Datenbanken konsolidiert in der Cloud

Heiko Eitner, Landesbetrieb Daten und Information Rheinland-Pfalz, und Marco Mischke, Robotron Datenbank Software GmbH

In der heutigen Zeit steigt die Zahl der Datenbanken ebenso wie die Anforderungen an deren Verfügbarkeit. Gleichzeitig sollen die Betriebskosten aber oftmals noch sinken. Diese konträren Zielsetzungen lassen sich durch den Einsatz eines Policy-managed RAC-Clusters erreichen. Die Autoren zeigen den beschrittenen Weg dorthin mit all seinen Fehlschlägen und Problemen.

Am Anfang des Weges stand ein vorhandener Fünf-Knoten-RAC-Cluster, der lediglich für eine einzige Datenbank reserviert war. Die Last auf dem Gesamtsystem war praktisch vernachlässigbar. Grund für den Einsatz von RAC war die Absicherung gegen Ausfälle einzelner Server. Daneben existierten noch eine ganze Reihe weiterer produktiver Datenbanken, die alle auf jeweils einem eigenen dedizierten Server liefen. Die Ausgangslage war also folgende:

- Fünf Server-RAC, eine Datenbank
- Neun Single Server, neun Datenbanken

Diese 14 Server mit ihren zehn Datenbanken stellen die Grundlage für den täglichen Betrieb dar. Darum gliederten sich noch entsprechende Test-, Entwicklungs- und Integrationssysteme, diese bleiben hier aber außen vor. Die produktiven Datenbanken können in drei verschiedene Klassen entsprechend ihrer Bedeutung für den täglichen Betrieb eingeteilt werden. Daraus resultiert diese Klassifizierung der Datenbanken:

- *High*
  Drei Datenbanken
- *Medium*
  Vier Datenbanken
- *Low*
  Drei Datenbanken

Die Rahmenbedingungen

Für die Konsolidierung der Datenbank-Landschaft wurde eine Reihe von Zielen definiert, die es möglichst effizient umzusetzen galt:

- *Hohe Flexibilität*
  Änderungen an der Landschaft wie neue Datenbanken, andere Priorisierung oder steigende Arbeitslasten sollen einfach handhabbar sein.
- *Optimale Auslastung der Hardware*
  Die Last soll möglichst gleichmäßig über alle Server verteilt werden können, sodass weder ruhende noch überlastete Server existieren.
- *Minimierung der finanziellen Aufwände*
  Kosten für Lizenzen, Anschaffungskosten der Server, Kosten für Energie und Kühlung sowie Personalkosten sollen soweit als möglich gesenkt werden.
- *Priorisierung entsprechend definierter SLAs*
  In Fehlerfällen müssen die Datenbanken mit dem höchsten SLA am längsten überleben beziehungsweise im Fall eines Desasters am schnellsten wieder verfügbar sein.
- *Optimierung der Administration*
  Einrichtung, Betrieb und Maintenance sollen standardisiert werden, um Aufwände und Fehler zu minimieren. Die nötigen Handlungsabläufe sollen entsprechend dokumentiert sein. Dies soll ebenfalls Vertretungsregelungen vereinfachen.
- *Verwendung verschiedener Releases*
  Die eingesetzten Applikationen stellen verschiedene Anforderungen und Bedingungen an die verwendete Datenbank-Version. Dies soll innerhalb der neuen Umgebung umsetzbar sein.

Die neue Lösung musste alle genannten Punkte abdecken. Ist ein Punkt nicht oder unzureichend umgesetzt, so wird die Lösung entsprechend verworfen.

Die Theorie

Um ein besseres Verständnis der Thematik zu bekommen, ist es wichtig, die Begriffe „Server“, „Serverpool“ und „Service“ zu verstehen. Ein Server repräsentiert einen physikalischen Rechner, der Teil eines Clusters ist. Serverpools fassen mehrere Server zu einer Gruppe zusammen. Dabei können minimale und maximale Anzahl der Server im Pool sowie die Priorität des Serverpools festgelegt werden. Diese Attribute beeinflussen die Startreihenfolge wesentlich.

Zuerst werden alle definierten Serverpools in der durch die Priorität festgelegten Reihenfolge bis zum Erreichen des Minimums mit Servern bestückt. Ist die minimale Anzahl von Servern in allen Serverpools erreicht, so wird der Serverpool mit der höchsten Priorität bis zum Erreichen der maximalen Anzahl an Servern befüllt. Danach folgen alle anderen Serverpools in der Reihenfolge ihrer Priorität.

