



RAC FAN Events

Yann Neuhaus . Senior Consultant
Urs Meier . Principal Consultant
08.09.2009

Note: Release 2.0 from this article, presenting some new statements:

- FAN event work with ODP .NET 10.2.0.20
- Better layout of the FAN architecture
- Presentation of service creation and configuration (DBMS_SERVICE)
- ONS configuration with racgons

1st Introduction

Since many years Oracle tries to find out a way to spread the load over the several instances of a Real Application Cluster. Until 10g Oracle only provided a "connect-time" load balancer. Moreover, Oracle was confronted to issues concerning the detection of instance outages. Functionalities like TAF (Transparent Application Failover) try to provide service continuity to the client. However, this works only for SELECT statements and in the case of a transaction the client must to handle instance failure autonomously.

2. Load balancing issues

The Oracle load balancer can be either configured on the client side (parameter LOAD_BALANCE=ON in tnsnames.ora) or on the server Side (parameter remote_listener). The client side load balancing algorithm allows spreading the client sessions over the several nodes, but the strategy is based on a "random" mechanism. This means that we cannot specify to which instance a client will connect. A client could eventually connect to an instance which is already under high load and therefore suffer from poor performances. Oracle took the issue into consideration and developed the server side load balancing algorithm. With this technology, each instance informs through the PMON process, the listeners located on the other nodes. The list of the listeners to inform is defined by the parameter "remote_listener" (example on INSTANCE1).

```
alter system set remote_listener=
' (ADDRESS_LIST=
 (ADDRESS=(PROTOCOL=TCP) (HOST=server2) (PORT=1521))
 (ADDRESS=(PROTOCOL=TCP) (HOST=server3) (PORT=1521))) '
scope=spfile sid='INSTANCE1'
```

The main drawback of these methods is that once a session is connected to an instance, it will never be able to "re-connect" to another instance, even if the load of the connected instance increases. Both load balancing methods are static ("connect time") load balancer. They do not take care of the CURENT load of the machine. This is often a problem for application servers, since they allocate their connection pool once and then constantly re-use the same connections without considering the current workload.



3. Failover issues

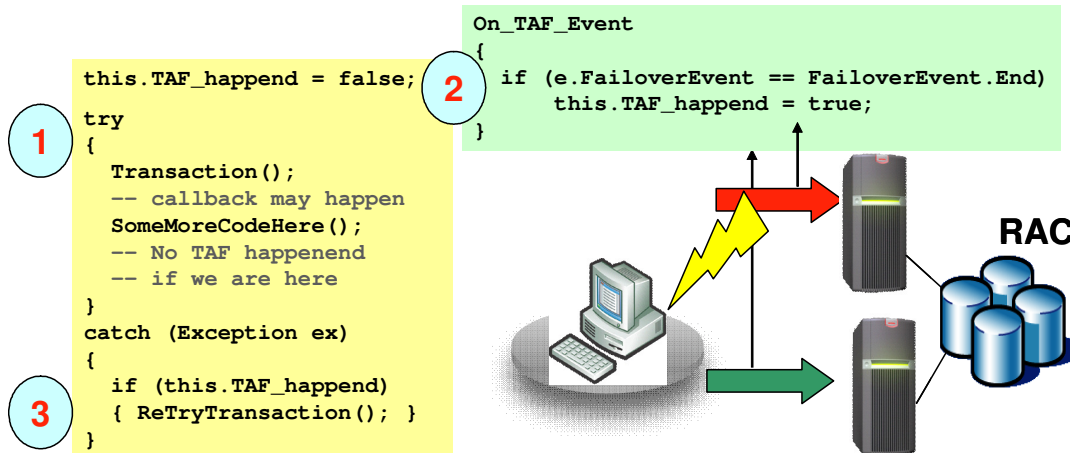
Transparent Application Failover (TAF) allows an Oracle session to automatically reconnect to another instance once the current instance fails. The result of a running SELECT will be returned to the session – even after an instance failure. In parallel Oracle had implemented a Virtual IP in order to detect quickly node failures. Without Virtual IP, an Oracle*Net session stays connected "tcp_keepalive" seconds until the failover occurs (default value for tcp_keepalive= 7200 (2hours) on most Operating Systems). However an opened transaction cannot be automatically fail over to another node. The application must be aware of the several ORA- errors in case of failure and restart its transaction. Note that all session settings (e.g. NLS) and package state must also be re-applied by the client logic.

Again this is often a problem for application servers using connection pools.

