An Oracle White Paper
January 2010

Automatic Workload Management with Oracle Real Application Clusters 11g Release 2

Introduction

Applications using a clustered database generally want to load balance their workload across the cluster. Starting with Oracle RAC 10g Release 2 the load balancing advisory provides real-time information to the application tier on the service level being provided by the database. Applications can utilize this information to provide the best possible throughput or transaction response time to the application using the assigned resources in the cluster.

Oracle Real Application Clusters (RAC) includes a highly available (HA) application framework that provides the necessary service and integration points between Oracle RAC and custom enterprise applications. One of the main principles of a highly available application is for it to be able to receive fast notification when something happens to critical system components (both inside and outside the cluster). This allows the application to execute event-handling programs. The timely execution of such programs minimizes the impact of cluster component failures, by avoiding costly connection time-outs, application timeouts, and reacting to cluster resource reorganizations, in both planned and unplanned scenarios.

This paper discusses the features of Oracle Real Application Clusters 11g Release 2 that can be used by applications to provide the best possible service levels to the end user by maximizing throughput or response time and minimizing the impact of cluster component failures. This paper also gives practical examples showing how you can integrate these features into your environment.

FAST APPLICATION NOTIFICATION (FAN)

Fast Application Notification (FAN), is a feature of Oracle Real Application Clusters (RAC) that further differentiates it for high availability and scalability. FAN enables the automated recovery of applications when cluster components fail. FAN solves the following problems:

- Applications waiting for TCP/IP time-outs when a node fails.
- Applications attempting to connect when services are down.
- Applications not connecting when services resume.
- Applications processing the last result after a node, instance, or service has gone down.
- Applications not balancing across available systems when services restart or expand.
- Attempting to execute work on slow, hung, and dead nodes

For cluster configuration changes, the Oracle RAC HA framework posts a FAN event immediately when a state change occurs in the cluster. Instead of waiting for the Application

Tier to poll the Database Tier and find a problem, using FAN your application tier will receive these events and react immediately to the event. For down events, the disruption to the application can be minimized as connections to the failed instance or node can be terminated. In-flight transactions are terminated and the application user immediately notified. Application users requesting connections are directed to available instances only. Server side callouts can be used to log trouble tickets or page Administrators to alert them of the failure. For Up events, when services and instances are started, new connections can be created so the application can immediately take advantage of the extra resources.

With Oracle RAC 10g Release 2, FAN was extended to assist applications in directing work requests to where the request can be best satisfied based on the current application workload. Applications can take advantage of the load balancing advisory FAN events to direct work requests to the instance in the cluster that is currently providing the best service.

You can take advantage of FAN in the following ways:

1. Your application can take advantage of FAN without any programmatic changes by utilizing an integrated Oracle Client. The integrated clients for FAN events include Oracle Database 11g JDBC, Oracle Universal Connection Pool (UCP) for Java, Oracle Database 11g ODP.NET, and Oracle Database 11g Oracle Call Interface (OCI).
2. Implement FAN server side callouts on your Database Tier.

Services

The use of FAN requires the use of Services. Services de-couple any hardwired mapping between a connection request and a RAC instance. Services are an entity defined for a RAC database that allows the workload for a RAC database to be managed. Services divide the entire workload executing in the Oracle Database into mutually disjoint classes. Each service represents a workload with common attributes, service level thresholds, and priorities. The grouping is based on attributes of the work that might include the application being invoked, the application function to be invoked, the priority of execution for the application function, the job class to be managed, or the data range used in the application function or job class. Do not use the default database service; create at least one service as described in this document.

Single Client Access Name (SCAN)

Single Client Access Name (SCAN)[1] is s a new Oracle Real Application Clusters (RAC) 11g Release 2 feature that provides a single name for clients to access an Oracle Database running in a cluster. The benefit is clients using SCAN do not need to change if you add or remove nodes in the cluster. Having a single name to access the cluster allows clients to use the

[1] For more information on SCAN please refer to http://www.oracle.com/technology/products/database/clustering/pdf/scan.pdf

EZConnect client and the simple JDBC thin URL to access any database running in the clusters independently of which server(s) in the cluster the database is active. SCAN provides load balancing and failover of client connections to the database. The SCAN works as an IP alias for the cluster.

```
sqlplus system/manager@sales1-scan:1521/oltp
jdbc:oracle:thin:@sales1-scan:1521/oltp
```

Figure 1 Sample EZConnect and Thin JDBC Connect Strings

Fast Connection Failover

Fast Connection Failover is a feature of Oracle clients that have integrated with FAN HA Events. This is the easiest way for an application to improve its availability by taking advantage of the ability to know a failure has occurred at the RAC database and recover from the failure as fast as possible. Utilizing this feature, applications can mask failures from the user. Oracle JDBC Implicit Connection Cache, Oracle Call Interface (OCI), and Oracle Data Provider for .Net (ODP.Net) include fast connection failover. With fast connection failover, when a down event is received, cached connections affected by the down event are immediately marked invalid and cleaned up. In flight transactions are aborted. The application can either propagate the error to the end user or retry the transaction and get a connection from the cache to a surviving instance. If the error is propagated to the user, the user can easily retry the transaction and get a connection to a surviving instance.

To make the best use of the cluster resources, applications need to spread the workload across multiple servers in the cluster. Oracle RAC provides advanced capability to notify the application of the current service level provided by each node so that it can intelligently send work requests to the instance that will provide the best service at the time of the request. The load balancing advisory continually analyzes the work being processed on each instance providing a service and publishes an event which includes a recommendation on the percentage of work to send to this instance and a flag indicating the quality of the data provided. The load balancing advisory adjusts distribution for nodes that have different power (I.E. different number of CPUs or different CPU speeds). It can react quickly to changes in the cluster configuration, changes in application workload, and hung or overworked nodes. The load balancing advisory is integrated with the Automatic Workload Repository built into Oracle Database 11g. The Automatic Workload Repository measures response time and CPU consumption for each service. The views V$SERVICEMETRIC and V$SERVICEMETRIC_HISTORY contain the service time for every service. These views are updated every 60s and contain 1 hour of history.

Load Balancing Advisory

The load balancing advisory publishes its information as FAN events. Oracle connection pools have been integrated with this load balancing advisory by subscribing to the FAN events to provide Runtime Connection Load Balancing for application. When an application gets a connection from the pool, it will get the best connection available to process the work request instead of a random connection.

Oracle Database 11g introduces an additional flag in the load balancing advisory event called affinity hint. The affinity hint is automatic when load balancing advisory is turned on through setting the goal on the service. This flag is for temporary affinity that lasts for the duration of a web session. Web conversations often connect and disconnect a number of times during the entire session. During each of these connects, it may access the same or similar data, e.g. a shopping cart, Siebel, etc. Affinity can improve buffer cache efficiency, which lowers cpu usage and transaction latency. The Affinity Hint is a flag that indicates if Affinity is active or inactive for a particular instance/service combination. Different instances offering the same service can have different settings for the Affinity Hint.

Oracle Database 11g Patchset 1 (11.1.0.7) introduces a new connection pool for Java called the Universal Connection Pool (UCP). UCP can be used against Oracle Database 10g or Oracle Database 11g. Applications using Oracle Database 11g and UCP, can take advantage of this new affinity feature. If the affinity flag is turned on in the Load Balancing Advisory event, then UCP will create an Affinity Context for the Web session such that when that session does a get connection from the pool, the pool will always try to give it a connection to the instance it connected to the first time it acquired a session. The choice of instance for the first connection is based on the current load balancing advisory information.

FAN Events

Before using some of the FAN features such as Server Side Callouts, you should understand FAN events. Oracle RAC FAN events consist of header and detail information delivered as a set of name-value pairs accurately describing the name, type and nature of the event. Based on this information, the event recipient can take concrete management, notification, or synchronization steps, such as shutting down the application connection manager, rerouting existing database connection requests, refreshing stale connection references, logging a trouble ticket, or sending a page to the database administrator.

FAN events are system events, sent during periods when cluster servers may become unreachable and network interfaces slow or non-functional. There is an inherent reliance on minimal communication channel overhead to send, queue and receive notifications quickly. The objective is to deliver FAN events so that they precede regular connection timeouts or typical polling intervals.

