RAC: verdeel en heers

Proof of Concept implementatie

Dit artikel is gebaseerd op de praktijkervaring van een aantal collegae met het installeren en beheren van enkele Real Application Cluster (RAC) databases. Het beoogt de lezer kennis te laten maken met enkele do's en don'ts van RAC, hem te helpen in zijn keuzetraject en praktische tips & tricks voor installatie en beheer aan te reiken.

Sinds met Oracle9i het verschijnsel Real Application Cluster (RAC) op de markt is gekomen hebben een aantal organisaties deze feature onder de loep genomen. In enkele gevallen is een zogenaamde Proof-of-Concept (PoC) uitgevoerd. Andere IT-organisaties hebben geïnstalleerde 9i databases reeds voorbereid op RAC door een cluster-database te creëren op een enkele machine, met als doel extra servers toe te kunnen voegen indien nodig. Dit artikel is gebaseerd op de ervaringen van enkele installaties en PoC-projecten op Linux, Tru64 en OpenVMS.

Definitie

In het kort betekent RAC dat een database door meerdere, kleine, machines wordt aangestuurd. Dit maakt het mogelijk een grote database te bouwen met relatief goedkope hardware, of een database te laten groeien zonder dat migratie naar een geheel nieuwe, grotere, server nodig is. Het gebruik van meerdere servers biedt tevens het voordeel dat de database bij uitval of onderhoud van één van de servers beschikbaar blijft. Men krijgt de hoge beschikbaarheid er gratis bij.

In één van de Proof of Concept-implementaties is het OCFS toegepast op een cluster van Intel gebaseerde Linux machines

De beoogde voordelen van RAC zijn:

- Hogere beschikbaarheid
- Schaalbaarheid, de mogelijkheid om extra hardware toe te voegen. Zogenaamde horizontale schaalbaarheid.
- Flexibiliteit, de mogelijkheid om ad-hoc, tijdelijk of permanent, extra processing power aan een database toe te voegen.

Onderliggende technologie

Een RAC database is dus verdeeld over meerdere servers. Er blijft echter één single-point-of-truth bestaan, namelijk de opgeslagen data op disk. Er is nog steeds maar één record voor "MILLER" in de EMP table. De servers zullen dus onderling moeten zorgen dat slechts 1 process tegelijk deze data kan modificeren.

Om dit mogelijk te maken heeft een RAC database twee belangrijke componenten nodig: een *geclusterd disk-systeem* en een *interconnect*.

Een *geclusterd disk systeem* betekent dat de meerdere servers de disks kunnen benaderen zonder dat daardoor integriteitsproblemen ontstaan. Vaak bestaat dit systeem uit zogenaamde RAW devices, steeds meer echter is een Clustered File Systeem beschikbaar, een CFS. Hierover straks meer.
Verder is voor de coördinatie tussen de servers een zogenaamde *interconnect* nodig. Dit is te beschouwen als een dedicated network dat processen op de servers in staat stelt snel met elkaar te communiceren. Oracle noemt dit Cache-Fusion omdat de inhoud van de Oracle data-cache wordt uitgewisseld tussen meerdere servers. Afgezien van een discussie over RAW-devices versus Clustered-File-Systems laten we in dit artikel deze onderliggende technologie buiten beschouwing. We concentreren ons op de database.

Kijk in de RAC database

Laten we eens kijken hoe een RAC database eruit ziet. Figuur 1 toont een Sql*Plus sessie waarbij de gebruiker verbonden is aan een RAC-database met twee actieve instances.

