# RAC, get real...

## *Performance hoort in het midden te liggen*

***Samen met Oracle 9i is het Real Application Cluster op de markt gekomen, de verbeterde versie van wat voorheen Oracle Parallel Server (OPS) heette. Aan de hand van een aantal criteria gaan we na in welke gevallen Oracle Real Application Cluster (RAC) zinvol is toe te passen. En hoewel RAC volgens Oracle met name voor schaalbaarheid en performance een grote belofte inhoudt, kijken we vanuit de praktijk ook naar alternatieven en naar de preventie van performanceproblemen.***

In enkele woorden komt RAC erop neer dat een database door meerdere servers (nodes, instances) benaderd kan worden waarmee de databaseserver niet langer de CPU- en memory-bottleneck van een applicatie is. Een en dezelfde databank kan nu op meerdere servers tegelijk draaien. Hiermee moet door het toevoegen van extra servers eenvoudig een hogere beschikbaarheid en betere schaalbaarheid mogelijk worden. Oracle presenteert RAC de als de laatste doorbraak in Oracle database technologie. De databasegigant laat zich moeilijk betrappen op harde uitspraken, maar de teneur van de boodschap is dat RAC 'de oplossing' is voor alle databaseproblemen.

Verder zien we dat de discussie rond OPS, de applicatiepartitionering, weinig aandacht meer krijgt. Applicatiepartitionering is de verdeling van de databasecomponenten (tabellen) over meerdere servers (nodes) binnen een OPS-systeem zodanig dat slechts minimale communicatie (block-pinging) nodig is om data-integriteit te waarborgen. Applicatie partitionering was lastig, maar noodzakelijk bij het goed toepassen van OPS. Met de zogeheten cache fusion lijkt de noodzaak van applicatiepartitionering voor RAC minder van toepassing te zijn. Wie goed leest zal echter ontdekken dat Oracle in de detaildocumentatie, met name in de deployment guide, nadrukkelijk adviseert om applicatiepartitionering toe te passen ter voorkoming van de overhead van cache fusion. De overhead bestaat dus nog steeds en het zou interessant zijn voor een gegeven applicatie uit te zoeken hoe de cache fusion overhead toeneemt naarmate meer nodes bijgeschakeld worden. In het volgende betoog willen we duidelijk maken dat de RAC een goede oplossing is voor bepaalde, specifiek te duiden situaties. Het is niet voor alle gevallen een geschikte oplossing.

***Indien men zeker weet dat de database de echte bottleneck is, kan RAC een uitweg bieden***

### Eisenpakket voor modern systeem

Volgens Oracle en andere leveranciers bestaat een moderne applicatie uit drie delen. Deel 1 is de front-end van de gebruikers: laptops, mobiele apparaten, altijd met een browser. Deel 2 is de middle tier: rekenkracht (CPU) en geheugenruimte (memory) om de logica uit te voeren en tegenwoordig ook om een deel van de presentatie te verzorgen. Deel 3 tenslotte omvat de database met daarin de enige en echte waarheid. Deze tier moet behalve rekenkracht en geheugen ook (snelle) toegang tot een opslagmedium bieden. In kader 1 wordt nadere toelichting gegeven op de 3 tier systeemarchitectuur.

Van dba goeroe en IOUG spreker Tim Gorman lenen we nu een aantal (technische) criteria waaraan de implementatie van een systeem of applicatie moet voldoen:

- High availability (e.g. altijd in de lucht). Hieronder groeperen we ook het criterium van MTTR (mean time to recovery, bijvoorbeeld herstel na een gesmolten CPU) welke Gorman apart beschouwt.
- Disaster recovery of uitwijkmogelijkheid (e.g. herstarten na een ramp, bijvoorbeeld een ondergelopen rekencentrum of een vuurwerkexplosie). Deze beschouwen we apart van high availability, ervan uitgaande dat de problemen na een echte ramp van een andere orde zijn dan een gebroken component.
- Cost/complexity of simpliciteit (e.g. is het systeem eenvoudig te bouwen en te onderhouden). We gaan er van uit dat een simpel systeem de voorkeur geniet.

