Connecting and using RAC

Martin Schmitter
Trivadis GmbH
Düsseldorf

Schlüsselworte

RAC, FAN, TAF, FCF, DB, SERVICE, LOAD BALANCING, JDBC, ODP, ORACLE, CONNECTION.

Einleitung

In vielen Kundenprojekten hat sich herausgestellt, dass die Einführung von Real Application Clusters (RAC) hohe Anforderungen an die Infrastruktur und DBAs stellt. Eine Herausforderung, die jedoch bei diesen Betrachtungen meistens übersehen wird, ist die Implementierung oder Anpassung der Applikationen.

Verbindungsmethoden und Datenbankservices müssen entsprechend konfiguriert und ausgewählt werden, um mit den Applikationen die Lastverteilungs- und Failover Funktionen wie z.B. Transparent Application Failover (TAF), Fast Application Notification (FAN) und Fast Connection Failover (FCF) effizient nutzen zu können.

Das Projekt endet nicht mit der Bereitstellung der Infrastruktur!

In diesem Artikel werden zunächst die notwendigen Anpassungen auf der Datenbankseite erläutert sowie eine Brücke zu den Anwendungsentwicklern und den Applikationsverantwortlichen geschlagen. Diese sollen in die Lage versetzt werden, die Möglichkeiten der Hochverfügbarkeits- Features zu bewerten und diese auf Applikationsseite einsetzen zu können.

Anhand der Erfahrungen aus zahlreichen Projekten und den daraus gewonnenen Best Practices, werden mit einfachen Beispielen die Möglichkeiten für Applikationen, die den ODP.Net - OCI oder JDBC Thin Treiber einsetzen, aufgezeigt.

Anmerkung:

Aufgrund der vielen Möglichkeiten und Unterschiede in den Datenbank-Versionen, wird in diesem Artikel hauptsächlich die Oracle Version 11gR2 mit einer Administrator Managed RAC Datenbank betrachtet. Diese sollte, aktuell die am weitesten verbreitete Variante sein. Eine Transformation auf andere Szenarien, z.B. Policy Managed DBs, ist ohne Probleme möglich.

Zur besseren Darstellbarkeit der Konfigurationsparameter, wird in den Beispielen auf eine Darstellung mit SCAN Adressen verzichtet. Im Prinzip ist das Verwenden von SCAN Adressen innerhalb der Version 11gR2 kein Problem und vereinfacht die Administration enorm.

Übersicht:

Oracle RAC bietet viele Vorteile im Vergleich zu einer einfachen Single Instance Datenbank.

Die wesentlichen Vorteile für eine Anwendung sind:

- Eine mögliche Lastverteilung über die vorhandenen Cluster Knoten,
  d.h. eine geschickte Verteilung der Last (aktiv/aktiv).
- Die Möglichkeit des Failovers
  Sollte ein Knoten ausfallen, kann auf einem anderen Knoten weitergearbeitet werden.

Diese Merkmale sind aber nicht bereits aktiviert, sondern müssen konfiguriert werden sowie den Datenbank Clients bereits bei der Verbindung zur Datenbank mitgegeben werden.

Ein klassischer TNS String für eine Verbindung zu einer Datenbank könnte folgendermaßen aussehen:

```
ORCL.FOO.BAR =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = foo.bar)(PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = ORCL.FOO.BAR)
   )
 )
```

In einem Cluster Umfeld würde man mehrere Knoten einsetzen. Das bedeutet, dass der Client darüber Informationen besitzen muss, welche Hosts existieren. Sollte einer der Hosts nicht erreichbar sein, wird auf einen verfügbaren umgeschwenkt. Das lässt sich in der Verbindungsbeschreibung entsprechend einstellen:

```
RACDB.FOO.BAR =
(DESCRIPTION =
  (ADDRESS_LIST =
  (FAILOVER = ON)
   (ADDRESS = (PROTOCOL = TCP)(HOST = foo1-vip.foo.bar)(PORT = 1521))
   (ADDRESS = (PROTOCOL = TCP)(HOST = foo2-vip.foo.bar)(PORT = 1521))
  )
 (CONNECT_DATA = (SERVICE_NAME = ORCL.FOO.BAR)
 )
)
```

Nachteilig ist im Beispiel, dass immer nur auf den ersten Knoten verbunden wird. Nur im Fehlerfall, also wenn der erste Cluster-Knoten nicht erreichbar ist, wird der zweite Knoten verwendet. Da der zweite Knoten jedoch auch aktiv ist, sollte man diesen ja auch nutzen können und somit die Last über zwei Cluster-Knoten verteilen.

```
RACDB.FOO.BAR =
(DESCRIPTION =
  (ADDRESS_LIST =
  (FAILOVER = ON)
  (LOAD_BALANCE=ON)
   (ADDRESS = (PROTOCOL = TCP)(HOST = foo1-vip.foo.bar)(PORT = 1521))
```

```
    (ADDRESS = (PROTOCOL = TCP)(HOST = foo2-vip.foo.bar)(PORT = 1521))
  )
 (CONNECT_DATA = (SERVICE_NAME = orcl.foo.bar)
 )
)
```

Der Load Balancing Parameter aktiviert nur ein zufälliges Load Balancing (Round Robin), ohne Rücksicht auf die aktuelle Last zu nehmen. Es könnte also passieren, dass man trotzdem mit einem Server verbunden wird, der eine besonders hohe Last hat.