Services sind ein Instrument, um Datenbanken anzusprechen. Man kann hier festlegen, welche Datenbanken welchen Service zur Verfügung stellen und auf welchem Serverpool der Service laufen soll. Weiterhin werden im Service entsprechende Attribute definiert, die das Verhalten im Fehlerfall bestimmen.

Erste Idee: Child Serverpools

Von Anfang an war klar, dass die neue Clusterlösung unter Verwendung von Serverpools realisiert werden muss. Dabei stießen die Autoren auf das Problem, dass ein Datenbank-Service nur genau ei-

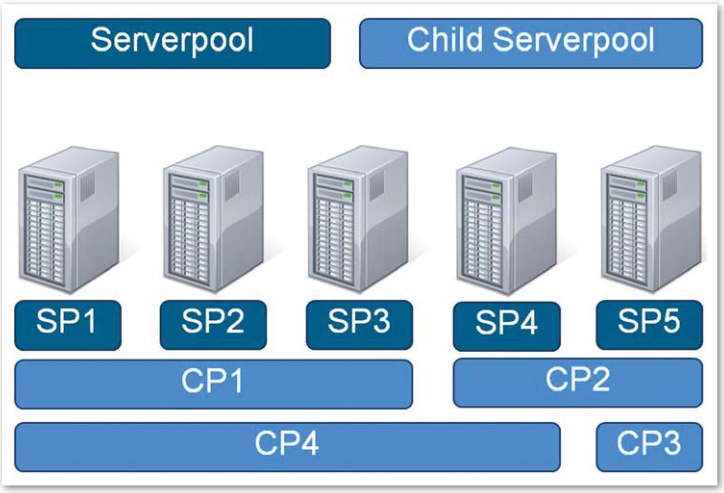

Abbildung 1: Cluster-Aufbau mit Child Serverpools

nem Serverpool zugeordnet werden kann. Es ist mit mehreren Serverpools also nie möglich, einen Service auf allen Servern im Cluster laufen zu lassen. Beim Studium der Clusterware-Dokumentation entdeckten sie dann die Möglichkeit, Child Serverpools einzurichten. Diese können mehrere Serverpools als Parent besitzen und es dürfen auch mehrere Child Serverpools auf einem Server laufen.

Die Idee war nun, einen Child Serverpool pro Service einzurichten. Dazu wurden fünf Serverpools mit absteigender Priorität und der Eigenschaft „min=max=1" erstellt. Das bedeutet, dass beim Starten des Clusters zuerst der Serverpool mit der höchsten Priorität auf dem ersten hochgefahrenen Server gestartet wird. Der zweite Serverpool wird dann entsprechend auf dem zweiten hochgefahrenen Server gestartet etc. Damit ist die Startreihenfolge der Serverpools unabhängig von der Startreihenfolge der Server.

Nun werden entsprechend Child Serverpools für jede Datenbank eingerichtet und diesen die gewünschten Serverpools entsprechend ihrer Einteilung zugewiesen. Dabei kann die Anzahl der Server im Child Serverpool sowie die Startreihenfolge und somit die Priorisierung frei gewählt werden. *Abbildung 1* zeigt den geplanten Aufbau im Cluster.

Diesem Aufbau entsprechend wurde nun versucht, die entsprechenden Serverpools anzulegen. Dazu musste das „crsctl"-Tool verwendet werden, da nur mit diesem das Festlegen eines Parent Serverpools möglich ist. Der erste Versuch scheiterte jedoch, da die so eingerichteten Child Serverpools nicht per „srvctl" einer Datenbank zugeordnet werden können. Die Clusterware erwartet offenbar intern noch den Prefix „ora." für diese Serverpools. *Listing 1* zeigt dies.

Daraufhin wurden die Child Serverpools entsprechend neu angelegt und nun jeweils der Präfix „ora." für die Namensgebung verwendet. Auch dieser Versuch scheiterte, wie in *Listing 2* zu sehen ist.

Wie sich herausstellte, darf nur „srvctl" zur Einrichtung von Serverpools für Policy-managed Datenbanken verwendet werden. Auch Tools wie „dbca" bieten die mit „crsctl" angelegten Serverpools beim Einrichten von Datenbanken gar nicht erst an.