*Figuur 1. Voorbeeld van een SQL*Plus query die twee instances toont*

Merk op dat de tweede instance dertig minuten later is opgestart. De tweede server/node/instance is pas later bijgeschakeld. Verder is de query is gedaan op een view met de prefix GV$ (we komen hier later op terug). Met de GV$PROCESS zien we de processen van beide instances:

```
SYSTEM @ RAC @ rac1 > select inst_id, program from gv$process ;

INST_ID PROGRAM
------- ------------------------
      1 PSEUDO
      1 oracle@pubnode1 (PMON)
      1 oracle@pubnode1 (DIAG)
      1 oracle@pubnode1 (LMON)
      1 oracle@pubnode1 (LMD0)
      1 oracle@pubnode1 (LMS0)
      1 oracle@pubnode1 (LMS1)
      1 oracle@pubnode1 (DBW0)
      1 oracle@pubnode1 (LGWR)
      1 oracle@pubnode1 (CKPT)
      1 oracle@pubnode1 (SMON)
      1 oracle@pubnode1 (RECO)
      1 oracle@pubnode1 (TNS V1-V3)
      1 oracle@pubnode1 (LCK0)
      1 oracle@pubnode1 (TNS V1-V3)
      1 oracle@pubnode1 (P000)
      1 oracle@pubnode1 (TNS V1-V3)
      2 PSEUDO
      2 oracle@pubnode2 (PMON)
      2 oracle@pubnode2 (DIAG)
      2 oracle@pubnode2 (LMON)
      2 oracle@pubnode2 (LMD0)
      2 oracle@pubnode2 (LMS0)
      2 oracle@pubnode2 (LMS1)
      2 oracle@pubnode2 (DBW0)
      2 oracle@pubnode2 (LGWR)
      2 oracle@pubnode2 (CKPT)
      2 oracle@pubnode2 (SMON)
      2 oracle@pubnode2 (RECO)
      2 oracle@pubnode2 (P000)
      2 oracle@pubnode2 (LCK0)
31 rijen zijn geselecteerd.

SYSTEM @ RAC @ rac1 >
```

Merk op dat er nog geen connecties gemaakt zijn naar de tweede instance (deze is net opgestart) en dat er een aantal processen zijn die niet voorkomen op single-instance databases (LMON, LMSn, LMDn). Verderop zullen we terugkomen op het monitoren van een multi-instance database.

Clustered File Systems versus RAW devices

Tot voor kort ging RAC, en voorloper OPS (Oracle Parallel Server), ervan uit dat clustering vrijwel altijd plaats moest vinden met behulp van zogenaamde RAW devices. RAW-devices zijn disks waarop geen file-system staat. Het nadeel is echter dat een RAW-device niet als een conventionele disk (lees: directory met files) is te beheren. Het onderhoud van RAW devices vereist specialistische kennis of specifieke tools. Verder is het maximum aantal RAW devices op sommige systemen beperkt tot 255. Dit vormt een beperking voor bijvoorbeeld de E-business suite die 300+ tablespaces omvat.

Een belangrijke conditie voor bredere toepassing van RAC is daarom de mogelijkheid om zogenaamde Clustered File Systems te gebruiken om de data op te slaan. Oracle heeft dit onderkend en heeft zelf voor Windows en Linux het Oracle Clustered File Systems (OCFS) beschikbaar gesteld. Voor Linux wordt deze component zelfs als open source software geleverd [5].
Op andere Unix-platforms kan een plaform-specifiek CFS ingezet worden (bijvoorbeeld advfs voor HP-Tru64) of kan een CFS van een derde partij betrokken worden (bijvoorbeeld Veritas VxFS). Op OpenVMS is clustering een *de facto* onderdeel van het operating system hetgeen beheer op dit platform zeer vergemakkelijkte.

Let bij uw combinatie van hard- en software speciaal op de certificatie-matrix van Oracle (metalink!). Controleer bovendien wat wel en niet (certified-) op een eventueel CFS kan worden opgeslagen. Zo mag de huidige versie van OCFS alleen gebruikt worden voor database-files.

Op andere Clustered File Systemen kunnen meestal zowel de Oracle-software (ORACLE_HOME), als de configuratie-files (initSID.ora) als de database-files geplaatst worden. Onderhoud van software (patches) en configuratie-files is dan eenvoudiger. Eenmalige installatie is eenvoudiger te beheren (minder componenten, makkelijker te patchen) maar men kan lastiger één node apart zetten om bijvoorbeeld een patch te testen. Zie ook het kader over rolling upgrades.

