http://www.coincoin.fr.eu.org/?FAN-vs-TAF

FAN vs TAF

- 1- Blog-Notes - Au boulot -

Date de mise en ligne : mercredi 1er décembre 2010

Non ni TAF ni FAN(FCF) ne rejouent les transactions DML (insert/update/delete) il faut les reprogrammer, mais ce n'est pas une obligation. Seul TAF permet la reconnection automatique

du client et uniquement sur une Query si configuré (pas DML) FAN ne le fait pas car FAN est un mecanisme programmatique

TAF permet le rejeu d'un select automatiquement si configuré, mais en réalité ne sert qu'en DataWarehouse lors de select conséquents. Je le configure ici pour éviter sa reprogrammation

donc il sera utile et utilisé pour les transactions de type Query

Une totale transparence transactionnelle est l'idéal mais il faut bien voir que sa programmation n'est pas des plus simple dans les deux cas (FAN ou TAF) : il faudrait que les developpeurs

par exemple puissent encapsuler une connexion database dans une classe spéciale pour les transaction en failover et dans une autre classe pour une logique sans failover s'ils veulent

categoriser les transactions à rejouer par exemple.

De plus la gestion des événements doit être maîtrisée et elle ne pourra se faire qu'au prix de tests sur une plateforme "crashable".

Je ne pense pas que la décision du rejeu d'une transaction dépende uniquement du développement mais soit de plus haut niveau.

Que cela soit TAF ou FAN la non prise en compte d'un code retour s'l 'est pas correctement trappée par l'exception correspondante terminera le programme.

La base commence toujours par défaire(rollback) les changements non commités (l' instance restante) une erreur sera remontée au client qui devrait à son tour faire son propre rollback.

Une seconde erreur est remontee pour indiquer que la connexion est perdue. Cette erreur là devra être trappée si l'on souhaite rejouer la transaction.

FAN

La difference entre TAF et FAN est notée ici :

– Application-level connection retries

Fast Connection Failover supports application-level connection retries. This gives the application control of responding to connection failovers. The application can choose whether to retry the connection or to rethrow the

exception. TAF supports connection retries only at the OCI/Net layer. => Avec FCF c'est à la charge du developpeur de se reconnecter. Avec TAF c'est automatique.

– Integration with the implicit connection cache

Fast Connection Failover is well-integrated with the implicit connection cache, which allows the connection cache manager to manage the cache for high availability. For example, failed connections are automatically invalidated in the cache. TAF works at the network level on a per-connection basis, which means that the connection cache cannot be notified of failures => ???

– Event-based

Fast Connection Failover is based on the Oracle RAC event mechanism. This means that Fast Connection Failover is efficient and detects failures quickly for both active and inactive connections.

– Load-balancing support

Fast Connection Failover supports UP event load balancing of connections and run-time work request distribution across active Oracle RAC instances.

FAN repose sur ONS (oracle notification service) qui permet la notification des évnemenents remonté par EVMD. ONS est deja configure sur les noeud du cluster et nécessite

une configuration supplementaire sur le middle pour être activé.

Il faut noter une difference essentielle entre TAF et FAN à savoir que FAN remonte les informations d'etat directement au niveau applicatif (push) alors

que TAF se contente de faire du polling (client). FAN est donc beaucoup plus réactif. FAN peut neanmoins être integre avec TAF

Pour programmer FAN il est possible d'utiliser soit C soit Java. Ensuite on peut utiliser soit ONS API C soit ONS Java API soit

implicit cache connection de JDBC et FCF et laisser la gestion des evenement FAN par la librairie JDBC (jdbc)(JDBC-OCI ou jdbc thin) ce qui evite d'avoir a gerer l'API ONS directement.

Dans le dernier cas il y a toujours une gestion des evenements mais FAN mais est plus "souple" à programmer.

DBCP fonctionne avec Oracle Cache connection et FCF (Fast Connection Failover) dans sa version actuellement implantée sur nos serveurs. Il faut activer implict cache connection et FCF.

TAF

On peut néanmoins s'éviter la programmation FAN en utilisant les callback et l'interface OracleOCIFailover pour programmer le rejeu du DML.

Il faut implementer la fonction callback fourni par la classe OracleOCIFailover

Dans une transaction classique :