- Rolling upgrades (e.g. uitrollen van nieuwe versies). Ter onderscheid, upgrades zijn geplande downtime, terwijl high availability (vrij naar Gorman) gaat over de betrouwbaarheid van componenten en operaties.
- Schaalbaarheid (e.g. is het systeem bestand tegen groei, hogere belasting en/of piekbelasting). Performance is vooral een kwestie van schaalbaarheid. Kan een systeem onder toenemende belasting van meer gebruikers en meer gegevens acceptabel blijven werken?

Graag willen we hier als applicatieontwikkelaars nog een (open deur?) criterium aan toevoegen:

- Integriteit (e.g. altijd correcte data en correcte functionaliteit) Dit laatste is met name van belang omdat caching en replicatie (applicatie-integratie, datamarts) ertoe kunnen leiden dat data bewust of onbewust enige tijd niet consistent zijn.

Voor elk van deze criteria willen we aangeven wat de invloed van RAC kan zijn, om zo tot argumenten te komen die aangeven in welke situatie RAC toegepast kan worden, en of er eventuele andere aantrekkelijke alternatieven zijn.

## 1. Integriteit

Hierop heeft RAC in principe geen directe invloed. Een databank met multiple-instances is normaliter net zo betrouwbaar als een single-instance databank. Hoogstens zou door toepassen van RAC als alternatief voor standby-databases of gerepliceerde databases een betere integriteit kunnen ontstaan. Natuurlijk staat of valt de integriteit ook met de kwaliteit van de "logica", met name de caching, op de middle tier, maar dat valt net buiten de scope van dit verhaal.
Indirect kan RAC (schaalbaarheid) er toe kan leiden dat meerdere losse, gekoppelde, systemen vervangen kunnen worden door een enkele, geïntegreerde, database. Als hierdoor minder interfaces en minder integratie-inspanningen nodig zijn, dan is dat een positief effect. Integratiespecialisten zullen opmerken dat er dan meestal nog steeds replicatie van het ene subsysteem naar het andere zal plaatsvinden, zij het niet langer via interfaces buiten de databank om. RAC heeft dus geen directe invloed op de integriteit van de database. Indirect kan het via schaalbaarheid een positief effect meebrengen.

## 2. Beschikbaarheid

Toepassing van RAC maakt de databasecomponent van een systeem minder kwetsbaar voor hardwareproblemen. De high availability die geboden wordt door RAC geldt echter alleen voor de systeemcomponenten die bij een server, of bij een zogenaamde node, horen: CPU(s), Memory, netwerkkaart(ten), voeding. Andere mogelijke storingen, zoals gebroken netwerk (-kabels), diskproblemen of user errors worden niet door een RAC-configuratie opgevangen. Boven een bepaald niveau zijn of worden CPU, geheugenbanken, voeding en netwerkkaarten hot-swappable, waardoor geen van deze hardwarecomponenten een single point of failure vormt. Dit pro-RAC argument wordt dus minder sterk naarmate de hardware relatief ongevoelig wordt voor storingen. Het argument "beschikbaarheid" voor RAC gaat dan alleen nog op wanneer een cluster samengesteld is uit relatief goedkope (NT, Linux) servers. Server hardware kan dermate degelijk worden uitgevoerd dat RAC nog maar weinig bijdraagt aan een betere beschikbaarheid.

## 3. Uitwijkmogelijkheden

Aangezien de nodes van een RAC systeem, meestal in dezelfde ruimte zullen staan levert RAC nog geen voordelen op voor een eventuele disaster recovery. Het bijhouden of ad hoc inrichten van een uitwijkcentrum wordt echter complexer, omdat idealiter ook op de uitwijksite alle componenten (servers, extra software) aanwezig moeten zijn. Voor efficiënte disaster recovery kan deze extra complexiteit een nadeel zijn.
Ter geruststelling: volgens het handboek kan een database die voor RAC geconfigureerd is, ook op een enkele server draaien, en de database kan ook single instance hersteld en beschikbaar gemaakt worden. Eventuele voordelen van RAC (performance, beschikbaarheid) verdwijnen dan grotendeels, maar de database is beschikbaar. Kortom, voor disaster recovery biedt RAC geen nadrukkelijke voordelen.

