Die Zeitschrift *database pro* wird nicht mehr aufgelegt und ist im Handel nicht mehr erhältlich. Bei Fragen zu diesem Artikel aus dem Jahr 2010 wenden Sie sich bitte an Thorsten.Grebe@twg-it.de.

Rubrik

TECHNOLOGIEN | Clusterdatenbanken

Inhalt

Clusterdatenbanken erlauben den simultanen, transaktionssicheren Zugriff mehrerer Clusterknoten auf den selben Datenbestand. Oracle Real Application Cluster und MySQL Cluster gehören zu den bekanntesten Vertretern. In der Umsetzung der Clustertechnik und im mitgelieferten Funktionsumfang unterscheiden sie sich grundlegend.

Plattform

Linux und das für die jeweilige Datenbank zertifizierte Betriebssystem

Technik/Anwendung

Clusterdatenbanken

oraussetzungen

Oracle, MySQL

Autor

Thorsten Grebe arbeitet seit 2002 bei der science+computing ag. Er ist Oracle Certified Professional und Sun Certified MySQL Cluster Administrator.

Oracle Real Application Cluster vs. MySQL Cluster

Clusterdatenbank-Analyse

Es gibt nur wenige, marktreife Clusterdatenbanken. Prominenteste Vertreter sind *Oracle Real Application Cluster* und *MySQL Cluster*, die beide zum Oracle-Imperium gehören. Thorsten Grebe

Der Begriff Clusterdatenbank ist weder geschützt noch klar definiert. Alternativ werden Sie als Datenbank-Cluster und geclusterte Datenbank bezeichnet. Darunter wird eine Ansammlung von unabhängigen Datenbankservern verstanden, die alle in einem dedizierten Raum stehen und abstrakt als „unser Datenbank-Cluster“ bezeichnet werden. Geclusterte Datenbank bezieht sich häufig auf aktiv/passiv Cluster, bei denen

ein Standby-System im Failover-Fall den Datenbestand von der ausgefallenen Produktivinstanz übernimmt. Bei einer anderen, neueren Spezies von Clusterdatenbanken werden mehrere Knoten im Lesezugriff gestartet, um rasant Datawarehousebestände analytisch durchzupflügen (z.B. Exasolution). Nicht immer ist klar, was die Hersteller genau darunter verstehen und wie sie es mit der Transaktionssicherheit halten und ob eventuell nur ein lesender Zugriff toleriert wird. Hier wird unter Clustrerdatenbank eine Datenbank verstanden, die den simultanen und transaktionssicheren Zugriff mehrerer unabhängiger Clusterknoten auf ein und denselben Datenbestand erlaubt. Änderungen, die ein Knoten durchführt sind augenblicklich für alle anderen Clustermitglieder sichtbar und ebenfalls änderbar. Der Anwender merkt nicht, ob er seine Abfragen gegen eine Einzelinstanz oder eine Clusterinstanz richtet. Der Cluster wirkt nach außen wie ein einzelnes System. Wird diese engere Auslegung zusätzlich auf Systeme beschränkt, die sich seit einigen Jahren am Markt bewährt haben und für die angenommen werden kann, dass sie auch noch in der nahen Zukunft unterstützt werden, dann bleiben nur wenige marktreife Produkte für Clusterdatenbanken übrig: zu den prominentesten zählen

- Oracle Real Application Cluster (alias RAC) und
- MySQL Cluster (alias NDB Cluster).

Die Motivation zur Clustertechnik sind in der Regel die größeren Prozessor- und Speicher-Ressourcen, die durch zusätzliche Clusterknoten bereitgestellt werden können, parallele Verarbeitung, erhöhte Verfügbarkeit und Ausfallsicherheit.

Mehr Online: Die Entwicklungsgeschichten - RAC und NDB Cluster

Sowohl Oracle RAC als auch MySQL Cluster besitzen unterschiedliche aber lange Entwicklungsgeschichten. Detaillierte Informationen dazu finden Sie als Bonus im Online-Bereich der database pro zur aktuellen Ausgabe.

