespaces
  - roles
- For example database services might be created for
  - Accounts Payable
  - Accounts Receivable
  - Customer Relationship Management
  - Sales
  - Warehouse

A *database service* is a logical grouping of sessions analogous to other logical groupings within Oracle, such as tablespaces and roles. Users with similar service-level requirements should all be assigned to the same service. The database services you define should reflect the main groups of resource consumers within your workload. For example, you may decide to create database services on a departmental basis, such as Accounts Payable, Accounts Receivable, and Customer Relationship Management. Alternatively, you may define database services to reflect types of workload, for example, Online, Reports, and Batch. With database services, users connect to a database without regard for which instance executes the SQL session.

Database Services
RAC Environment

- In a RAC environment
  - You can control which instances are allocated to each database services at different times
  - Database services can be shared across one or more instances
  - Additional instances can be made available dynamically in response to failures or changes in workload

However, you can fully realize the potential of database services in a RAC environment by controlling which instances are allocated to different database services at different times]Database services are available continuously with the workload shared across one or more instances. Additional instances can be made available dynamically in response to failures or changes in workload. This helps you reflect business priorities and needs in your management of the database.

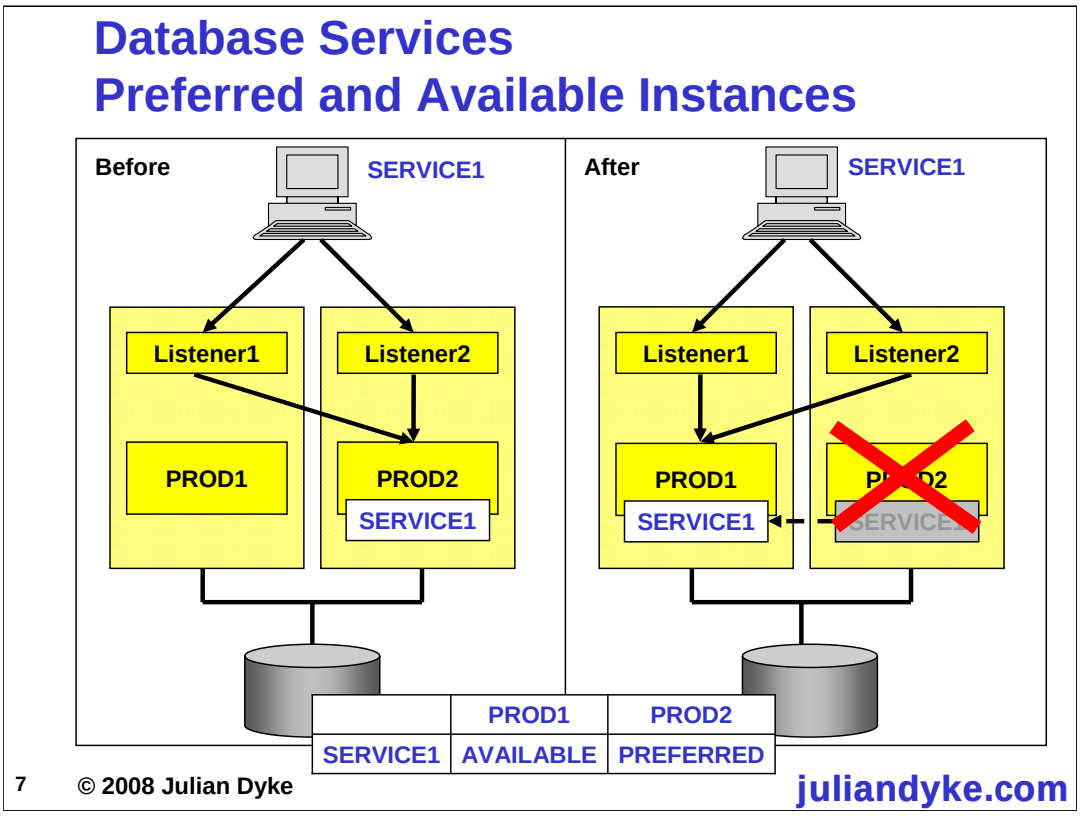
Database Services
Preferred and Available Instances

- Each database service can be assigned to
  - One or more preferred instances - these will be used by the service if they are available
  - One or more available instances - these will be used if the preferred instance(s) is not available
- You can also designate instances that a database service can never use
- Database services which are relocated to an available instance are not automatically restored to the preferred instance
  - If the preferred instance is restored, the database service must be relocated manually

Each database service is initially assigned to one or more instances known as the PREFERRED instances, and they will be used when they are available. You can also assign the database service to one or more secondary instances which can be used if one of the PREFERRED instances is unavailable. If a PREFERRED instance fails, then Oracle will automatically move the database service from the failed instance to a surviving secondary instance. Secondary instances are rather confusingly also known as AVAILABLE instances.

If a database service is relocated to an AVAILABLE instance following the failure of a PREFERRED instance, and the PREFERRED instance is subsequently restarted, the database service will not necessarily be relocated back from the AVAILABLE to the PREFERRED instance; this avoids a second service outage. It is possible to override this behavior manually by configuring FAN callouts.

Clients and middle-tier applications make connection requests by specifying a global database service name. In the event that one or more of the preferred instances is unavailable, the connection will transparently be redirected to any available alternate instance.

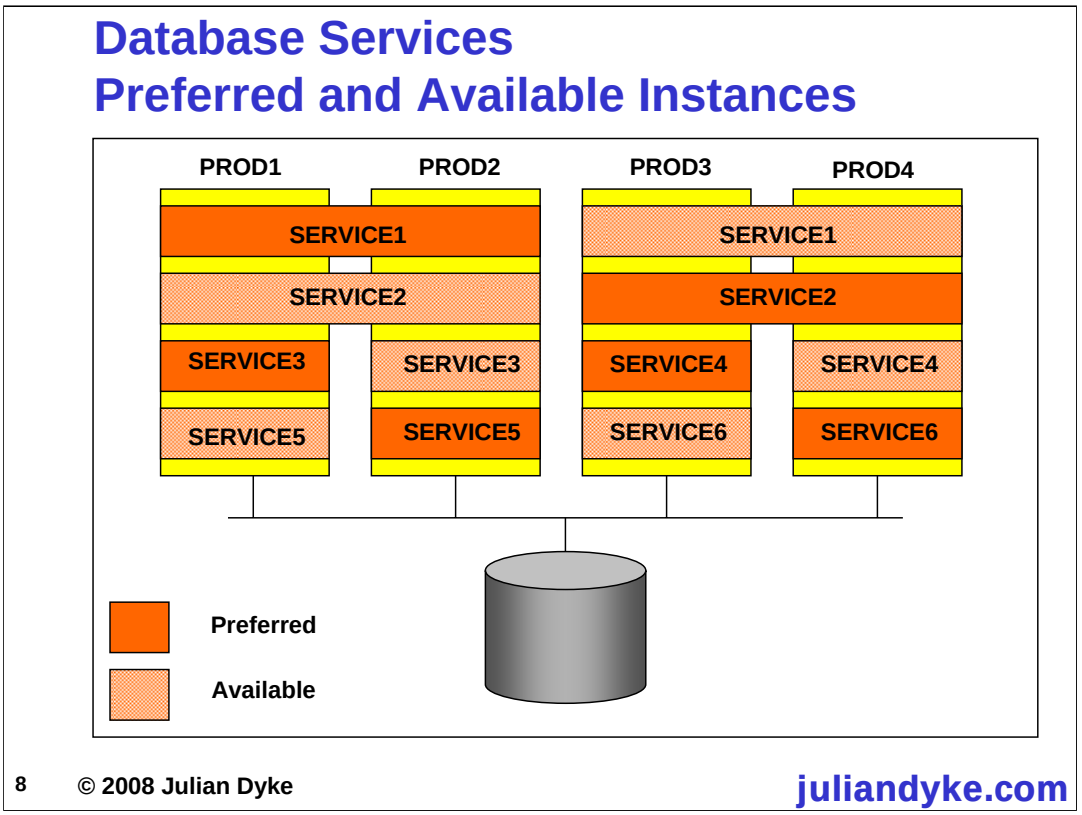


This slide shows how services can be used to specify preferred and available instances. In the above example SERVICE1 has one preferred instance (PROD2) and one available instance (PROD1).

During normal operation, all connections to the service will be sent to PROD2. In this example, the client is performing client-side load balancing so connections can be sent to the listener process on either node. The listener is aware that SERVICE1 is currently running on PROD2 and therefore forwards all incoming SERVICE1 connections to that instance.

If PROD2 is shutdown or fails then Oracle Clusterware will automatically relocate SERVICE1 to the available instance, in this casePROD1. The listener processes on both nodes will now send all incoming SERVICE1 connections to PROD1.

Note that if PROD2 is subsequently restarted, SERVICE1 will not be automatically relocated back to PROD1. SERVICE1 will continue to run on PROD1 until it is manually relocated back to PROD2 using Enterprise Manager or SRVCTL.



The diagram above shows a possible configuration for a 4 instance database supporting six services.

The configuration is as follows:

Service	Preferred Instances	Available Instances
SERVICE1	PROD1, PROD2	PROD3, PROD4
SERVICE2	PROD3, PROD4	PROD1, PROD2
SERVICE3	PROD1	PROD2
SERVICE4	PROD3	PROD4
SERVICE5	PROD2	PROD1
SERVICE6	PROD4	PROD3

Database Services
Preferred and Available Instances

- **Consider the following configuration for SERVICE1**

Service	PROD1	PROD2	PROD3	PROD4
SERVICE1	Preferred	Preferred	Available	Available

- **SERVICE1 will normally run on instances PROD1 and PROD2**

```
srvctl status service -d PROD -s SERVICE1
Service SERVICE1 is running on instance(s) PROD1, PROD2
```

- **If either PROD1 or PROD2 are unavailable then SERVICE1 can be relocated to PROD3 or PROD4.**
- **For example if PROD2 fails, SERVICE1 might be relocated to PROD3:**

```
srvctl status service -d PROD -s SERVICE1
Service SERVICE1 is running on instance(s) PROD1, PROD3
```


This slide shows a sample database service configuration.

SERVICE1 will normally run on instances PROD1 and PROD2. If either PROD1 or PROD2 is unavailable then SERVICE1 can run on PROD3 or PROD4

You can check the configuration for a service using

srvctl config service -d <database> [-s <service>]

If no service is specified then the configuration of all services will be displayed

For the above example:

$ srvctl config service -d PROD -s SERVICE1

SERVICE1 PREF: PROD2 PROD1 AVAIL: PROD4 PROD3

Database Services
Preferred and Available Instances

- However when the PROD2 instance is restarted, the service will not automatically be relocated:

```
$ srvctl start instance -d PROD -i PROD2

$ srvctl status service -d PROD -s SERVICE1
Service SERVICE1 is running on instance(s) PROD1, PROD3
```

- New connections for SERVICE1 will continue to be redirected to instances PROD1 and PROD3
- SERVICE1 must be manually relocated from PROD3 to PROD2

```
$ srvctl relocate service -d PROD -s SERVICE1 -i PROD3 -t PROD2

$ srvctl status service -d PROD -s SERVICE1
Service SERVICE1 is running on instance(s) PROD2, PROD1
```

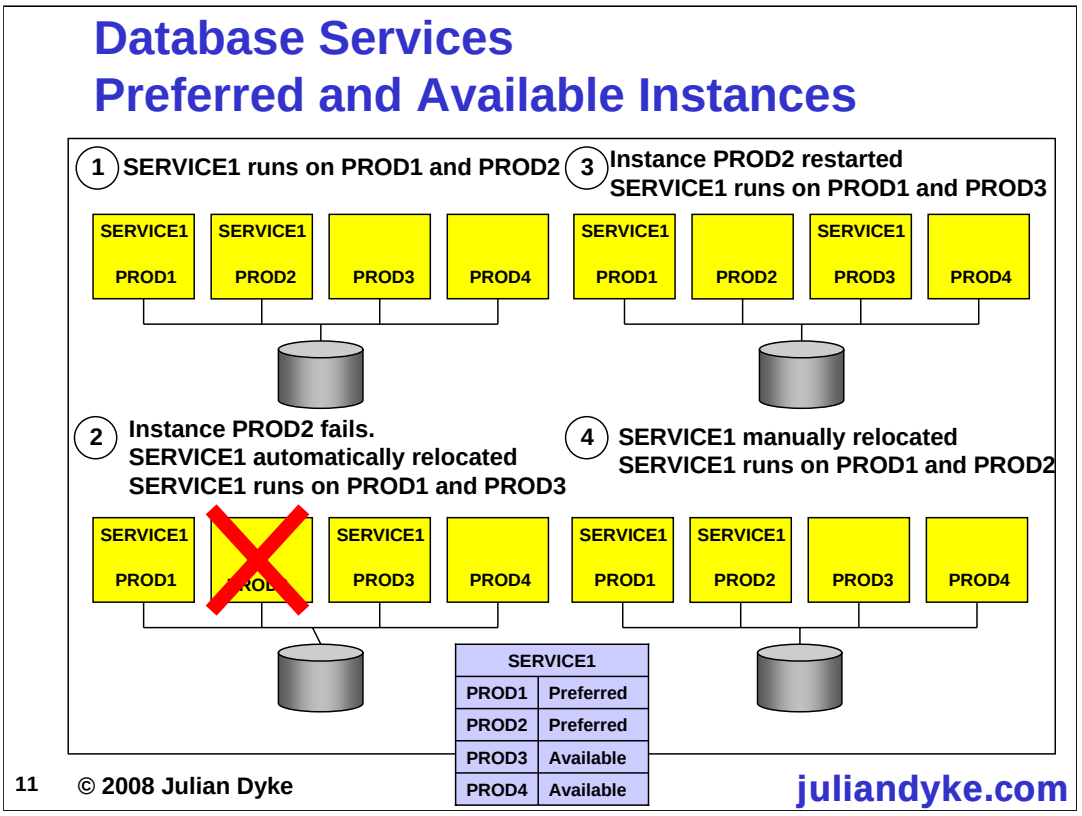
- New connections for SERVICE1 will now be directed to instances PROD1 and PROD2

10 juliandyke.com

If a service is relocated to an available instance it will not automatically be relocated back to the preferred instance when the preferred instance is restarted. The service must be relocated manually.