***Bevatten alle nodes te allen tijde compatibele softwareversies?***

## 4. Kosten en complexiteit

Voor degenen die de tijd en energie kunnen investeren in het lezen van meer dan alleen de white papers zal duidelijk zijn dat het inrichten van RAC helaas weinig of niet verschilt van het oude OPS. Correcte installatie en configuratie kost tijd en stelt hoge eisen aan dba en systeembeheerder. Het gebruik van RAC leidt voor de beheerders tot:

- additionele software componenten (de clustermanagement-software, deels afkomstig van de hardware vendor)
- management van raw-devices (op de meeste platforms, dit kan veranderen als er meer clustered-file systems beschikbaar komen)
- configuratie van de RAC-instances en het doorgronden en bijhouden van meer configuratie-informatie

Gebruik van RAC (of OPS onder 8i) houdt in dat het aantal te beheren componenten toeneemt (meer componenten) en

## Moderne applicaties, 3-tier toepassing

In de typische toepassing volgens Oracle (en andere vendors) bestaat een moderne applicatie uit drie delen:

1) De desktops en de on/offline mobile apparatuur van de eindgebruiker, liefst met alleen de browser
2) De middle tier met daarop de applicatielogica
3) De database met daarin de echte en enige waarheid

Globaal gezien heeft elke tier zijn eigen taak

- De front end moet leuk presenteren en presteren. Deze moet glad, licht en sexy zijn, liefst ook draadloos en mobiel. De front end bevat idealiter alleen een browser.
- Op de middle tier bevindt zich de applicatielogica. Hier worden de berekeningen uitgevoerd waarmee data bewerkt en gepresenteerd wordt. Deze component heft vooral rekenkracht (CPU) en geheugenruimte nodig. Oracle positioneert hier 9iAS met J2EE, webforms en eventuele andere componenten.
- De database tenslotte bevat de kroonjuwelen van de moderne organisatie. Hierin zit de enige en absolute waarheid van de applicatie, *het single point of truth.* Hier wordt de lijst van klanten en prospects bijgehouden, de status van de orders, de voorraad in het magazijn, en waarschijnlijk ook de lijst met debiteuren. De database stelt de hoogste eisen met betrekking tot degelijkheid, integriteit, opslagruimte en toegangssnelheid.

Idealiter is een applicatie zo opgebouwd dat beschikbaarheid en schaalbaarheid bereikt kunnen worden door meer middle tier machines (klonen) in te zetten. Dit kan leiden tot een zogenaamde server farm en vormt op termijn een uitdaging voor de systeembeheerder(s) omdat de middle tier machines bij voorkeur groepen van twee of meer identieke machines moeten zijn en blijven. Zie ook de: side-box over software versies op clusters.

We kunnen het niet laten om hierbij nog maar eens op te merken dat eventuele business-logica die kritisch is voor de integriteit van de applicatie bij voorkeur op de database geïmplementeerd moet worden. Al was het maar om te voorkomen dat incidentele ODBC toegang bepaalde business-rules per ongeluk omzeild en de data-integriteit aantast.

Geen enkele middle tier oplossing kan garanderen dat zijn ingebouwde logica door alle front-ends gebruikt zal worden. Het is altijd mogelijk dat een nieuwe component de database direct gaat benaderen via ouderwets SQL*net of NETx. Met de moderne architecturen en complexe business-logica ontstaat (terecht) de neiging om functionaliteit weer op de applicatieserver onder te brengen. Dit mag echter niet ten koste van de Integriteit gaan. Als een business rule werkelijk cruciaal is, moet men overwegen om hem te allen tijde op de databank onder te brengen in de vorm van constraints, triggers, table API's en stored procedures. Deze laatste wijsheid dateert trouwens al uit de periode van Oracle 7 toen constraints in de database ondergebracht werden.