So wurde das Serverpool-Konzept erneut mit „srvctl" eingerichtet. Nachdem die Serverpools entsprechend eingerichtet und eine Datenbank in einen zukünftigen Child Serverpool gelegt war, musste nur noch das „PARENT_POOL"-Attribut dieses Serverpools per „crsctl" modifiziert werden, da „srvctl" keine solche Option besitzt. Allerdings wechselte die Datenbank dadurch plötzlich von „policy managed" in „administrator managed" *(siehe Listing 3)*.

Ein Service Request stellte klar, dass die Benutzung von Child Serverpools für Datenbanken nicht unterstützt wird. Der zugehörige Bug 16369884 „ADDING SUBPOOL CHANGES POLICY MANAGED TO ADMINISTRATOR MANAGED DB" wurde entsprechend mit „Status: 32 – Not a Bug. To Filer" geschlossen.

Finale Idee: Policy-managed Databases

Nach diesen Fehlschlägen waren die Autoren gezwungen, ihr Konzept zu überdenken. Letztendlich haben sie für jede der drei eingangs definierten Gruppen (high, medium, low) einen Serverpool mit entsprechender Kardinalität und Priorität definiert. Die Services der Datenbanken wurden von ihnen auf die passenden Serverpools verteilt. Eine Datenbank sollte jedoch per Kundenanforderung auf allen Servern im Cluster laufen, für diese Datenbank haben sie dementsprechend Services in allen Serverpools eingerichtet. *Abbildung 2* zeigt die neue Architektur. *Listing 4* verdeutlicht noch einmal das Anlegen der einzelnen Serverpools sowie die Zuordnung der Services zu den entsprechenden Serverpools.

Aus diesem Aufbau ergab sich nun das Problem, eine Datenbank über mehrere

```
crsctl add serverpool S1 -attr "IMPORTANCE=9,MIN_SIZE=1,MAX_SIZE=1"
crsctl add serverpool S2 -attr "IMPORTANCE=8,MIN_SIZE=1,MAX_SIZE=1"
crsctl add serverpool S3 -attr "IMPORTANCE=7,MIN_SIZE=1,MAX_SIZE=1"

crsctl add serverpool CP1   -attr \
   "IMPORTANCE=9,MIN_SIZE=1,MAX_SIZE=5,PARENT_POOLS=S1 S2 S3"
crsctl add serverpool CP2   -attr \
   "IMPORTANCE=9,MIN_SIZE=1,MAX_SIZE=2,PARENT_POOLS=S1 S2"
crsctl add serverpool CP3   -attr \
   "IMPORTANCE=8,MIN_SIZE=1,MAX_SIZE=2,PARENT_POOLS=S1"

srvctl add database -d DB1 -g CP1 -o $ORACLE_HOME
PRCR-1039 : Server pool ora.CP1 does not exist
```

Listing 1

```
crsctl add serverpool ora.CP1 -attr \
   "IMPORTANCE=9,MIN_SIZE=1,MAX_SIZE=5,PARENT_POOLS=S1 S2 S3"
crsctl add serverpool ora.CP2 -attr \
   "IMPORTANCE=9,MIN_SIZE=1,MAX_SIZE=2,PARENT_POOLS=S1 S2"
crsctl add serverpool ora.CP3 -attr \
   "IMPORTANCE=8,MIN_SIZE=1,MAX_SIZE=2,PARENT_POOLS=S1"

srvctl add database -d DB1 -g CP1 -o $ORACLE_HOME
PRKO-3150 : The server pool(s) CP1 specified with the -g cannot be used
to add an administrator-managed database
```

Listing 2

```
srvctl add serverpool -g S1  -i 10 -l 1 -u 1
srvctl add serverpool -g CP1 -i 10 -l 1 -u 5
srvctl add database -d DB1 -g CP1

srvctl status database -d DB1
Database is policy managed

crsctl modify serverpool ora.CP1 -attr „PARENT_POOL=ora.S1"

srvctl status database -d DB1
Database is administrator managed
```

Listing 3

Services transparent vom Client aus anzusprechen. Wie also muss ein Eintrag in der „tnsnames.ora" aussehen, um mehrere Services einer Datenbank in einem Alias zusammenzufassen? Dazu kam eine „DESCRIPTION_LIST" zum Einsatz *(siehe Listing 5)*.