So betrachtet ist man sehr flexibel. Jedoch haben die bis jetzt beschriebenen Möglichkeiten einige Nachteile:

1. Die Einstellungen sind nur beim Verbindungsaufbau relevant (Connection Time).
2. Failover und Load Balancing werden clientseitig konfiguriert und können auf anderen Clients vergessen werden. Änderungen müssen aufwändig administriert werden.
3. Der Failover findet nur beim Verbindungsaufbau statt. Eine einmal verbundene Session wird im Fehlerfall nicht automatisch neu auf einen verbleibenden Cluster-Knoten verbunden. Gegebenenfalls kann die Applikation unter den geänderten Umständen nicht weiter betrieben werden und muss neu gestartet werden. Ein gutes Exception-Handling ist damit obligatorisch.
4. Das Load Balancing findet nur zur Verbindungszeit (Connect Time) statt. Auf sich verändernde Umgebungsbedingungen wird nicht eingegangen. Gerade für Applikationsserver mit Connection Pools, die Verbindungen über einen langen Zeitraum aufrechterhalten, ist das ein Problem. Verändert sich die Last auf den Servern, werden die Verbindungen nicht lastabhängig neu verteilt.

Failover automatisch! Geht das?

Wäre es nicht besser, wenn bei einem Serverausfall die Sessions automatisch auf einem anderen Knoten neu erzeugt werden?

Kein Problem! Solch eine Möglichkeit gibt es schon! Sie nennt sich Transparent Application Failover (TAF).

```
RACDB.FOO.BAR =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (FAILOVER=ON)
      (LOAD_BALANCE=ON)
      (ADDRESS = (PROTOCOL = TCP)(HOST = foo1-vip.foo.bar )(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = foo2-vip.foo.bar )(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = ORCL.FOO.BAR )
      (FAILOVER_MODE =
         ( TYPE = SELECT )
         ( METHOD = BASIC )
         ( RETRIES = 3600 )
         ( DELAY = 1 )
         ( ENABLE = BROKEN )
      )
    )
  ))
```

TAF ermöglicht es einer Oracle Session bei einem Verbindungsfehler, automatisch einen Reconnect auf eine andere Instanz durchzuführen. Ein Select-Statement kann sogar an der Stelle des Abbruchs wieder aufgenommen werden. Der User muss im besten Fall nur etwas länger auf das Ergebnis warten.

Jedoch können offene Transaktionen nicht automatisch neu gestartet werden. Die Applikation muss hier auf die ORA- Fehler Meldungen reagieren und die richtigen Transaktionen neu ausführen. Auch gehen die Session Parameter und NLS-Settings verloren und müssen neu gesetzt werden. Eine Transaktion in diesem Sinne ist eine Oracle Operation die dem ACID Prinzip genügt. Business-Transaktionen hingegen sind auf Applikationsseite auf Konsistenz zu überprüfen.

Für Database Links lässt sich das Feature leider nicht einsetzen.

Wird ein Connection Pool verwendet, gibt es keine aktive Kommunikation von der Datenbank an den Pool. Der Pool öffnet z.B. 20 Verbindungen. Fordert nun ein Client eine der Verbindungen aus dem Pool an, wird irgendeine Verbindung durch den Pool zurückgegeben. Die Instanz könnte jedoch mittlerweile nicht mehr erreichbar sein. Trotzdem wird versucht eine Verbindung aufzubauen. Erst die tatsächliche Verwendung der Verbindung führt zum Fehler und erst nun erhält der Pool davon Kenntnis und setzt die Verbindung zurück. In der Applikation muss die entsprechende Exception gefangen werden und eine neue Verbindung angefordert werden.

Wie kann man der Situation entgegenwirken?

Bei der Verwendung eines Oracle OCI Treibers kann der Parameter `ENABLE=BROKEN` bei der proaktiven Erkennung von gestörten Verbindungen unterstützen. Ohne den Parameter können bis zu 2 Stunden bis zum TCP-Timeout (abhängig vom gesetzten Timeout) vergehen.

Verwendet man die VIP-Ressource der Oracle Clusterware, wird die Fehlererkennung über die Cluster Software geregelt. Bei einem Netzwerkfehler wird die VIP auf einem anderen Knoten gestartet und der ARP Cache auf dem Switch zurückgesetzt. Daher gibt es das Problem des langen Wartens auf den TCP-Timeout an dieser Stelle nicht.

Der Oracle Data Provider for .Net (ODP .Net) bietet für Microsoft .Net darüber hinaus einen Event Handler an, den der Anwendungsentwickler verwenden kann, um den Status eines Failover abzufragen. So hat man die Möglichkeit den Failover abzuwarten und kann eine validierte Verbindung nutzen um die Transaktion erneut auszuführen.

Wenn ein Oracle JDBC Thin Treiber verwendet wird, kann das TAF Feature nicht verwendet werden. Es ist hierfür schlichtweg nicht implementiert. Wird auf Applikationsserver-Seite ein Connection Pool eingesetzt, wird in der Praxis ein Dummy-Select ausgeführt, um proaktiv die Verbindungen zu validieren. Im Fehlerfall kann der Pool seine Verbindungen neu sortieren.