Keerzijde hiervan is dan weer dat, wanneer steeds meer logica in de database zelf terechtkomt, de belasting van de databaseserver, de derde tier, steeds groter wordt. Een voorbeeld van en applicatie die in zijn huidige vorm beperkt schaalbaar zal blijken is waarschijnlijk Oracle Portal. Vrijwel alle logica is hierbij ondergebracht in stored procedures. Hiermee neemt het risico toe dat de databaseserver de (CPU-) flessenhals van de schaalbaarheid wordt. RAC pretendeert hier dan weer een oplossing voor te zijn, omdat het onbeperkte schaalbaarheid van de database belooft. Een mogelijke testlab casus dient zich aan.

diversifieert (meer verschillende componenten). Dit argument van extra complexiteit kan zwaar wegen in organisaties waar DBA en systeembeheer toch al in een spagaat staan om de lopende systemen gaande te houden. Uit oogpunt van complexiteit en kosten biedt RAC hier geen evidente voordelen.

## 5. Rolling upgrades

De RAC documentatie stelt in de versie ten tijde van het schrijven (v904) dat rolling upgrades nog niet ondersteund worden. Dit betekent dat een RAC- database, net als een gewone database down moet voor upgrades en patches. Op geclusterde file systemen waar niet alleen raw devices, maar ook echte file clustering mogelijk is (wie kent nog VMS?) is het waarschijnlijk wel mogelijk om voor minor patches een rolling upgrade toe te passen. Dit is echter vooralsnog expliciet niet ondersteund. Merk op dat enkele hardware vendors hun clustertechnologie wel positioneren met de mogelijkheid van rolling upgrades. Dit geld dan echter alleen voor upgrades op operating system- of applicatieniveau wanneer de applicatie zich daartoe leent. Samenvattend: RAC ondersteunt nog geen rolling upgrades.

## 6. Het killer-argument: schaalbaarheid en performance

RAC maakt het mogelijk om meer rekenkracht en meer geheugen aan een databank te koppelen in de vorm van meerdere nodes (clustered servers) waarop zogenaamde instances draaien. Op het eerste gezicht is dit het beste argument om RAC in te zetten: RAC maakt de database schaalbaar. Indien een systeem door groei van data of door toename van gebruik problemen vertoond, en indien de (performance-) bottleneck

aantoonbaar op de database terug te voeren is (observeren, meten, analyseren), dan is toepassing van RAC een mogelijke oplossing. Schaalbaarheid lijkt zelfs het meest voor de hand liggende argument om RAC toe te passen.

## Alternatieven voor RAC

Omdat na lezing van de (overigens zeer complete) Oracle Documentatie de toepassing van RAC, net als bij OPS, niet iets is voor een rustige vrijdagmiddag, willen we toch enkele over-

## Versiebeheer van software op clusters en server farms

Zowel RAC-databases als IAS-deployed applicaties gaan uit van een constructie waarin meerder servers eenzelfde taak uitvoeren. Bij RAC draaien meerder nodes de instances van een database. Bij IAS kunnen meerdere servers ingezet worden om de middle tier schaalbaar te maken. Andere software vendors passen met name op de middle tier eenzelfde load balancing constructie toe. Dit leidt steeds meer tot een wildgroei aan (kopieën van) softwareversies, al of niet gepatched. De Oracle Installer werkt hieraan mee door ingeval van geclusterde componenten (9iAS en RAC) software automatisch te repliceren over de gedefinieerde nodes. Dit leidt er toe dat elke node in een cluster of server farm zijn eigen directory-structuur met ORACLE_HOME, IAS_HOME en CLASSPATH heeft. Eventuele applicatiesoftware (J2EE, webforms) moet in dit geval ook n-maal gekopieerd worden, met alle risico's op fouten van dien.