Three categories of events are supported in Oracle RAC FAN:

- Service events, which includes both application services and database services
- Node events, which includes cluster membership states and native join/leave operations
- Load Balancing Events sent from the RAC Load Balancing Advisory

RAC standardizes the generation, presentation, and delivery of events pertaining to managed cluster resources. The following examples show the structure of a FAN event:

Parameter	**Description**
Version	Version of the event payload. Used to identify release changes.
Event type	**SERVICE, SERVICE_MEMBER, DATABASE, INSTANCE, NODE, ASM, SRV_PRECONNECT** Note that Database and Instance types provide the database service – db_unique_name.db_domain
Service name	The service name. Matches the service in DBA_SERVICES.

Database Unique Name	The unique database supporting the service. Matches the initialization parameter value for db_unique_name, which defaults to the value of the initialization parameter DB_NAME.
Instance	The name of the instance supporting the service. Matches the ORACLE_SID
Node name	The node name supporting the service or the node that has gone down. Always matches the node name known to CSS.
Status	Values are UP, DOWN, NOT_RESTARTING, PRECONN_UP, PRECONN_DOWN, UNKNOWN
Cardinality	Number of service members – included on all UP events.
Reason	Failure, Dependency, User, Autostart, Boot
Incarnation	For node down events – the new cluster incarnation
Timestamp	Date and time stamp (local time zone) to order notification events

Figure 2 HA FAN Event

Parameter	Description
Version	Version of the event payload. Used to identify release changes.
Event type	SERVICE_METRICS
Service	Matches the service in DBA_SERVICES.
Database Unique Name	The unique database supporting the service. Matches the initialization parameter value for db_unique_name, which defaults to the value of the initialization parameter DB_NAME.
Timestamp	Date and time stamp (local time zone) to order events
The following fields are repeated.	
Instance	The name of the instance supporting the service. Matches the ORACLE_SID
Percent	The percentage of work requests to send to this database and instance
Flag	Indication of the service quality relative to the service goal – Values are GOOD (metrics are valid), VIOLATING (AWR ELAPSED or CPU thresholds are violated), NO DATA (instance did not send metrics), UNKNOWN (AWR has no data for service)
Affinity Hint	Indicates if client should use session affinity. Values are TRUE or FALSE.

Figure 3 Load Balancing FAN Event

To check on HA FAN events, check the Agent log for the Oracle User ($GRID_HOME/log/<hostname>/agent/crsd/oraagent_<userid>/oraagent_<userid>.log). Search for [EonsSub ONS].

To monitor the Load Balancing advisory events, use the following example:

```
set pages 60 space 2 lines 132 num 8 verify off feedback off
column user_data heading "AQ Service Metrics" format A60 wrap
break on SERVICE_NAME skip 1
select
 to_char(ENQ_TIME, 'HH:MI:SS') Enq_time, user_data
from SYS.SYS$SERVICE_METRICS_TAB
order by 1 ;
```

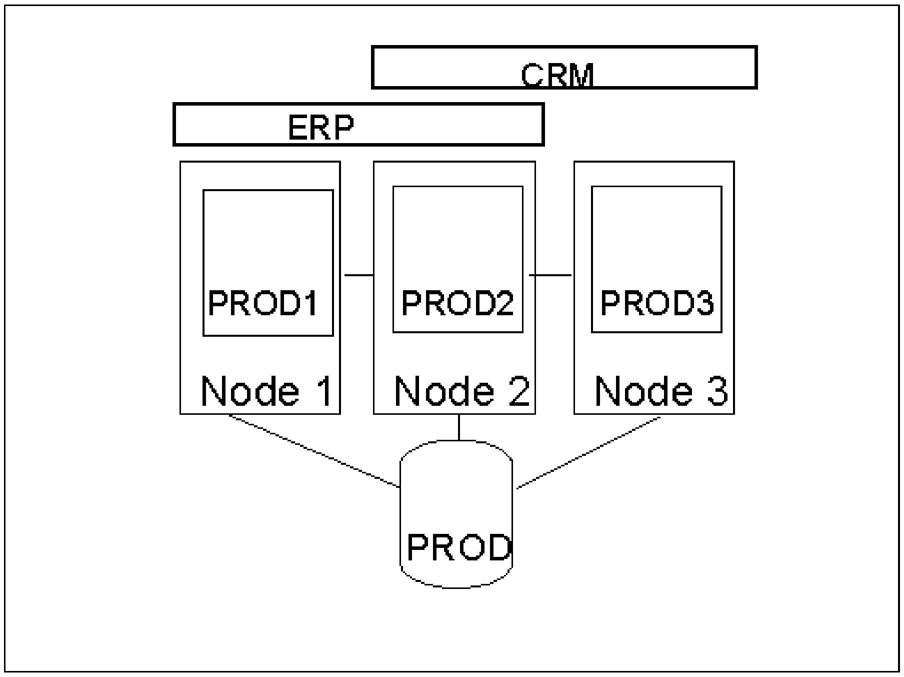


Figure 4 Example Application Setup

To help understand what FAN events look like, let us look at a sample configuration that is pictured in Figure 5. In this example, we have a 3 node RAC database called PROD with 2 Services. Service ERP is defined as Primary=Node1,Node2 and Available=Node3. Service CRM is defined as Primary=Node2, Node3 and Available=Node1. With this configuration, the ERP service will fail over to Node3 if instance PROD1 or PROD2 fail and the CRM service will fail over to Node1 if instance PROD2 or PROD3 fail. Normal operation is to have the ERP service running on PROD1,PROD2 and the CRM service to be running on PROD2,PROD3.

If Node1 fails, the ERP service will failover to Node3, service PROD will stop on Node1. The following down events are sent:

Event 1: FAN event type: instance

Properties: version=1.0 service=PROD database=PROD instance=PROD1 host=node1 status=down

Event 2: FAN event type: service_member

Properties: version=1.0 service=ERP database=PROD instance=PROD1 host=node1 status=down

After service ERP fails over to instance PROD3, service member ERP is up on instance PROD3 and the event is sent as follows:

Event 3: FAN_event type: service_member

Properties: version=1.0 service=ERP database=PROD instance=PROD3 host=node3 status=up

A load balancing advisory event for the above configuration would be:

Event 4: FAN-event type: service_metrics

Properties: version=2.0 service=ERP database=PROD instance=PROD1 percent=70 service_quality=GOOD instance=PROD2 percent=30 service_quality=GOOD

Event 5 :FAN-event type: service_metrics

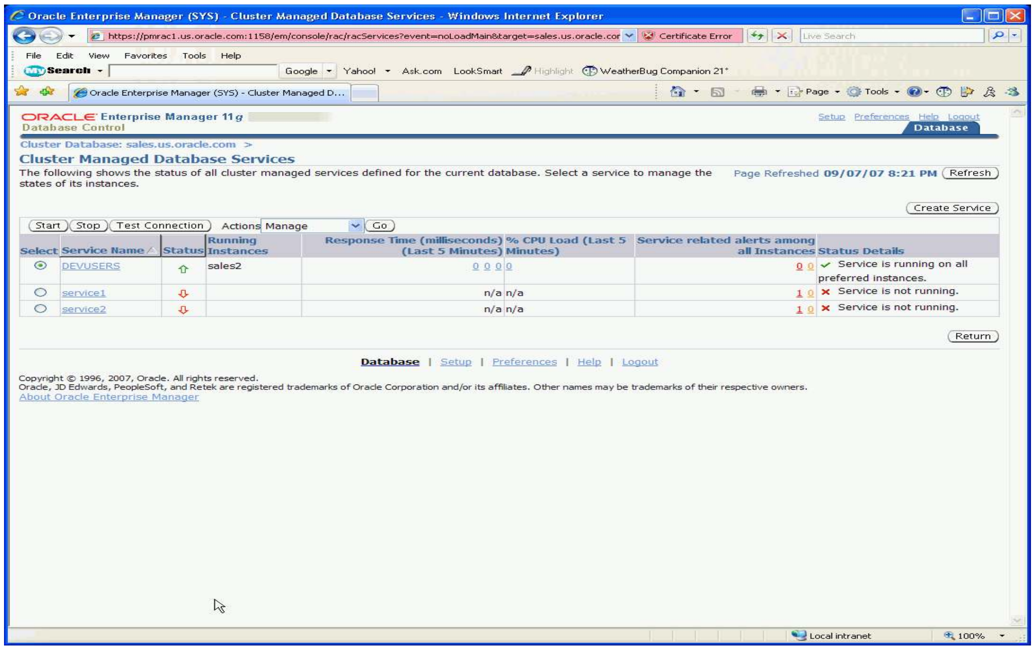
Properties: version=2.0 service=CRM database=PROD instance=PROD2 percent=30 service_quality=GOOD instance=PROD3 percent=70 service_quality=GOOD

Database Tier

Oracle Database 11g provides the infrastructure to make your application data highly available through the Oracle Real Application Clusters (RAC) option. Oracle Real Application Clusters provides the ability to have multiple instances running on multiple nodes accessing the same database. In the event of hardware or software failure, access to your data is not lost and only a subset of application sessions are affected by the failure. The Oracle Clusterware will automatically try to restart the failed system or process. Often the system is recovered before the Administrator is aware a problem occurred.