Dummy Select für Thin JDBC (Tomcat mit Connection Pool):

```
<Resource name="jdbc/myDatabase" auth="Container"
 type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
 url="jdbc:oracle:thin:@
(DESCRIPTION=(ADDRESS_LIST=
(ADDRESS=(PROTOCOL=TCP)(HOST=10.0.0.1)(PORT=1521))
(ADDRESS=(PROTOCOL=TCP)(HOST=10.0.0.2)(PORT=1521))
(LOAD_BALANCE=yes))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl_jdbc)
))"
username="SCOTT" password="TIGER" maxActive="100" maxIdle="10"
```

```
maxWait="-1" validationQuery="SELECT 1 FROM DUAL" testOnBorrow="true"/>
```

Bis jetzt haben wir uns weitestgehend nur die Client-Seite angeschaut. Im Idealfall würden sich die Instanzen untereinander austauschen und über aktuelle Last und Verfügbarkeit informieren. Darüber hinaus könnte man doch serverseitig die Verbindungsparameter für die Clients und Session hinterlegen. So müsste man nicht mühselig alle Clients einzeln konfigurieren, was sehr fehleranfällig ist. Auch würde es dann nicht mehr passieren, dass ein Client sich auf die am stärksten belastete Instanz verbindet, wenn man an zentraler Stelle die Belastungsinformationen hinterlegt und den Clients zur Verfügung stellt.

Sie ahnen es bereits… Gibt es schon!

Fast Application Notification (FAN)

FAN Events informieren die Clients über Konfigurationsänderungen (z.B. Cluster Node up/down, Instanz up/down und Database up/down) und die aktuelle Auslastung. Die Informationen werden via Advanced Queuing oder über den Oracle Notification Service (ONS) bereitgestellt. Clients können diese Informationen auswerten und darauf reagieren.

FAN kann man nicht wie mit einem Schalter einschalten, sondern wird aus einer Reihe von zu konfigurierenden Bedingungen bereitgestellt. Unter anderem muss das serverseitige Load Balancing mit dem DB Parameter Remote Listener aktiviert werden. Der ONS Service muss überprüft und für den Oracle Data Provider das Advanced Queuing eingeschaltet werden. Die notwendigen Schritte werden später anhand von zwei Beispielen vorgestellt.

Bei FAN geht es in der Hauptsache darum, Informationen über den Zustand der Instanzen bereitzustellen.

Fast Connection Failover (FCF)

Fast Connection Failover kann man sich als die Client Implementation, für die Verarbeitung der FAN Events im Connection Pool vorstellen.

Der Connection Pool, kann dann FAN Events auslesen und darauf weitestgehend automatisch reagieren. Bereits fertige Implementierungen des FCF werden von Oracle für Java und .Net bereitgestellt, so dass nicht jedes Event manuell und mit viel Aufwand verarbeitet werden muss. Mögliche Lösungen sind hier zum Beispiel der Universal Connection Pool (UCP) und der Oracle Data Provider (ODP) für Microsoft .Net. Beide werden von Oracle auf der Oracle Webpage bereitgestellt.

Eine notwendige Bedingung, um FCF nutzen zu können, ist der Einsatz von Oracle RAC oder Oracle Restart/HAS mit einer Single Instance Datenbank in Verbindung mit einem Connection Pool.

Hier ein kleiner Auszug, was FCF zu leisten vermag:

- Im Fehlerfall werden gestörte Verbindungen umgehen erkannt und vom Pool entfernt bzw. neu auf verwendbare Verbindungen verteilt. Aktive Transaktionen erhalten eine Exception.
- Geplante Unterbrechungen können mittels FCF und dem Pool Manager verwaltet werden. Aktive Verbindungen (Transaktionen) werden erst geschlossen, wenn alle Arbeit getan ist. Erst dann werden diese freigegeben, bzw. beendet.
- Der Status von Verbindungen kann mittels API `(isValid)` abgefragt werden.
- Neue Instanzen werden unmittelbar erkannt und in den Pool integriert. Wird eine Instanz entfernt, werden die Verbindungen automatisch auf die verbleibenden Knoten verteilt.

- Abhängig von der aktuellen Auslastung und der Load Balancing-Algorithmen werden Verbindungen über alle Instanzen verteilt.

Serverseitige Parametrierung

Die oben genannten Features, wie z.B. TAF, können sowohl clientseitig als auch serverseitig konfiguriert werden. Für die Verwendung des FAN/FCF-Features ist jedoch eine Konfiguration auf der Server-Seite unumgänglich.

TAF kann auf Client-Seite aber auch auf Server-Seite via DB Service konfiguriert werden.

FAN muss mittels DB Service konfiguriert werden.

Viele Dinge sprechen für eine serverseitige Konfiguration, z.B. will man sicherstellen, dass sich alle Clients mit den richtigen Parametern verbinden. Parameteränderungen sind unmittelbar aktiv und können an zentraler Stelle umgesetzt werden. Unterschiede und/oder fehlerhafte Konfigurationen auf den Clients werden vermieden.