Once a program (i.e a java bean) requires a connection from the connection pool, it must make sure that the session is still available. For this purpose most of the Application Servers start each minute a "dummy" SQL statement against the database (i.e: select 1 from dual;) to keep the connections online and to force a TAF failover in case of instance crash. It could still be possible that a program gets a connection currently performing a failover at the moment of the "getConnection" method, or even a "bad" connection, which has not yet failed over.

TAF Event – not tough enough

At OCI level (therefore also available for JDBC and .Net) Oracle introduced the TAF Event. This is a callback event linked to a connection descriptor which starts a function in case of failover. We can therefore ask the status of the failover and wait until this operation completes. The program or even the transaction can therefore easily be restarted in an exception handler which is guaranteed to use a valid and "failed over" session.



Both the load balancing and failover possibilities are not satisfactory. First of all, the client could stay connected to an overloaded instance (load balance issue). Second, the client/application server must perform SQL statements to force failover (this could overload the database in large environments) or handle errors via the TAF Event once it tries to use an invalid connection.

The question is: Why do the RAC/Clusterware layer not inform the clients or application servers when the node/instances status changed (availability or load) ?



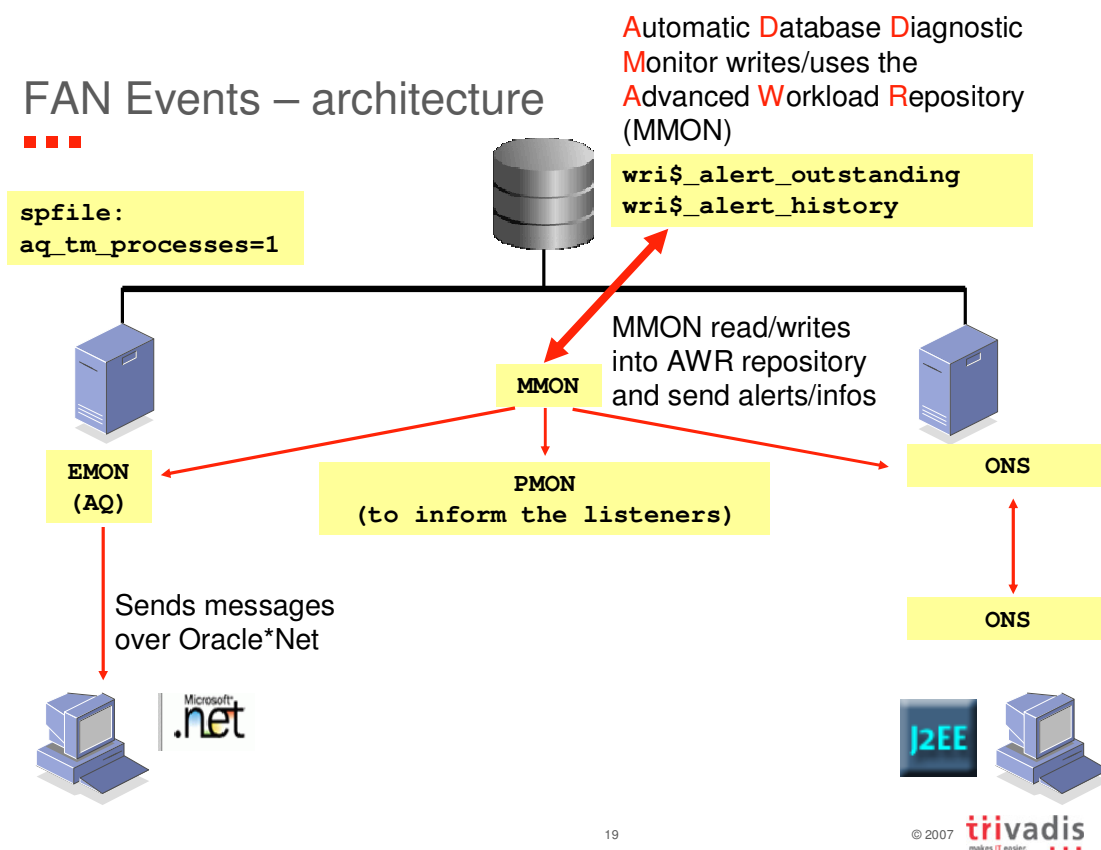
4th FAN Events

Notifying clients about the RAC availability and instance (actually service) performance is the purpose of the FAN (Fast Application Notification) events. The client is not actively checking the availability or load of an instance and is no more glued to an instance once connected. The nodes directly inform the application server about which instance is able to provide a defined "Quality Of Service".