Services can be relocated using SRVCTL as shown in the example above or they can be relocated using Enterprise Manager.

This prevents excessive activity on the restarted instance. It also allows time for troubleshooting on the restarted node.



This slide shows how service failover is managed when there is more than one preferred instance and more than one available instance. In this example SERVICE1 has been configured with preferred instances PROD1 and PROD2 and available instances PROD3 and PROD4

1- SERVICE1 is running on preferred instances PROD1 and PROD2.

2 - Instance RAC2 fails (I killed the PMON background process). SERVICE1 is relocated by Oracle Clusterware from instance PROD2 to instance PROD3. SERVICE1 is now running existing preferred instance PROD1 and available instance PROD3.

3 - Node 2 (PROD2) is automatically restarted by Oracle Clusterware. However, failed-over services are not automatically relocated so SERVICE1 continues to run on existing preferred instance PROD1 and available instance PROD3.

4 - SERVICE1 is manually relocated from instance PROD3 to instance PROD2 using:

srvctl relocate service -d PROD -s SERVICE1 -i PROD3 -t PROD2

SERVICE1 is now running on preferred instances PROD1 and PROD2 again.

Database Services
Preferred and Available Instances

- **Now consider the following configuration for SERVICE2**

Service	PROD1	PROD2	PROD3	PROD4
SERVICE2	Preferred	Preferred	Not Used	Not Used

- **SERVICE2 will normally run on instances PROD1 and PROD2**

```
srvctl status service -d PROD -s SERVICE2
Service SERVICE2 is running on instance(s) PROD1, PROD2
```

- **If PROD2 is unavailable SERVICE2 can still run on PROD1.**

```
srvctl status service -d PROD -s SERVICE2
Service SERVICE2 is running on instance(s) PROD1
```

- **If PROD1 is unavailable then SERVICE2 cannot run.**

```
srvctl status service -d TEST -s SERVICE2
Service SERVICE2 is not running
```


If both PROD1 and PROD2 instances are unavailable, then attempts to connect to SERVICE2 will result in the following error:

ORA-12514: TNS:listener does not currently know of service requested in connect descriptor

Database Services
Load Balancing

- In Oracle 10.2 and above each database service can be associated with a load balancing goal
- Connections can be made to instances based on:
  - Service quality
    - response time
  - Overall system throughput
    - completing jobs as efficiently as possible
- Database services can also be associated with a load balancing method which can be
  - LONG - sessions connected once and then execute a potentially large number of statements
  - SHORT - sessions connect, execute a small number of statements and then disconnect again

Database Services
Transparent Application Failover

- Database services can be configured to use Transparent Application Failover (TAF)
- Following failure of a node TAF specifies
  - whether sessions can resume processing SELECT statements on another node
  - whether sessions should pre-connect to a failover instance
  - number of retry attempts during a failover
  - timeout between retry attempts
- Service TAF configuration takes precedence over client TAF configuration

A session can also have associated Transparent Application Failover (TAF) characteristics. These characteristics specify whether, following a node or instance failure, sessions can resume processing SELECT statements on another node, and also whether sessions should pre-connect to any failover instances. It is also possible to specify the maximum number of retry attempts in the event of a failover and a time-out between attempts.

Database Services Resource Manager

- Database services can be assigned to Resource Manager consumer groups
- When a user connects with a specific database service, they will automatically be assigned to the associated Resource Manager consumer group
- You can restrict resource usage by a database service within each instance

Database services can be assigned to Resource Manager consumer groups, allowing you to restrict resource usage by database service within each instance. When a user connects with a specific database service, that user will automatically be assigned to the associated Resource Manager consumer group.

Database Services
Other Features

- In Oracle 10.1 and above, a number of other Oracle features are aware of database services including:
  - Parallel Execution
  - Scheduler
  - Advanced Queuing
  - Streams

In Oracle 10.1 and above, a number of other Oracle features are aware of database services and can use them to manage their workloads, including Parallel Execution, the Job Scheduler, Advanced Queuing, and Streams.

Database Services
Service Names

- In Oracle 10.1 and above, a maximum of 64 database services can be configured
  - Oracle creates two default services
  - The remaining 62 can be user-defined
- There are two default services
  - SYS$BACKGROUND - Used by background processes only
  - SYS$USERS - Default service for user sessions that are not associated with any other database users
- Database service names are stored in SERVICE_NAMES initialization parameter
- Maximum length of parameter value is 4096 bytes

In Oracle 10.1 and above, a maximum of 64 database services can be created. Oracle creates two default services; the remaining 62 database services can be user-defined. The two default services are as follows:

• SYS$BACKGROUND: Used by background processes only

• SYS$USERS: Default service for user sessions that are not associated with any other database services

These services are always available on every instance within the database unless the instance is running in restricted mode. You cannot modify these database services or their attributes, and you also cannot stop or disable them

Database service names are stored by Oracle in the SERVICE_NAMES initialization parameter, which has a maximum size of 4KB. The maximum length of the service names of all services assigned to an instance therefore cannot exceed 4KB. You should not attempt to modify the SERVICE_NAMES parameter manually.

Database Services Administration

- Database Services can be administered using
  - DBCA
  - Enterprise Manager (Oracle 10.2 and above)
  - SRVCTL
  - SQL*Plus

Database services can be administered using DBCA or, in Oracle 10.2. and above, using Enterprise Manager. In the past you could also use a combination of SRVCTL and SQL*Plus to create and delete services. However, I do not recommend this as the data dictionary entries created by the DBMS_SERVICE package rely on some internal sequence numbers.

DBCA and Enterprise Manager both configure entries in the TNSNAMES.ORA file for the new services. If the service is created using SQL*Plus, you will need to add the services manually to TNSNAMES.ORA.

Database Services Administration using DBCA

- In Oracle 10.1 and 10.2 DBCA can be used to
  - Create and Delete Database Services
  - Assign Preferred, Available and Unused Instances
  - Configured Transparent Application Failover
- As the oracle user in an X-Windows session start the DBCA
  - On the Welcome page select the Oracle Real Application Clusters database option and click Next to continue
  - On the Operations page select Services Management and click Next
  - On the List of Databases page, select the cluster database to be configured and press Next

You can use the DBCA Service Management page to manage database service assignments, instance preferences, and TAF policies. You can use DBCA to modify database service configurations while the database is started and also when it is shut down.

To add or modify database services, create an X Window terminal session and start the DBCA tool at the command line:

london1$ dbca

Then follow these steps:

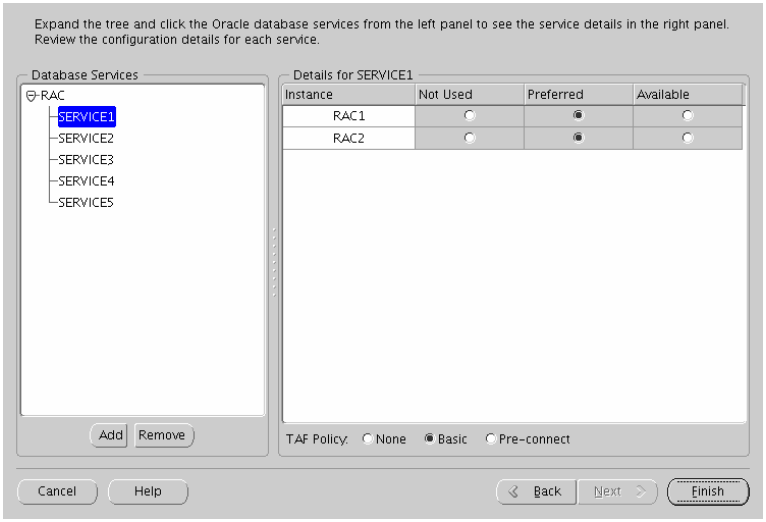
1. On the Welcome page, select the Oracle Real Application Clusters database option and click Next.

2. On the Operations page, select Services Management and click Next.

3. On the List of Databases page, select the cluster database you wish to configure and click Next.

4 See next slide

Database Services Administration using DBCA



4 . On the Database Services page, any database services that have already been assigned to the database will be displayed. Here you can select any database service and specify the preferred and available instances. You can also select the TAF policy for the instance.

To add a new database service, click the Add button. You will be prompted to enter a name for the new service. The new service will be added to the Database Services page. You can then set the instance and TAF details as just described.

You can also use this page to remove a database service by clicking the Remove button. The highlighted service will be removed.

5 Click the Finish button to display a summary message, and then click OK to apply the changes. A progress message is displayed while the database services are configured.

When you use DBCA to add database services, it adds them to the OCR, updates the Net Service entries for each service, and starts the services. When you use DBCA to remove database services, it stops the services, removes the Net Service entries, and deletes them from the OCR.

Database Services Administration using Enterprise Manager

- In Oracle 10.1 and above Enterprise Manager can be used to
  - view current database service configuration including
    - preferred and available instances
    - TAF policy
  - enable / disable database services
  - start / stop database services
  - relocate database services
- In Oracle 10.2 and above Enterprise Manager can also create and delete database services
- In Oracle 11.1 Enterprise Manager is the only GUI tool capable of administering database services

In Oracle 10.1 and above Enterprise Manager can be used to

• view the current database service configuration including the preferred and available instances and the TAF policy

• enable / disable database services

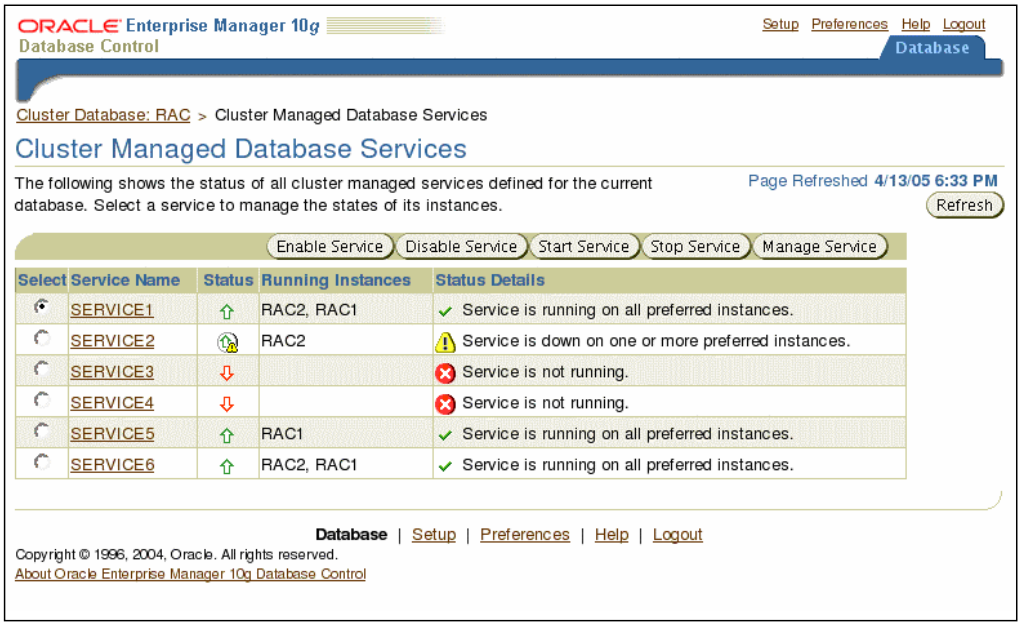
• start / stop database services

• relocate database services

However, in Oracle 10.1 Enterprise Manager cannot be used to create or delete services.