We proberen een alternatief te schetsen.

## Aannames en uitgangspunten

- Een clusterplatform waarbij de verschillende nodes disk-sharing kunnen doen, e.g. op bepaalde disks kunnen alle nodes de files lezen en zo mogelijk schrijven. Nog niet alle hardware vendors zijn hier klaar voor.
- Een cluster met minimaal twee, maar waarschijnlijk meer nodes.
- Streven naar een minimum aantal kopieën van software. E.g. de IAS_HOME subdir en de applicatie jar-files komen liefst slechts 1x voor op het gehele systeem. In de praktijk is 2x handig, in verband met upgrades en eventuele menselijke fouten zoals "rm *".
- De wens om zoveel mogelijk identieke componenten (servers, nodes) te hebben. Het exclusief reserveren van een node (server) voor een taak (bijvoorbeeld report server) is ook een mogelijkheid, maar creëert een single-point-of-failure. Probeer dan om minimal twee identieke report servers in te richten in een cluster- of failover constructie.
- De wens om upgrades zo geruisloos mogelijk door te voeren (zie rolling upgrades).

Elke node van de cluster wordt vervolgens ingericht met zijn eigen rootdisk. Hierop komt alleen de essentiële software die nodig is om een node te laten draaien en om hem uniek te definiëren (e.g. operating system, host file en node-specifieke parameterfiles).
Bij het opstarten van het systeem wordt vanuit elke node een link gelegd naar de benodigde ORACLE_HOME, IAS_HOME en andere applicatiesoftware directories. Door alle nodes naar dezelfde directories, bv op mountpoint /sw1, te laten linken ontstaat de zekerheid dat ze allemaal dezelfde software versie gebruiken. Geen onzekerheden meer over wel-of-niet-complete installaties of replicaties.
Tenslotte moet het voor rolling-upgrades mogelijk zijn om een node uit de cluster te halen voor een upgrade. Om deze redenen zullen de software subdirs (ORACLE_HOME, IAS_HOME, CLASSPATH) twee keer op het systeem aanwezig moeten zijn. De te upgraden cluster wordt zodanig gemodificeerd dat hij zijn software van de andere directory boom, bijvoorbeeld /sw2, leest. De losse node kan nu gebruikt worden om deze tweede software boom te upgraden terwijl de andere clusters gewoon doorgaan met het bedienen van de gebruikers.
Als de upgrade van de software achter de rug is kan de node terug in het cluster, waarbij hij gebruik blijft maken van de nieuwe versie, /sw2, van de software. Als bij testen blijkt dat er geen fouten optreden dan kunnen vervolgens de andere nodes van de cluster één voor één omgeschakeld worden om gebruik te maken van de nieuwe versie op mountpoint /sw2. Merk op dat hierbij geen automatische replicatie hoeft plaats te vinden terwijl install- of upgrade toch maar één keer wordt uitgevoerd. De oude versie van de software kan enige tijd blijven staan op /sw1, tot men besluit dat terugrollen echt niet nodig is. Hierna dan met eenvoudige file-copy of met een aparte upgrade procedure ook deze software aangepast worden.

Tenslotte kan men bij performanceproblemen of piek belasting besluiten om de helft van de clusternodes te koppelen aan /sw1 en de andere helft aan /sw2 om een eventuele disk-bottleneck minder nijpend te maken.

Merk nogmaals op dat dit alleen mogelijk is op hardware-platforms die file-system-clustering aankunnen. RAC gaat, net als eerder OPS, uit van clustering op raw-devices omdat meer hardware platformen zich hiervoor lenen. Bovenstaande constructie is ook mogelijk met NFS, maar waarschijnlijk te traag voor praktische toepassing.

wegingen en alternatieven de revue laten passeren. Conventionele (archive log-) backups of een standby database zijn goede alternatieven als de criteria van "beschikbaarheid"en "uitwijk" zwaar wegen.