Tests haben gezeigt, dass die Client-Verbindungen entsprechend über alle Services verteilt werden. Damit benutzen die Clients auch alle Server im Cluster. Mit dieser Lösung sind nun alle Anforderungen an das neue System erfüllt.

Erfahrungen aus dem Alltag

Im täglichen Betrieb dieser neuen Umgebung wurden bereits einige Erfahrungen gesammelt. Die Bereiche erstrecken sich von Backup über Monitoring bis hin zu eingesetzten Versionen und Patching. Für die Backups der Datenbanken im Cluster wird RMAN verwendet. Dieser verbindet sich über den Standard-Service zur jeweiligen Datenbank. Auf die Einrichtung eines separaten Service wurde zugunsten der Übersichtlichkeit verzichtet, da dies keinen nennenswerten Mehrwert bedeutet.

Das Monitoring des Systems erfolgt mit Cloud Control Release 3. Dieses ist ohnehin im Einsatz und scheint am besten geeignet, eine solche Umgebung effektiv zu überwachen und zu warten. Trotzdem gibt es einige Fallstricke bei der Einrichtung der Umgebung im Cloud Control. In älteren Releases wurden die Scan Listener im Cluster als „Down" gemeldet, wenn diese aus irgendeinem Grund auf einen anderen Server geschwenkt waren. Cloud Control versuchte dann nach wie vor, die Scan Listener auf dem ursprünglichen Server zu überwachen. Dazu existiert eine My Oracle Support Note 1493823.1; das zugrunde liegende Problem ist in Release 3 behoben. Trotzdem findet ein erneutes Discovery die vorhandenen Scan Listener als neue Listener. Dies kann und sollte man einfach ignorieren.

Ein größeres Problem stellt das Einrichten der Datenbanken dar. Cloud Control findet nicht immer alle Instanzen zu einer Datenbank, sondern teilt die Instanzen auf mehrere Datenbanken auf. Das Cloud Control prüft die Verfügbarkeit der Instanzen immer anhand des Servers, der zum Zeitpunkt des Discovery aktuell war. Werden Instanzen im Cluster umverteilt, hinzugefügt oder entfernt, weil Änderungen an der Kardinalität der Serverpools vorgenommen werden oder einzelne Server offline gehen müssen, so verliert Cloud Control praktisch völlig den Überblick und es bleibt nur ein Rediscovery als Weg, die Umgebung wieder sauber einzurichten. Das Problem ist bei Oracle als Bug in Bearbeitung.

Das Release- und Patch-Management erfordert ebenfalls einige Vorüberlegungen. Da die Version der Grid Infrastructure immer mindestens der Datenbank-Version entsprechen muss, haben sich die Autoren entschieden, immer die neueste Version der Grid Infrastructure einzuset-