In Oracle 10.2 and above Enterprise Manager can create and delete database services

Database Services
Administration using Enterprise Manager



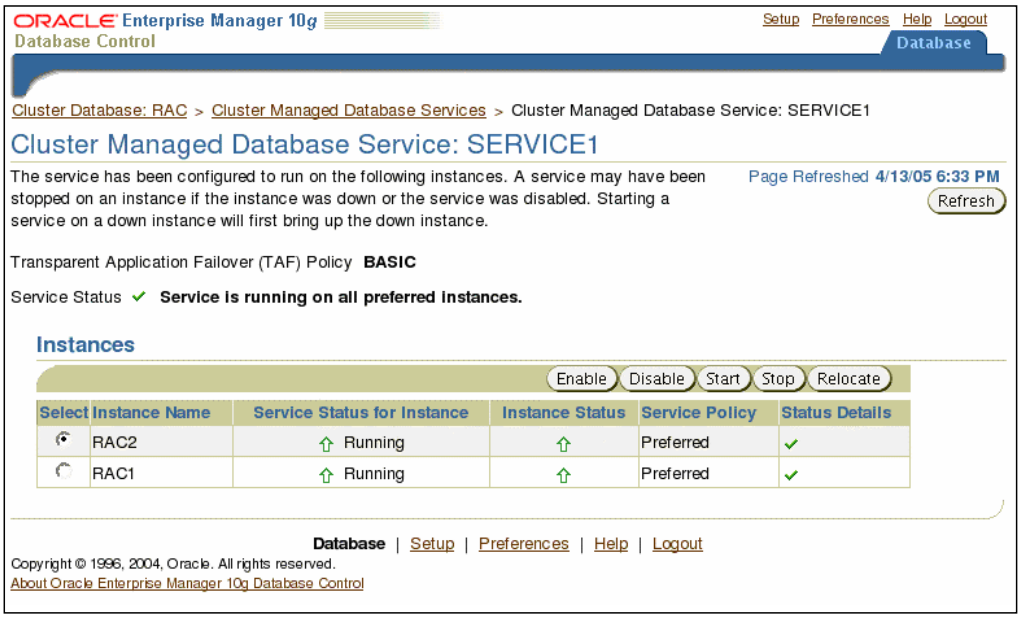
In Oracle 10.1 and above, you can use the Enterprise Manager Service Administration page to view the current database service configuration; enable, disable, start, stop, and relocate services; and for each database service, view the preferred instance, available instance, and TAF policy.

To access the Cluster Managed Database Services page from the Enterprise Manager Database Control, home page click the Administration tab. Under the High Availability options list, click Cluster Managed Database Services.

At this stage, you may be prompted for your operating system credentials, in which case enter your operating system username and password. The Cluster Managed Database Services page will be displayed, as shown above. On this page, you can enable, disable, start, and stop services.

If you click either the Service Name hyperlink or the Manage Service button, you will be taken to the service details page for the currently specified database service as shown on the next slide.

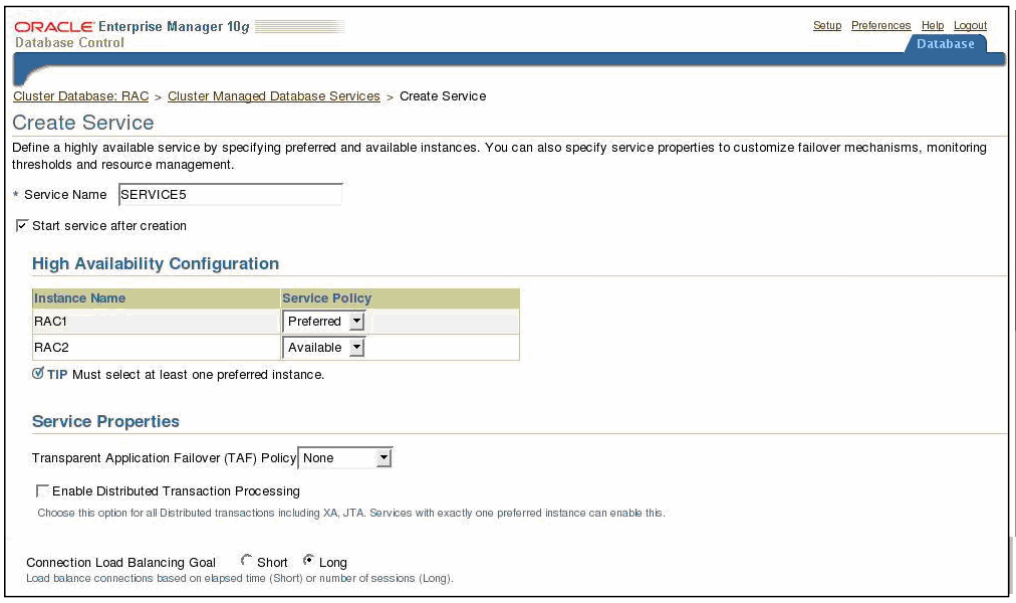
Database Services
Administration using Enterprise Manager



This service details page shows the allocation of instances for the specified database service. You can use this page to start, stop, and relocate database services on a specific instance.

You can use EM to administer database services after they have been created using DBCA or SRVCTL. In Oracle 10.1, you cannot use EM to create database services or to specify TAF policies

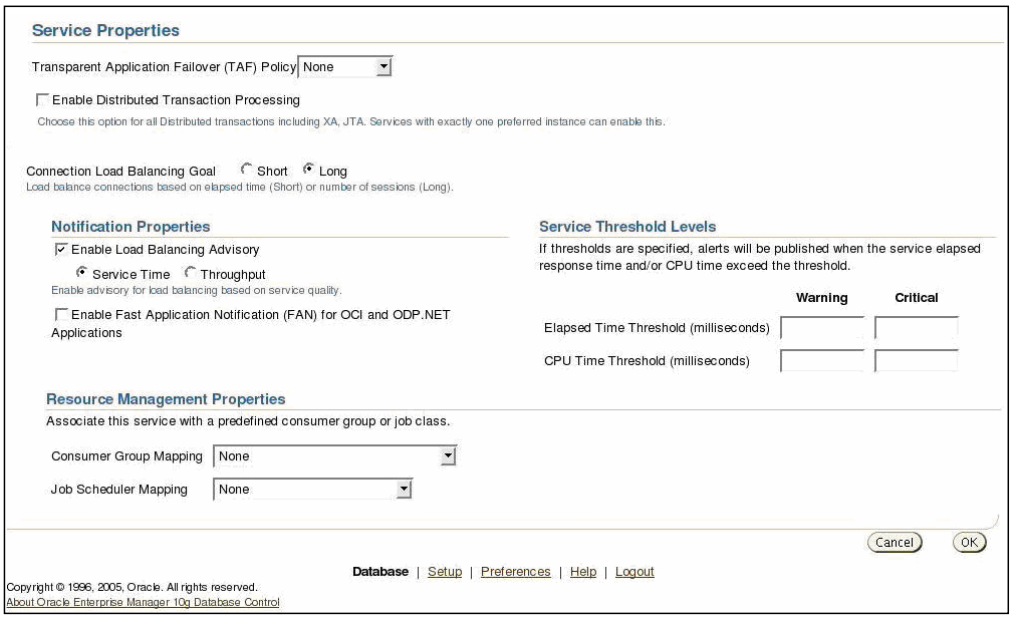
Database Services Administration using Enterprise Manager



In Oracle 10.2 and above, Enterprise Manager Database Control has been enhanced to include the creation, configuration, and deletion of database services. In addition, the Cluster Managed Database Services page has moved. You can now access this page by selecting Maintenance ~TRA Cluster Managed Database Services.

In Oracle 10.2, the Cluster Managed Database Services page includes buttons to start, stop, and test the connection to the selected database service. It also includes an Actions drop-down list which allows you to manage (the default), ,delete, enable, disable and edit the properties of selected database services .Finally, a new Create Service button allows you to create a new service. If you click this button, you will be taken to the Create Service page the top half of which is shown on this slide and the bottom half of which is shown on the next slide.

Database Services Administration using Enterprise Manager



The Create Service page allows you to specify the following:

• A database service name

• A database service policy for each instance: Preferred (default), Available, or Not Used

• A TAF policy: None (default), Basic, or Preconnect

• A Connection Load Balancing Goal: Short (the default in EM, but not in the CREATE_SERVICE procedure of DBMS_SERVICE) or Long

You can also do the following:

• Enable the load balancing advisory, which is disabled by default, and specify the load balancing goal, which can be Service Time or Throughput.

• Enable Fast Application Notification (FAN) for connection pooling applications.

• Specify a consumer group mapping and a Job Scheduler mapping for Resource Manager.

• Specify warning and critical values for service threshold levels for elapsed time and CPU time. When these thresholds are exceeded, alerts will be published.

You can edit the same set of properties at any time by selecting an existing database service from the Cluster Managed Database Service page, selecting Edit Properties from the Action drop-down, and clicking Go.

Database Services Administration

- Database services can also be administered using command line tools
- For example to create a new service called SERVICE1
  - On each node add an entry to $ORACLE_HOME/network/admin/tnsnames.ora

```
NET_SERVICE1 =
   (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server3-vip)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = server4-vip)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = server11-vip)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = server12-vip)(PORT = 1521))
      (LOAD_BALANCE = yes)
      (CONNECT_DATA =
         (SERVER = DEDICATED)
         (SERVICE_NAME = SERVICE1)
      )
   )
```


Note that in this example the service name is SERVICE1 and the net service name is NET_SERVICE1. This is for illustrative purposes only. Normally the service name and the net service name will be identical.

Database Services Administration

- **Service creation (continued):**
- **Use SQL*Plus to create the service in the database**

```
DBMS_SERVICE.CREATE_SERVICE ('SERVICE1','NET_SERVICE1');
```

- **Use SRVCTL to create and start the service in the OCR**

```
[oracle@server3]$ srvctl add service -d TEST -s SERVICE1 \
-r "TEST1,TEST2" -a "TEST3,TEST4"
```

```
[oracle@server3]$ srvctl start service -d TEST -s SERVICE1
```

- **Check the status of the new service:**

```
[oracle@server3]$ srvctl status service -d TEST -s SERVICE1
Service SERVICE1 is running on instance(s) TEST1, TEST2
```


Additional attributes can be specified for service creation including

• TAF configuration

• Distributed Transaction Processing (DTP) configuration

• AQ HA Notification

These attributes can also subsequently be modified using the DBMS_SERVICE.MODIFY_SERVICE

Services can be deleted using DBMS_SERVICE.DELETE_SERVICE

You can confirm that the service is correctly configured using the SYS_CONTEXT built-in function. For example:

sqlplus user1/user1@SERVICE1

SELECT SYS_CONTEXT ('USER_ENV','SERVICE_NAME')
AS SERVICE_NAME FROM dual;

SERVICE_NAME

SERVICE1

Database Services
Listener

- **Services currently supported by a listener can be displayed using LSNRCTL SERVICES**
- **For example:**

```
Service "SERVICE1" has 2 instance(s).
  Instance "TEST1", status READY, has 2 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
      "DEDICATED" established:0 refused:0 state:ready
         REMOTE SERVER
         (ADDRESS=(PROTOCOL=TCP)(HOST=server3.juliandyke.com)(PORT=1521))
  Instance "TEST2", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:1 refused:0 state:ready
         REMOTE SERVER
         (ADDRESS=(PROTOCOL=TCP)(HOST=server4.juliandyke.com)(PORT=1521))
```


Additional information can be displayed by setting the DISPLAYMODE to VERBOSE

For example:

LSNRCTL> SET DISPLAYMODE VERBOSE

LSNRCTL> SERVICES

Further information can be displayed by setting the DISPLAYMODE to RAW

For example:

LSNRCTL> SET DISPLAYMODE RAW

LSNRCTL> SERVICES

Database Services Views

- Information about services configured within a database is reported in the DBA_SERVICES data dictionary view
- A subset of that information is reported in the V$SERVICES dynamic performance view
- Currently active services are reported in the V$ACTIVE_SERVICES dynamic performance view

The DBA_SERVICES data dictionary view provides information about currently configured database services

You can also check which database services are currently enabled using the V$SERVICES dynamic performance view.

The GV$ACTIVE_SERVICES dynamic performance view shows the currently active services on each instance in the cluster.

In all three views the GOAL, DTP, ENABLED, AQ_HA_NOTIFICATION and CLB_GOAL columns were introduced in Oracle 10.2.

Database services will appear in GV$ACTIVE_SERVICES once for each instance on which they are currently active, for example:

```
SELECT inst_id, service_id, name
FROM gv$active_services
WHERE name LIKE 'SERVICE%';

INST_ID SERVICE_ID NAME
------- ---------- ----------------------
      1          7 SERVICE1
      1          8 SERVICE2
      2          7 SERVICE1
      2          9 SERVICE3
```

In the example, SERVICE1 is running on instances 1 and 2. SERVICE2 is only running on instance 1, and SERVICE3 is only running on instance 2.

Database Services
Dynamic Performance Views

- **V$SERVICE_STATS Contains limited statistics for all configured services**
- **In Oracle 10.2, service statistics include:**

application wait time	parse time elapsed
cluster wait time	physical reads
concurrency wait time	physical writes
db block changes	redo size
DB CPU	session cursor cache hits
DB time	session logical reads
execute count	sql execute elapsed time
gc cr block receive time	user calls
gc cr blocks received	user commits
gc current block receive time	user I/O wait time
gc current blocks received	user rollbacks
logons cumulative	workarea executions - multipass
opened cursors cumulative	workarea executions - onepass
parse count (total)	workarea executions - optimal

V$SERVICE_STATS includes statistics for the default services

SYS$BACKGROUND

SYS$USERS

It also shows statistics for the database and individual services. For example, if the database is called RAC and has four services called SERVICE1, SERVICE2, SERVICE3 and SERVICE4, there will be a total of seven sets of statistics; two for the default services, four for the user-defined database services and one for the RAC service.