Vergleicht man die Entwicklungsgeschichten von Oracle RAC und MySQL Cluster, fällt ein Unterschied ins Auge: Die Clusterversion von Oracle verwendet dieselben Programmdateien wie die Einzelinstanz – es ist dieselbe Software in geänderter Konfiguration. Die Clusterlösung von MySQL wurde dagegen hinzugekauft. Eine unangenehme Konsequenz hieraus ist, dass der Funktionsumfang der Einzelinstanz von MySQL mit den Speichermodulen MyISAM und InnoDB nicht deckungsgleich ist mit dem Funktionsumfang von NDB Cluster ist. Eine Anwendung, die für einen MySQL-Server entwickelt wurde, lässt sich dementsprechend nur eingeschränkt transparent auf die Clustervariante von MySQL portieren.

- http://www.databasepro.de/Aktuelles-Heft

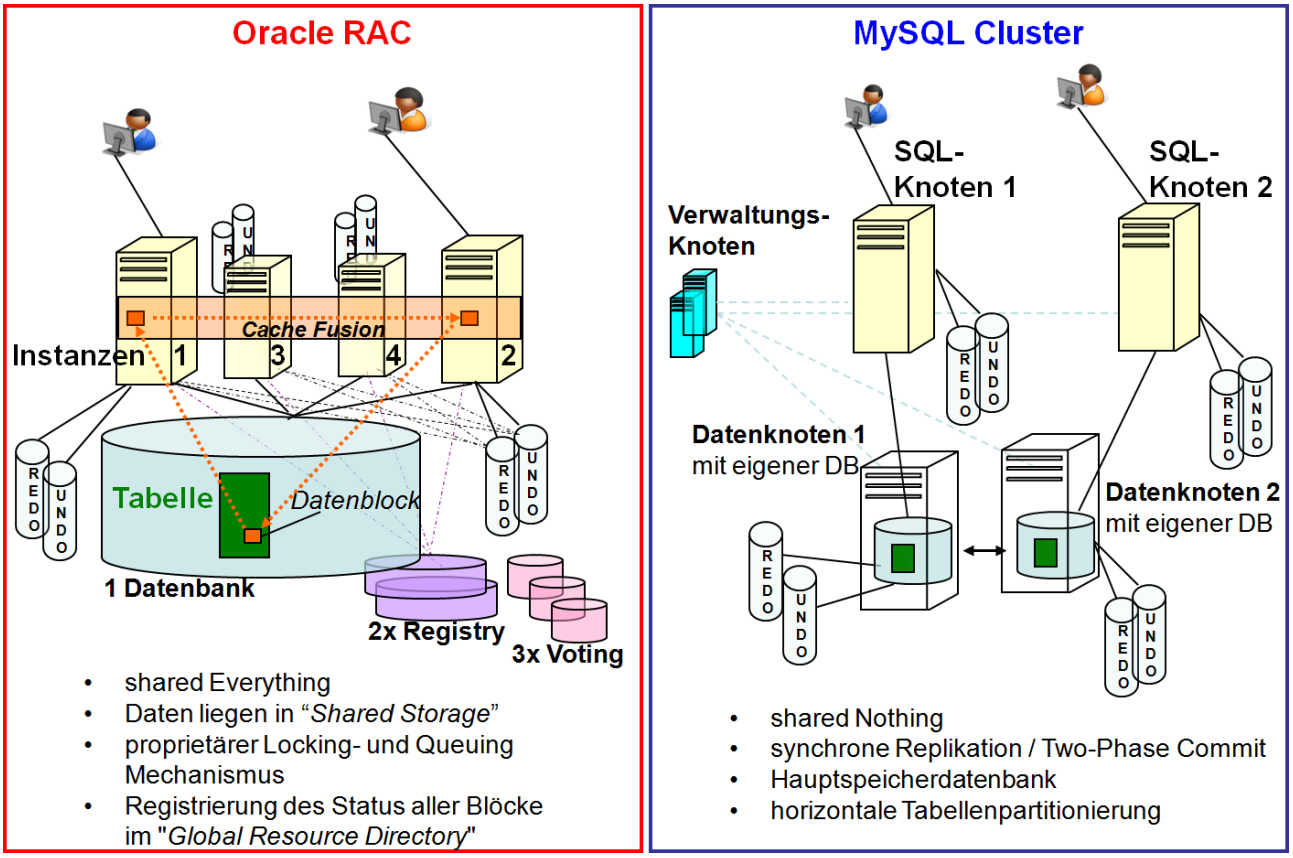
Architekturen im Vergleich

In beiden Entwürfen ist es gelungen, einen Datenbestand gleichzeitig durch mehrere Clusterknoten zu öffnen und zu bearbeiten. Bemerkenswert sind die unterschiedlichen Ansätze, mit denen Oracle RAC und MySQL Cluster dies bewerkstelligen. Während Oracle RAC den *Shared-Everything*-Ansatz wählt, bei denen alle speicherresistenten Komponenten des Systems auf einem gemeinsamen clusterfähigen Speichermedium

liegen müssen, verfolgt MySQL den *Shared-Nothing*-Ansatz, bei dem jeder Knoten autark ist und keine gemeinsamen Komponenten teilt.

RAC-Architektur

Beim *Real Application Cluster* werden alle dauerhaften Bestandteile des Clusters, Datendateien, Kontrolldateien, Parameterdateien, Undo-Tablespaces, Redologs, die Cluster-Registry und das Quorum (Cluster-Manager-Komponente, die die Datenintegrität bei einem Teilausfall sicherstellt) auf einer gemeinsamen, clusterfähigen Storage abgelegt (Bild 1).



Architekturen im Vergleich (Bild 1)