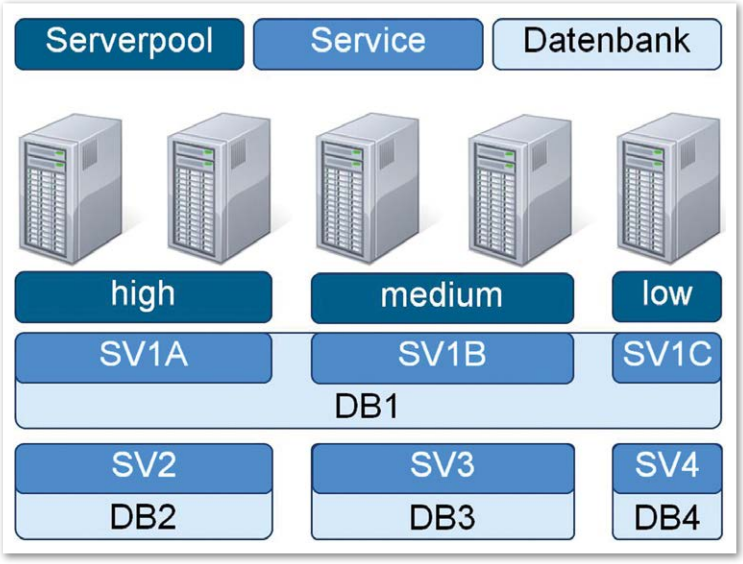

Abbildung 2: Clusteraufbau mit Serverpools und Services

zen. Die Upgrades der Grid Infrastructure können „rolling“ erfolgen. Das bedeutet, es entstehen keine Ausfallzeiten für die Applikationen. Datenbanken werden immer per „out of place“-Upgrade aktualisiert. Das wird nicht nur für Releases und Patchsets praktiziert, sondern auch für das Einspielen von Patches oder Patch Bundles. Man installiert in solchen Fällen immer ein neues Oracle-Home, das nach Bedarf gepatcht wird. Danach erfolgt die Umstellung der Datenbanken auf das neue Oracle Home. Dadurch verringert sich die nötige Downtime auf ein Minimum und es entstehen keinerlei Beeinträchtigungen für andere, nicht betroffene Datenbanken.

Fazit

Die neue Cloud-Lösung deckt alle im Vorfeld gesteckten Rahmenbedingungen vollständig ab. Es entstand eine standardisierte Umgebung für alle produktiven Datenbanken, die Verteilung der Instanzen erfolgt automatisch über die den Services zugeordneten Serverpools entsprechend den definierten SLAs und ermöglicht einen kontinuierlichen, unterbrechungsfreien Betrieb für alle Datenbanken. Zuvor war dies nur für eine Datenbank der Fall, die als RAC lief. Dabei war keine einzige zusätzliche Lizenz notwendig, da der Fünf-Knoten-Cluster bereits vorhanden war. Man konnte sogar Lizenzen einsparen, da eine ganze Reihe Datenbanken statt auf dedizierten Servern nun mit in diesem Cluster laufen.

Ausblick

Mit der hier vorgestellten Umgebung werden Features benutzt, die schon seit Längerem in 11*g* R2 etabliert sind und ausgereift erscheinen. Eine interessante Möglichkeit stellt die mit Database 12*c* neu eingeführte Multitenant-Option dar. Damit ließe sich eine noch bessere Flexibilität erzielen. Allerdings gibt es derzeit noch einige Einschränkungen beim Klonen von Datenbanken; die Quell-Datenbank ist während des Klonens für die Anwendung nicht verfügbar. Diese Tatsache, zusammen mit den zusätzlichen Lizenzkosten, ist der Grund, warum diese Option aktuell nicht verwendet wird. Die Autoren verfolgen die Entwicklung der Multitenant-Option aber mit Spannung.

```
srvctl add srvpool -g high   -i 10 -l 1 -u 2
srvctl add srvpool -g medium -i 8  -l 1 -u 2
srvctl add srvpool -g low    -i 6  -l 1 -u 1

srvctl add service -d DB1   -s SV1A    -g high   -c uniform
srvctl add service -d DB1   -s SV1B    -g medium -c uniform
srvctl add service -d DB1   -s SV1C    -g low    -c uniform

srvctl add service -d DB2   -s SV2     -g high   -c uniform
srvctl add service -d DB3   -s SV3     -g medium -c uniform
srvctl add service -d DB4   -s SV4     -g low    -c uniform
```

Listing 4

```
ALIAS=
(DESCRIPTION_LIST=
   (DESCRIPTION=
       (ADDRESS=(PROTOCOL=TCP)(HOST=cloud-scan)(PORT=1521))
       (CONNECT_DATA=
           (SERVICE_NAME= SV1A))
   )
   (DESCRIPTION=
       (ADDRESS=(PROTOCOL=TCP)(HOST=cloud-scan)(PORT=1521))
       (CONNECT_DATA=
           (SERVICE_NAME= SV1B))
   )
   (DESCRIPTION=
       (ADDRESS=(PROTOCOL=TCP)(HOST=cloud-scan)(PORT=1521))
       (CONNECT_DATA=
           (SERVICE_NAME= SV1C))
   )
)
```

Listing 5

Daneben bietet die Clusterware in der Version 12*c* die Möglichkeit, Serverpools dynamisch über Serverkategorien nach bestimmten Hardware-Attributen zusammenzusetzen. Es lassen sich außerdem verschiedene Konfigurationen der Serverpools mittels Policy verwalten und nach Bedarf aktivieren. Diese Features werden mittelfristig zum Einsatz kommen, um den Betrieb der Umgebung weiter zu vereinfachen und zu automatisieren.

Heiko Eitner
heiko.eitner@ldi.rlp.de

Marco Mischke
marco.mischke@robotron.de