Die serverseitige Konfiguration funktioniert mittels der Datenbank Services. Ein Datenbankservice kann spezielle Eigenschaften annehmen und an die Sessions weitergeben. Für uns ist an dieser Stelle besonders relevant, dass die Möglichkeit besteht, die Verbindungsparameter an die Client Sessions zu übergeben und das serverseitige Load Balancing zu konfigurieren. Die Servicekonfiguration überschreibt die Clientkonfiguration auf Session Ebene. Datenbankadministratoren haben so die Möglichkeit alle wesentlichen Einstellungen an zentraler Stelle zu hinterlegen und können die eine oder andere Nachlässigkeit bei der Clientkonfiguration ausgleichen.

Z.B. könnte man die TAF Konfiguration, die wir als Client Konfiguration im Beispiel gesehen haben auch über einen Datenbank Service einrichten:

```
srvctl add service -d orcl -s orcl_taf.foo.bar -r ORCL1, ORCL2 \
                   -P BASIC -e SELECT -m BASIC -z 3600 -w 1
```

Ab der Oracle Version 11gR2 ist die Verwendung des PL/SQL Packages DBMS_SERVICES veraltet. Die Konfiguration sollte ab 11gR2 nur noch mittels `SRVCTL` vorgenommen werden, damit diese konsistent in den Eigenschaften der Cluster Ressourcen und letztendlich in der Cluster Registry sind. In den vorherigen Versionen waren noch nicht alle Optionen mittel `SRVCTL` verfügbar, sie waren fehlerhaft, mangelhaft oder gar nicht implementiert. Daher wurde der Service mittels `SRVCTL` angelegt und im Anschluss mittels DBMS_SERVICE parametriert.

Ob die Service-Einstellungen Wirkung zeigen, kann man über die `gv$session` View überprüfen:

```
select
  schemaname,
  osuser,
  machine,
  program,
  failover_type,
  failover_method,
  failed_over
from gv$session
where service_name = 'orcl_taf.foo.bar';
```

```
SCHEMANAME  OSUSER MACHINE    PROGRAM              FAILOVER_TYPE FAILOVER_METHOD FAILED_OVER
----------  ------ ---------- -------------------- ------------- --------------- -----------
```

```
SCOTT       foobar appserver1 tigershop.service.exe SELECT        BASIC           NO
SCOTT       foobar appserver2 tigershop.service.exe SELECT        BASIC           NO
SCOTT       foobar appserver1 tigershop.service.exe SELECT        BASIC           NO
SCOTT       foobar appserver2 tigershop.service.exe SELECT        BASIC           NO
```

Serverseitiges Load Balancing

Mit setzen des Datenbank Parameters `REMOTE_LISTENER` aktiviert man das serverseitige Load Balancing. Jede DB Instanz kann damit über den PMON Prozess den Listener auf den anderen Knoten über die aktuelle Auslastung informieren.

```
alter system set remote_listener=
'(ADDRESS_LIST=
(ADDRESS=(PROTOCOL=TCP)(HOST=foo2-vip.foo.bar)(PORT=1521))'
 scope=spfile sid='ORCL1';
```

Zum Zeitpunkt des Verbindungsaufbaus (Connect Time) kann nun ein Load Balancing mit Hilfe der zur Verfügung gestellten Auslastungsinformationen der Instanzen erfolgen. Wir sprechen nach wie vor vom Connect Time Load Balancing, welches mit dem DB Service Parameter `CLB_GOAL=LONG (default)|SHORT` beeinflusst werden kann.

Verwendet man homogene Abfragen und Transaktionen, oder einen Connection Pool mit lang anhaltenden Verbindungen, will man die Verbindungen in der Regel gleichmäßig über alle Instanzen verteilen. Der Connection Pool erzeugt stellvertretend für die Clients eine bestimmte Anzahl von Verbindungen. Welche Aufgaben übermittelt werden, wie lange diese dauern und welche Ressourcen diese benötigen, ist nicht vorhersehbar. Es wird daher versucht, die Sessions gleichmäßig über alle Instanzen zu verteilen.

Das SHORT Attribute ist für heterogene Aufgaben geeignet. Kurze unterschiedliche Aufgaben wie sie z.B. bei einem Web Shop auftreten. Neue Verbindungen werden auf die Instanz mit der geringsten Auslastung gelenkt. Die Aufgabe wird in möglichst kurzer Zeit erledigt und die Verbindung wieder geschlossen.

Um die FAN Features mit einem Connection Pool nutzen zu können, wird nicht nur das Connect Time Load Balancing und Failover aktiviert, sondern außerdem muss auch das Runtime Connection Load Balancing mittels Load Balancing Advisory Goal (`GOAL`) eingestellt werden. Damit wird dem Client Connection Pool ermöglicht, während der Laufzeit auf veränderte Auslastung zu reagieren und inaktive Verbindungen im Pool neu zu organisieren.

```
desc dba_services;

Name                 Null Type
-------------------- ---- -------------
SERVICE_ID                NUMBER
NAME                      VARCHAR2(64)
NAME_HASH                 NUMBER
NETWORK_NAME              VARCHAR2(512)
CREATION_DATE             DATE
CREATION_DATE_HASH        NUMBER
FAILOVER_METHOD           VARCHAR2(64)
FAILOVER_TYPE             VARCHAR2(64)
FAILOVER_RETRIES          NUMBER(10)
FAILOVER_DELAY            NUMBER(10)
MIN_CARDINALITY           NUMBER
MAX_CARDINALITY           NUMBER
```

```
GOAL                           VARCHAR2(12)
DTP                            VARCHAR2(1)
ENABLED                        VARCHAR2(3)
AQ_HA_NOTIFICATIONS            VARCHAR2(3)
CLB_GOAL                       VARCHAR2(5)
EDITION                        VARCHAR2(30)
```

Bei der Service Qualität (Load Balancing Advisory Goals) wird nach den Kriterien der schnellsten Reaktionszeit und der maximalen möglichen Durchsatzrate unterschieden (`GOAL=SERVICE_TIME|THROUGHPUT|NONE`).