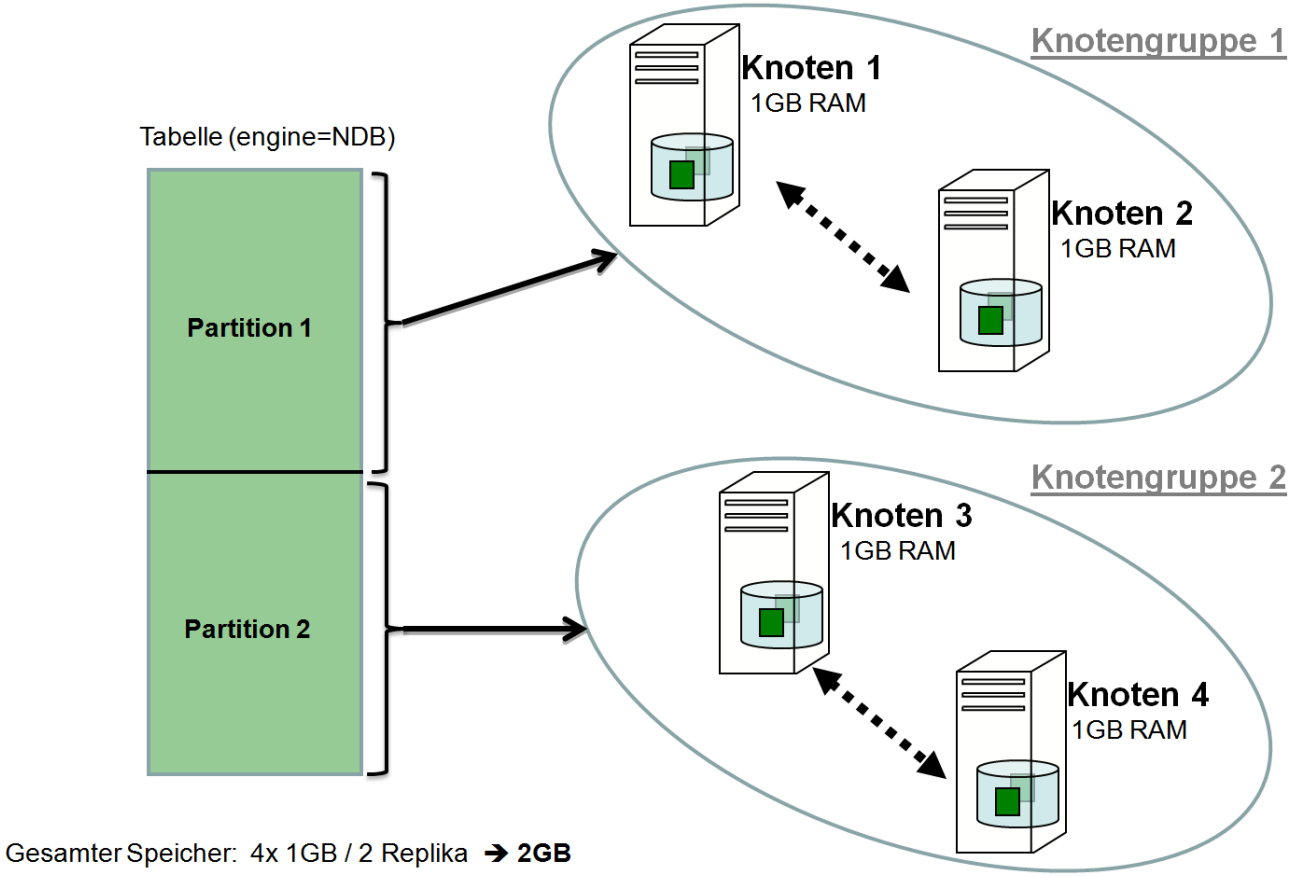
Oracle legt die Verwendung des Oracle Storage Managers ASM nahe, der im Gegensatz zu anderen Lösungen (Clusterdateisystem auf Hardware-RAID) die E/A-Last auf Extentebene optimal regulieren kann. Mit Ausnahme von Undo- und Redo-Dateien müssen alle Komponenten für alle Knoten les- und schreibbar sein. Für Undo- und Redo-Dateien gilt, dass nur der besitzende Knoten darin hineinschreiben darf. Alle Knoten müssen in der Lage sein, die Undo- und Redo-Information ihrer Nachbarknoten zu lesen, denn beim Instanzen-Crash eines Knotens müssen alle Nachbarknoten fähig sein, ein automatisches Instance-Recovery für den ausgefallenen Knoten auszuführen. Eine zentrale Rolle spielt die *Oracle Cluster Registry* (OCR), in der alle Cluster Ressourcen mit ihrem aktuellen Status inventarisiert werden. Hier steht beispielsweise auf welchem Knoten ein Service aktiv ist oder welche IP-Adresse deaktiviert ist, obwohl sie auf einer Instanz

konfiguriert ist. Die binäre OCR-Datei muss von allen Knoten aktualisierbar sein. Wegen ihrer Cluster-Bedeutung wird sie doppelt vorgehalten. Bei Verlust müsste der Cluster neu aufgesetzt werden. Das Quorum besteht aus den von Oracle so bezeichneten *Voting Disks*, von denen mindestens drei vorhanden sein sollten, die ebenfalls von allen Knoten gelesen und beschrieben werden müssen. Die *Voting Disks* entscheiden über die Mitgliedschaft im Cluster. Knoten, die den Zugriff auf das Quorum verlieren, werden von der Clusterware aus dem Clusterverbund ausgeschlossen.