We tested these FAN events on the ODP .Net and JDBC layer.

In order to limit the scope and the complexity of the article, we intentionally only focused on the workload FAN events and not on the availability FCF events (Fast Connection Failover). These events inform the clients about service presence or failure.

Below the FAN framework which will be detailed in the next chapters





5th Service definition

In order to use the FAN events an Oracle Service must be configured. This service is a kind of application or application type to which the clients can connect to. A service can run on one or more instances (preferred nodes) and could failover to one or many "available" instances. The Oracle clients will connect to a "service" but won't know on which node this service is started and won't know the defined service quality. All these settings are defined by the Database or Grid administrators, like presented below. The Transparent Application Failover parameters can also be defined on the server side for each service. This is the server side alternative to a tnsnames.ora configuration on the client and eases the deployment of TAF settings.

Such a service can be created with :

```
srvctl add service -d RAC_SITE1 -s OLTP -r RAC1,RAC2,RAC3 -a RAC4
```

A service called OLTP has been created on the instances RAC1, RAC2 and RAC3 (preferred instance : -r). In case of failure of one of these instances the service could also run on RAC4 (available instance).

The client will connect with the following Oracle*Net alias:

```
OLTP.TRIVADISTRAINING.COM =  
(DESCRIPTION=  
  (LOAD_BALANCE=ON)  
  (ADDRESS_LIST=  
    (ADDRESS=(PROTOCOL=TCP) (HOST=server1-v) (PORT=1521))  
    (ADDRESS=(PROTOCOL=TCP) (HOST=server2-v) (PORT=1521))  
    (ADDRESS=(PROTOCOL=TCP) (HOST=server3-v) (PORT=1521))  
    (ADDRESS=(PROTOCOL=TCP) (HOST=server4-v) (PORT=1521))  
  )  
  (CONNECT_DATA=  
    (SERVICE_NAME=OLTP) ) )
```

In a next step the service must be configured according to the application and infrastructure requirements with the package DBMS_SERVCE as presented below.

5th1 Connection load balance goal (CLB_GOAL)

The services have a defined Connection Load Balance algorithm (CLB_GOAL) which privileges either the long connected sessions (from a connection pool, CLB_GOAL=LONG) or shortly connected sessions (CLB_GOAL=SHORT).

5th2 Service Quality goal (GOAL)

Also, the services must be configured to either provide the best response time (SERVICE_TIME) or the better throughput (THROUGHPUT). With SERVICE_TIME Oracle will privilege the least loaded node which corresponds to the one able to answer faster, with THROUGHPUT the requests will be redirected to the node with the highest number of calls per second. In theory, the one being able to perform the highest transaction rate (may be because of a stronger machine in the cluster).



For instance, if the OLTP service is accessed through an application server and should respond as fast as possible to the incoming requests, configure it as follow:

```
EXEC DBMS_SERVICE.MODIFY_SERVICE(service_name=>'OLTP',CLB_GOAL=>
DBMS_SERVICE.CLB_GOAL_LONG);

EXEC DBMS_SERVICE.MODIFY_SERVICE(service_name=>'OLTP',goal=>
DBMS_SERVICE.GOAL_SERVICE_TIME);
```

6th FAN & ODP .Net

ODP.NET is based on OCI and reads the FAN event using the Oracle Advanced Queing (AQ) protocol.

This information is delivered by the MMON (Master Monitor) process which reads in the AWR table and informs the EMON (Advanced Queuing) process.

The FAN configuration for the ODP .Net layer needs advanced queuing to be configured on the server side :

```
aq_tm_processes=1
```

Also the HA AQ NOTIFICATION parameter must be enabled for this service:

```
EXEC DBMS_SERVICE.MODIFY_SERVICE (
SERVICE_NAME => 'OLTP', AQ_HA_NOTIFICATIONS => true);
```

The client connection pool must be setup with the following properties in the connection string:

```
Load balacing=true -- for the FAN workload events
HA Events=true -- for the FCF events
```

It is also **VERY IMPORTANT** to make sure that no firewall exists between the client and the RAC instances. The reason is that the Advanced Queuing framework returns the FAN event to an undefined port to the client. If a firewall exists, these event won't be caught by the .Net client and the application will perform a simple round robin session allocation over the several instances. The port used to receive the event will be configurable in the next ODP .Net releases.