Die Service Time-Metrik zielt darauf ab, wie schnell ein Task beendet werden kann. Insbesondere wird hier auf die Reaktionszeit der Datenbank Wert gelegt. Die Load Balancing Advisory Daten basieren auf der verstrichenen Zeit, die mit dem Service benötigt wird und der zur Verfügung stehenden I/O-Bandbreite. Die Service Time Metrik ist am besten für Online Transaction Processing (OLTP) Anwendungen geeignet. Hier gibt es viele heterogene Verbindungen mit unterschiedlichen Laufzeiten.

Die Throughput-Metrik eignet sich im Gegensatz dazu, besonders für Data Warehouses (DWH) und Batch-Verarbeitung. Die Metrik überwacht ebenfalls, wie schnell ein Datenbank-Task beendet werden kann. Darüber hinaus werden aber auch die Leistungsfähigkeit der einzelnen Instanzen und die zur Verfügung stehende Prozessorzeit überwacht. Wird eine neue Aufgabe übermittelt, wird diese an die Instanz mit der am meisten zur Verfügung stehenden Prozessorzeit und der höchsten I/O-Leistungsfähigkeit weitergeleitet.

Beispiel einer Service Konfiguration mit der Service Time-Metrik für OLTP und dem `CLB_GOAL=LONG` für langfristige Verbindungen mit einem Connection Pool.

```
srvctl add service -d orcl -s orcl_rlb.foo.bar -r ORCL1,ORCL2 \
      -B SERVICE_TIME -j LONG
```

Wir haben bis zum jetzigen Zeitpunkt eine Menge über die Grundlagen des Verbindungsaufbaus und der Parametrisierung von Client- oder Server-Seite gelernt. Wir wissen bereits, wie wir Failover- und Load Balancing-Mechanismen konfigurieren und bereitstellen können, jetzt ist es an der Zeit, diese auch zu nutzen.

Man könnte die FAN Events mittels verschiedener Methoden auslesen und für die Applikation bereitstellen. Diese sind hinreichend im OCI Referenz Guide dokumentiert. Man könnte aber auch einfach etwas nutzen, dass bereits in der Lage ist, die FAN Events zu interpretieren und automatisch darauf reagiert (FCF). Hierfür bieten sich Connection Pools an. Es gibt zurzeit zwei Lösungen, die sich förmlich aufdrängen. Die Eine ist die Implementation mit dem Oracle Data Provider for Net (ODP.Net), die Andere ist der Universal Connection Pool (UCP), der sich im Schwerpunkt an JAVA-Lösungen mit dem OJDBC Thin Treiber richtet. Beide Lösungen nutzen grundsätzlich unterschiedliche Methoden und müssen daher auch unterschiedlich parametriert werden. Darauf wird im den folgenden Beispielen eingegangen.

Oracle Data Provider for .Net (ODP.Net)

Der Oracle Data Provider stellt auf Basis des Oracle Call Interface (OCI) einen FCF-fähigen Connection Pool zur Verfügung. ODP findet in .Net Projekten Anwendung und bietet einen vereinfachten Zugriff auf die Oracle Datenbank an.

Für uns sind zwei wesentliche Features relevant. Zum einen das Runtime Connection Load Balancing und zum anderen das Fast Connection Failover.

Das Runtime Connection Load Balancing verteilt Verbindungen über alle RAC Instanzen auf Basis des Load Balancing Algorithmus (Advisor) und des Service GOALs zur Laufzeit. Siehe dazu auch serverseitiges Load Balancing.

FCF gibt automatisch benutzte Verbindungen und Ressourcen im Connection Pool frei, die durch gestorbene oder heruntergefahrene Datenbanken, DB Services oder Knoten nicht mehr benutzt werden können. Ohne dieses Feature würde der Connection Pool die gestörten Verbindungen nicht aus dem Pool entfernen und Administratoren müssten bei jedem Fehler den Pool manuell zurücksetzen.

Vorbereitung auf Server-Seite

Um diese Features nutzen zu können, müssen zunächst auf Server-Seite einige Bedingungen erfüllt werden.

Neben der bereits besprochen Datenbank Service Konfiguration werden die FAN Events via Advanced Queue bereitgestellt. Dieses geschieht mittels des MMON (Master Monitor) Prozess, der die notwendigen Informationen aus dem Automatic Workload Repository (AWR) ausliest. Diese werden dann an den EMON (Advanced Queuing) Prozess übergeben.

Daher darf der DB Parameter `AQ_TM_PROCESSES` nicht auf 0 stehen.

```
alter system set AQ_TM _PROCESSES=1;
```

Die FAN Events werden über die Streams Queue Tabellen ALERT_QT und SYS$SERVICE_METRICS_TAB bereitgestellt. .Net und OCI Applikationen können sich registrieren, um diese Informationen zu bekommen.