Services

Traditionally an Oracle database provided a single service and all users connected to the same service. A database will always have this default database service that is the database name. This service cannot be modified and will always allow you to connect to the database. Do not use the default database service, create at least one service as described in this document. A database can have many services (up to a maximum of 115 per database). The services hide the complexity of the cluster from the client by providing a single system image for managing work. Applications and mid-tier connection pools select a service by using the Service Name in their connection data. The service must match the service that has been created using add service with Enterprise Manager, or SRVCTL. It is recommended that you use Enterprise Manager to create and manage services by selecting the Cluster Managed Database Service screen in Enterprise Manager. You can check the active services by querying the V$ACTIVE_SERVICES view. You can check the sessions using a service by querying the V$SESSION view. You may find features of Oracle database create their own services such as the 2 internal services SYS$BACKGROUND which is used by the background processes only and SYS$USERS which is the default service for user sessions that are not associated with any application service. You will not be able to manage these internal services. As a rule of thumb, if you did not create the service, you will not be able to manage it. Enterprise Manager supports creating, modifying, viewing, and operating services as a whole, with drill down to the instance-level when needed. Figure 4 below shows the Enterprise Manager 11g screen for database services, which is the recommended way to manage services. Click on the Availability Tab from your clustered Database, and then click on the Clustered Managed Services link. A service can span one or more instances of an Oracle database and a single instance can support multiple services. The number of instances offering the service is managed by the DBA independent of the application. All client connections should connect to a service.



Cluster Database: sales.us.oracle.com >

Cluster Managed Database Services

The following shows the status of all cluster managed services defined for the current database. Select a service to manage the states of its instances.

Page Refreshed 09/07/07 8:21 PM

Select	Service Name	Status	Running Instances	Response Time (milliseconds) (Last 5 Minutes)	% CPU Load (Last 5 Minutes)	Service related alerts among all Instances	Status Details
●	DEVUSERS	↑	sales2	0 0 0	0	0 0	✓ Service is running on all preferred instances.
○	service1	↓		n/a	n/a	1 0	✗ Service is not running.
○	service2	↓		n/a	n/a	1 0	✗ Service is not running.

Copyright © 1996, 2007, Oracle. All rights reserved.
Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
About Oracle Enterprise Manager

Figure 5 Enterprise Manager 11g Cluster Managed Services Screen

Services enable the automatic recovery of an application's ability to connect to the database. This is achieved according to business rules. Following outages, the service is recovered quickly and automatically at the surviving instances. When instances are later repaired, services that are not running are restored quickly and automatically by RAC HA Framework. Immediately when a service changes state, either up or down, a fast application notification (FAN) event is available for applications using the service. Applications can use this notification to trigger recovery and load balancing actions.

The use of Services is required to take advantage of the load balancing advisory and runtime connection load balancing features of Oracle RAC. When a service is created, a service level goal is defined. There are 3 options available

NONE – Default setting, you are not taking advantage of this feature

THROUGHPUT – Work requests are directed based on throughput. THROUGHPUT should be used when the work in a service completes at homogenous rates. An example is a trading system where work requests are similar lengths.

SERVICE_TIME – Work requests are directed based on response time.   SERVICE_TIME should be used when the work in a service completes at various rates. An example is as internet shopping system where work requests are various lengths

Oracle Net Services provides the ability to load balance connections to the database at connection time. You must configure Oracle Net Services for connection load balancing by setting the local_listener and remote_listener parameters. This will be automatically done if you used DBCA to create your database. Since Oracle Database 10g Release 2, a goal for connection load balancing is set when a service is created. There are two options for the Connection Load Balancing goal:

SHORT - used for application connections that are short in duration. This should be used with connection pools integrated with the load balancing advisory.

LONG - used for application connections that are connected for a long period such as third party connection pools and SQL*Forms applications. This is the default value.

Note: If you use the shared services feature of Oracle Net Services and set the service attribute of the dispatchers parameter in the init.ora, the service name specified in this parameter cannot be managed. IE srvctl stop service will show the service stopped in crsctl stat res -t however the listener will still allow connections.

Transparent Application Failover (TAF) can be defined at the service. TAF is a feature of Oracle Net Services that will failover a database session to a backup connection should the session be disconnected. Note: TAF will not happen on a normal shutdown of the database. If a service has a defined TAF policy, this will override a TAF policy on the client connection. The minimum requirement to turn TAF on is to set the failover_type=session or select. TAF restrictions have not changed and TAF is therefore only valid for OCI connections (do not use with JDBC FCF). The TAF failover method of PRECONNECT is not supported when defining a TAF policy for a service, you must use BASIC. TAF will use the FAN HA events to provide fast failover for database sessions. With Oracle RAC 11g Release 2, you must use SRVCTL to add the TAF policy.

```
$srvctl modify service -d crm -s  gl.us.oracle.com -q TRUE -P BASIC -e
SELECT -z 180 -w 5 -j LONG
```

Figure 6 Sample srvctl command to add TAF policy to Service

With Oracle RAC, there is a special service option to support Distributed Transaction Processing applications (IE XA transactions[2]). By defining the DTP property of a service, the service is guaranteed to run on one instance at a time in an Oracle RAC database. All global distributed transactions performed through the DTP service are ensured to have their tightly-coupled branches running on a single RAC instance. For Administrator – Managed database, define only one instance as the preferred instance. You can have as many AVAILABLE instances as you want. For Policy-Managed databases, use a singleton service.

Fast Application Notification (FAN) Server Side Callouts

Server-side callouts provide a simple, yet powerful integration mechanism with the High Availability Framework that is part of the Oracle Clusterware. Server side callouts can be deployed with minimal programmatic effort. A callout is essentially a shell script or pre-compiled executable written in any programming language. Simply place an executable in the directory $GRID_HOME/racg/usrco on every node that runs the Oracle Clusterware. If using scripts, remember to set the shell as the first line of the executable. A callout script can be as simple as callout.sh[3]:

[2] Best Practices for XA Transactions with RAC can be found
http://www.oracle.com/technology/products/database/clustering/pdf/bestpracticesforxaandrac.pdf

[3] Sample Code can be found
http://www.oracle.com/technology/sample_code/products/rac/index.html

```
#! /bin/ksh
FAN_LOGFILE= [your path name]/admin/log/`hostname`_uptime.log
echo $* "reported="`date` >> $FAN_LOGFILE &
```

When callout.sh is executed, it will produce output such as:

```
NODE VERSION=1.0 host=sun880-2 incarn=23 status=nodedown reason=
timestamp=08-Oct-2004 04:02:14 reported=Fri Oct 8 04:02:14 PDT 2004
```

All executables in the directory $GRID_HOME/racg/usrco will execute immediately in an asynchronous fashion when a condition occurs. A condition occurs when an HA event is received in the cluster through the Oracle Notification Service (ONS). The Oracle RAC HA framework posts a FAN event to ONS immediately when a state change occurs. A state change could be the start or stop of a service, instance, database, or when a node leaves the cluster. Oracle recommends event-handling programs with similar semantics or whose executions must be performed in a particular order, be invoked from a common callout. You should carefully test each new callout for execution performance before formal deployment.