Die größte Herausforderung ist der koordinierte Zugriff mehrerer Instanzen auf ein und denselben Datenbestand. Wenn eine Instanz ein Update auf eine Tabelle durchführen möchte, ist zunächst die Lage der betroffenen Datenblöcke zu ermitteln. Liegen diese noch auf der Festplatte oder wurden sie bereits in den *Buffer Cache* geladen? Falls sie bereits im Buffer Cache liegen, ist zu klären, in welchem (bei einem vier Knoten-RAC können die Blöcke über die Speicher von vier Instanzen verteilt sein). Ist der Blockbesitz festgestellt, ist dessen Status zu prüfen. Werden die einzelne Blöcke durch ihre besitzenden Instanzen bereits bearbeitet und sind mit einem exklusiven Lock belegt? Falls auf Lock-Freigaben zu warten ist, sind andere Instanzen auch bereits an dem einen oder anderen Datenblock interessiert? Ein ausgeklügelter Locking- und Queueing-Mechanismus ist hier gefordert, der performant sein muss und vor allen Dingen keine Fehler machen darf. Hat die Instanz, die das Update durchführen möchte, die erforderlichen Sperren gesetzt, dann ist die Statusänderung der betroffenen Blöcke sofort für alle anderen Instanzen zu veröffentlichen. Ein schnelles In-Memory Blockinventar muss hier stets für alle Instanzen lesbar auf dem aktuellen Stand gehalten werden. Oracle nennt dieses Blockinventar das *Global Resource Directory*. Nun kann ein komplexes Update, das in der *WHERE*-Klausel mehrere Tabellen verknüpft, mit einem Schlag Tausende von Blöcken anfassen. Möglicherweise führen viele Anwender ähnliche Vorgänge zur gleichen Zeit aus. Der Datenverkehr über den Cluster-Interconnect kann schwindelerregende Dimensionen annehmen.

MySQL-Cluster-Architektur

In der Architektur unterscheidet sich das MySQL-Cluster vom RAC in zwei wesentlichen Punkten: Zum einen verwendet es einen *Shared-Nothing*-Ansatz, zum anderen handelt es sich um eine Hauptspeicherdatenbank. Das Gegenstück zur *Shared Storage* beim RAC sind die Datenknoten (Bild 1). Es handelt sich hierbei um vollwertige Rechner mit eigenem Betriebssystem und einer kompletten Kopie der Datenbank bei zwei Knoten oder Teilen der Datenbank bei mehr als zwei Knoten. Ab vier Knoten werden NDB-Tabellen horizontal partitioniert und auf die Hälfte der Knoten verteilt (Bild 2). Die Datenbestände zwischen den Knoten werden über ein Two-Phase-Commit-Protokoll stets synchron gehalten. Dies bedeutet, dass ein erfolgreicher Commit nach einer DML-Anweisung die erfolgreiche synchrone Replizierung der Änderung auf einen Nachbarknoten voraussetzt, ansonsten gilt der Commit nicht als vollendet.



Horizontale Tabellenpartitionierung in MySQL (Bild 2)

Auch bei MySQL-Cluster gibt es Undo- und Redo-Dateien, in die (wie beim RAC) nur ihre Eigentümer hineinschreiben dürfen. Anders als beim RAC gilt hier das *Shared-Nothing*-Prinzip. Das Undo und Redo bleibt für Nachbarknoten unsichtbar. Dies ist hier in Ordnung, denn Dank der synchronen Replikation entfällt nach einem Knotenabsturz die Notwendigkeit für ein Instance-Recovery durch den überlebenden Knoten. So wenig, wie sich Anwender beim RAC direkt an der *Shared Storage* anmelden (was bei einem SAN, NFS-Mount oder einer ASM-Blackbox auch gar nicht möglich wäre), genauso wenig melden sich Anwender direkt an den Datenknoten im MySQL-Cluster an. Zwar könnten sie es, da die Datenknoten vollwertige Rechner mit vollständigen Betriebssystemen sind, jedoch ist ein Zugriff von dort auf die Clusterdaten nicht möglich. Anwender müssen sich an SQL-Knoten (auch API-Knoten) anmelden, um über diese mit den Clusterdaten zu arbeiten. Auf diesen läuft ein konventioneller MySQL-Server, der um eine Schnittstelle zur NDB-Engine angereichert ist.

Gewöhnungsbedürftig ist die Notwendigkeit bei MySQL Cluster-Funktionen einer Applikation teilweise auf die SQL-Knoten auszulagern. Dies betrifft Volltext-Indizes, Trigger, Views, Methoden und Events, die von der NDB-Engine nicht unterstützt werden. Aus diesem Grund wird ein Teil der Undo- und Redo-Information auch auf den SQL-Knoten erzeugt. Werden mehrere SQL-Knoten betrieben, hat sich der Administrator selbst um die Replikation dieser Objekte zu kümmern.