OCFS: interessant

In één van onze PoCs hebben we het OCFS toegepast op een cluster van Intel gebaseerde Linux machines. Zoals gezegd mag OCFS expliciet (nog) niet gebruikt worden voor het opslaan van initSID.ora of ORACLE_HOME, of enige andere file. Toen we probeerden onze initSID.ora centraal op het OCFS te plaatsen bleek manipulatie van deze file (copy, open, save) irritant

traag te worden. Ten faveure van OCFS moet gezegd worden dat het gratis is en binnen de gegeven specificaties en beperkingen uitstekend lijkt te presteren. Oracle heeft beloofd deze ontwikkeling door te zetten en volgt daarmee een duidelijke pro-Linux koers. Dit is belangrijk genoeg om in de gaten te houden.

Voorbereidingen en installatie

Aan het installeren van de RAC-software gaan een aantal beslissingen en handelingen vooraf. Zo moet gekozen worden voor hard- en software en disk-subsystems (RAW versus CFS). Alle keuzes moeten gecertificeerd zijn of op zijn minst compatibel. Verder moet de Unix kernel op elke node geprepareerd worden met de juiste parameters. Bij Linux kan dit eenvoudig met een script in de init.d dat de parameters op de gewenste waarden zet. Zie figuur 2.

Figuur 2. Op Linux kunnen de kernel parameters met een script aangepast worden.

Voor andere Unix platformen kan het nodig zijn een nieuwe kernel te bouwen en te installeren middels een reboot. Verder is het altijd aan te bevelen om de meegeleverde installatiehandleiding én de beschikbare metalink-informatie te lezen. Voor vrijwel elk platform is een step-by-step RAC installatie handleiding te vinden die de meegeleverde documentatie aanvult. Een paar uur studie vooraf kan veel tijdsverlies voorkomen. Nuttig is ook het script InstallPrep.sh, waarvan de laatste versies te vinden is op Metalink.

Voor installatie raden we aan de hoogste versie (9.2.0.1 bij het schrijven van deze tekst) en de laatste patches (momenteel 9.2.0.2 en wat platform-specifieke patches) te gebruiken. RAC technologie is nog zo nieuw, dat er nog regelmatig kleine veranderingen aan de software plaatsvinden. Zo leidde een recente bug ertoe dat bij uitvallen (of reboot) van een server/node de database wel in de lucht bleef, maar de instance op de betreffende node niet meer te (her)starten was, waardoor de gehele database uiteindelijk toch even down moest (Bug 2563543).

Installatie van disk verloopt altijd sneller dan installatie vanaf een CD. Geef er als het enigszins mogelijk is dus de voorkeur aan de CD's op disk te kopiëren en de installer van disk op te starten. Installeer als eerste component altijd de cluster-manager en start de gsd-deamon op, een door Oracle geleverde cluster-component. Pas hierna staat de installer toe dat men de RAC-opties installeert. Indien de Oracle software niet op een CFS geïnstalleerd kan worden dan kan men ervoor kiezen via de installer alle cluster-nodes te voorzien van software (aanbevolen) of zelf de software naar de andere nodes te kopiëren. Let in dit geval op files waar de hostname in voorkomt (bijvoorbeeld listener.ora).

Nogmaals, installatie van slechts één enkele ORACLE_HOME op een CFS heeft onze voorkeur. Automatische of handmatige replicatie van geïnstalleerde software is een oplossing voor een probleem dat we liever voorkomen dan oplossen. Hopelijk is dit ook bij volgende versies van OCFS voor Linux mogelijk.

Creatie van de database

Voor het aanmaken van de database is het gebruik van de DataBase Create Assistant (DBCA) aanbevolen. Echter, de DBCA heeft de neiging om alle voorbeelden en opties op te nemen, dus het aanmaken van "scripts-only" is om twee redenen een aanrader. In de eerste plaats is het doorlezen en bewaren van de create-scripts leerzaam. Ten tweede kan het verwijderen van de interessante maar niet-gebruikte opties (Java, XML) het aanmaken van de database aanzienlijk versnellen. Relevante parameters bij het aanmaken zijn vooral de waarden voor MAXINSTANCES en MAXLOGFILES, aangezien die samen bepalen hoeveel instances (lees: servers) de database kan omvatten. Overigens, met het create-controlfile kunnen deze parameters nog steeds opgehoogd worden, maar dan moet de database wel even uit de lucht.