Note: As a security measure, it is recommended that the callout directory has *write* permissions only to the system user who installed Oracle Clusterware, and that each callout executable or script contained therein has *execute* permissions only to the same Oracle Clusterware user.

Writing server-side callouts involves the following steps:

1. Parse the event argument list. A sample Bourne shell script that parses the event argument list is:

```
# Scan and parse HA event payload arguments:
#
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 ;; # catch event date
        ??:??:??) NOTIFY_LOGTIME=$PROPERTY ;; # catch event time (hh24:mi:ss)
esac
done
```

2. Filter incoming FAN events

Instead of propagating every FAN event to the local event-handling program, you can filter the incoming events based on payload information. In the following Bourne shell script example, a trouble ticket system (using the second filtering example above) is invoked only when RAC HA framework posts a SERVICE, DATABASE or NODE event type, with status either “down”, “not_restarting”, “restart_failed” or “nodedown”, and only for two application service names: HQPROD and FIN_APAC:

```
# Only FAN events with the following conditions will be inserted
# into the critical trouble ticket system:
# NOTIFY_EVENTTYPE => SERVICE | DATABASE | NODE
# NOTIFY_STATUS => down | not_restarting | restart_failed | nodedown
# NOTIFY_DATABASE => HQPROD | FIN_APAC
#
if ((( [ $NOTIFY_EVENTTYPE = "SERVICE" ] ||
        [ $NOTIFY_EVENTTYPE = "DATABASE" ] || \
        [ $NOTIFY_EVENTTYPE = "NODE" ] \
        ) && \
        ( [ $NOTIFY_STATUS = "down" ] || \
        [ $NOTIFY_STATUS = "not_restarting" ] || \
        [ $NOTIFY_STATUS = "restart_failed" ] || \
        [ $NOTIFY_STATUS = "nodedown " ] \
        )) && \
        ( [ $NOTIFY_DATABASE = "HQPROD" ] || \
        [ $NOTIFY_DATABASE = "FIN_APAC" ] \
        ))
then
        << CALL TROUBLE TICKET LOGGING PROGRAM AND PASS RELEVANT NOTIFY_*
        ARGUMENTS >>
fi
```

3. Executing event-handling programs

You must decide the actions that need to be taken once an event is received based on your business requirements. These can generally be fit into three categories of event handlers you may want to implement:

- Event logging: Receives the FAN event locally and copies a subset of the payload data into some persistent repository, e.g. a local log file, operating system logging service, or a remote database table.
- Event paging: Receives the FAN event locally and issues a notification message to a remote device, such as a pager, e-mail client or SNMP management console.
- Event start/stop action: Receives the FAN event locally and issues a start or stop command to a remote daemon or process. This requires that the local event handler be a trusted proxy application running on each RAC cluster node, capable of self-authentication and issuance of start/stop commands, through a secure network connection and an API that the remote daemon or process exposes. For example, the local component execution program may be a small, custom compiled program, or a perl script posting a URL through HTTPS.

Oracle Net Services Integration with FAN Events

The Connection Manager (CMAN) and Oracle Net Services Listeners take advantage of FAN. This allows the Listener and CMAN to immediately de-register services provided by the failed instance and avoid erroneously sending connection requests to failed instances. The Listener uses the load balancing advisory when load balancing connections where the service has CLB_GOAL=SHORT, and GOAL=SERVICE_TIME or THROUGHPUT. If CLB_GOAL=LONG then the listener will load balance based on number of sessions and the GOAL setting will not be used.

Data Guard Integration with FAN Events

Oracle Database 11g Data Guard Broker is integrated with FAN. When the role changes for a database that is part of a Data Guard physical standby configuration, an event is posted to help administrators automate post role change activities and notify applications when the database it is connected to is no longer operating in the primary role. The standby database can be either a single instance or RAC database. The Data Guard Broker posts a database down event for the lost primary. The down event is posted on successful completion of the failover operation and is posted by the database that is operating in the new primary database role on behalf of the old primary. Oracle Call Interface (OCI) has integrated with the Data Guard database down event. OCI clients that have registered for HA events, will receive this event. If the OCI connection is using Transparent Application Failover (TAF), then the client session can be failed over to the new primary database.

Oracle Notification Services on the Database Tier

Oracle Notification Service (ONS) uses a simple publish/subscribe method to produce and deliver event messages for both local and remote consumption. ONS daemons run locally sending messages to and receiving messages from a configured list of nodes (where other ONS daemons are active). Oracle Clusterware and RAC utilize ONS to propagate FAN messages both within the RAC cluster, and to client or mid-tier machines. ONS is installed with Oracle Real Application Clusters (RAC), and the Oracle Clusterware resources to manage the ONS daemon are created automatically during the RAC installation process. Oracle Clusterware automatically starts the ONS daemon during a reboot. Ensure your ONS is configured on each node running the Oracle Clusterware.

The ONS daemon is running as a node application. To check node applications use the command: `srvctl status nodeapps`. Your results should be similar to the following:

```
>srvctl status nodeapps
VIP rac1-vip is enabled
VIP rac1-vip is running on node: rac1
VIP rac2-vip is enabled
VIP rac2-vip is running on node: rac2
Network is enabled
Network is running on node: rac1
Network is running on node: rac2
GSD is disabled
GSD is not running on node: rac1
GSD is not running on node: rac2
ONS is enabled
ONS daemon is running on node: rac1
ONS daemon is running on node: rac2
eONS is enabled
eONS daemon is running on node: rac1
eONS daemon is running on node: rac2
```

Use onsctl ping to check that the ONS daemon is active.

```
>onsctl ping
ONS is running
```

The ONS configuration file, $ORACLE_HOME/opmn/conf/ons.config, contains the ONS configuration. This file is managed by the Oracle Clusterware ONS Agent and you should always use srvctl command to modify attributes of your ONS configuration. The important pieces of the ONS configuration are `localport`, the port that ONS binds to on the localhost interface to talk to local clients, `remoteport`, the port that ONS binds to on all interfaces for talking to other ONS daemons, and `nodes`, a list of other ONS daemons to talk to specified as either hostnames or IP addresses plus ports. The host information is updated by the Oracle Clusterware ONS Agent when it starts the ONS process. It will contain all active nodes at the time the ONS process was started.

```
      localport=6100          # line added by Agent
      allowgroup=true
      usesharedinstall=true
      remoteport=6200         # line added by Agent
      nodes=rac1:6200,rac2:6200        # line added by Agent
```

Figure 7 ONS Configuration with Oracle RAC

To see what your current configuration of the ONS is, use the srvctl config nodeapps command.

```
$srvctl config nodeapps
VIP exists.:rac1
VIP exists.: /rac1-vip/10.1.1.11/255.255.255.0/eth0
VIP exists.:rac2
VIP exists.: /rac2-vip/10.1.1.12/255.255.255.0/eth0
GSD exists.
ONS daemon exists. Local port 6100, remote port 6200
eONS daemon exists. Multicast port 19733, multicast IP address
234.19.232.48, listening port 2016
```

Figure 8 Sample output of srvctl config nodeapps command

Modifications of the ONS ports or nodes should be done using the srvctl modify nodeapps command.

```
srvctl modify nodeapps –l 6100 –r 6200 –t rac1:6200,rac2:6200
```

Using Oracle Streams Advanced Queuing for FAN Event Publication

Oracle Real Application Clusters publishes FAN events to a system Alert queue in the database. ODP.NET and OCI client integration use this method to subscribe to FAN events.

To have FAN HA events for a Service posted to the alert queue, the notification must be turned on for the service using srvctl.

```
srvctl modify service -d crm -s gl.us.oracle.com -q TRUE
```

To view the FAN HA events that are published, use the view `DBA_OUTSTANDING_ALERTS` or `DBA_ALERT_HISTORY`. When the GOAL is set on a service to either THROUGHPUT or SERVICE_TIME, load balancing advisory events are sent.

Application Tier

The easiest way to receive the benefits of FAN with your application is to use an integrated Oracle Client. Oracle Database 11g has integrated the connection pools for the JDBC (supports both the JDBC thick and thin driver), ODP.NET and OCI clients. If you are not using one of the integrated clients, an API is available which allows your application to subscribe directly to FAN events. All client integration requires that your database service be defined with the appropriate flags set and that you have Oracle Net Services configured for connection load balancing.