Met een standby database is zelfs een "rolling upgrade" met minimale downtime te realiseren. Neem in overweging dat elke belangrijke databank, ook een RAC, sowieso een goede backupstrategie nodig heeft. Er zal dus altijd tijd en moeite geïnvesteerd moeten worden in backup en uitwijk. Deze inspanning kan worden hergebruikt voor rolling upgrades. Heeft men eenmaal een

***Elke node van de cluster wordt vervolgens ingericht met zijn eigen rootdisk***

standby databases, dan is het additioneel inzetten van een RAC voor redenen van beschikbaarheid, uitwijk en rolling-upgrades al bijna overbodig. Kortom, een standby databank is altijd een goed idee, en maakt toepassing van RAC minder noodzakelijk.

De behoefte aan schaalbaarheid geeft impliceert dat een systeem draait op te weinig of te kleine hardware. Een voor de hand liggend alternatief is dus: sterkere hardware (in plaats van meer van hetzelfde om een RAC te bouwen). Op dit punt komen performance, kosten en complexiteit samen: een RAC vergt behalve investering in additionele hardware ook een additionele inspanning met betrekking tot systeeminrichting, hetgeen leidt tot meer complexiteit. Het "eenvoudig" vergroten van de server(s) is wellicht aantrekkelijker. Deze afweging moet door de beheerder gemaakt worden, maar de volgende overwegingen kunnen meespelen:

- Grotere hardware is meestal gemakkelijker te beheren dan een clustersysteem
- Grotere hardware is meestal van betere kwaliteit, hetgeen voordelen biedt voor beschikbaarheid

Samenvattend: Grotere hardware, indien nog mogelijk, is conceptueel aantrekkelijk.
Anderzijds kan de behoefte aan schaalbaarheid voortkomen uit het feit dat te veel systemen draaien op een enkel systeem. In dit geval is de remedie het neerzetten van meerdere, liefst identieke systemen. Dit zal het geval zijn als meerdere databanken op een enkele server draaien, of als middle tier en datatier eenzelfde systeem delen. In het algemeen geldt dat een goed ontworpen 3-tier systeem goed schaalbaar is door meer middle tier capaciteit in te zetten. Alvorens RAC in te zetten om een schaalbaarheidsprobleem op te lossen kan het verstandig zijn de gehele architectuur nog een kritisch te bekijken. En bij het onderbouwen van de beslissingen geldt, zoals altijd: meten en analyseren alvorens te beslissen (look before you leap).
Schaalbaarheid kan ook bereikt worden door het splitsen van systemen (tiers) over servers, indien de architectuur dit toelaat.

***Wel is RAC een goede aanvulling op de bestaande 3-tier strategie van Oracle***

## Rolling upgrades, de uptime-belofte die bijna vervuld is

Op de Compaq Support website vonden we de (Engelse) definitie van een rolling upgrade. Het is overigens opvallend, dat we op dit moment nog geen goede Nederlandse vertaling hebben. De definitie luidde:
*"A rolling upgrade is a software upgrade of a cluster that is performed while the cluster is in operation. Clients accessing services are not aware that a rolling upgrade is in progress. One member at a time is rolled and returned to operation while the cluster transparently maintains a mixed-version environment for the base operating system, cluster, and Worldwide Language Support (WSL) software."*

Merk op dat deze definitie gericht is op het operating system, maar dezelfde argumentatie is van toepassing op een groep van twee of meer middle tier machines, applicatie-servers met 9iAS erop. Men kan een server upgraden terwijl de anderen doordraaien, waardoor de gebruiker niets hoeft te merken.

Analoog hieraan lijkt RAC nu ook de mogelijkheid te openen om de database zonder downtime te upgraden door de nodes of instances een voor een log te koppellen, te upgraden en weer terug aan te koppelen. Als we vervolgens de 'Oracle 9i RAC concepts guide' erop naslaan vinden in de 'installation and configuration guide' de definitie:
*"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 1 (9.0.1) of Oracle9i Real Application Clusters does not support rolling upgrade."*

N.B.: Dit verhindert niemand om het toch eens te proberen, echter voor eigen risico.