Registrer la fonction callback : => registerTAFCallback()

```
 // Fonction qui crée une nouvelle connexion
void handleDBConnections() {
    try {
              DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
              con = (OracleConnection)
DriverManager.getConnection("jdbc:oracle:oci:@DBDATA","user","passwd");
              if (con != null)
                    this.RegisterFailover();
    } catch (SQLException e) {
          cat.debug("Error Occurred while registring failover.");
          e.printStackTrace();
    }
}

// Fonction qui execute une query
void runQuery() {
    ......
    try {
        String query = "insert into essai values (1)";
       rs = stmst.executeQuery(query);
    } catch (SQLException e) {
        if  ((e.getErrorCode() == 1012) || (e.getErrorCode() == 1033) ||  // Ces codes d'erreur sont
relatif à une node failure ou connexion failure
             (e.getErrorCode() == 1034) || (e.getErrorCode() == 1089) ||
             (e.getErrorCode() == 3113) || (e.getErrorCode() == 3114) ||
             (e.getErrorCode() == 10203) || (e.getErrorCode() == 12500) ||
             (e.getErrorCode() == 12571) || (e.getErrorCode() == 25408))
              cat.debug("Node Failed pendant l'execution d'une instruction INSERT/DELETE/UPDATE");
               //rollback
              rollback();
              // Obtenir une nouvelle connection
              handleDBConnections(); <= on redemande une nouvelle connexion
              // Réexecuter la transaction un certain nombre de retry a programmer soi meme
              runQuery();
    } else
      // L'erreur n'est pas une node failure
          rollback();
    }
}
```

Enfin la fonction callback (que j'ai remis dans un exemple complet mais dont la premiere partie n'est la que pour

montrer le register de la callback)

La seconde partie montre l'implantation de la callback.

L'implementation de la callback n'est pas obligatoire si on ne veut pas rejouer du DML mais elle peut être là pour logger le comportement du failover OU permettre

de redemander une connexion en cas de probleme du failover ou bien d'aborter la demande. Dans ce cas precis FO_ERROR = 5 l'application peut redemander une reconnexion

du nombre de fois spécifié par le parametrage de tnsnames.ora (RETRIES) je pense et sleep pendant le temps indique dans le programme

si specifié (ou parametre DELAY ?)

La doc oracle n'etant pas tres precise et prolixe en la matiere cela est à tester.

Les points d'interrogation sont le comportement du programme en cas FO_ERROR = 5. Le comportement de TAF est de recreer une connexion automatiquement en cas de failover sur une query pas sur DML . Dans le cas ou l'on implémente une Query c'est TAF via tnsnames qui automatiquement recrée une connexion mais rien

n'interdit d'aborter le retry dans le cas FO_ERROR = 5, il suffit de ne pas envoyer un FO_RETRY.

L'exemple que je fournis est sur du DML et c'est à la charge du programme de redemander une connexion via la fonction handleDBConnections()) si souhaité

Dans ce cas je pense aussurement que le parametre RETRIES et DELAY ne sont plus geres par tnsnames mais par l'application qui doit elle meme gerer ses compteurs de retry et de sleep

Encore une fois on peut aborter le nombre de retries dans le cas du FO_ERROR si souhaité.