De RAC database is verdeeld over meerdere servers, maar er blijft één single-point-of-truth bestaan

Verder wil elke RAC-instance over zijn eigen UNDO tablespace beschikken. Ter geruststelling: voor applicaties die een hardgecodeerd (groot) RBS willen gebruiken is het ook mogelijk de parameter "undo_suppress_errors" toe te passen. Tevens kan het gebruik van Locally Managed Tablespaces (LMTs) worden aangeraden. De overhead van Dictionary-

Managed tablespaces (DMTs) weegt extra zwaar in een RAC omgeving omdat de system-tablespace een bottleneck vormt. Inspecteer vooraf ook de init.ora file, en kies bewust voor het al dan niet gebruiken van de spfile-feature. Indien een CFS ontbreekt is een centrale spfile aan te raden. Op een CFS is een init.ora wellicht handiger. Zie het kader over spfile voor meer informatie en een anecdote.

Monitoren van een RAC database

Om de correcte werking van een geclusterde database in de gaten te houden zijn de volgende zaken van belang:

1. Functioneren alle servers/nodes naar behoren?
   Is op alle nodes nog voldoende processor-capaciteit en geheugen beschikbaar?
2. Is de disk-toegang voldoende snel?
3. Functioneert de interconnect en is de communicatie voldoende snel (bandbreedte en latency)?

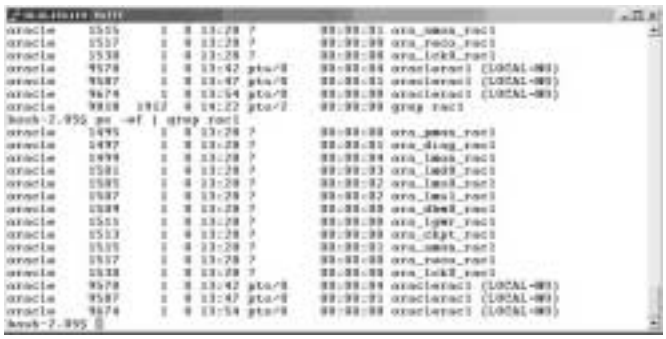
De eerste twee punten zijn ook van toepassing op een non-RAC database. Bewaking hiervan gebeurt normaliter op twee niveaus:

- Op het operating system met de beschikbare tools. Bij een Unix/Linux systeem meestal met vmstat, iostat, sar, top en ps. Voor de VMS fans: mon proc /topc
- Vanuit de database via de wait-statistics.

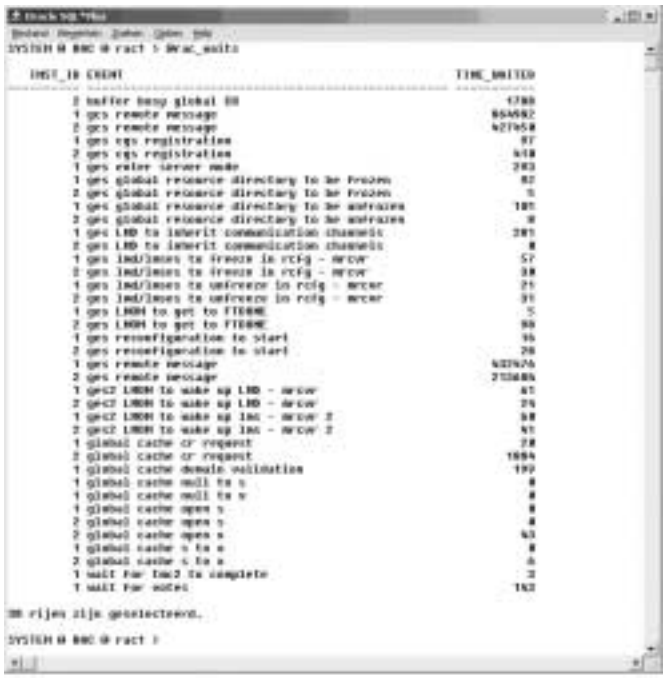
Het monitoren van een RAC database verschilt dus niet veel van het beheer van een gewone database. Voornaamste verschillen zijn dat meer servers/instances in de gaten gehouden moeten worden en dat de gezamenlijke componenten, de interconnect en de disks, een kritische rol spelen. Controleer of de tools die men voor monitoring gebruikt wel van toepassing zijn op RAW devices of Clustered File Systems. Sommige tools rapporteren bijvoorbeeld niet de i/o die plaatsvindt op RAW devices, waardoor een disk-bottleneck niet tijdig onderkend wordt.