Uit bovenstaande blijkt, dat RAC niet de oplossing is, indien de database performance niet de kern van het probleem is, dan is RAC niet de oplossing.

## Wanneer dan wel?

In welke gevallen is toepassing van RAC dan uiteindelijk nog wel aantrekkelijk? Waarschijnlijk zijn de mogelijke toepassingen voor RAC onder te verdelen in twee categorieën.

- Hardwarelimiet bereikt
- Piekbelasting

## Hardwarelimiet bereikt

Een systeem kan gegroeid zijn tot aan de limiet van de gegeven hardware (budget, policy). Indien men zeker weet dat de database de echte bottleneck is (meten, analyseren, bijvoorbeeld bij toepassing van Oracle Portal), dan kan RAC een uitweg bieden. Dit zal typisch optreden bij gevallen waar organisatie, budget of techniek de toepassing van "grotere" hardware onmogelijk maken, maar waar het bijzetten van "identieke" machines wel acceptabel is. Denk aan een Wintel-only organisatie. In dit geval geldt weer: eerst nadenken. Clustering op goedkope hardware is nog maar gedeeltelijk robuust, en cache fusion over TCP/IP is veel minder efficiënt dan over dedicated hardware.

***Oracle adviseert applicatie partitionering ter voorkoming van overhead van cache fusion***

## Piekbelasting

Indien een systeem plotselinge extreme piekbelasting moet aankunnen is RAC een interessante optie. Men kan namelijk de databank in RAC configuratie aanmaken en met slechts 1 of 2 nodes laten draaien. Indien nu plotselinge piekbelasting op treedt, bijvoorbeeld door verwerken van verkiezingsuitslagen, maandafsluitingen of extreem nieuws (news sites op 911), dan is het mogelijk om andere systemen in de organisatie in te zetten als additionele nodes in een RAC. Zo kan men bijvoorbeeld de development-servers, of servers die normaal voor andere toepassingen gebruikt worden bijschakelen. Let op: dit stelt hoge eisen aan inrichting en beheer. Maar het kan een goed alternatief zijn voor overdimensionering van het productiesysteem. En ook hier geldt weer: think before you configure. Is de hardwarestrategie hierop berekend of hierop aan te passen (interconnects, gebruik van een SAN)? Kan de beheersorganisatie het aan (onder stress een clusternode bijschakelen)? Bevatten alle nodes te allen tijde compatible softwareversies?

## Tenslotte

Uiteindelijk zal RAC als technologie een toepassing vinden voor specifiek gevallen. Wellicht zijn dat de hierboven beschreven situaties. Indien RAC overwogen wordt, herlees dan de eisen/wensen nog eens, en houd vooral in de gaten:

- Als de database niet de bottleneck is, dan is RAC niet de oplossing. RAC lost geen problemen op die op of door de middle tier veroorzaakt worden.
- RAC stelt (nog) hoge eisen aan de beheersorganisatie.

Wel is RAC een goede aanvulling op de bestaande 3-tier strategie van Oracle en andere leveranciers. De middle tier is al bijna per definitie schaalbaar (mits goed ontworpen) en met RAC belooft Oracle ook de data tier schaalbaar te maken. We zijn benieuwd hoe zich dit de komende jaren gaat ontwikkelen.

## Referenties


Titel: High Availability And Scalability Of Your Database Oracle 9i RAC documentation (v9.0.4) concepts, admin guide

Auteur: Tim Gorman

Hyperlink: www.orsweb.com/memberarea/techpapers/tdha.pdf



**Piet de Visser**

is als dba werkzaam bij CMG Trade Transport and Industry in Rotterdam en kan worden bereikt onder piet.de.visser@cmg.nl.