Only a limited set of 28 statistics are maintained for each service in V$SERVICE_STATS. This limits the amount of SGA memory required to store the statistics and also the amount of CPU required to maintain them.

Database Services
Session Management

- In Oracle 11.1 and above all sessions connected to a specific service can be disconnected from the current instance

```
DBMS_SERVICE.DISCONNECT_SESSION (service_name,option');
```

- service_name is mandatory
- option can be:
  - DBMS_SERVICE.POST_TRANSACTION (default)
  - DBMS_SERVICE.IMMEDIATE
- For example:

```
DBMS_SERVICE.DISCONNECT_SESSION ('SERVICE1");
```

- disconnects all sessions connected to SERVICE1 on the current instance only
- Internally calls ALTER SYSTEM DISCONNECT SESSION

DBMS_SERVICE.DISCONNECT_SESSION disconnects all sessions connected to the local instance with the specified service name. By default it waits until all corresponding sessions have been disconnected.

At the time of wrting the documentation for DBMS_SERVICE.DISCONNECT_SESSION was incorrect.

More reliable documentation can be found in the package header which is $ORACLE_HOME/rdbms/admin/dbmssrv.sql

DBMS_SERVICE.DISCONNECT_SESSION calls ALTER SYSTEM DISCONNECT SESSION sid, serial, option which is described on the next slide.

DBMS_SERVICE.DISCONNECT_SESSION can be used with single instance databases as well as RAC databases.

Database Services
Session Management

- **ALTER SYSTEM DISCONNECT SESSION has the following syntax**

```
ALTER SYSTEM DISCONNECT SESSION 'sid, serial'
IMMEDIATE | POST_TRANSACTION
```

- **Either POST_TRANSACTION or IMMEDIATE must be specified**
- **For example:**

```
ALTER SYSTEM DISCONNECT SESSION '131,28' IMMEDIATE;
ALTER SYSTEM DISCONNECT SESSION '120,42' POST_TRANSACTION
```

- **If POST_TRANSACTION is specified the calling session will wait until the target session completes its transaction**

Either POST_TRANSACTION or IMMEDIATE must be specified for the option otherwise the following error will be raised:

ORA-02000: missing POST_TRANSACTION or IMMEDIATE keyword

ALTER SYSTEM DISCONNECT SESSION IMMEDIATE causes a

ORA-03135: connection lost contact

to be raised on the target session next time it attempts to execute a SQL statement.

ALTER SYSTEM DISCONNECT SESSION POST_TRANSCATION will allow transactions to complete and to COMMIT or ROLLBACK. However the next statement executed after the COMMIT/ROLLBACK will receive the following error:

ORA-00028: your session has been killed

Database Services
Session Management

- In Oracle 11.1 and above ALTER SYSTEM KILL SESSION can be used to kill sessions on another instance.
- The extended syntax is

```
ALTER SYSTEM KILL SESSION 'sid, serial,@instance' [option];
```

- For example

```
ALTER SYSTEM KILL SESSION '120,51,@2';
ALTER SYSTEM KILL SESSION '120,54,@2' IMMEDIATE;
```

- ALTER SYSTEM KILL SESSION does not have a POST_TRANSACTION option

Without the IMMEDIATE clause, the command:
ALTER SYSTEM KILL SESSION '120,51,@2';
will cause the following error to be raised in the target session
ORA-00028: your session has been killed
With the IMMEDIATE clause, the command:
ALTER SYSTEM KILL SESSION '120,54,@2' IMMEDIATE;
will cause the following error to be raised in the target session
ORA-03135: connection lost contact

Connection Management
Workload Balancing

- Goals of workload balancing:
  - Distribute workloads across nodes in cluster
  - Maximize scalability and throughput
  - Minimize effect of individual node failures on overall throughput
- In an ideal system workload would be evenly distributed across each node in cluster
  - Can be achieved for some applications
    - All transactions have similar resource requirements
  - For most applications
    - Some transactions more resource intensive than others
    - Asymmetric workload balancing required

Workload management involves the distribution of workloads across the different nodes in the cluster.

In an ideal world, the workload would be evenly distributed across each node in the cluster. For some applications this is possible. Normally these applications will have simple atomic transaction types and all transactions will use a similar amount of resources. This type of application is often found within telcos. Some internet applications have similar resource profiles. In these cases a round-robin algorithm or a random algorithm should be sufficient to guarantee a well-balanced workload.

However, most applications have transactions with differing resource requirements. There might be a mix of OLTP and reporting workloads. Alternatively users might be able to write ad-hoc reports. Complex processing might be performed offline for example in E-Business suite. Different sessions and/or transactions might have different durations. In cases such as these, simply directing a connection to the next node on a round-robin basis is not sufficient and a more sophisticated solution is required. This inevitably requires an asymmetric approach to connection balancing to ensure that new connections are directed to the least loaded node.

Connection Management
Connection Balancing

- Oracle uses connection load balancing
  - All connection balancing activity is performed before a session establishes a connection
  - Once a connection is established with a specific instance, session will communicate with same instance until
    - Session terminates
    - Instance terminates
  - Establishing a connection is relatively expensive
    - System resources
    - CPU
  - Understand how you application uses connections

Oracle uses connection load balancing. All connection balancing activity is performed before a session establishes a connection. Once a connection is established with a specific instance, session will communicate with same instance until either the session terminates or the instance terminates.

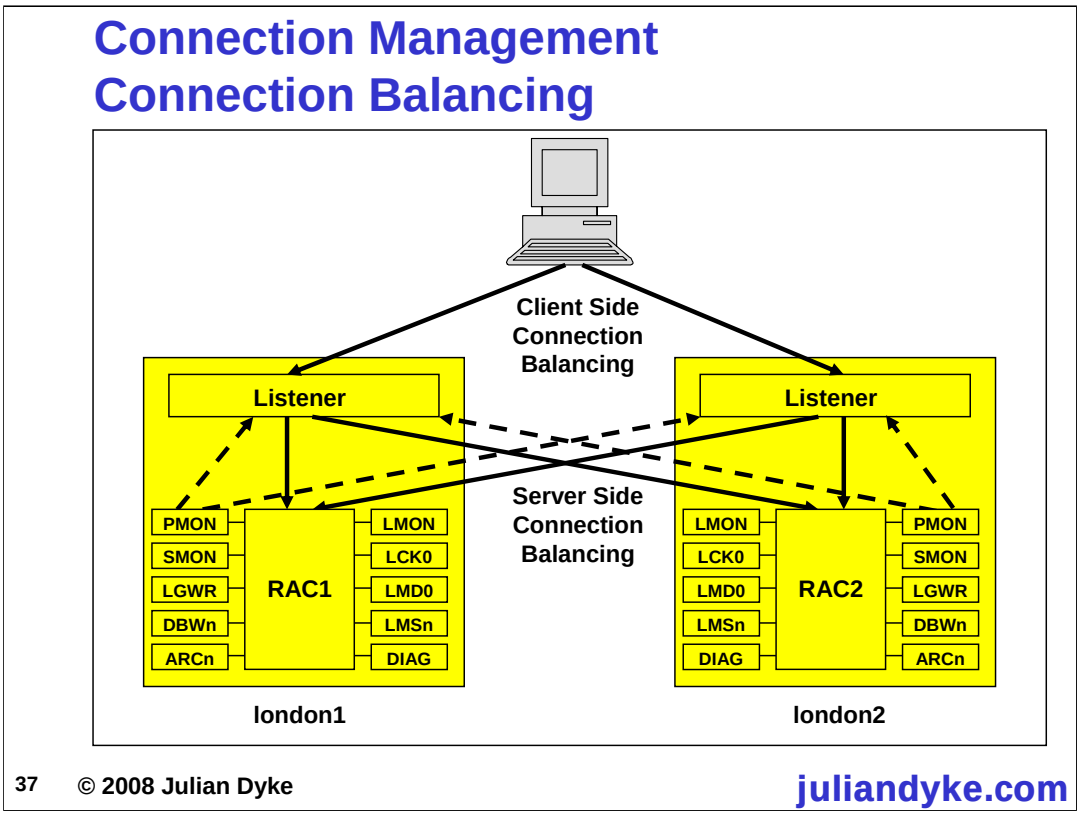
Establishing a connection is relatively expensive in terms of system resources and CPU, so it is important to understand how your application uses connections.

Connection Management
Connection Balancing

- Connection load balancing can be performed on
  - client-side
  - server-side
  - both client-side and server-side

Connection load balancing can be performed on the client-side, the server-side or both.

The following slides discuss client-side and server-side connection load balancing.



This slide shows the difference between client side and server side connection balancing.

Client-side connection balancing is entirely performed within the client and is a function of Oracle Net Services. The client selects a listener process at random from the list of available listener processes in the local TNSNAMES.ORA file or in the connection string. In the above example, on average 50% of connection requests will be sent to the listener process on node london1 and the other 50% of connection requests will be sent to the listener process on node london2.

Server side load balancing is performed within the listener process. The PMON background process informs all listeners in the cluster of the current workload in the instance. The PMON background processes sends these messages to the listener on the same node which is specified in the LOCAL_LISTENER parameter and to all listeners on other nodes in the cluster which are specified in the REMOTE_LISTENER parameter. The listeners use the data supplied by the PMON background process in conjunction with the listener configuration (Oracle 10.1 and below) or the database service configuration (Oracle 10.2 and above) to determine which instance the connection should be directed to.

Connection Management
Client-Side Connection Balancing

- Client-side connection balancing:
  - configured in clients local TNSNAMES.ORA file
  - alternatively configured in connection string
  - uses LOAD_BALANCE parameter
  - effectively selects a random instance
  - does not use load metrics

Client-side connection balancing works well if all instances are registered. Prior to Oracle 10.1, client-side connection balancing did not detect blocked, failed, or hung instances, which could lead to skewed connection loads on each node. In Oracle 10.1, the combination of Virtual IP (VIP) and Fast Application Notification (FAN) can be used to detect failed instances, resulting in a much more even distribution of connections across the nodes

Client-side connection balancing is unaware of the existing load on the nodes and therefore does not take account of how many other clients are currently connected to the node or of the current workload on the node.

Another issue with client-side connection balancing is that the random algorithm is based on time slices that are relatively long. Therefore, if a large number of clients are attempting to connect concurrently, they may all be directed to the same node. This issue is particularly prevalent during scalability testing, where care must be taken to design tests that ramp up over a suitably long time period to ensure a realistic distribution of connections across all nodes in the cluster. Consequently, client-side connection balancing may be unsuitable for applications that add connections in batches.

Connection Management
Client-Side Connection Balancing

- Example of a service description
  - $ORACLE_HOME/network/admin/tnsnames.ora

```
SERVICE1 =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = server6-vip)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = server7-vip)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = server8-vip)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = server9-vip)(PORT = 1521))
        (LOAD_BALANCE = YES)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = SERVICE1)
        )
    )
```


This slide shows an example of client-side load balancing. In this example, the client can choose a listener process on any of four nodes.

The LOAD_BALANCE = YES parameter specifies that a listener should be chosen at random from the list of available addresses.

The LOAD_BALANCE parameter can also be set to ON or TRUE which both have the same effect as YES.

If the LOAD_BALANCE parameter is set to NO (OFF or FALSE), then the list of addresses will be processed in the order that they are listed in the net service name descriptor.

Alternatively you can specify an address list e.g.

```
SERVICE1 =
    (DESCRIPTION =
        (ADDRESS_LIST=
            (ADDRESS = (PROTOCOL = TCP)(HOST = server6-vip)(PORT = 1521))
            (ADDRESS = (PROTOCOL = TCP)(HOST = server7-vip)(PORT = 1521))
            (ADDRESS = (PROTOCOL = TCP)(HOST = server8-vip)(PORT = 1521))
            (ADDRESS = (PROTOCOL = TCP)(HOST = server9-vip)(PORT = 1521))
        )
        (LOAD_BALANCE = YES)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = SERVICE1)
        )
    )
```

Specifying an ADDRESS_LIST is equivalent to specifying FAILOVER=ON

Connection Management
Server-side Connection Balancing

- Server-side connection balancing
  - Available in Oracle 9.0.1 and above
  - Significantly enhanced in Oracle 10.2
- Implemented by listener processes
  - Have information on blocked, failed or hung instances
  - Have current loading and session status for each instance
- Server-side load balancing can be configured for
  - short connections - e.g. internet queries
  - long connections - e.g. connection pools

Server-side load balancing is performed by listener processes on each node. The current load on each instance is updated by the PMON background process

Well-designed applications should connect once and remain connected because of the high cost of establishing a connection. This behavior is typical of application designs that implement middle-tier connection pooling. However, many applications still in use connect, execute a single statement or transaction, and then disconnect again.

In Oracle 10.2 server-side load balancing is incorrectly configured by DBCA.

Connection Management
Server-side Connection Balancing

- In Oracle 10.1 and below
  - driven by metrics
  - possible metrics are
    - workload per node (CPU / run queue length)
    - number of current connections
- Instances register and report loads for server processes
- Metric used depends on anticipated lifetime of connection
  - Short connections should be directed to least loaded instance
  - Long connections should be distributed evenly across all instances

In server-side connection balancing, the TNS listener process performs intelligent load balancing when establishing connections between clients and servers. The database registers and reports load for both dedicated servers and shared servers

The metric used to assign connections to instances should not vary over the lifetime of the connection. If the connection is expected to persist for a relatively long time, the current workload on the node is not a good metric, as the workload could be highly variable over the lifetime of the connection. Therefore, the connections should be distributed evenly across all nodes.