Once load is generated on one of the instance, the ODP .Net layer reacts directly and avoids the loaded instance by distributing the statement over the other instances. The loaded instance is less, or no more used, the FAN events are reported correctly to the client. We used ODP.NET version 10.2.0.20

7th FAN & JDBC

After these encouraging results, the JDBC layer (jdbc thin driver 1.4 with Oracle jdbc driver 10.2.0.2.0) has been tested. The used service was the same. However, the JDBC layer uses another communication stack based on the ONS (Oracle Notification Server). This ONS is installed per default during a RAC/Clusterware setup on the server side. The configuration is done in the \$ORA_CRS_HOME/opmn/config/ons.config file or through the racgons binary which writes the Clusterware ONS configuration in the Clusterware repository OCR (Trivadis advised method). Also, on the middle tier side, ONS could be installed and configured with the same list of nodes (including the RAC and middle tier nodes). If it is not wished to install the ONS servers on the



middle tier, the JAVA program can register itself directly to the Oracle Clusterware ONS layer as presented below.

ONS is delivered by a “simple” Oracle client installation (developer version is enough).

Example of ONS configuration file required on all nodes (also middle tier)

```
oracle:~/product/crs-10.2.0/opmn/conf/ [crs1020] cat ons.config
localport=6100
remoteport=6200
loglevel=9
nodes=server1:6200,server2:6200, app_server1:6200
```

Another configuration possibility is to use the OCR (Oracle Cluster Repository) to store the ONS configuration. First of all configure ons.config with useocr=on:

```
oracle@server1:~/product/crs/opmn/ [crs] cat conf/ons.config
localport=6114
remoteport=6200
useocr=on
```

Then add the configuration in the OCR with racgons:

```
racgons add_config server1:6200 server2:6200 app_server1:6200
```

The java client subscribes to the ONS daemon process by specifying the property ‘oracle.ons.oraclehome’ through system.setProperty or with the ‘java -d’ option. You must also include the ons.jar in your CLASSPATH. The ONS daemon process has to be started with ‘onsctl start’.

```
private OracleDataSource _ods = null;

private void initialize()
{
    try {
        System.setProperty("oracle.ons.oraclehome", "C:\\oracle\\product\\10.2.0\\client_2");
        _ods = new OracleDataSource();
        _ods.setUser(AppProperties.getInstance().getUserName());
        _ods.setPassword(AppProperties.getInstance().getPassword());
        _ods.setURL(AppProperties.getInstance().getConnectionString());

        _ods.setConnectionCachingEnabled(true);
        _ods.setFastConnectionFailoverEnabled(true);

        Properties properties = new Properties();
        properties.setProperty("MinLimit", "30");
        properties.setProperty("MaxLimit", "100");
        properties.setProperty("InitialLimit", "30");
        _ods.setConnectionCacheName("MyCache");
        _ods.setConnectionCacheProperties(properties);
    }
}
```

Or use java -D

Enable implicit connection cache

It is also important to use the implicit JDBC connection pool manager in order to make the



connection pool aware of the FAN events. This is simply done by calling `setConnectionCachingEnabled` (for load balancing) and `setFastConnectionFailover` (for re-balancing connections on instance up/down) on the `OracleDataSourceObject`.

Starting with Oracle 10g Release 2, java clients can subscribe to ONS without ONS being installed on the middle tier. This is called 'remote subscription' and is performed by calling 'setONSConfiguration' on the `OracleDataSource` instance. 'setONSConfiguration' accepts the same configuration string (`nodes=server1:6200,server2:6200`) as we have seen with `ons.config` file.

With remote subscription you don't need to install ONS. However, you don't have any trace and debug methods of ONS either.

Our tests simulated a JDBC application server running an application against a RAC cluster. Until Oracle 10g there were no possibilities to distribute the sessions of a connection pool on a regular basis over the several instances (i.e : 10-10-10 on a 3 nodes RAC with 30 connections in the connection pool). We could use the client side `tnsnames.ora` parameter `load_balance=on` which is based on "random" method, therefore not predictable. With Oracle RAC 10g the service can be created with a Connection Load Balancing Goal (CLB_GOAL) set to LONG as presented above. Used in combination with the `remote_listener` parameter it allows to build up a connection pool well distributed over the several instances.