In der Datenbank muss die Option `AQ_HA_NOTIFICATIONS` aktiviert werden. Erst dann erscheinen die Events in der Queue `SYS.SYS$SERVICE_METRICS` und können von dem ODP Client verwendet werden.

```
-q <AQ HA notifications> (TRUE | FALSE)

srvctl add service -d orcl -s orcl_odp –r orcl1,orcl2\
-P basic -e select -m basic -z 3600 -w 1 \
-q TRUE -B SERVICE_TIME -j LONG
```

Mittels des SELECTs erfolgt die Gegenprobe:

```
select enq_time, user_data from SYS.sys$service_metrics_tab;
…
26.08.13 10:41:44,887738000  SYS.SYS$RLBTYP('orcl_odp.foo.bar','VERSION=1.0
database=ORCL service=orcl_odp.foo.bar {
{instance=ORCL2 percent=38 flag=GOOD aff=TRUE}
{instance=ORCL4 percent=22 flag=GOOD aff=FALSE}
{instance=ORCL3 percent=33 flag=GOOD aff=TRUE}
{instance=ORCL1  percent=9  flag=GOOD  aff=TRUE}  }  timestamp=2013-08-26
12:41:44')

…
```

Die unterschiedlichen Zeitangaben kommen dadurch zustande, dass `ENQ_TIME` in `UTC` angegeben ist und `timestamp` der `local time` entspricht.

Der Wert `percent` stellt die Verteilung an die Cluster Knoten dar. Im Beispiel wird die Situation eines 4-Knoten-Clusters beschrieben. Die größte Auslastung ist auf dem Knoten mit der Instanz ORCL1. Dieser nimmt nur 9 Prozent alle neuen Verbindungen entgegen. Instanz ORCL2 ist am wenigsten ausgelastet und nimmt 38 Prozent der neuen Verbindungen entgegen. Dieses kann mit einem einfachen SELECT überprüft werden:

```
select inst_id, count(*)
from gv$session where
service_name = orcl_odp.foo.bar'
group by inst_id order by inst_id;

   INST_ID   COUNT(*)
---------- ----------
         1          2
         2         18
         3         16
         4          6
```

Da aktive Verbindungen nicht neu verteilt werden und auch andere DB Services und Applikationen auf dem Cluster betrieben werden können, ist die Abfrage nur geeignet eine Tendenz festzustellen!

Im Beispiel werden die bereits besprochen TAF-Features ebenfalls mitkonfiguriert. TAF kann mit FAN/FCF kombiniert werden. TAF beschleunigt die Erkennung von Verbindungsfehlern und sollte daher genutzt werden. Die Verwendung ist aber nur mit OCI-fähigen Client-Verbindungen verwendbar. Für JDBC Thin kann, wie schon beschrieben, TAF nicht verwendet werden.

.Net und OCI Applikationen können sich nun an dem Service registrieren und die FAN Events auslesen. Dazu muss den Applikationsusern aber noch die Berechtigung auf die Queue gegeben werden.

```
BEGIN
 DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('DEQUEUE','SYS.SYS$SERVICE_METRICS',
'SCOTT');
END;
/
```

Client-Seite

Für die Client Verbindung zur Datenbank kann man die Verbindungs-Parameter mittels TNSNAMES.ORA-Eintrag bekannt geben. Es spricht aber auch nichts dagegen, den Connect String direkt als `Data Source`-Attribute zu übermitteln. Da auf dem Service die TAF- und Load Balancing-Parameter gesetzt sind, braucht man diese im Connect String nicht zu berücksichtigen. Wichtig ist nur den richtigen Service auszuwählen und das Connect Time Load Balancing einzuschalten. Das Connect Time Load Balancing verhindert, dass alle neuen Client Verbindungen immer mit dem ersten Knoten verbunden werden. Dieses würde den Listener bei einem Failover überlasten, wenn z.B. mehrere hundert Verbindungen sich auf einen Schlag neu verbinden würden. Die Verwendung von SCAN löst dieses Problem im Übrigen auch.

```
RACDB.FOO.BAR =
  (DESCRIPTION =
    (ADDRESS_LIST =
```

```
      (FAILOVER=ON)
      (LOAD_BALANCE=ON)
      (ADDRESS = (PROTOCOL = TCP)(HOST = foo1-vip.foo.bar )(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = foo2-vip.foo.bar )(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = orcl_odp.foo.bar )
      )
    )
  ))
```

Nun steht einem Verbindungsaufbau aus der Applikation nichts mehr im Wege. In dem Beispiel sehen wir einen Verbindungsaufbau in C#. Die Attribute `HA Events` und `Load Balancing` schalten FCF und Realtime Connection Load Balancing ein. Voraussetzung ist die Verwendung von Connection Pooling. Connection Pooling wird mit dem Attribute `Pooling` aktiviert und ist im Standard eingeschaltet.

```
// C#
using System;
using Oracle.DataAccess.Client;

class ConnectionSample
{
 static void Main()
 {
   OracleConnection con = new OracleConnection();

   //using connection string attributes to connect to Oracle Database
con.ConnectionString = "User Id=scott;Password=tiger;Data Source=RACDB; HA
Events=true;Load Balancing=true";
   con.Open();
   Console.WriteLine("Connected to Oracle" + con.ServerVersion);

   // Close and Dispose OracleConnection object
   con.Close();
   con.Dispose();
   Console.WriteLine("Disconnected");
 }
}
```

Erhält man beim Verbindungsaufbau einen ORA-01017, fehlt dem User möglicherweise das Zugriffsrecht auf die Queue Table `SYS.SYS$SERVICE_METRICS`.