Connection Management
Server-side Connection Balancing

- In Oracle 10.1 and below
  - Connection persistence is configured at listener level
  - Specified in $TNS_ADMIN/listener.ora
  - For long connections specify:

```
PREFER_LEAST_LOADED_NODE = OFF
```

  - For short connections optionally specify:

```
PREFER_LEAST_LOADED_NODE = ON
```

  - This is the default
  - Minimizes impact of logon storms

If each connection is expected to be long, then distribute the workload across all available nodes by setting the PREFER_LEAST_LOADED_NODE parameter to OFF in $TNS_ADMIN/listener.ora. However, if each connection is expected to be relatively short, you can send it to the least loaded node. by setting the PREFER_LEAST_LOADED_NODE parameter to ON which is the default value.

Logon storms occur when large numbers of clients attempt to establish connections over a very short period of time. If a system is subject to logon storms, then connecting to the least loaded node is not appropriate, because an interval of time is required to assess system load and communicate it back to the listener process. During this interval, the state of the database may have changed due to newly connected sessions

Do not attempt to configure PREFER_LEAST_LOADED_NODE=OFF if you are setting a load balancing goal in Oracle 10.2 and above.

Connection Management
Server-side Connection Balancing

- In Oracle 10.1 and below:
  - Database metrics reported by PMON background process
  - Sends updated SERVICE_REGISTER information to listener including
    - load
    - maximum load
    - instance load
    - maximum load per instance
  - PMON background process
    - inspects database metrics every three seconds
    - only posts listener process when there has been a significant change
    - checks for shutdown listeners every minute

Database metrics are reported by the PMON background process, which sends updated SERVICE_REGISTER information to the listener including the load, maximum load, instance load, and maximum load for the instance. PMON inspects the database metrics once every three seconds; however, it posts the listener process only when there has been a significant change. If a listener is shut down or fails, PMON checks whether it has been restarted once per minute.

Because the PMON process can register with remote listeners, each listener process is always aware of all instances and dispatchers, irrespective of the node on which they are running. The listener uses the load information to determine which instance to establish a connection with and, in the case of shared servers (MTS), which dispatcher to assign to the client.

If a database service has multiple instances on multiple nodes, the listener selects the least loaded instance on the least loaded node. If a shared server is configured, then the least loaded dispatcher of the selected instance is chosen. In addition, if shared servers are configured, all dispatchers on each instance must be cross-registered with the other listeners on the other nodes using the LISTENER attribute of the DISPATCHERS parameter.

Connection Management
Server-side Connection Balancing

- In Oracle 10.2 and above:
  - Server-side connection balancing enhanced
  - Mechanism by which listener informed of current workload for each node improved
- Load balancing advisory
  - supplies information to OCI, ODP.NET and JDBC connection pools
  - third-party applications can subscribe to load balancing advisory using Oracle Notification Service (ONS)
  - monitors current workload for each service in instance and creates FAN events
- FAN events
  - sent to connection pools for change of configuration or status of any instance providing a service

In Oracle 10.2 and above, server-side load balancing has been enhanced. As in Oracle 10.1, the listener process distributes connection requests among active instances based on the current workload of each node and instance in the cluster. However, the mechanism by which the listener process is informed of the current workload for each node has been improved.

In Oracle 10.2 and above, RAC includes a load balancing advisory which can supply information to OCI, ODP.NET, and JDBC connection pools. Third-party applications can subscribe to the load balancing advisory using the Oracle Notification Service (ONS). The load balancing advisory can be used in a number of standard architectures where work is allocated including connection load balancing, transaction processing monitors, application servers, connection concentrators, hardware and software load balancers, job schedulers, batch schedulers, and message queuing systems. Load balancing advisory events include the current service level that an instance is providing for a service and a recommendation of how much of the workload should be sent to that instance.

Connection Management
Server-side Connection Balancing

- In Oracle 10.2 and above:
  - Load balancing advisory events include:
    - current services level instance is providing for database service
    - recommendation of how much of workload should be sent to instance
  - For each service you can specify
    - Load balancing goal
    - Connection load balancing goal
  - Specify load balancing goals using
    - MODIFY_SERVICE procedure in DBMS_SERVICE
    - Enterprise Manager

The load balancing advisory monitors the current workload for each service in the instance and creates FAN events, which are sent to the connection pools whenever there is a change of configuration or status of one of the instances providing the service. This allows the connection pools to select the most appropriate instance when creating new connections.

Connection Management
Server-side Connection Balancing

- In Oracle 10.2 and above:
  - Load balancing goal can be
    - GOAL_NONE - disable load balancing advisory
    - GOAL_SERVICE_TIME - load balancing advisory based on elapsed time for work done for service plus available bandwidth to service
      - Workload sent to least-loaded node
    - GOAL_THROUGHPUT - load balancing advisory based on rate work completed in service plus available bandwidth to service
      - Workload sent to fastest node
      - Useful where some nodes are faster than others

The load balancing advisory is configured by specifying a load balancing goal when creating or modifying a service using the DBMS_SERVICE package or in Enterprise Manager (EM). The load balancing goal can take the value of NONE, SERVICE_TIME, or THROUGHPUT.

If the goal is SERVICE_TIME then requests will be directed to the least loaded node. In theory this is the node that is able to service the request fastest.

If the goal is THROUGHPUT then requests will be redirected to the node with the highest numbers of calls per second. In theory this will be the node able to perform the highest transaction rate. For example the target node may have faster processors or more memory than others in the cluster.

Connection Management
Server-side Connection Balancing

- In Oracle 10.2 and above:
  - Connection load balancing goal can be:
    - CLB_GOAL_LONG - balances number of connections per instance using session count per service
    - CLB_GOAL_SHORT - uses load balancing advisory if goal is GOAL_SERVICE_TIME or GOAL_THROUGHPUT
  - CLB_GOAL_LONG - should be specified for applications with connections of long duration
    - Connection pools
    - Most OLTP and batch applications
    - Default connection load balancing goal
  - CLB_GOAL_SHORT - should be specified for applications with connections of short duration
    - Web applications not using connection pool

The LONG connection load balancing goal should be specified for applications with connections of a long duration. This would include connection pools and also most OLTP and batch applications. LONG is the default connection load balancing goal.

The SHORT connection load balancing goal should be specified for applications with connections of a short duration, such as web applications that do not use a connection pool.

You can set the load balancing goal and the connection load balancing goal in the CREATE_SERVICE and MODIFY SERVICE procedures in the DBMS_SERVICE package. When you create a database using DBCA, server-side load balancing is configured and enabled by default. Services created using DBCA have the default settings of GOAL=NONE and CLB_GOAL=CLB_GOAL_LONG. These are also the default values if you create the service manually using the DBMS_SERVICE package.

Connection Management
Server-side Connection Balancing

- To modify a service use DBMS_SERVICE e.g.:

```
EXECUTE dbms_service.modify_service -
( -
     service_name => 'SERVICE1', -
     goal=> DBMS_SERVICE.GOAL_THROUGHPUT, -
     clb_goal => DBMS_SERVICE.CLB_GOAL_LONG -
);
```

- Current values can be checked in DBA_SERVICES e.g.:

```
SELECT name, goal, clb_goal FROM dba_services;
```

- Also in V$SERVICES and V$ACTIVE_SERVICES:

```
SELECT name, goal, clb_goal FROM v$services;
```


You can verify the current value of the connection load balancing goal by querying the DBA_SERVICES table, for example:

SQL> SELECT name, goal, clb_goal FROM dba_services;

You can also verify the current load balancing goal settings for each service in the V$SERVICES and V$ACTIVE_SERVICES dynamic performance views.

Connection Management
Server-side Connection Balancing

- On each node configure listener names in
  - $ORACLE_HOME/network/admin/tnsnames.ora

```
LISTENER_LONDON1 =
  (ADDRESS = (PROTOCOL = TCP)(HOST = london1-vip)(PORT = 1521))

LISTENER_LONDON2 =
  (ADDRESS = (PROTOCOL = TCP)(HOST = london2-vip)(PORT = 1521))

LISTENER_LONDON3 =
  (ADDRESS = (PROTOCOL = TCP)(HOST = london3-vip)(PORT = 1521))

LISTENER_LONDON4 =
  (ADDRESS = (PROTOCOL = TCP)(HOST = london4-vip)(PORT = 1521))

LISTENERS_RAC =
 (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = london1-vip)(PORT = 1521))
   (ADDRESS = (PROTOCOL = TCP)(HOST = london2-vip)(PORT = 1521))
   (ADDRESS = (PROTOCOL = TCP)(HOST = london3-vip)(PORT = 1521))
   (ADDRESS = (PROTOCOL = TCP)(HOST = london4-vip)(PORT = 1521))
 )
```


In order for workload balancing to work correctly with database services, it is necessary to specify the LOCAL_LISTENER and REMOTE_LISTENER parameters for each instance in the cluster.

In the above slide one alias has been added for each local listener (LISTENER_LONDON1 etc). This is not strictly necessary as the full description can be used directly in the LOCAL_LISTENER parameter.

Even when the TNSNAMES.ORA file is used, only the local listener alias for the local node is actually required.

However, in my opinion, it is easier to specify all four listeners so that the TNSNAMES.ORA file is identical on all nodes and can therefore be copied between nodes without modification.

The LISTENERS_RAC service is used by the REMOTE_LISTENER parameter and is identical on all nodes in the cluster.

Connection Management
Server-side Connection Balancing

- The following commands update the LOCAL_LISTENER parameters for the new service:

```
ALTER SYSTEM SET local_listener = LISTENER_LONDON1
SCOPE=SPFILE SID='RAC1';

ALTER SYSTEM SET local_listener = LISTENER_LONDON2
SCOPE=SPFILE SID='RAC2';

ALTER SYSTEM SET local_listener = LISTENER_LONDON3
SCOPE=SPFILE SID='RAC3';

ALTER SYSTEM SET local_listener = LISTENER_LONDON4
SCOPE=SPFILE SID='RAC4';
```

- The following commands update the REMOTE_LISTENER parameters for the new service:

```
ALTER SYSTEM SET remote_listener = LISTENERS_RAC
SCOPE=SPFILE;
```


In the above slide one alias has been added for each local listener (LISTENER_LONDON1 etc). This is not strictly necessary as the full description can be used directly in the LOCAL_LISTENER parameter.

Even when the TNSNAMES.ORA file is used, only the local listener alias for the local node is actually required.

However, in my opinion, it is easier to specify all four listeners so that the TNSNAMES.ORA file is identical on all nodes and can therefore be copied between nodes without modification.

The LISTENERS_RAC service is used by the REMOTE_LISTENER parameter and is identical on all nodes in the cluster.

Transparent Application Failover (TAF) Overview

- Transparent Application Failover (TAF)
  - Feature of OCI driver
  - If connection fails
    - automatically reconnects to another instance for same database
    - active transactions rolled back
    - new database connection is identical to original
  - Introduced in Oracle 8.1.5
  - Particularly applicable in RAC environment
  - Works with Java and C
  - Must use OCI library
  - Configured in TNSNAMES.ORA

Transparent Application Failover (TAF) is a feature that was introduced in Oracle 8.1.5. In the event of an instance failure, TAF allows applications to automatically reconnect to another instance. The new connection will be identical to the original. However, any uncommitted transactions existing at the time of failure will be rolled back.

Transparent Application Failover (TAF) Basic Configuration

- **Basic client configuration generated by DBCA**

```
# TAF BASIC
SERVICE1 =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = london1-vip)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = london2-vip)(PORT = 1521))
        (LOAD_BALANCE = yes)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = SERVICE1)
            (FAILOVER_MODE =
                (TYPE = SELECT)
                (METHOD = BASIC)
                (RETRIES = 180)
                (DELAY = 5)
            )
        )
    )
```


A TAF connection must be configured in the tnsnames.ora Oracle Net configuration file. This located in $ORACLE_HOME/network/admin, unless this location has been overridden by the $TNS_ADMIN environment variable. In Oracle 10.1 and above, the Service Management page of DBCA can generate appropriate TNS entries for TAF.

Transparent Application Failover (TAF) FAILOVER_MODE Clause

- Defines failover properties
- Includes the following attributes:
  - TYPE - defines behaviour following a failure. Can be SESSION, SELECT or NONE
  - METHOD - defines when connections are made to the failover instance. Can be BASIC or PRECONNECT
  - RETRIES - specifies number of times a connection should be attempted before returning an error
  - DELAY - specifies time in seconds between each connection retry

In order to configure TAF, the TNS entry must include a FAILOVER MODE clause that defines the properties of the failover as follows:

• The TYPE attribute defines the behavior following a failure. It can take the values SESSION, SELECT, or NONE.

•The METHOD attribute defines when connections are made to the failover instance. It can take the value BASIC or PRECONNECT.

• The RETRIES attribute specifies the number of times that a connection should be attempted before returning an error.

• The DELAY attribute specifies the time in seconds between each connection retry.