JDBC

For Java applications, you can use the Oracle Database 11g JDBC Fast Connection Failover feature of the Implicit Connection Cache. The steps required to implement FAN with Oracle Database 11g JDBC are:

1. Enable JDBC Implicit Connection Cache and JDBC Fast Connection Failover in your datasource. For faster failover of the client connection set TCP connect timeout property.
2. Configure the ONS on each RAC node to be aware of your application tiers.
3. Define the JDBC data source parameters to point to the remote ONS in the RAC cluster.
4. When starting the application, ensure that the ons.jar file is located on the application CLASSPATH.

JDBC Fast Connection Failover, JDBC Implicit Connection Cache and the ONS are discussed in detail in the following sections of this paper.

Turning on the Fast Connection Failover feature with JDBC includes Runtime Connection Load Balancing. The JDBC connection pool subscribes to the FAN Load Balancing events automatically when you configure fast connection failover. Instead of randomly assigning a free connection, the connection pool will choose the connection that will give the best service according to the latest information it has received. If a node becomes hung, it will gravitate connections from the hung node to other nodes in the cluster.

Oracle Application Server 10g (10.1.2) users with Java applications using servlets or beans, will receive the benefits of FAN by utilizing the JDBC Fast Connection Failover feature. Oracle Application Server 10g (10.1.3) fully integrates FAN features.

JDBC Fast Connection Failover (FCF)

The JDBC connection pool when configured to use Fast Connection Failover, automatically subscribes to FAN events, therefore it can react to the up or down events from the database cluster. The application will always be given a valid connection to an active instance providing the database service requested. When a down event is received, all connections to that instance using that service are terminated. Any connections that were in use at the time of failure are cleaned up so the application will receive the failure immediately. Database recovery will take care of rolling back any in-flight uncommitted transactions. The connection pool will create additional connections only when it requires them (i.e. a getConnection comes in and there are no idle connections in the pool). When that instance is restarted, an up event is sent and the connection pool will create new connections to the database. The number of new connections is a portion of the connections it held to the instance when it failed. If you have connection load balancing setup correctly, the new connections should go to the instance that was restarted. When an up event is received for a new instance, the connection pool tries to re-balance its connections by retiring a portion of connections and then creating new connections. Assuming you are using load balancing, the new connections will go to the instance that triggered the up event and work is immediately directed to the added instance with no application changes. The database connection string used with Fast Connection

Failover cannot use Transparent Application Failover (TAF) and the service used, should not have TAF attributes defined.

The pre-requisites for using Fast Connection Failover are:

- The implicit connection cache is enabled. Fast Connection Failover works in conjunction with the JDBC connection caching mechanism. This helps applications manage connections to ensure high availability.
- The application uses service names to connect to the database.
- JDBC Datasource is configured to for remote ONS subscription (or a ONS is running on the application tier as with Oracle AS 10g)
- The underlying database should be at a minimum Oracle Real Application Clusters (RAC) 10g (10.1.0.4). If failover events are not propagated, connection failover cannot occur.

Universal Connection Pool (UCP)

Oracle Database 11g (11.1.0.7) introduced a new Java connection pool called the Universal Connection Pool. The Universal Connection Pool (UCP) is a Java based connection pool that supports any type of connection (JDBC, LDAP, JCA), to any type of database (Oracle or non-Oracle) with any middle tier (Oracle or non-Oracle) and also support stand-alone deployments such as Toplink or BPEL. UCP includes integration to features of Oracle Database such as Oracle Real Application Clusters including Fast Connection Failover, Runtime Connection Load Balancing and Connection Affinity for RAC Instances. To take advantage of FCF and Runtime Connection Load Balancing, you need to set the Pool Datasource property for FastConnectionFailoverEnabled and the ONSConfiguration as shown below in Figure 7 Turning on FCF with UCP. Applications will need both UCP.JAR and ONS.JAR in their classpath.

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
Pds.setConnnectionPoolName("FCFSampleUCP");
pds.setONSConfiguration("nodes=racnode1:6201,racnode2:6201");
pds.setFastConnectionFailoverEnabled(true);
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin:@sales1-scan:1521/oltp");
......
```

Figure 9 Turning on FCF with UCP

Applications may increase performance by taking advantage of the connection affinity feature of UCP. There are two types of affinity, Web Session and XA. The first connection request is load balanced using the advice from the Load Balancing Advisory, subsequent requests for the duration of the web session or XA transaction are directed to the same instance as the first request. The affinity is only a hint, if there are no available connections to the desired instance, the pool will use choose a connection to the least loaded instance based on the information from the load balancing advisory. UCP will enforce affinity across Oracle Application Server OC4J instances. For XA transactions, applications will not have to specify a DTP service. To use affinity, you must be using Oracle RAC 11g, Oracle JDBC 11g, and UCP. A connection affinity callback must be registered with the UCP. It is recommended that you configure both Fast Connection Failover and Runtime Connection Load Balancing when using affinity. [4] Applications should also take advantage of the JDBC connect timeout property and the isFatalConnectionError described on the next page.

Implicit Connection Cache (ICC)

To take advantage of FAN and FCF with Oracle Database 10g JDBC drivers, you must be using the Implicit Connection Cache[5]. An application turns the implicit connection cache on by invoking `OracleDataSource.setConnectionCachingEnabled(true)`. After implicit caching is turned on, the first connection request to the `OracleDataSource` transparently creates a connection cache.

```
OracleDataSource ods = new OracleDataSource();
// Set DataSource properties
ods.setUser("Scott");
ods.setConnectionCachingEnabled(true); // Turns on caching
ctx.bind("MyDS", ods);
// ...
```

To take advantage of Fast Connection Failover, a second Data source property FastConnectionFailoverEnabled must be set to true. Note: ConnectionCacheName should come after the ConnectionCachingEnabled and FastConnectionFailover properties.

```
    OracleDataSource ods = new OracleDataSource()
    ...
    ods.setUser(“Scott”);
    ods.setPassword(“tiger”);
    ods.setConnectionCachingEnabled(True);
    ods.setFastConnectionFailoverEnabled(True);
    ods.setConnectionCacheName(“MyCache”);
    ods.setConnectionCacheProperties(cp);
    ods.setONSConfiguration("nodes=racnode1:6201,racnode2.:6201");
    ods.setURL("jdbc:oracle:thin:@sales1-scan:1521/oltp");
```

JDBC TCP Connect Timeout Property

Due to the retry logic in TCP, JDBC clients encounter delays between the virtual IP failing over, and the client completing a successful connection. These delays are avoided by setting the oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR property. This time is in milliseconds:

```
Properties prop = new Properties ();
prop.put (oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR,
"" + (1 * 1000)); // 1 second
dbPools[ poolIndex ].setConnectionProperties ( prop );
```

Application Usage of Fast Connection Failover

[4] For more information on using the Universal Connection Pool for Java, please read the Oracle Universal Connection Pool for JDBC Developers Guide. E10788-01

[5] With the introduction of UCP, ICC will be deprecated in a future release

When a connection is closed by the implicit connection cache or the universal connection pool, the application receives a SQLException on each connection . The application can invokes isFatalConnectionError(SQLException e) to determine:

- if each connection is still usable
- if connection is not usable any more, retry connection request, and obtain a good connection from the pool

At this stage, the application may replay transaction.