Zoals eerder aangegeven bestaat een instance van een RAC database uit meer processen dan een ongeclusterde database (zie de eerder getoonde output van gv$process). Op Unix-niveau zijn die processen zichtbaar te maken met het ps commando. Zie figuur 3 voor het resultaat van een ps commando op één van onze systemen. Ook hier zijn de extra processen zichtbaar.

Met Oracle9i-RAC is het Cache-Fusion mechanisme in de plaats gekomen van het trage, disk-IO gelimiteerde, block-pinging in eerdere versies. Dit betekent dat nu de interconnect tussen de nodes de nieuwe bottleneck voor schaalbaarheid is. Het monitoren van de interconnect op operating system niveau verschilt per platform, maar de aandachtspunten zijn dezelfde: bandbreedte en snelheid (latency).



Figuur 3. De Unix achtergrond processen van een instance.



Figuur 4. RAC-specifieke events en wachttijden van de twee instances van een RAC-database.

Global dynamic performance views: GV$

Op database- en instance niveau blijft het controleren van de wachttijden het belangrijkst. Gewoonlijk gebeurt dit met de informatie uit de dynamic performance views, de v$ views. Echter de v$ views tonen alleen de gegevens van de instance waaraan de huidige sessie verbonden is. Oracle heeft daarom voor vrijwel alle v$ views een tegenhanger gemaakt met de prefix gv$. Elke gv$ view bevat een extra column inst_id die aangeeft op welke instance de gegevens van toepassing zijn en toont als het ware de union van de v$ views over alle actieve instances. Deze gv$ views worden de global dynamic performance views genoemd. Ze waren ook al aanwezig in OPS-databases in oudere versies. Let op dat bij het bevragen van de gv$ views de responsetijden lager kunnen zijn dan bij het gebruik van de v$ views. Opvragen van gv$ views leidt op rustige systemen ook tot zichtbare activiteit over de interconnect.

Oplettende DBA's hebben al geconstateerd dat single-instance databases ook de gv$ views bevatten. Elke DBA met eigen scripts kan dus zijn favoriete code gemakkelijk ombouwen voor toepassing in een RAC omgeving. De scripts blijven ook goed toepasbaar voor non-RAC databases. Speciaal voor RAC databases zijn een aantal additionele wait-statistics van belang. Een overzicht van deze statistics wordt gegenereerd door het bijgevoegde script rac_waits.sql.

```
Doc
rac_Waits.sql:
                                events specifically for RAC
#

select inst_id, event, time_waited
from gv$system_event
where event like 'gcs%'
   or event like 'ges%'
   or event like '%global%'
   or event like '%tmc2%'
   or event like '%vote%'
   or 1=2
order by  2, 1
/
```

De output hiervan is opgenomen in figuur 4. We zien hier het verschil tussen de instances 1 en 2. Instance 2 is net opgestart en heeft voor een aantal events lagere waarden (de instance is nog ongebruikt) maar ook voor enkele parameters een hogere waarde (de nieuwkomer heeft bij instance 1 geïnformeerd naar de status van de database). Verder illustreert het (nog) ontbreken van een aantal events op inst2 dat events en wait-statistics in een instance pas opvraagbaar zijn nadat het event is opgetreden en afgerond. Van praktisch belang is vooral om de events van figuur 4 in de gaten te houden. Lange wachttijden op deze events duiden op een te trage interconnect of op onvoldoende bandbreedte. Om nog verder in te zoomen op eventuele interconnect-bottlenecks zijn de views v$cache_transfer en v$file_cache_transfer van belang. De RAC deployment-guide geeft veel aanwijzingen om database en applicaties to monitoren, en indien nodig, te tunen voor toepassing op RAC. Indien men opstart-parameters wil wijzigen dan kan dat in versie 9i vrijwel altijd on-the-fly. In figuur 5 is dit te zien. Merk op dat de parameter alleen voor de huidige instance, dat wil zeggen de instance waaraan de sessie verbonden is, gewijzigd wordt.