In addition, if the METHOD attribute is PRECONNECT, then you can also specify the BACKUP attribute, which defines the service that should be used for secondary passive connections.

Transparent Application Failover (TAF) TYPE Attribute

- Defines behaviour following a failure
  - Takes values SESSION (default), SELECT or NONE
  - SESSION (default)
    - Any uncommitted transactions are rolled back
    - Session is connected to another instance
  - SELECT
    - Any uncommitted transactions are rolled back
    - Session is connected to another instance
    - Any SELECT statement executing at time of failure will resume on new instance at point of failure
    - SELECT statement continues to return remaining rows to client

The TYPE attribute defines the behaviour of TAF following a failure. It takes the values SESSION (default), SELECT or NONE.

If the value is SESSION which is the default, then in the event of a failure, any active transactions on the failed instance are rolled back and the session reconnects to another instance.

If the value is SELECT, then in the event of a failure, any active transactions on the failed instance are rolled back and the session reconnects to another instance. Any SELECT statement that was executing at the time of failure will resume on the new instance at the point of failure. When the TYPE attribute is set to SELECT, every Oracle Net client records the number of rows fetched from each cursor. This adds a minor CPU overhead to the cost of each fetch operation. In the event of a node failure the SELECT statement is re-executed on the new instance using the same SCN. Any rows already fetched on the previous instance are discarded and the SELECT statement resumes fetching rows at the point at which the failure occurred.

Note that enabling this attribute can put additional stress on the new instance at the point of failover, especially of the SELECT statement was resource intensive (such as performing a large sort or hash join) or had already returned a large number of rows.

Transparent Application Failover (TAF) Method Attribute

- Can be BASIC or PRECONNECT
  - BASIC
    - Basic connections are performed at time of failover
    - Can be highly resource intensive
    - Impacts performance during failover
  - PRECONNECT
    - Sessions are pre-connected to failover instance
    - Reduces impact on performance of failover
    - Requires additional sessions on failover instance

BASIC - Connection to backup node established at failover time. If there are a large number of processes to failover, this can cause a long delay while the new sessions are established

PRECONNECT - Connection to backup node is established at same time that original connection is established. Enables faster failover, but requires that backup node can support all connections concurrently

Failing over a large number of connections consumes a significant amount of resources. If the cluster is already busy, then failing over all the connections from a failed node will affect sessions currently connected on the remaining nodes. If impacting existing sessions is not acceptable, then it is possible to pre-connect sessions to a failover node. Pre-connecting sessions reduces resource consumption at failover time, but clearly requires additional resources, in the form of sessions and processes, to be permanently allocated to the backup nodes.

Transparent Application Failover (TAF) Preconnect Configuration

- **Preconnect client configuration generated by DBCA**

```
# TAF- PRECONNECT
SERVICE2 =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = london1-vip)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = london2-vip)(PORT = 1521))
        (LOAD_BALANCE = yes)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = SERVICE2)
            (FAILOVER_MODE =
                (BACKUP = SERVICE2_PRECONNECT)
                (TYPE = SELECT)
                (METHOD = PRECONNECT)
                (RETRIES = 180)
                (DELAY = 5)
            )
        )
    )
```


In order to pre-connect sessions to a secondary node, you must specify PRECONNECT for the METHOD attribute. In Oracle 10.1 and above, you can also use DBCA to generate TNS entries for TAF pre-connection. Two entries will be generated: one for the primary service and another for the pre-connect service.

The above slide shows the TNS entry automatically generated by DBCA for SERVICE2. The following slide shows the TNS entry automatically generated by DBCA for SERVICE2_PRECONNECT.

Transparent Application Failover (TAF) Preconnect Configuration

- **Preconnect client configuration generated by DBCA (continued)**

```
# TAF- PRECONNECT
SERVICE2_PRECONNECT =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = london1-vip)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = london2-vip)(PORT = 1521))
        (LOAD_BALANCE = yes)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = SERVICE2_PRECONNECT)
            (FAILOVER_MODE =
                (BACKUP = SERVICE2)
                (TYPE = SELECT)
                (METHOD = BASIC)
                (RETRIES = 180)
                (DELAY = 5)
            )
        )
    )
```


Note, however, that the pre-connect service defines a FAILOVER_MODE METHOD of BASIC. The introduction of virtual IP addresses in Oracle 10.1 simplifies the configuration of pre-connections, as you can specify all nodes in the cluster for each service. Oracle will manage the assignment of sessions to each node.

Transparent Application Failover (TAF)
JDBC OracleOCIFailover Interface

```
public interface OracleOCIFailover
{
    // Failover Types
    public static final int FO_SESSION = 1;
    public static final int FO_SELECT = 2;
    public static final int FO_NONE = 3;

    // Failover Events
    public static final int FO_BEGIN = 1;
    public static final int FO_END = 2;
    public static final int FO_ABORT = 3;
    public static final int FO_REAUTH = 4;
    public static final int FO_ERROR = 5;
    public static final int FO_RETRY = 6;

    public int callbackFn
    (
        Connection connection,
        Object ctxt,     // Anything the user wants to save
        int type,        // One of the above possible Failover types
        int event        // One of the above possible Failover events
    );
}
```


TAF works with both Java and C applications. In order to use TAF, the client must be using the OCI library, so if you are using JDBC, you will need to connect using the OCI thick client.

A special interface called OracleOCIFailover defines the constants and callback method that should be used in JDBC TAF programs.

Transparent Application Failover (TAF) Failover Types

- **The following types of failover are defined:**
  - **FO_SESSION - Only the user session is reauthenticated on the server-side while open cursors in the OCI application must be re-executed - equivalent to FAILOVER_MODE=SESSION in tnsnames.ora.**
  - **FO_SELECT - The user session is reauthenticated on the server-side and open cursors in the OCI can continue fetching. Client-side maintains fetch-state for each open cursor - equivalent to FAILOVER_MODE=SELECT in tnsnames.ora.**
  - **FO_NONE - Default. No failover functionality enabled. Can also be explicitly specified to prevent failover from occuring - equivalent to FAILOVER_MODE=NONE in tnsnames.ora**

The OracleOCIFailover interface defines the following types of failover:

• FO_SESSION: The user-session will be re-authenticated by the server. Any open cursors in the client must be re-executed by the application. FO_SESSION is equivalent to FAILOVER_MODE=SESSION in tnsnames.ora.

• FO_SELECT: The user session will be re-authenticated by the server. Any cursors open for SELECT statements in the client can continue fetching. The client side maintains the fetch state for each open customer. FO_SELECT is equivalent to FAILOVER_MODE=SELECT in tnsnames.ora.

• FO_NONE: Disables all failover functionality. FO_NONE is the default failover type and is equivalent to FAILOVER_MODE = NONE. You might wish to specify this value to explicitly prevent failover from occurring.

• FO_TYPE_UNKNOWN - invalid failover type returned from OCI driver

Transparent Application Failover (TAF) Failover Events

- The following failover events are defined:
  - FO_BEGIN - indicates failover has detected a lost connection and failover is starting
  - FO_END - indicates successful completion of failover
  - FO_ABORT - indicates failover was unsuccessful and there is no option of retrying
  - FO_REAUTH - indicates that a user-handle has been reauthenticated
  - FO_ERROR - indicates that failover was temporarily unsuccessful, but it gives the application the opportunity to handle the error and retry failover The usual method of error handing is to issue sleep () method and retry by returning the value FO_RETRY
  - FO_RETRY - see above

The OracleOCIFailover interface defines the following failover events:

• FO_BEGIN: Indicates that failover has detected a lost connection and that failover is starting.

• FO_END: Indicates that failover has completed successfully.

• FO_ABORT: Indicates that failover has completed unsuccessfully and that there is no possibility of retrying.

• FO_REAUTH: Indicates that the user session has been reauthenticated on the failover server.

• FO_ERROR: Indicates that failover was unsuccessful. However, this state might only be temporary. The client has the opportunity to sleep for a suitable period and then retry failover.

• FO_RETRY: This value is returned by the application if it wishes to retry failover following receipt of an FO_ERROR message.

• FO_EVENT_UNKNOWN - invalid failover event

Transparent Application Failover (TAF) TAF Callbacks

- TAF Callbacks
  - callbacks that are registered in case of failover
  - used in event of failure of one database connection and failover to another database connection
  - called during failover to notify JDBC application of events generated
- Application has limited control of failover process

The callbackFn method defines the behavior of the application in the event of a failover. Callbacks are registered in case of a failover and are called during the failover to notify the application of the events that are being generated.

Transparent Application Failover (TAF) Sample Program (1 of 6)

```
import java.sql.*;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.OracleOCIFailover;
import oracle.jdbc.pool.OracleDataSource;

public class TAF1
{
    static final String user = "<user>";
    static final String password = "<password>";
    static final String URL = "jdbc:oracle:oci8:@SERVICE1";

    public static void main (String[] args) throws Exception
    {
        CallBack callback = new CallBack ();
        String msg = null;
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        // Create an OracleDataSource instance and set properties
        OracleDataSource ods = new OracleDataSource ();
        ods.setUser (user);
        ods.setPassword (password);
        ods.setURL (URL);
```


The program on the next six slides assumes that tnsnames.ora contains an entry similar to the one shown above for SERVICE1 on each node in the cluster. The program performs the following steps:

1. Create an OracleDataSource object and initialize the values of the user, password, and URL attributes. Remember to substitute appropriate values for these if you are trying this program on your own system.

2. Create a database connection and register the callback function declared in the CallBack class against the connection.

3. Use the SYS_CONTEXT function to obtain the number of the instance to which the connection has been made.

4.Select rows from the ALL_OBJECTS dictionary view, fetching them one row at a time and pausing for half a second between each row.

When the select statement has been running for a few seconds, shut down the instance on which the statement is running using SHUTDOWN ABORT:

$ sqlplus "/ as sysdba"

SQL> SHUTDOWN ABORT

Alternatively, you can use SRVCTL to shutdown the instance, for example:

$ srvctl stop instance -d RAC -i RAC2 -o abort

The program should pause for a few seconds and then resume returning rows from the point at which it stopped. The length of the interval depends on a number of factors including the number of instances, the size of the buffer cache, and the latency of the interconnect.

Transparent Application Failover (TAF) Sample Program (2 of 6)

```
 // Connect to the database
connection = ods.getConnection ();

// Register TAF callback function
((OracleConnection)connection).registerTAFCallback (callback,msg);

// Create a statement
statement = connection.createStatement ();

// Determine current instance
resultSet = statement.executeQuery
("SELECT SYS_CONTEXT ('USERENV','INSTANCE') FROM dual");


while (resultSet.next ())
{
     // Print current instance
     System.out.println ("Instance = " + resultSet.getInt (1));
}
resultSet.close ();
```


This slide shows the database connection and TAF callback function registration.

The program then determines the current instance using the USERENV context of the SYS_CONTENT built-in function.

Transparent Application Failover (TAF) Sample Program (3 of 6)

```
        // Print each object id and object name for each row in all_objects
        resultSet = statement.executeQuery
        ("SELECT object_id, object_name FROM all_objects ORDER BY object_id");
        while (resultSet.next ())
        {
            System.out.println (resultSet.getInt (1) + " " + resultSet.getString (2));

            // Pause for 0.5 seconds
            Thread.sleep (500);
        }

        // Close the resultset
        resultSet.close ();

        // Close the statement
        statement.close ();

        // Close the connection
        connection.close ();
    }
}
```


On this slide the program executes the following SELECT statement:

```
SELECT object_id, object_name
FROM all_objects
ORDER BY object_id;
```

This statement will return around 50,000 rows in Oracle 10.2.

The program fetches one row at a time from the cursor. In order to demonstrate the effect of using TAF, the program pauses for half a second between each fetch.

Transparent Application Failover (TAF) Sample Program (4 of 6)

```
class CallBack implements OracleOCIFailover
{
    public int callbackFn (Connection connection, Object context, int type, int event)
    {
        String failover_type = null;

        switch (type)
        {
            case FO_SESSION:
                failover_type = "SESSION";
            break;
            case FO_SELECT:
                failover_type = "SELECT";
            break;
            default:
                failover_type = "NONE";
            break;
        }
```


This slide shows the callback function that is called, if registered, in the event of a instance failure. The code required by the application to handle the instance failure should be included in this function.

The switch statement determines the value of the TYPE attribute for the current session - this can be SESSION, SELECT or NONE.

Transparent Application Failover (TAF) Sample Program (5 of 6)

```
switch (event)
{
    case FO_BEGIN:
        System.out.println (context + ": " + failover_type + " failing over");
    break;
    case FO_END:
        System.out.println (context + ": failover ended");
    break;
    case FO_ABORT:
        System.out.println (context + ": failover aborted");
        // Return FO_ABORT to calling method
        return FO_ABORT;
    break;
    case FO_REAUTH:
        System.out.println (context + ": failover reauthorized");
    break;
```


This slide shows the code for the callback function that determines which event has occurred. This can include

• FO_BEGIN - A failover event has started

• FO_END - The failover event has ended

• FO_ABORT - The failover event was aborted. The callback registered should return the FO_ABORT event if the FO_ERROR event is passed to it

• FO_REAUTH - The failover has been reauthorized

• FO_ERROR - Failover is not yet complete.