```
try {
conn = getConnection();
// do some work
} catch (SQLException e) {
handleSQLException(e)
}
...
handleSQLException (SQLException e)
{
if
(OracleConnectionCacheManager.isFatalConnectionError(e))
ConnRetry = true; // Fatal Connection error detected,
}
```

Note: Oracle Application Server handles connection retry transparently for Container Managed Persistence

Oracle Notification Service (ONS) on the Application Tier

The Oracle JDBC Implicit Connection Cache, when configured for Fast Connection Failover (FCF), requires an ONS daemon from which to receive event messages. The absence of ONS will prevent the connection cache from receiving events. If you are using Oracle Application Server, the ONS is installed as part of the Application Server. If you are not using Oracle Application Server, you can install ONS on the application tier (in this case the JVM in which your JDBC instance is running must have oracle.ons.oraclehome set to point to your ORACLE_HOME where the ONS files were installed) or use a remote ONS subscription by setting the ONS subscription DataSource property (recommended). Remote ONS subscription offers the following advantages:

- Support for an All Java mid-tier stack
- No ONS daemon needed on the client machine. No need to manage this process
- Simple configuration via DataSource property

When using remote ONS subscription for Fast Connection Failover, an application invokes `setONSConfiguration(String remoteONSConfig)` on an Oracle DataSource instance as in the following example:

```
ods.setONSConfiguration("nodes=racnode1:4200,racnode2.:4200");
```

The ONS.JAR must be included in the CLASSPATH on the client. The ONS.JAR can be found as part of the Oracle Client installation. There can only be one setONSConfiguration in your datasource definition. If you have more than 3 Oracle RAC nodes to register with, you will need to increase the property `oracle.ons.maxconnections` to the number of registrations (connections) that ONS needs to make.

Information on managing the ONS is found in Appendix A of this document. For both Oracle Application Server and ONS installed from Oracle Database 10g Client, you will need to update the configuration file.

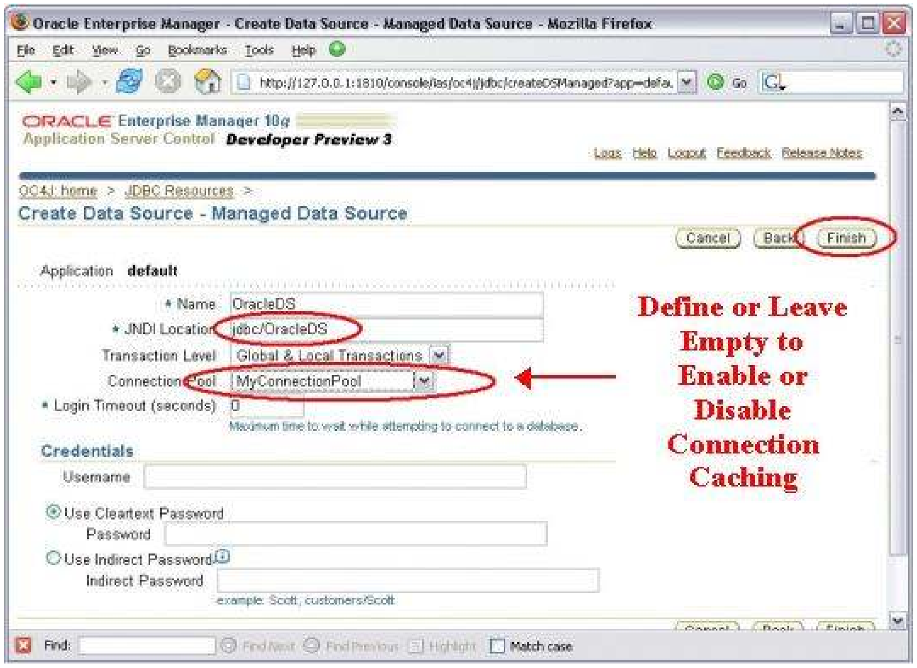
OC4J Data Sources

Oracle Containers for J2EE (OC4J) data sources integrate the new Implicit Connection Caching and Fast Connection Failover features in Oracle Database 10g JDBC. For Oracle AS 10g versions 9.0.4.x and 10.1.2, Implicit Connection Caching support is limited to native data sources. Other types of data sources (e.g., emulated and non-emulated data sources) do not support this feature and use an older form of caching. For Oracle AS 10g (9.0.4.x) versions, since the default Oracle JDBC driver is older than Oracle 10g, users must upgrade the default driver to at least Oracle Database 10g JDBC in order to take advantage of this feature.

Implicit Connection Cache

Configuring Implicit Connection Caching for these Oracle AS versions is mostly done in a declarative way by modifying the data-sources.xml file directly. The affected OC4J instance has to be restarted for the changes to take effect.

Oracle Application Server 10g 10.1.3 provides improved integration with FCF. OC4J 10.1.3 offers two simple types of data sources: managed data source and native data source. Implicit Connection Caching is supported in both types of data sources. The primary tool for configuring Implicit Connection Caching in OC4J 10.1.3 data sources is the user-friendly Oracle Enterprise Manager (EM) 10g Application Server Control Console. Central management of the data sources via the Console significantly lowers administrative cost. Standard JMX-based management supports the dynamic creation, deletion, and modification of both data source types, as well as the associated connection caches, without OC4J restart. Alternatively, Implicit Connection Caching can still be enabled or disabled declaratively within the data-sources.xml deployment descriptor file for any OC4J data source. Because the descriptor syntax has changed in 10.1.3, this would be different from how it is done in earlier Oracle Application Server versions. For more details on OC4J Data Sources with Implicit Connection Caching and Fast Connection Failover, see the article on OTN at
http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_data_sources/oc4j_ds.htm


Oracle Enterprise Manager - Create Data Source - Managed Data Source - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:1810/console/ias/oc4j/jdbc/createDSManaged?app=defa...
ORACLE Enterprise Manager 10g Application Server Control Developer Preview 3
Logs Help Logout Feedback Release Notes
OC4J: home > JDBC Resources >
Cancel Back Finish
Find: Find Next Find Previous Highlight Match case

Create Data Source - Managed Data Source

Application **default**

* Name	OracleDS
* JNDI Location	jdbc/OracleDS
Transaction Level	Global & Local Transactions
Connection Pool	MyConnectionPool
* Login Timeout (seconds)	0

Maximum time to wait while attempting to connect to a database.

Credentials

Username

Use Cleartext Password

Password

Use Indirect Password

Indirect Password

example: Scott, customers/Scott

Define or Leave Empty to Enable or Disable Connection Caching

Figure 10 Enterprise Manager Application Server Control

```
<connection-pool  name="myConnectionPool">
   <connection-factory
      factory-class="oracle.jdbc.pool.OracleDataSource"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@localhost:1521:oracle"/>
   </connection-factory>
</connection-pool>
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   description="Managed DataSource">
   connection-pool-name="myConnectionPool"/>
```

Add or Remove this Attribute to Enable or Disable Connection Caching

Figure 11 Sample data-sources.xml deployment descriptor

Fast Connection Failover (FCF)

When using "oracle.jdbc.pool.OracleDataSource" as the native data source, there is a common way in all Oracle AS versions to enable Implicit Connection Caching and Fast Connection Failover at the same time, with all the connection cache properties taking their default values. That is, setting the system property "oracle.jdbc.FastConnectionFailover" to true when launching an OC4J instance. For example,

```
java –Doracle.jdbc.FastConnectionFailover=true –jar oc4j.jar
```

Configuring Fast Connection Failover for these Oracle AS versions is done in a declarative way by modifying the data-sources.xml file directly. The affected OC4J instance has to be restarted for the changes to take effect. The Implicit Connection Cache must be enabled in order to use Fast Connection Failover.

To configure Fast Connection Failover, follow these two steps:

1. Specify "oracle.jdbc.pool.OracleDataSource" as the "class" attribute in a <data-source> element;
2. Specify the "fastConnectionFailoverEnabled" property with value "true" within the same <data-source> element.

```
<data-source
  class="oracle.jdbc.pool.OracleDataSource"
  name="OracleDS"
  location="jdbc/OracleCache"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="scott"
  password="tiger"
  url="jdbc:oracle:thin:@localhost:1521:orcl">
    <property  name="connectionCacheName"  value="ICC" />
    <property  name="connectionCachingEnabled"  value="true" />
    <property  name="fastConnectionFailoverEnabled"  value="true" />
</data-source>
```

For OC4J 10.1.3 data sources, declarative configuring of Fast Connection Failover is very similar to how it is done in earlier Oracle Application Server versions. Specifically, it can be done in two steps, depending on the data source type.

For managed data sources:

1. specify oracle.jdbc.pool.OracleDataSource as the factory-class attribute in a <connection-factory> element for a configured <connection-pool>;
2. specify the fastConnectionFailoverEnabled property with value true within the same <connection-factory> element.