It is very important to precise that all our tests were performed with this combination of `LOAD_BALANCE=ON` on the client side and `remote_listener` on the server side. If the service is configured with `CLB_GOAL=LONG`, the sessions will be spread out anyway over the several nodes according to the number of opened sessions. However if the SHORT method is wished, it means that the sessions should get the least loaded node AT CONNECTION TIME (because they just connect, do something and disconnect). For this purpose the parameter `remote_listener` will take the lead by finding the least loaded node (therefore it must be configured). The client side parameter `LOAD_BALANCE=ON` just allows to avoid to overload always the same listener with connection requests. With `LOAD_BALANCE=OFF`, all the clients would try to connect to the RAC in same order: (e.g. `listener1`, `listener2`, `listener3`, and so on ...) So we can easily imagine that the listener 1 will be quite fast overloaded by connection requests.

Also note that `LOAD_BALANCE` is per default set to ON for an Oracle*Net descriptor.

By setting the trace level to 9 in the `ons.config` file, we can observe the percentage of the load which can be accepted by each instance. Without load on the RAC server, each instance accepts 33 percent of the charge. As soon as load is generated on the instance1 (RAC1), using hammerora for example, the percentage of requests accepted by this instance will strongly decrease:

```
VERSION=1.0 database=RAC10_SITE1 { {instance=RAC3 percent=49  
flag=GOOD}{instance=RAC2 percent=46 flag=GOOD}{instance=RAC1  
percent=6 flag=GOOD} } timestamp=2006-09-22 10:49:00
```

In this example RAC1 (with load) accepts only 6% of the requests against the database and the other instances accept: 49% for RAC3 and 46 for RAC2.

Our Java program has been written to use a connection pool spread over the three instances of the cluster. The connection pool manager (`OracleDataSource` object) regularly checks to which instance a connection from the connection pool is connected. As soon as the load on the instance 1 (RAC1) has been signaled by the ONS, the JDBC connection pool delivered connections from the instance RAC1 less often, but still keeps the connection open. As soon as this instance was totally overloaded, no more connection against this instance has been used at all.



Below the screenshots of the Java and .Net applications we used to test the connection pools while load was generated on RAC1:

RACSimpleTest

Settings
Connection String
jdbc:oracle:thin:@(DESCRIPTION =(ADDRESS_LIST =(FAILOVER = ON)(LOAD_BALANCE = ON)(ADDRESS = (PROTOCOL = TCP)(HOST = cougar-b-v.trivadistraini

Iterations: 900 Frequency (secs): 2 Run Tests

Session Distribution

Instance	Sessions
RAC3/cougar-b	7
RAC2/tiger-b	8
RAC1/lion-b	15

Runtime Load Balancing

Instance	Count
RAC1	65
RAC3	412
RAC2	423

RAC Simple Test

Settings
Connection String
Data Source=DOTNET.TRIVADISTRAINING.COM ; User ID=dotnet; Password=dotnet; Min Pool Size=30; HA events=true; load balancing=true;

Iterations: 900 Frequency (secs): 2 Run Tests

Session Distribution

INSTANCE_NAME	HOST_NAME	COUNT(*)
RAC3	cougar-b	12
RAC2	tiger-b	10
RAC1	lion-b	8

Runtime Load Balancing

instance_name	total
RAC1	14
RAC3	446
RAC2	440

Conclusion

The JDBC and ODP .Net behavior with the FAN events seems quite encouraging and really allows us to think about a real RUNTIME load balancer. The Oracle*Net layer was not able to manage the load distribution dynamically therefore Oracle shifted this responsibility to the OCI, .Net and JDBC layers (this article covered .Net and JDBC). Oracle will need some time to fix out all the small issues in order to provide a perfectly working layer, specially concerning some issues with the FCF (Fast Connection Failover) events, out of scope in this article, but the technological trend in which Oracle is moving is very interesting.

In order to get the best load balancing we recommend to combine the client side mechanism (tnsnames.ora load_balance=on), the server side mechanism (remote_listener) and the FAN events with services.

Do you feel the Grid :-)



More about RAC, FAN and FCF events, please join the Trivadis Real Application Clusters lesson.

Yann Neuhaus

Trivadis AG

Elisabethenanlage 9

CH-4051 Basel

Internet: www.trivadis.com

Tel: +41-44-808 70 20

Fax: +41-44-808 70 21

Mail: yann.neuhaus@trivadis.com

Urs Meier

Trivadis AG

Europa-Strasse 6

CH-8152 Glattbrugg (Zürich)

Internet: www.trivadis.com

Tel: +41-44-808 70 20

Fax: +41-44-808 70 21

Mail: urs.meier@trivadis.com