Transparent Application Failover (TAF) Sample Program (6 of 6)

```
            case FO_ERROR:
            {
                System.out.println (context + ": failover error. Sleeping...");
                try
                {
                    Thread.sleep (100);
                }
                catch (InterruptedException e)
                {
                    System.out.println ("Failed to sleep");
                    System.out.println (e.toString ());
                }
                // Return FO_RETRY to calling method
                return FO_RETRY;
            }
            default:
                System.out.println (context + " invalid failover event");
            break;
        }
        return 0;
    }
}
```


Handling the FO_ERROR event - In case of an error while failing over to a new connection, the JDBC application is able to retry failover. Typically, the application sleeps for a while and then it retries, either indefinitely or for a limited amount of time, by having the callback function return FO_RETRY

Transparent Application Failover Server-side TAF

- In Oracle 10.2 (?) and above it is possible to configure server-side TAF
  - TAF can be centrally administered
  - Configuration stored in SYS.SERVICE$
  - Configuration reported in DBA_SERVICES
  - No changes to connection string required
  - No changes to TNSNAMES.ORA required
- For example

```
dbms_service.modify_service
(
     service_name=>'SERVICE2',
     failover_method=>DBMS_SERVICE.FAILOVER_METHOD_BASIC,
     failover_type=>DBMS_SERVICE.FAILOVER_TYPE_SELECT,
     failover_retries=>5,
     failover_delay=>10
);
```


failover_method can be:

• FAILOVER_METHOD_NONE

• FAILOVER_METHOD_BASIC - Basic server-side TAF supported.

It is not possible to specify pre-connection for server-side TAF

failover_type can be:

• FAILOVER_TYPE_NONE

• FAILOVER_TYPE_SESSION

• FAILOVER_TYPE_SELECT

failover_retries should be integer value specifying the number of times that TAF should attempt the re-connect and re-authenticate.

failover_delay should be integer value specifying the number of seconds that TAF will wait if the re-connect / re-authentication fails.

Transparent Application Failover (TAF) Summary

- TAF only works in very limited circumstances
  - Application is read-only
  - Application can store and re-execute all DML statements executed within each transaction
- TAF does not restore values for the following:
  - Session parameters
  - PL/SQL package variables
  - User-defined type variables
- If any of the above are necessary, then the application must restore the values following a failover

If you are considering using TAF, you should remember that it is not suited to all applications. In fact, it only applies in very limited circumstances. While TAF works well for read-only applications, it presents some issues for applications that modify the database. In the event of a failure, any uncommitted transactions will be rolled back. Therefore, the application must be capable of detecting the failure and, if necessary, reapplying DML statements up to the point of failure. If this capability is a requirement, then the application must be capable of recording all statements issued in a transaction together with the values of any bind variables.

Note also that, while TAF can reauthorize a session, it does not restore the session to its previous state. Therefore, following an instance failure, you will need to restore any session variables, PL/SQL package variables, and instances of user-defined types. TAF will not restore these values automatically, so you will need to extend your applications to restore them manually.

For the reasons discussed previously, you may find that the areas within your applications that can take advantage of TAF are fairly limited. In Oracle 10.1 and above, there has been some de-emphasizing of the capabilities of TAF in favor of FAN and, specifically for JDBC applications, Fast Connection Failover.

Fast Application Notification (FAN) Introduction

- Introduced in Oracle 10.1
- Enables
  - Automatic recovery of applications
  - Load balancing when cluster configuration changes
- Avoids
  - Applications waiting for TCP/IP timeouts when node fails without closing sockets
  - Applications attempting to connect to services which are down
  - Applications not connecting when services resume
  - Applications processing last result after node, instance or service has gone down
  - Applications not balancing across available systems when services restart or expand

In Oracle 10.1 and above, RAC includes the High Availability (HA) Application Framework. This provides service and integration points between RAC and application. The framework is able to receive fast notification of events affecting critical system components. This allows applications to execute error-handling programs which

• minimizes impact of component failures

• avoids connection and application time-outs

• handles both planned and unplanned cluster reorganizations

Fast Application Notification (FAN) FAN Events

- Objective is to deliver FAN events so they precede regular connection timeouts or typical polling intervals
- FAN events
  - Use database services
  - Posted immediately a state change occurs
  - Require minimal communication channel overhead
  - Are processed immediately by application tier
    - Application can react immediately
    - Application does not need to poll database tier
  - Can be processed by Listener and Connection Manager (CMAN)

FAN events are processed by Oracle Net Services Listeners and Connection Manager (CMAN) in Oracle 10.1.0.3 and above

Event recipient can take appropriate action such as:

• shutdown application connection manager

• reroute existing database connection requests

• refresh stale connection references

• log trouble tickets

• page database administrator

Fast Application Notification (FAN) FAN Events

- There are two categories of FAN events
  - Service events
    - Include application services and database services
      - Service events (including pre-connections)
      - Database events
      - Instance events
      - ASM events
  - Node events
    - Include cluster membership states and node join/leave operations

For up events

• New connections are created so application can immediately use new services and instances

For down events

• Disruption to application can be minimized

• Connections to failed instance or node can be terminated

• Uncommitted transactions are terminated and users immediately notified

• Application users requesting connections are directed to available instances only

• Server side callouts can log tickets or page administrators

Fast Application Notification (FAN) Uses

- FAN events can be used in 3 ways:
  - Using Oracle 10g JDBC Fast Connection Failover feature of Implicit Connection Cache on the application tier
  - Using Oracle Notification Service Application Programming Interface (ONS API) to subscribe to FAN events and execute event-handling actions
  - Using FAN server side callouts on the database tier

FAN events can be used in three ways:

• Application can use FAN without any programmatic changes using Oracle 10g JDBC Fast Connection Failover feature of Implicit Connection Cache on Application Tier

• Application can use FAN programmatically using Oracle Notification Service Application Programming Interface (ONS API) to subscribe to FAN events and execute event-handling actions

• FAN server side callouts can be implemented on the database tier

Fast Application Notification (FAN)
FAN Event Structure

- FAN Events
  - Consist of
    - Header
    - Payload
      - Set of name-value pairs
  - Payload structure is as follows:

```
<Event_Type> VERSION=<n.n>
service=<serviceName.dbDomainName>
[database=<db_unique_name> [instance=<instance_name>]]
[host=<hostname>]
status=<Event_Status>
reason=<Event_Reason> [card=<n>]
timestamp=<eventDate> <eventTime>
```

Fast Application Notification (FAN)
FAN Event Structure

- **Event Payload Descriptors**

Event Resource Identifier	Description
<event_type>	See next slide
VERSION=<n.n>	Event payload version (currently VERSION 1.0)
service=<service_name.domain_name>	Name of primary or preconnect service (not for NODE events)
database=<db_unique_name>	Name of RAC database (not for NODE events)
instance=<instance_name>	Name of RAC instance (not for SERVICE_DATABASE and NODE events)
host=<host_name>	Name of cluster node (not for SERVICE and DATABASE events)
status=<event_status>	See following slide
reason=<event_reason>	See following slide
card=<n>	Service membership cardinality (only for SERVICE status=up)
timestamp=<eventDate> <eventTime>	Server-side data and time when event was detected

Fast Application Notification (FAN)
FAN Event Structure

- **FAN Event Types**

Event Type	Description
SERVICE	Primary application service event
SRV_PRECONNECT	Preconnect application service event
SERVICEMEMBER	Application service on a specific instance event
DATABASE	Oracle Database Event
INSTANCE	Oracle Instance Event
ASM	Oracle ASM Instance Event
NODE	Oracle Cluster Node Event

- **FAN Event Status Descriptions**

Event Status	Description
status=up	Managed resource up
status=down	Managed resource down
status=preconn_up	Preconnect application service up
status=preconn_down	Preconnect application service down
status=nodedown	Managed node down
status=not-restarting	Managed resource cannot fall over to a remote node
status=unknown	Unrecognized status

SRV_PRECONNECT- mid-tier and TAF using primary and secondary instances

Fast Application Notification (FAN)
FAN Event Structure

- **FAN Event Reasons**

Event Status	Activity Type	Description
reason=user	Planned	User-initiated commands such as srvctl and sqlplus
reason=failure	Unplanned	Managed resource polling checks detecting a failure
reason=dependency	Unplanned	Dependency of another managed resource that triggered a failure condition
reason=unknown	Unhandled	Unknown or internal application state when event is triggered
reason=autostart	CRS boot	Initial cluster boot (Managed resource has profile attribute AUTO_START=1 and was offline before the last CRS shutdown)
reason=boot	CRS boot	Initial cluster boot (Managed resource was running before the last CRS shutdown)

Fast Application Notification (FAN) Diagnostics

- To enable diagnostics for node events
  - Set and export the parameter _USR_ORA_DEBUG in $ORA_CRS_HOME/bin/racgwrap

```
_USR_ORA_DEBUG=1 & export _USR_ORA_DEBUG
```

  - Additional diagnostic output will be written to files in
    - $ORA_CRS_HOME/log/<node>/racg
- To enable diagnostics for database and service events
  - Set and export the parameter _USR_ORA_DEBUG in $ORACLE_HOME/bin/racgwrap

```
_USR_ORA_DEBUG=1 & export _USR_ORA_DEBUG
```

  - Additional diagnostic output will be written to files in
    - $ORACLE_HOME/log/<node>/racg

In Oracle 10.1, FAN events for nodes are reported to:

$ORA_CRS_HOME/racg/dump

In Oracle 10.1, FAN events for services and databases are reported to:

$ORACLE_HOME/racg/dump

Fast Application Notification (FAN) Diagnostics

- **evmwatch is a utility that displays events as they are generated by Oracle Clusterware**
- **For example:**

```
evmwatch -A -t "@timestamp @@"

"28-Dec-2007 19:32:00CRS ora.TEST.SERVICE1.cs
is transitioning from state OFFLINE to state ONLINE on member server7"

"28-Dec-2007 19:33:26CRS ora.TEST.SERVICE1.TEST1.srv
is transitioning from state OFFLINE to state ONLINE on member server6"

"28-Dec-2007 19:33:26CRS ora.TEST.SERVICE1.TEST2.srv
is transitioning from state OFFLINE to state ONLINE on member server7"

"28-Dec-2007 19:32:00RAC: ora.TEST.SERVICE1.TEST1.srv: up: "

"28-Dec-2007 19:33:26RAC: ora.TEST.SERVICE1.cs: up: "

"28-Dec-2007 19:32:00CRS ora.TEST.SERVICE1.cs started on member server7"

"28-Dec-2007 19:32:00CRS ora.TEST.SERVICE1.TEST1.srv was modified"

"28-Dec-2007 19:33:26CRS ora.TEST.SERVICE1.TEST1.srv started on member server6"

"28-Dec-2007 19:33:27RAC: ora.TEST.SERVICE1.TEST2.srv: up: "

"28-Dec-2007 19:33:27CRS ora.TEST.SERVICE1.TEST2.srv was modified"

"28-Dec-2007 19:33:27CRS ora.TEST.SERVICE1.TEST2.srv started on member server7"
```


The above example shows the output generated by:

srvctl start service -d TEST -s service1

You can test the event mechanism using the evmpost command. For example in one session run:

evmwatch -A -t "@timestamp@@"

In another session execute the following command:

evmpost -u "This is a test"

The following output will be displayed by evmwatch (with a different timestamp of course)

"28-Dec-2007 21:26:10EVM user msg (root): This is a test"

Both evmwatch and evmpost are in $ORA_CRS_HOME/bin. They should be executed by the root user.