```
<managed-data-source
  jndi-name="jdbc/ManagedDS"
  description="Managed DataSource">
  connection-pool-name="myConnectionPool"
  name="ManagedDS"/>
<connection-pool
  name="myConnectionPool"
  min-connections="10"
  max-connections="30"
  inactivity-timeout="30">
  <connection-factory
   factory-class="oracle.jdbc.pool.OracleDataSource"
   user="scott"
   password="tiger"
   url="jdbc:oracle:thin:@localhost:1521:oracle"/>
<property name="fastConnectionFailoverEnabled" value="true"/>
</connection-factory>
</connection-pool>
```

For native data sources:

1. specify oracle.jdbc.pool.OracleDataSource as the data-source-class attribute in a <native-data-source> element;

2. specify the fastConnectionFailoverEnabled property with value true within the same <native-data-source> element.

```
<native-data-source
  name="nativeDataSource"
  jndi-name="jdbc/nativeDS"
  description="Native DataSource"
  data-source-class="oracle.jdbc.pool.OracleDataSource"
  user="scott"
  password="tiger"
  url="jdbc:oracle:thin:@localhost:1521:oracle">
   <property name="connectionCacheName" value="ICC"/>
<property name="connectionCachingEnabled" value="true"/><property
 name="fastConnectionFailoverEnabled" value="true"/>
</native-data-source>
```

Using TAF/FAN with JDBC Thick Driver

If you would like to use TAF with the JDBC thick driver, then you should use the OCI integration with FAN instead of FCF. To use TAF, first set the TAF policy for the service with the srvctl command and set –q true. You can also take advantage of the OCI callback functionality with your application program. Do not use both JDBC FCF and TAF together.

```
srvctl modify service -d crm -s gl.us.oracle.com -q TRUE -P
BASIC -e SELECT -z 180 -w 5 -j LONG
```

Figure 12 Using SRVCTL to set TAF policy

Oracle Data Provider for .NET (ODP.NET)

For .NET applications, starting with Oracle Database 10g Release 2, ODP.NET provides the ability to take advantage of FAN events for high availability and connection pool load balancing. Oracle Data Provider for .NET (ODP.NET) connection pools subscribe to FAN notifications from RAC that indicate when nodes are down and when services are up or down. Based on these notifications, ODP.NET connection pools make idle connections, connections that were previously connected to nodes that failed, available again. It also creates new connections to healthy nodes if possible. In the case of a DOWN event, Oracle cleans up sessions in the connection pool that is connected to the instance that stops. It will then create new connections to an UP instance if it exists and the number of connections in the pool is below min_pool_size. The steps required to implement FAN with Oracle Database 11g Release 2 are:

1. Turn on AQ HA event notifications

   ```
   srvctl modify service -d crm -s gl.us.oracle.com -q TRUE -
   -j LONG –B THOUGHPUT
   ```

2. Grant permission to the application user(s) to de-queue the messages. This is the user who is connecting from the .NET application.

   ```
   execute
   dbms_aqadm.grant_queue_privilege('DEQUEUE','SYS.SYS$SERVICE_METRICS',
   <your user>);
   ```

3. Enable Fast Connection Failover for ODP.NET connection pool by subscribing to FAN HA events. Set the `HA Events` string to true. This can be done at connect time or in the data source definition. Note this will only work with if you are using connection pools, I.E. `"pooling=true"` attribute is set.

```
"user id=scott;password=tiger;data source=crm;HA events=true;"

// C#
using System;
using Oracle.DataAccess.Client;
class ConnectionPoolingSample
{
static void Main()
{
OracleConnection con = new OracleConnection();
//Open a connection using ConnectionString attributes
//related to connection pooling.
con.ConnectionString =
"User Id=scott;Password=tiger;Data Source=crm;" +
"Min Pool Size=10;Connection Lifetime=120;Connection Timeout=60;" +
"HA events=true", "Incr Pool Size=5; Decr Pool Size=2";
con.Open();
Console.WriteLine("Connection pool successfully created");
// Close and Dispose OracleConnection object
con.Close();
con.Dispose();
Console.WriteLine("Connection is placed back into the pool.");
}
}
```

For faster failover of the client, there are a couple of SQLNet parameters that should be set. Set the SQLNet Connect Timeout Property on the Client to specify the time, in seconds, for a client to establish an Oracle Net connection to the database instance. (DO NOT SET on Oracle RAC Server). If an Oracle Net connection is not established in the time specified, the connect attempt is terminated. The client receives an `ORA-12170:TNS:Connect timeout occurred` error. The value is seconds and should be set in the SQLNET.ORA.

```
       SQLNET.OUTBOUND_CONNECT_TIMEOUT = 5
```

Set the SQLNet TCP connet timeout to specify the time, in seconds, for a client to establish a TCP connection to the database server. If a TCP connection to the database host is not established in the time specified, the connect attempt is terminated. The client receives an `ORA-12170:TNS:Connect timeout occurred` error.

```
       TCP.CONNECT_TIMEOUT=3
```

NOTE: `TCP.CONNECT_TIMEOUT` must be less than `SQLNET.OUTBOUND_CONNECT_TIMEOUT`

With Oracle RAC 11g Release 2, you can specify the connect timeout parameter on the tnsnames entry instead of in the SQLNET.ORA (which affects all TNSNAMES entries).

```
sales.mycompany.com =(DESCRIPTION=
(CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
(ADDRESS_LIST= (LOAD_BALANCE=on)(FAILOVER=ON)
 (ADDRESS=(PROTOCOL=tcp)(HOST=scan1)(PORT=1521))
 (ADDRESS=(PROTOCOL=tcp)(HOST=scan2)(PORT=1521)))
  (CONNECT_DATA=
   (SERVICE_NAME= sales.mycompany.com)))
```

Figure 13 Sample TNSNAMES.ORA entry with CONNECT_TIMEOUT

ODP.NET provides runtime connection load balancing to provide enhanced load balancing of the application workload. Instead of randomly selecting an available connection from the connection pool, it will choose the connection that will provide the best service based on the current workload information. The steps required to implement connection pool load balancing, are:

1. Turn on event notifications and set a goal for your service:

   ```
   srvctl modify service -d crm -s gl.us.oracle.com -q TRUE –B THROUGHPUT
   ```

2. Enable Runtime Connection Load Balancing by subscribing to FAN Load Balancing events. This can be done at connect time or in the data source definition. Note this will only work with if you are using connection pools, I.E. `"pooling=true"` attribute is set.

   ```
   "user id=scott;password=tiger;data source=erp;load balancing=true;"
   ```

Oracle Call Interface

The Oracle Call Interface (OCI) provides integration with FAN HA and Load Balancing Advisory events with Oracle RAC 11g. To take advantage of the Load Balancing Advisory, you need to enable the OCI Session Pool. OCI clients can register to receive notifications about RAC high availability events and respond when events occur. This improves the connection failover response time in OCI and also removes terminated connections from connection and session pools. This feature works for all OCI client applications. OCI clients with transparent application failover (TAF) enabled are recommended to enable this feature for fast failover. Note: SQLPLUS is an OCI client and FAN can be enabled by using the –F (FAILOVER) parameter when starting SQLPLUS. On receipt of a down event for an instance or node, OCI will

- Terminate affected connections at the client
- Remove connections from the OCI connection pool and OCI session pool (The session pool maps each session to a physical connection in the connection pool. There can be multiple sessions per connection)
- If TAF is configured and it is a failure, the connection will failover, if not, the client receives an error such as ORA-12543
- If a TAF callback has been registered, then the failover retries and failover delay are ignored. If an error occurs, TAF will continue to attempt to connect and authenticate as long as the callback returns a value of OCI_FO_RETRY. Any delay should be coded into the callback logic.

Client applications must connect to a RAC instance to enable event notification. Clients that have enabled high availability event notification can optionally register client EVENT callbacks. This reduces the time that it takes to detect a connection failure.

For faster failover of the client, there are a couple of SQLNet parameters that should be set. Set the SQLNet Connect Timeout Property on the Client to specify the time, in seconds, for a client to establish an Oracle Net connection to the database instance. (DO NOT SET on RAC Server). If an Oracle Net connection is not established in the time specified, the connect attempt is terminated. The client receives an `ORA-12170:TNS:Connect timeout occurred` error. The value is seconds and should be set in the TNSNAMES.ORA or SQLNET.ORA.