Toevoegen van een node

Met het toevoegen van een node, worden de grote voordelen van RAC waargemaakt: schaalbaarheid en flexibiliteit, indien nodig onder hoge (tijds-)druk. In één geval hebben we ad-hoc in enkele minuten een nieuwe instance aan een database toegevoegd. De betreffende database stond op een Clustered File System waardoor de software (ORACLE_HOME) en de configuratie-files (initRAC.ora) reeds zichtbaar waren voor de toegevoegde machine. Verder was de database reeds aangemaakt met voldoende hoge waarden voor MAXINSTANCES en MAXLOGFILES.

De relevante acties waren:

- Het aanmaken van een initrac3.ora in de dbs directory (in ons geval een soft-link naar de centrale initrac.ora).
- Toevoegen van drie parameters aan initrac.ora:
  rac3.thread=3
  rac3.instance_name='rac3'
  rac3.undo_tablespace='UNDOTBS3'
- Het aanmaken van de extra undo-tablespace.
- Het aanmaken van de extra log-thread:
  alter database add logfile thread 3
  group 7 '/d01/rac/redo07.log' size 200M,
  group 8 '/d01/rac/redo08.log' size 200M,
  group 9 '/d01/rac/redo09.log' size 200M;
- Het beschikbaar maken van de extra thread:
  alter database enable public thread 3;
- Het opstarten van een listener-process op de 3e node. Aanpassen van de listener.ora is niet nodig omdat een instance zichzelf bij de (default-) listener aanmeld.
- Het opstarten van de instance.

Alles bij elkaar zijn zeven stappen en drie minuten werk nodig om een extra instance, een extra server in te zetten om gebruikers of batchjobs op deze database te bedienen!
Het concept van capaciteit-op-aanvraag is in zicht. Uit de tests die we hebben uitgevoerd blijkt overigens een goede schaalbaarheid. Het toevoegen van een tweede (en heel even ook een derde) node aan een systeem leidde tot vrijwel lineaire verhoging van de capaciteit. Hierbij moet aangetekend dat de betreffende applicatie, het batchgewijs verwerken van inkomende data, zich uitstekend leende voor deze vorm van schaalbaarheid.

Figuur 5. Het dynamisch aanpassen van een parameter gebeurt alleen op de instance waar men mee verbonden is.

Rolling upgrades, de uptime-belofte die bijna vervuld is

Bij clusters van webservers of applicatie-servers is het reeds mogelijk om de servers één voor één te upgraden zonder dat de "service" daaronder te lijden heeft, een zogenaamde rolling upgrade. De onderliggende servers worden stuk voor stuk "losgekoppeld", aangepast en weer terug "aangekoppeld".

Analoog hieraan lijkt RAC nu ook de mogelijkheid te bieden de database zonder downtime te patchen of te upgraden. Men kan de servers/nodes/instances loskoppelen, upgraden en weer aankoppelen.
Als we de 'Oracle RAC Setup And Configuration Guide' erop naslaan vinden we in hoofdstuk 4:
"The term rolling upgrade refers to upgrading different databases or different instances of the same database in Oracle9i Real Application Clusters one at a time, without stopping the database. Release 2 (9.2) of Oracle9i Real Application Clusters does not support rolling upgrade."
We zijn van mening dat een RAC database in principe een rolling-upgrade of rolling patch moet kunnen doorstaan. Men kan immers de Oracle software (ORACLE_HOME) op één node patchen terwijl de andere nodes de database beschikbaar houden. In een CFS configuratie betekent dit dat men (tijdelijk) twee ORACLE_HOMEs nodig heeft (en ook niet meer dan twee). De instances kunnen één voor één herstart worden met de nieuwere versie van de software. Pas als de laatste node/instance met de nieuwe software herstart is kan een eventueel upgrade-sql script (catpatch.sql) toegepast worden.
Het is dus mogelijk een patch of upgrade door te voeren zonder dat de database-service onderbroken wordt. Het wachten is op de bevestiging van Oracle dat deze feature ook daadwerkelijk ondersteund wordt.