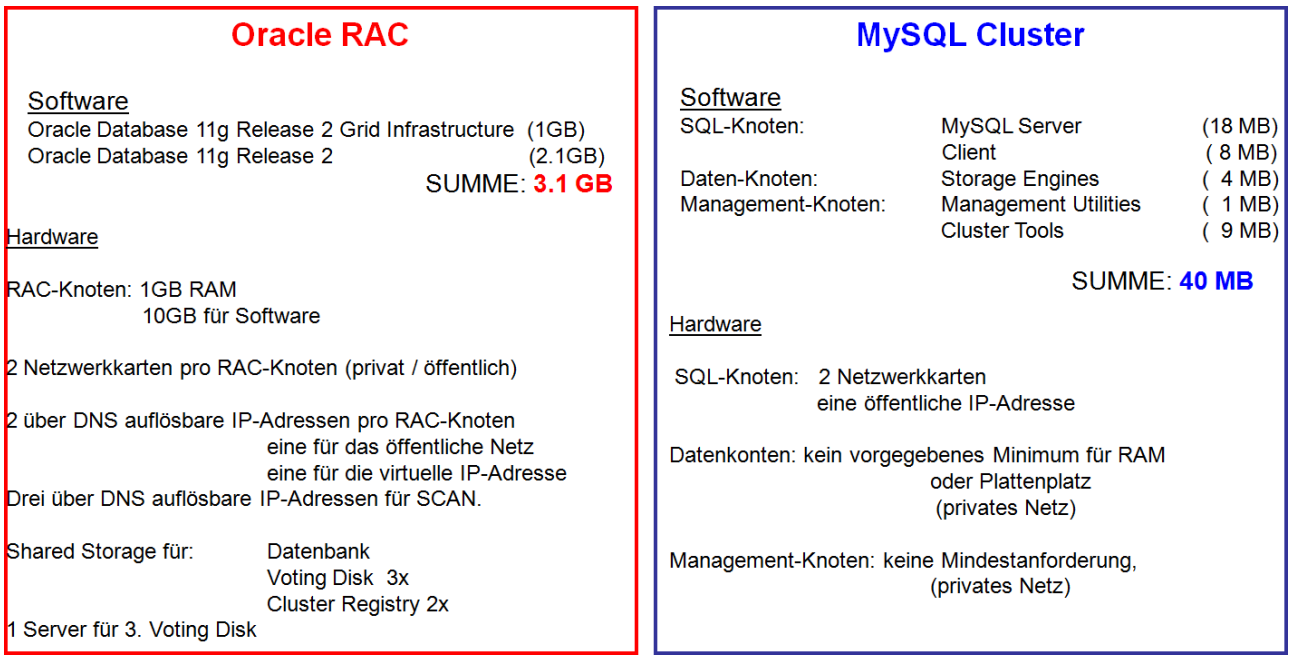
Die Rolle der *Cluster Registry* und *Voting Disks* beim RAC wird im MySQL-Cluster von einer dritten Knotenspezies, dem Verwaltungsknoten, übernommen. Auf diesem liegt die Konfigurationsdatei des Clusters, die, anders als beim RAC, eine übersichtliche, bearbeitbare Textdatei ist. SQL- und Datenknoten, die dem Cluster beitreten wollen, müssen sich beim Verwaltungsknoten anmelden und um Einlass bitten. Zur Erhöhung der Ausfallsicherheit können Verwaltungsknoten redundant betrieben werden. Bei einer Netzwerkunterbrechung zwischen den Datenknoten entscheidet der Verwaltungsknoten auf welchem Datenknoten der NDB-Prozess beendet wird, um eine Split-Brain-Situation zu verhindern. Beim RAC verhindern *Voting Disks* und Clusterware das Entstehen von Split-Brain-Szenarien.