Enfin la doc n'est pas tres claire s'il faut fournir un rollback au niveau applicatif ou pas (comme je l'ai indiqué dans la fonction runQuery())

```
-  * This sample demonstrates the registration and operation of
-  * JDBC OCI application failover callbacks
-  * Note: Before you run this sample, set up the following
-  *       service in tnsnames.ora:
-  *       inst_primary=(DESCRIPTION=
-  *             (ADDRESS=(PROTOCOL=tcp)(Host=hostname)(Port=1521))
-  *             (CONNECT_DATA=(SERVICE_NAME=ORCL)
-  *                          (FAILOVER_MODE=(TYPE=SELECT)(METHOD=BASIC))
-  *             )
-  *           )
-  *       Please see Net8 Administrator's Guide for more detail about
-  *       failover_mode
-  *
-  * To demonstrate the the functionality, first compile and start up the sample,
-  *    then log into sqlplus and connect /as sysdba. While the sample is still
-  *    running, shutdown the database with "shutdown abort;". At this moment,
-  *    the failover callback functions should be invoked. Now, the database can
-  *    be restarted, and the interrupted query will be continued.
-  *
-  */
- // You need to import java.sql and oracle.jdbc packages to use
- // JDBC OCI failover callback
- import java.sql.*;
- import java.net.*;
- import java.io.*;
- import java.util.*;
- import oracle.jdbc.OracleConnection;
- import oracle.jdbc.OracleOCIFailover;
- import oracle.jdbc.pool.OracleDataSource;
- public class OCIFailOver {
-   static final String user = "hr";
-   static final String password = "hr";
-   static final String URL = "jdbc:oracle:oci8:@inst_primary";
-   public static void main (String[] args) throws Exception {
-     Connection conn = null;
-     CallBack   fcbk= new CallBack();
-     String     msg = null;
-     Statement  stmt = null;
-     ResultSet rset = null;
-     // Create a OracleDataSource instance and set properties
-     OracleDataSource ods = new OracleDataSource();
-     ods.setUser(user);
-     ods.setPassword(password);
-     ods.setURL(URL);
-     // Connect to the database
-     conn = ods.getConnection();
-     // register TAF callback function
-     ((OracleConnection) conn).registerTAFCallback(fcbk, msg);
-     // Create a Statement
-     stmt = conn.createStatement ();
-     for (int i=0; i<30; i++) {
-       // Select the NAMEs column from the EMPLOYEES table
```

```
-       rset = stmt.executeQuery ("select FIRST_NAME, LAST_NAME from EMPLOYEES");
-       // Iterate through the result and print the employee names
-       while (rset.next ())
-         System.out.println (rset.getString (1) + " " +
-                             rset.getString (2));
-       // Sleep one second to make it possible to shutdown the DB.
-       Thread.sleep(1000);
-     } // End for
-     // Close the RseultSet
-     rset.close();
-     // Close the Statement
-     stmt.close();
-     // Close the connection
-     conn.close();
-   } // End Main()
- } // End class jdemofo
- /*
-  * Define class CallBack
-  */
- class CallBack implements OracleOCIFailover {
-    // TAF callback function
-    public int callbackFn (Connection conn, Object ctxt, int type, int event) {
-      /*************************************************************************
-       * There are 7 possible failover event
-       *   FO_BEGIN = 1   indicates that failover has detected a
-       *                  lost connection and faiover is starting.
-       *   FO_END = 2     indicates successful completion of failover.
-       *   FO_ABORt = 3   indicates that failover was unsuccessful,
-       *                  and there is no option of retrying.
-       *   FO_REAUTH = 4  indicates that a user handle has been re-
-       *                  authenticated.
-       *   FO_ERROR = 5   indicates that failover was temporarily un-
-       *                  successful, but it gives the apps the opportunity
-       *                  to handle the error and retry failover.
-       *                  The usual method of error handling is to issue
-       *                  sleep() and retry by returning the value FO_RETRY
-       *   FO_RETRY = 6
-       *   FO_EVENT_UNKNOWN = 7  It is a bad failover 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";
-      }
-      switch (event) {
-        case FO_BEGIN:
```

```
-          System.out.println(ctxt + ": "+ failover_type + " failing over...");
-          break;
-        case FO_END:
-          System.out.println(ctxt + ": failover ended");
-          break;
-        case FO_ABORT:
-          System.out.println(ctxt + ": failover aborted.");
-          break;
-        case FO_REAUTH:
-          System.out.println(ctxt + ": failover needs reauthorization");
-          break;
-        case FO_ERROR:
-          System.out.println(ctxt + ": failover error gotten. Sleeping...");
-          // Sleep for a while of 2s
-          try {
-            Thread.sleep(2000);
-          }
-          catch (InterruptedException e) {
-            System.out.println("Thread.sleep has problem: " + e.toString());
-          }
-          return FO_RETRY;
-        default:
-          System.out.println(ctxt + ": bad failover event.");
-          break;
-      }
-      return 0;
-    }
- }

- Sample code to test FCF

- The code sample below shows the way how FCF works, it may not necessarily be the best way to do, but it
should give you a general idea of what is needed.
Make sure to change the following strings in the program, , and <service_name>.

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;
import oracle.jdbc.pool.OracleDataSource;
public class FCFDemo {
  public staticvoid main(String[] args) throws InterruptedException {
    try {
      OracleDataSource ods = new OracleDataSource();
      ods.setUser("scott");
      ods.setPassword("tiger");
      String dbURL="jdbc:oracle:thin:@"
                 +"(DESCRIPTION=" +
                  "(ADDRESS=(PROTOCOL =TCP)" +
                  "(HOST=)(PORT=1521))" +
                  "(ADDRESS=(PROTOCOL =TCP)" +
```

```
                "(HOST=)(PORT=1521))" +
                "(LOAD_BALANCE=ON)(CONNECT_DATA=(SERVICE_NAME=<service_name>)))";
      ods.setURL(dbURL);
      ods.setConnectionCachingEnabled(true);
      ods.setConnectionCacheName("cache");
      Properties prop = new Properties();
      prop.setProperty("MinLimit", "5");
      prop.setProperty("MaxLimit", "40");
      prop.setProperty("InitialLimit", "10");
      ods.setConnectionCacheProperties(prop);
      ods.setFastConnectionFailoverEnabled(true);
      Connection conn = ods.getConnection();
      System.out.println(conn);
      Statement stmt=null;
      ResultSetrs = null;
      try {
        while (true) {
          rs =stmt.executeQuery("select instance_name from v$instance");
          while(rs.next()) {
            System.out.println("Instance name: " + rs.getString(1));
          }
          Thread.sleep(1000);
        }
      }
      catch (SQLException sqle) {
        conn =ods.getConnection(); //Re-get the conn
        stmt =conn.createStatement();
      }
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
 #!/bin/sh
count=0
while [ $count -lt 5 ]                                  # Set up a loop control
do                                                      # Begin the loop
    count=expr $count + 1 # Increment the counter
   sqlplus -s scott/tiger@R2d1C@test.sql
done

col "Instance" format a20
col "Host" format a20
col "Service Name" format a20
select sys_context('userenv', 'instance_name') "Instance",
      sys_context('userenv', 'server_host') "Host",
      sys_context('userenv', 'service_name') "Service Name"
from dual
/
exit;
```

Note :

Driver-type dependency : TAF is in fact a OCI failover mechanism exposed to Java through JDBC-OCI. FCF is driver-type independent (i.e., works for both JDBC-Thin and JDBC-OCI).