NB: Dit verhindert niemand om het toch eens te proberen, voorlopig echter (nog) voor eigen risico.

Failover: de applicatie moet ook meedoen

Het failover concept dat bij RAC geïmplementeerd is wordt Transparant Application Failover (TAF) genoemd. Het komt erop neer dat een gebruiker of applicatie die een connectie heeft met een instance van een RAC database niets hoeft te merken van het eventuele uitvallen van de instance of van de onderliggende server. De connectie met de database blijft intact en wordt overgenomen door een andere server/node/instance. Uit testen bleek dat deze functionaliteit werkt "as advertised". Eén van de PoC's was speciaal hierop gericht. Het betreffende systeem heeft tests met een database op zowel OpenVMS als op Tru64 uitstekend doorstaan. Ook op Linux werkte de fail-over prima en we twijfelen er niet aan dat deze functionaliteit ook op andere platformen werkt.

Dit is een zeer goede prestatie gezien de onderliggende complexiteit. Het is een grote stap voorwaarts vergeleken met de "5-minuten-node-failover-instance-herstart-session-reconnect" constructies die hardware-leveranciers reeds jaren aanbieden. De implementatie van TAF loopt via tnsnames.ora of de Oracle names service, alwaar voor een connectie één of meer failover adressen gespecificeerd worden. De voornaamste kanttekening bij TAF is dat een eventuele failover tijdens een lopende query een hickup geeft van enkele seconden, tot zelfs enkele tientallen seconden.

Men moet echter rekening houden met de volgende beperkingen van het TAF mechanisme:
- Het is niet van toepassing op thin-JDBC clients, aangezien deze geen tns-entries gebruiken.
- In het geval van fail-over gaan de niet-gecommitteerde gegevens van een transactie verloren. Hoewel een gebruiker of batch-proces dus zijn connectie niet kwijt raakt kunnen er wel data verloren gaan.

Een eventueel batch process moet dus over goede error-afhandeling beschikken en herstartbaar zijn (altijd al een noodzaak). Data-entry schermen moeten de gebruiker eveneens attent maken op eventuele rollbacks (ook een goede gewoonte). TAF staat dus (nog) niet voor TransActie Failover.

Conclusies en opinies

Na enkele ervaringen met RAC installaties en configuratie, en na (vaak gedwongen) bestudering van veel documentatie durven we de volgende conclusies en aanbevelingen voor onze rekening te nemen:
1. RAC is for real: RAC werkt "as advertised". RAC maakt een (nog) hogere beschikbaarheid mogelijk (almost can't break it), maar is vooral interessant omdat het de database schaalbaar en flexibel maakt.
2. Capaciteit op aanvraag behoort nu tot de mogelijkheden. Een RAC database kan beschikbare hardware flexibel, op aanvraag, benutten. Dit is vooral aantrekkelijk bij een slecht voorspelbare groei of bij variabele belasting. Beschikbare hardware kan tijdens minder hoge belasting elders worden benut.
3. Linux op Intel-processoren, in combinatie met OCFS of een ander certified CFS is een serieus en kostenefficiënt alternatief voor middelgrote en grote databases. We concluderen dit uit onze eigen ervaringen, maar een sterke referentie is ook de businesscase van Amazon.com die is gepresenteerd op Oracle OpenWorld.

Uit onze verschillende installaties leidden we verder een aantal best-practices af. Enkele voorbeelden:

- Bouw een server-park van identieke machines en identiek geconfigureerde operating systems. Uitwisselbaarheid maakt flexibiliteit mogelijk.
- Besteed aandacht aan de disk-componenten. Als de server-component (CPU, memory) een massaproduct is dan resteert de disk-component als het meest kritieke punt van een database systeem.
- Gebruik van een SAN vereenvoudigt de opbouw van een cluster en de uitwisselbaarheid van servers.
- Een Clustered File System is sterk aanbevolen. Het beheer wordt eenvoudiger en kent minder beperkingen (het is bijvoorbeeld niet gelimiteerd tot 255 RAW devices per cluster). OCFS is een goed alternatief voor gebruik op Linux.
- Een single-install van ORACLE_HOME vereenvoudigt het beheer. We adviseren voor patches of upgrades eventueel een tweede ORACLE_HOME aan te houden (zie ook het kader over rolling upgrades).