Wir nutzen nun die RAC Features TAF und FAN mit FCF mit dem Oracle Data Provider und einem Connection Pool!

JDBC Thin mit UCP

Der Universal Connection Pool (UCP) ist der neue Oracle Connection Pool mit Unterstützung für FAN und FCF. Er basiert auf dem JDBC Thin Treiber und hat keine OCI Implementierung, d.h., dass die OCI-Funktionen nicht verwendet werden können. Wie könnte also eine Kommunikation über die Belastung und den Cluster Status erfolgen, wenn es keinen Zugriff auf die FAN Events über OCI gibt?

Die Lösung ist einfacher als es vielleicht den Anschein hat. Man hängt sich einfach in die Cluster Kommunikation, d.h. in den Mechanismus den der Cluster nutzt um Information auszutauschen, den

Oracle Notification Service (ONS). Der ONS Service wird bei der Cluster Installation automatisch als Daemon installiert und konfiguriert.

Mit dem Befehl `ORACLE_HOME/opmn/bin/onsctl` kann der Daemon gesteuert und konfiguriert werden. Eine Überprüfung, ob der Daemon erreichbar ist, wäre wie folgt möglich:

```
>onsctl ping
```

ONS is running

Sollte der Daemon nicht erreichbar sein, kann man mittels:

```
>onsctl start/stop
```

versuchen, den ONS zu starten. Alternativ lässt sich die Konfiguration in der Datei `ORACLE_HOME/opmn/conf/ons.config` überprüfen:

```
>cat ORACLE_HOME/opmn/conf/ons.config.
localport=6100                                          # line added by Agent
allowgroup=true
usesharedinstall=true
remoteport=6200                                         # line added by Agent
nodes=foo1:6200,foo2:6200,foo3:6200,foo4:6200           # line added by Agent
```

Der aktuelle Status des ONS Daemons kann mittels `ONSCTL DEBUG` detailliert abgefragt werden.

Ab 11gR2 ist die Konfiguration des ONS dynamisch. Der Clusterware-ONS-Agent schreibt einige Parameter direkt in die ons.config. Dieses ist an dem Merkmal „`# line added by Agent`" erkennbar. Der Kommandozeilen-Befehl-`SRVCTL` löst den `RACONS` Befehl für die Konfiguration der Ports in der Oracle Cluster Registry (OCR) ab und führt konsequent den Ansatz fort, alle Cluster relevanten Einstellungen möglichst über ein Interface zu steuern.

Musste in der Vergangenheit der Parameter `useocr=true` in der ONS Konfiguration gesetzt werden, ist dieser nun veraltet und die Konfiguration wird immer an die OCR übertragen. Letztendlich wird damit erreicht, dass die ONS Konfiguration gegenüber dem Cluster jederzeit transparent ist und neue Cluster Knoten hinzugefügt oder entfernt werden können und somit alle beteiligten Knoten voneinander wissen.

```
>srvctl config nodeapps -s
ONS exists: Local port 6100, remote port 6200, EM port 2016
```

Nachdem der ONS verifiziert ist, muss auf der Server-Seite ein Datenbank Service für die Interaktion mit dem Connection Pool auf der Client-Seite konfiguriert werden. Im Gegensatz zur ODP .Net Lösung kann beim JDBC Thin Treiber nicht auf die TAF Features zurückgegriffen werden. Auch wird kein Zugriff auf die Advanced Queue und die `sys.sys$service_metrics_tab` erfolgen, daher müssen die AQ_HA_NOTIFICATIONS nicht aktiviert werden.

Beispiel Service Konfiguration:

```
srvctl add service -d orcl -s orcl_jdbc –r orcl1,orcl2 –a orcl3 -B
SERVICE_TIME -j SHORT
```

Es gibt ein bekanntes Problem mit dem Service-Namen. Dieser ist aus Sicht des UCPs case-sensitive und sollte daher immer klein geschrieben werden. Der UCP fragt diverse Metadaten, wie z.B. den Service Namen, ab und verknüpft diese mit den FAN Events. Ist dort der Service Name nicht exakt gleich geschrieben, funktioniert dieses möglicherweise nicht.

```
select name from dba_services;

NAME
---------------
SYS$BACKGROUND
SYS$USERS
ORCL
orcl_jdbc
```

Auch beim JDBC Thin wird für das Connect Time Load Balancing beim Aufbau des UCPs der REMOTE_LISTENER_Parameter (siehe Serverseitiges Load Balancing) benötigt. Ist der Pool erst einmal initialisiert, wird mit Hilfe der Information der FAN Events weiter gearbeitet.

Client-Seite:

Für das Setup des UCPs auf Applikationsservern werden drei Dateien benötigt. Die Dateien `ojdbc6.jar` (JDK 1.6) für den JDBC Treiber und `ucp.jar` für den Universal Connection Pool. Diese Dateien können auf der Webseite von Oracle heruntergeladen werden.