Fast Application Notification (FAN) Server-Side Callouts

- Server-side callouts
  - Provide simple mechanism to integrate with HA framework
  - Can be deployed with minimal programmatic effort
  - Can be a shell script or pre-compiled executable written in any programming language
  - Stored in $ORA_CRS_HOME/racg/usrco
  - Must be included on every node in cluster
- A state change can be
  - start or stop of service
  - start or stop of instance
  - start or stop of database
  - node leaving cluster

RAC HA framework posts a FAN event to ONS immediately when a state change occurs

A condition occurs when an event is received in the cluster through the Oracle Notification Service (ONS)

When a condition occurs all executables in $ORA_CRS_HOME/racg/usrco will be executed asynchronously

Oracle recommends event-handling programs with similar semantics or whose executions must be performed in a particular order be invoked from a common callout

Test each new callout for execution performance before deployment

Fast Application Notification (FAN) Server-Side Callouts

- For example create a script called callout1.sh in $ORA_CRS_HOME/racg/usrco with the following contents

```
#! /bin/bash
echo $*  >> /tmp/fan.log
```

- This callout writes all arguments to /tmp/fan.log
- Note that the !/bin/bash shell directive is mandatory
- Set the permissions on the callout

```
chmod 755 $ORA_CRS_HOME/racg/usrco/callout1.sh
```

- Copy to the same location on the other nodes
- Subsequent events will be written to /tmp/fan.log
- Alternatively messages can be logged to /var/log/messages using the following callout script:

```
#! /bin/bash
logger "ONS:" $*
```


The callout directory should have write permissions only to the system user who installed CRS

Each callout executable or script should have execute permissions only for the same user

Fast Application Notification (FAN) Server-Side Callouts

- Consider SERVICE1 configured with
  - preferred instances TEST1 (server6) and TEST2 (server7)
  - available instances TEST3 (server8) and TEST4 (server9)
- If server6 fails a NODE event is sent to the remaining nodes

```
NODE VERSION=1.0 host=server6 incarn=7 status=nodedown
reason=member_leave timestamp=28-Dec-2007 20:26:21 reported=Fri
Dec 28 20:26:22 GMT 2007
```

- SERVICE1 will be restarted on one of the existing nodes; in this case server9
- A SERVICEMEMBER event is sent to server9

```
SERVICEMEMBER VERSION=1.0 service=SERVICE1 database=TEST
instance=TEST4 host=server9 status=up reason=unknown card=2
timestamp=28-Dec-2007 20:25:01
```


The following shell script extract demonstrates how to handle the arguments in a callout:

```
NOTIFY_EVENTTYPE=$1             # Event type is handled differently

for ARGS in $*; do
    PROPERTY=`echo $ARGS | $AWK -F"=" '(print $1)'`
    VALUE =`echo $ARGS | $AWK -F"=" '(print $2)'`

    case $PROPERTY in
        VERSION|version)        NOTIFY_VERSION=$VALUE;;
        SERVICE|service)        NOTIFY_SERVICE=$VALUE;;
        DATABASE|database)      NOTIFY_DATABASE=$VALUE;;
        INSTANCE|instance)      NOTIFY_INSTANCE=$VALUE;;
        HOST|host)              NOTIFY_HOST=$VALUE;;
        STATUS|status)          NOTIFY_STATUS=$VALUE;;
        REASON|reason)          NOTIFY_REASON=$VALUE;;
        CARD|card)              NOTIFY_CARDINALITY=$VALUE;;
        TIMESTAMP|timestamp)    NOTIFY_LOGDATE=$VALUE;;
        ??:??:??)               NOTIFY_LOGTIME=$PROPERTY;;
    esac
done
```

Oracle Notification Service (ONS) Introduction

- Oracle Clusterware uses the Oracle Notification Service (ONS) to propagate FAN messages
  - within RAC cluster
  - to client and mid-tier systems

- ONS:
  - must be configured on each node in the cluster
  - installed with RAC
  - runs as a CRS resource
  - uses simple publish / subscribe method to produce and deliver event messages for both local and remote consumption

Oracle Notification Service (ONS)
ONS Daemon

- ONS daemon
  - executable is $ORA_CRS_HOME/opmn/bin/ons
  - runs as a node application
  - started automatically by CRS during reboot
  - sends and receives messages from other ONS daemons in a configurable list of nodes
  - can be controlled using ONSCTL
- Under normal operation should be controlled by Oracle Clusterware
  - Can also be started using SRVCTL

```
srvctl start nodeapps -n <node_name>
srvctl stop nodeapps -n <node_name>
srvctl status nodeapps -n <node_name>
```


Options are

-d Daemon mode - forks and detaches OPMN server processes

-a <cmd> Administrative mode

where <cmd> in ping, shutdown, reload, debug

To check status of node applications use:

$ srvctl status nodeapps -n <nodename>

For example:

```
$ srvctl status nodeapps -n server3
VIP is running on node: server3
GSD is running on node: server3
Listener is running on node: server3
ONS daemon is running on node: server3
```

Oracle Notification Service (ONS) ONSCTL

- **ONSCTL can be used to control the ONS daemon**
- **Options include:**

start	Start ONS daemon
stop	Stop ONS daemon
ping	Check if ONS daemon is running
debug	Display debug information for ONS daemon
reconfig	Reload ONS configuration
help	Print short syntax description
detailed	Print verbose syntax description

- **To display debug information for the ONS daemon use:**
  - **ONSCTL DEBUG**

To check if the ONS daemon is active use:

```
onsctl ping
```

For example:

```
$ onsctl ping
ons is running ...
```

Oracle Notification Service (ONS) Configuration

- On database tier ONS can be configured in $ORACLE_HOME/opmn/conf/ons.config
- Must contain values for
  - localport - port that ONS binds to on local interface to talk to local clients
  - remoteport - port that ONS binds to on all interfaces for talking to other ONS daemons
  - nodes - list of other ONS daemons to talk to specified as either hostname | port or IP address | port

```
localport=6100          # port ONS is writing to on this node
remoteport=6200         # port ONS is listening on this node
# This is the list of hosts and ports ONS is posting to.
# Include RAC and client nodes
nodes=server6.juliandyke.com:6200,server7.juliandyke.com:6200,
server8.juliandyke.com:6200,server9.juliandyke.com:6200
```


ons.config mandatory parameters:

• *nodes* should include all nodes in RAC cluster and any other nodes where ONS is running to receive FAN events for applications e.g. mid-tier

• Ports specified in *nodes* should be remote ports

ons.config optional values include

• *loglevel* - specifies level of messages that should be logged by ONS. Range 1 (minimum) to 9 (maximum), default value 3. For example:

loglevel = 3

• *logfile* - specifies file to which log messages will be written
Default value is $ORACLE_HOME/opmn/logs/opmn.log
For example:

logfile=/private/oraclehome/opmn/logs/myons.log

• *walletfile* - specifies wallet file used by Oracle SSL layer to store SSL certificates. For example:

You can also specify comments e.g.

This is a comment

Oracle Notification Service (ONS) Configuration

- ons.config optional values include
  - useocr - specifies where to store node information
    - on - Oracle Cluster Registry (OCR)
    - off - ONS configuration file (ons.config)

```
useocr=on
```

- If useocr=on then racgons utility must be used to add ONS configuration information to OCR
- Do not use useocr=on on client side

87 © 2008 Julian Dyke juliandyke.com

In Oracle 10.1.0.3 only do not set usrocr=on otherwise the OCR can become unavailable during recovery. In Oracle 10.1.0.3, define nodes and ports in ons.config

If RAC installed on shared file system then the $ORACLE_HOME/opmn/conf directory must be created locally on each node. This is a workaround for bug 3406450. You can use a symbolic link, for example:

```
mv $ORACLE_HOME/opmn/conf $ORACLE_HOME/opmn/conf.orig
mkdir $HOME/opmn/conf
cp $ORACLE_HOME/opmn/conf.orig/* $HOME/opmn/conf
ln -s $HOME/opmn/conf $ORACLE_HOME/opmn/conf
```

Oracle Notification Service (ONS) Adding and Deleting Nodes

- By default nodes configured in OCR use port 4948
- Nodes can be added using:

```
racgons add_config hostname:port [hostname:port] ...
```

- For example:

```
racgons add_config server9.juliandyke.com:6200
```

- Nodes can be deleted using:

```
racgons remove_config hostname[:port] [hostname:port] ...
```

- For example:

```
racgons remove_config server9.juliandyke.com:6200
```

- If no port is specified then all ports are removed for hostname

ONS can dynamically recognize a new ONS daemon

If you add a node to the cluster, you do not need to restart the ONS on the existing nodes - when new ONS publishes an event to the existing nodes they will dynamically add new ONS to their configuration

You do not need to add all RAC nodes and all mid-tier nodes to ons.config. However, you must have at least one mid-tier and one RAC node for ONS to work

Fast Connection Failover Introduction

- Introduced in Oracle 10.1
- Provides:
  - Rapid detection and cleanup of invalid cached connections (DOWN event processing)
  - Load balancing of available connections (UP event processing)
  - Runtime work request distribution to all active RAC instances
- Allows JDBC connection pools to subscribe to FAN events
- JDBC Connection pool
  - Can react to up and down events from database cluster
  - Application will always receive valid connection to an active instance providing requested database service

Modern applications use connection pools. In the past applications created a connection to the database every time they needed to complete some work in the database. When a connection pool is created the application uses an existing connection from the pool. This reduces the connection overhead on the database server and also improves response times.

Fast Connection Failover provides client integration with RAC for RAC/HA event notification mechanisms

Fast Connection Failover
Up and Down Events

- When up event is received:
  - New connections should go to restarted instance
  - Connection pool attempts to rebalance connections by retiring some connections and creating new ones If using load balancing
  - Connections should go to instance triggering up event
  - Work should be immediately directed to new instance with no application changes
- When down event is received:
  - All connections to instance are terminated.
  - Connections that were in use are cleaned up so application will receive failure immediately
  - Database will rollback uncommitted transactions

Database connection strings used with Fast Connection Failover cannot use TAF

Fast Connection Failover Prerequisites

- JDBC Implicit Connection Cache is enabled
  - can use thick or thin JDBC drivers
- Application uses service names to connect to database
- Underlying database is 10.1.0.3 or above
- Oracle Notification Service (ONS) is configured and available on client node
  - required to propagate database events and notify JDBC
- Application must specify oracle.ons.oraclehome property
  - specifies $ORACLE_HOME where ONS files are installed

Configuration steps are:

• Enable JDBC Implicit Connection Cache in datasource

• Enable JDBC Fast Connection Failover in datasource

• Configure ONS on each RAC node to be aware of application tier nodes

• Install and configure ONS on each application tier node

• Ensure that ons.jar is on CLASSPATH for application

• Specify system property oracle.ons.oraclehome when starting application

Fast Connection Failover is driver independent; it supports both the thin and OCI JDBC drivers. The JDBC Implicit Connection Cache must be used which may mean updating the application to use a more recent version of the Oracle JDBC client. Integration with the application is easy; it is only necessary to enable Fast Connection Failover in the OracleDataSource.

The oracle.ons.oraclehome system property must be specified when starting the applcation. It should specify the location of the ONS home:

Doracle.ons.oraclehome=<location of ons home>

Fast Connection Failover JDBC

- **Enable Implicit Connection Cache using:**

```
OracleDataSource.setConnectionCachingEnabled (true);
```

- **Enable Fast Connection Failover using:**

```
OracleDataSource.setFastConnectionFailoverEnabled (true);
```

- **For example:**

```
OracleDataSource ods = New OracleDataSource ();
ods.setUser ("US01");
ods.setPassword ("US01");
ods.setConnectionCachingEnabled (true);
ods.setFastConnectionFailoverEnabled (true);
ods.setConnectionCacheName ("MyCache");
ods.setConnectionCacheProperties (cp);
ods.setURL ("jdbc:oracle:thin:@(DESCRIPTION=
    (LOAD_BALANCE=on)
    (ADDRESS=(PROTOCOL=TCP)(HOST=VIP1)(PORT=1521)
    (ADDRESS=(PROTOCOL=TCP)(HOST=VIP2)(PORT=1521)
    (CONNECT_DATA=(SERVICE_NAME=service_name)))");
```


When implicit caching is enabled the first connection request to the OracleDataSource transparently creates a connection cache

When Fast Connection Failover is enabled it cannot be disabled during the lifetime of the cache

Fast Connection Failover
ONS Configuration

- ONS must be configured within the application tier in:
  - $ORACLE_HOME/opmn/conf/ons.config
- For example:

```
localport=6100          # port ONS is writing to on this node
remoteport=6200         # port ONS is listening on this node
nodes=server6.juliandyke.com:6200,server7.juliandyke.com:6200,
server8.juliandyke.com:6200,server9.juliandyke.com:6200,
laptop3.juliandyke.com
```

- When the starting application
  - specify the system property
    -Doracle.ons.oraclehome=<location_of_ons_client>
  - add $ORACLE_HOME/opmn/lib/ons.jar to $CLASSPATH

93 © 2008 Julian Dyke juliandyke.com

On the database tier, ONS can be configured in the OCR or in the ons.config file. On the middle tier ONS is always configured in the ons.config file which is located in

$ORACLE_HOME/opmn/conf/ons.config

The hostname of the current node can be included in nodes list in ons.config. It will be ignored when list is read At a minimum the ONS client must be aware of at least one RAC node. However, it is recommended that all RAC nodes are included

In order to use Fast ConnectionFailover, the Oracle JDBC Implicit Connection Cache requires a local ONS daemon from which to receive event messages The connection cache will not receive messages if ONS daemon is not running

ONS shipped with Oracle Database 10.1.0.2 and above

ONS not shipped with Oracle Client 10.1.0.2. To use it with this version, you must obtain patch for bug 3848905 or alternatively upgrade to Oracle Client 10.1.0.3 or above.

To check that ONS daemon is active on client use:

onsctl ping

Fast Connection Failover Failure Detection

- When a node or instance fails:
  - Database detects error and rolls back transaction
  - FAN event is propagated to JDBC connection cache
  - Cache manager cleans up all invalid connections
  - When an application holding invalid connection tries to use connection it receives ORA-17008 Closed Connection
  - When application receives ORA-17008 it should
    - Retry connection request
    - Repeat transaction

It is not necessary to rollback the transaction when ORA-17008 is received

Fast Connection Failover Comparison with TAF

- FCF supports application-level connection retries.
  - Application can decide whether to retry connection or throw exception. TAF supports connection retries only at OCI/Net layer
- FCF is integrated with Implicit Connection Cache
  - Allows connection cache manager to manage cache for HA. Failed connections are invalidated in the cache. TAF works at network level on per-connection basis. With TAF connection cache cannot be notified of failures
- FCF is based on RAC event mechanism
  - more efficient - can detect failures quickly for both active and inactive connections. TAF is based on network call mechanism
- FCF supports UP event load balancing
  - of connections and runtime work request distribution across active RAC instances