Uitkijken met de SPFILE

Bij gebruik van een CFS is het mogelijk alle instances met één enkele initDB.ora file te laten werken: alle instances kunnen hun eigen initDB<n>.ora met een (soft-)link naar een centrale init.ora laten verwijzen. In de initDB.ora moeten instance specifieke parameters als volgt opgenomen worden:

```
rac2.thread=2
rac2.instance_name="rac2"
rac2.db_cache_size=500m
```

Een enkele initDB.ora is eenvoudiger en overzichtelijker in beheer, is eenvoudig te backuppen en men kan historie en commentaar opnemen in de file zelf. Tegenover het goed gedocumenteerde voordeel van de spfile (persistentie van ad-hoc aanpassingen) staat een slecht geadverteerd risico: zowel bij RAC als bij gewone databases, is dat het mogelijk parameters ongemerkt zodanig te wijzigen dat een instance niet meer (her)startbaar is. Probeer bijvoorbeeld eens:

```
Alter system set db_cache_size=0 scope=spfile;
```

Er volgt geen foutmelding, dus alles lijkt in orde, maar de instance zal bij de volgende herstart weigeren op te starten. De fix hiervan bestaat uit, inderdaad, het opnieuw aanmaken van een initDB.ora file. Mocht u dit overkomen: de data in de spfile is over het algemeen "leesbaar", al moet de inhoud misschien met het unix dump-commado (dd) van een RAW-device gehaald worden. Geharde DBA's weten bovendien dat alle non-default parameters ook nog in de alert-file staan.

- Gebruik LOG_ARCHIVE_FORMAT om te zorgen dat de archives van de verschillende threads het thread-nummer in de naam hebben. Wanneer dit niet gebeurt, dan kunnen de archives van verschillende threads identieke namen krijgen, met alle risico's van dien.

Tenslotte spreken we de hoop uit dat Oracle ook nog een aantal onderwerpen zal verbeteren. Zo hopen we dat in toekomstige versies van RAC een rolling upgrade tot de mogelijkheden gaat behoren (zie kader). Dit zal Larry's claim van "unbreakable" verder ondersteunen en zal de Oracle database nog betrouwbaarder maken. Verder hopen we dat het OCFS voor Linux in volgende versies gebruikt kan worden om ORACLE_HOME en andere reguliere files op te installeren (e.g. init.ora, adump, bdump, cdump, udump, …).
De ontwikkeling van OCFS zal Oracle een betere positie in de Linux-markt geven. Ook zal de ontwikkeling van OCFS de klanten van Oracle een grotere keuzemogelijkheid geven in hardware- en platformkeuze. Toepassing van RAC op goedkope hardware kan hiermee een interessante kostenbesparing opleveren. We zullen de ontwikkelingen aandachtig blijven volgen.

Referenties en leesmateriaal

[1] Oracle 9i RAC documentation (v9.2) "concepts", "admin-guide", "deployment and performance" en "setup and configuration".
[2] Metalink : http://metalink.oracle.com
[3] Technet : http://otn.oracle.com
[4] Oracle OpenWorld 2002, diverse presentaties over RAC en Linux. : http://otn.oracle.com/oracleworld/linux
[5] Oracle Clustered File System: http://otn.oracle.com/tech/linux/open_source.html en http://www.ocfs.org/ocfs/

NB: de chief-architect van OCFS spreekt Vlaams.

Frits Hoogland
is als database administrator werkzaam bij LogicaCMG Managed Services in Hoofddorp en kan worden bereikt onder Frits.Hoogland@logicacmg.com.
Piet de Visser
is als database administrator werkzaam bij LogicaCMG Trade Transport and Industry in Rotterdam en kan worden bereikt onder piet.de.visser@logicacmg.com.

De auteurs danken vele collega's voor hun commentaar. Met name Robert Blok, Mark Wormgoor, Bas Varkevisser en Jeroen Schipperijn hebben ons veel extra werk bezorgd ;-). Waarvoor dank.