Darüber hinaus, wird die `ons.jar` Datei zur Kommunikation mit dem ONS benötigt. Diese ist Bestandteil des ORACLE_HOMEs der GRID-Infrastructure und kann einfach herauskopiert werden.

Alle Dateien müssen über den Java Classpath referenziert und für die Applikation erreichbar sein.

Fast Connection Failover ist abhängig vom ONS, um Datenbank Events zu empfangen. Dieses kann auf zwei Weisen geschehen. Auf Client-Seite kann ein eigener ONS Daemon installiert werden, der sich am DB Server registriert. Der bevorzugte Weg ist aber eine Remote Konfiguration, d.h. der Connection Pool verbindet sich direkt mit dem ONS auf DB Server Seite.

Applikationen, die eine Remote-Konfiguration verwenden, müssen das `oracle.ons.oraclehome` System Property auf den Pfad des `ORACLE_HOME` auf der RAC Seite setzen.

```
java -Doracle.ons.oraclehome=$ORACLE_HOME ...
```

Nun sind wir an dem Punkt, dass aus der Applikation der Connection Pool initialisiert werden kann:

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionPoolName("fcf_pool");
pds.setFastConnectionFailoverEnabled(true);
pds.setONSConfiguration("nodes=foo1:6200,foo2:6200\nwalletfile=
/oracle11/onswalletfile");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin@(DESCRIPTION= "+
"(LOAD_BALANCE=on)"+
"(ADDRESS=(PROTOCOL=TCP)(HOST=foo1-vip) (PORT=1521))"+
```

```
"(ADDRESS=(PROTOCOL=TCP)(HOST=foo2-vip) (PORT=1521))"+
"(CONNECT_DATA=(SERVICE_NAME=orcl_jdbc)))");
```

FCF wird mit dem Parameter aktiviert:

```
pds.setFastConnectionFailoverEnabled(true);
```

Die FAN Events können, wie bei dem ODP .Net Treiber, ausgegeben werden. Diese werden in das `$GRID_IF_HOME/opmn/logs/ons.log` geschrieben. Das Trace Level muss dafür auf den Wert 9 gesetzt werden.

Eine genaue Auflistung aller benötigten Java Klassen und ein vollständiges Beispiel wäre an dieser Stelle zu umfangreich. Eine detaillierte Anleitung kann man jedoch dem Oracle Universal Connection Pool for JDBC Developer's Guide entnehmen.

Zusammenfassung

Wir haben einen ersten Eindruck gewonnen, was möglich und nötig ist, um die Hochverfügbarkeits-Features der Oracle Datenbank zu nutzen. Wir können verschiedene Arten der Lastverteilung nutzen. Wir können fehlgeschlagene Sessions identifizieren und automatisiert auf einem Cluster Knoten neu verbinden. Wir können laufende `SELECT`s wiederholen, ohne dass der Client etwas davon mitbekommt. Wir können einen Connection Pool automatisiert aufräumen und Verbindungen, Last bezogen, neu verteilen.

All diese Features können aber nur in Zusammenarbeit zwischen DBA und Anwendungsentwickler realisiert werden. Es nützt nichts, wenn der DBA entsprechende Services und Technologien bereitstellt, der Anwendungsentwickler jedoch davon nicht weiß und seine Applikation nicht so gestaltet, dass Business Transaktionen auf Konsistenz überprüft werden und das Exception-Handling sauber implementiert ist.

Was wir nämlich nicht können ist, laufende Transaktionen erneut auszuführen. Wird eine Transaktion durch einen Verbindungsverlust unterbrochen, muss der Anwendungsentwickler ein entsprechendes Exception-Handling implementiert haben und die Transaktion gegebenfalls neu ausführen.

Nur dann können unterbrochene Datenbank Transaktionen sauber wieder aufgenommen werden und die Business Transaktionen bleiben konsistent.

Wäre es nicht toll, wenn man auch Datenbank Transaktionen transparent für die Applikation erneut ausführen könnte? Sie ahnen es…

Oracle hat mit Einführung der neuen Version 12c eine neue Technologie, unter der Überschrift Application Continuity, namens Transaction Guard eingeführt. Diese ermöglicht es, den Status einer aktiven Transaktion über einen Verbindungsverlust hinweg aufzuzeichnen.

Für den Client ist der Status einer Transaktion, die nicht committed wurde, fraglich. Beispielsweise könnte eine Prozedur im Hintergrund arbeiten und den Verbindungsverlust nicht bemerken. Würde man die Prozedur einfach erneut ausführen, droht Dateninkonsistenz. Transaction Guard versucht das Problem zu lösen, indem es mittels eines Logical Transaction Identifiers (LTXID) den Status der Transaktion nachhält. So kann eine Entscheidung getroffen werden, ob diese erneut ausgeführt werden muss oder nicht.

Das Thema wird also um einiges komplexer. Versprochen!

To be continued…

Kontaktadresse:

Martin Schmitter
Trivadis GmbH
Werdener Straße, 4
D-40227 Düsseldorf

Telefon: +49 (0) 211-58 66 64 70
Fax: +49 (0) 211-58 66 64 71
E-Mail martin.schmitter@trivadis.com
Internet: www.trivadis.com