```
TNSNAME.ORA:
      sales.mycompany.com =(DESCRIPTION=
      (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
      (ADDRESS_LIST= (LOAD_BALANCE=on)(FAILOVER=ON)
       (ADDRESS=(PROTOCOL=tcp)(HOST=scan1)(PORT=1521))
       (ADDRESS=(PROTOCOL=tcp)(HOST=scan2)(PORT=1521)))
        (CONNECT_DATA=
         (SERVICE_NAME= sales.mycompany.com)))
```

SQLNET.ORA:

```
      SQLNET.OUTBOUND_CONNECT_TIMEOUT = 5
```

Set the SQLNet TCP connet timeout to specify the time, in seconds, for a client to establish a TCP connection to the database server. If a TCP connection to the database host is not established in the time specified, the connect attempt is terminated. The client receives an `ORA-12170:TNS:Connect timeout occurred` error.

```
      TCP.CONNECT_TIMEOUT=3
NOTE:  TCP.CONNECT_TIMEOUT must be less than
SQLNET.OUTBOUND_CONNECT_TIMEOUT (or CONNECT_TIMEOUT)
```

Perform the following three steps to configure FAN notifications to an OCI client:

1. Configure the service at the server to set the value for the parameter `AQ_NOTIFICATIONS` to `TRUE`. For example:

```
srvctl modify service -d crm -s gl.us.oracle.com -q TRUE –B THROUGHPUT
```

2. Enable `OCI_EVENTS` at environment creation time, this tells Oracle you may be interested in HA events:

```
/* Need to init w/ OCI_EVENTS to receive HA events */

  if (checkerr(NULL, OCIInitialize((ub4) OCI_EVENTS, (dvoid *)0,
                     (dvoid * (*)(dvoid *, size_t)) 0,
                     (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                     (void (*)(dvoid *, dvoid *)) 0)))
    goto terminate;
```

3. In your program, you will need to check if an event has occurred, this code gets called when an HA event occurs:

```
void evtcallback_fn(ha_ctx, eventhp)
dvoid *ha_ctx;
OCIEvent *eventhp;
{
  OCIServer *srvhp;
  OCIError *errhp;
  sb4 retcode;
  OraText *hostname;
  OraText *dbname;
  OraText *instname;
  OraText *svcname;
  OCIDate timestmp;
  OCIEnv  *envhp = (OCIEnv *)ha_ctx;
  ub4 sizep;
  printf("HA event received.\n");
```

```
    if (OCIHandleAlloc( (dvoid *)envhp, (dvoid **)&errhp,
                     (ub4) OCI_HTYPE_ERROR,
                     (size_t) 0, (dvoid **) 0))
      return;
    if (retcode = OCIAttrGet(eventhp, OCI_HTYPE_EVENT, (dvoid *)&srvhp,
                             (ub4 *)0,
                             OCI_ATTR_HA_SRVFIRST, errhp))
      checkerr (errhp, (sword)retcode);
    else{
      printf("found first server handle.\n");
      /*get associated instance name, */
      if (retcode = OCIAttrGet(srvhp, OCI_HTYPE_SERVER, (dvoid *)&instname,
                               (ub4 *)&sizep,
                               OCI_ATTR_INSTNAME, errhp))
        checkerr(errhp, (sword)retcode);
      else
        printf("instance name is %s.\n", instname);
    }
    while(!retcode){
      if (retcode = OCIAttrGet(eventhp, OCI_HTYPE_EVENT, (dvoid *)&srvhp,
                               (ub4 *)0,
                               OCI_ATTR_HA_SRVNEXT, errhp))
        checkerr (errhp, (sword)retcode);
      else{
        printf("found another server handle.\n");
        /*get associated instance name, */
        if (retcode = OCIAttrGet(srvhp, OCI_HTYPE_SERVER, (dvoid *)&instname,
                                 (ub4 *)&sizep,
                                 OCI_ATTR_INSTNAME, errhp))
          checkerr(errhp, (sword)retcode);
        else
          printf("instance name is %s.\n", instname);
      }
    }
    OCIHandleFree((dvoid *)errhp, OCI_HTYPE_ERROR);
    printf("Finished event callback function.\n");
  }
```

4. Application must then decided what it wants to do when it receives an HA event

```
  /*Registering HA callback function. */
    if (checkerr(errhp, OCIAttrSet(envhp, (ub4) OCI_HTYPE_ENV,
                             (dvoid *)evtcallback_fn, (ub4)0,
                             (ub4)OCI_ATTR_EVTCBK, errhp)))
    {
      printf("Failed to set register EVENT callback.\n");
      return EX_FAILURE;
    }
    if (checkerr(errhp, OCIAttrSet(envhp, (ub4) OCI_HTYPE_ENV,
                               (dvoid *)envhp, (ub4)0,
                               (ub4)OCI_ATTR_EVTCTX, errhp)))
    {
      printf("Failed to set register EVENT callback context.\n");
      return EX_FAILURE;
    }
    return EX_SUCCESS;
  }
```

5. Client applications must link with the client thread or operating system thread library. I.E. libthread or libpthread

Runtime Connection Load Balancing with OCI Session Pools

Runtime connection load balancing is basically routing work requests to sessions in a session pool that best serve the work. It comes into effect when selecting a session from an existing session pool. Runtime connection load balancing is enabled by default with Oracle Database 11g Release 1 or higher client talking to a server of Oracle Database 10g Release 2 or higher. Setting the `mode` parameter to `OCI_SPC_NO_RLB` when calling `OCISessionPoolCreate()` disables runtime connection load balancing.

To utilize runtime connection load balancing with your OCI Session Pool, the following steps required:

- The application must have been linked with the threads library.
- The OCI environment must be created in `OCI_EVENTS` and `OCI_THREADED` mode.
- Configure the RAC service with a GOAL, CLB_GOAL , and `AQ_NOTIFICATIONS` to `TRUE`. For example:

```
srvctl modify service -d crm -s gl.us.oracle.com -q TRUE –B
THROUGHPUT –j SHORT
```

Conclusion

Oracle Real Application Clusters provides many features that enterprise applications can take advantage of for very high availability and scalability. The high availability of your application depends on your notification and repair policies for the computing environment running the application. On the database server, the RAC HA framework provides notifications of any change in the cluster configuration. The application can subscribe to FAN events and react quickly so their users can immediately take advantage of additional resources and are unaffected (or minimally affected) by a reduction in available resources. Applications are provided with greater flexibility to manage the various workloads that the database must execute within given service levels. Using the FAN callouts and FAN event system, repair processes are invoked immediately when the fault is detected, and applications including disaster recovery are integrated end to end, eliminating time-outs and wasted time spent retrying.

Appendix A Turning on Logging with JDBC

If you are testing your JDBC environment and want to see more detail of what is actually happening, you can turn on logging from your JDBC Data Source. Here are the steps with a sample properties file. The JDBC demo dir also has a sample logging properties file -- OracleLog.properties.

1. Use at least JDK 1.4
2. Use debug jar ojdbc14_g.jar from $ORACLE_HOME/jdbc/lib. I.E. rename the ojdbc14.jar (something like ojdbc14.jar_save, and then rename ojdbc14_g.jar to ojdbc.jar). This needs to go in your CLASSPATH

Include a properties file, with contents (be careful of the word line wrapping):

```
============
handlers= java.util.logging.ConsoleHandler
# default file output is in user's home directory
java.util.logging.FileHandler.pattern = jdbc.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
# Setting this to SEVERE avoids duplicate output from
#  default logger
java.util.logging.ConsoleHandler.level = SEVERE
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
oracle.jdbc.level = FINEST
oracle.jdbc.pool.level = FINEST
==============
```

Some output goes to StandardOut so you may want to redirect to a file.

3. Set the following when starting the test:

```
 ...-Doracle.jdbc.trace=true
-Djava.util.logging.config.file=<properties file location> ...
```

Your JDBC log should show:

- Calls to initFailoverParameters - this shows you that Fast Connection failover is enabled
- FCF eventType and eventBody (showing the full event) – this shows you JDBC got the event when a failure occured
- Calls to abortConnection – this shows you that JDBC is cleaning up the connections to the failed instance.

Automatic Workload Management with Oracle Real Application Clusters
January 2010
Author: Barb Lundhild
Contributing Authors: Carol Colrain, Troy Anthony, Daniel Semler, Rajkumar Irudayaraj

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

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