Anders als beim RAC ist die Datenkapazität des MySQL-Clusters nicht durch die verfügbare Festplattenkapazität bestimmt, sondern in erster Linie durch das verfügbare RAM. Ist die Kapazität eines MySQL-Clusters erhöht werden, muss RAM erweitert werden. Dies erreichen Sie entweder durch Aufstocken des lokalen RAMs oder durch das Hinzufügen zusätzlicher Knoten. Erhöhen Sie die Knotenzahl von 2 auf 4, verdoppelt sich die Speicherkapazität des Clusters. Ab 4 Datenknoten wird der Datenbestand aufgeteilt, indem Tabellen horizontal (über Datensätze) partitioniert werden (Bild 2). Dabei wird jede Tabellenpartition redundant in einer Knotengruppe abgelegt. Die beiden Rechner innerhalb einer Knotengruppe decken sich gegen Systemausfälle, bilden also jeweils Ausfallgruppen. Tabellen werden in so viele Partitionen geteilt, wie es Knotengruppen gibt. Dies bedeutet, je mehr Knotengruppen es gibt, desto größer können die Tabellen werden und desto höher ist der Parallelitätsgrad beim Zugriff auf die Clusterdaten. Solange jeweils ein Rechner einer Knotengruppe überlebt läuft der Cluster weiter (in dem Szenario in Bild 2 dürfen insgesamt zwei Rechner ausfallen). Zusätzlich können zur Steigerung der Kapazität Daten in festplattenbasierte Tabellen ausgelagert werden.

Installation und Komplexität

Die Clusterdatenbanken unterscheiden sich in ihrer Komplexität. Während RAC in der Version 11gR2 einen Komplexitätsgrad erreicht hat, der eine mehrmonatige Einarbeitung erfordert, zeichnet sich der MySQL-Cluster durch ein extrem geradliniges und wohltuend schlankes Design aus, dass innerhalb von Tagen bis Wochen zu verstehen ist. Der Komplexitätsunterschied deutet sich bereits im Downloadvolumen der beiden Clusterdatenbanken an. Beträgt diese für eine RAC-Installation insgesamt 3,1 GByte, so sind es beim MySQL-Cluster lediglich 40 MByte. Auch die Anforderungen an die Hardware unterscheiden sich stark (Bild 3). In den frühen Zeiten der 10g-Version gab es eine kurze Bewegung zu geteilten Oracle Software-Homes. Da diese kein rollierendes Upgrade zulassen, geht der Trend zu eigenständigen Homes pro Knoten, so dass jeder RAC-Knoten eine komplette Softwareinstallation erhält. Nur bei einer hohen Knotenanzahl lohnt es sich über geteilte Homes nachzudenken. Pro RAC-Knoten werden zwei Netzwerkkarten benötigt, eine für den öffentlichen Zugriff und eine für den privaten Cluster-Interconnect. Zwei über DNS auflösbare IP-Adressen sind pro Knoten erforderlich, eine für den administrativen Zugriff und eine für die virtuelle IP-Adresse, über die Clients auf den Cluster zugreifen. Seit 11gR2 wird darüber hinaus mindestens eine weitere über DNS auflösbare IP-Adresse für den *Single Cluster Access Name* (SCAN), ein nach außen einheitlicher

Zugangspunkt zum Cluster, erwartet. Zugriffe sollen zukünftig über den SCAN erfolgen. Die Clusterware selbst regelt, welche Knoten-IP-Adressen hinter der SCAN stehen. Oracle empfiehlt drei öffentliche IP-Adressen für den SCAN zu registrieren. Schließlich muss viel Planung für die *Shared Storage*, den Datenbanksoeicher, *Voting Disk* und *Cluster Registry* aufgewandt werden. Mit der Storage steht und fällt die Performance des RAC. Haben Sie die Storage für eine erhöhte Ausfallsicherheit auf zwei Standorte verteilt, sollte noch ein weiterer Server eingeplant werden, der die dritte *Voting Disk* beherbergt, um gegen Split-Brain-Szenarien gewappnet zu sein.



Installationsvoraussetzungen (Bild 3)

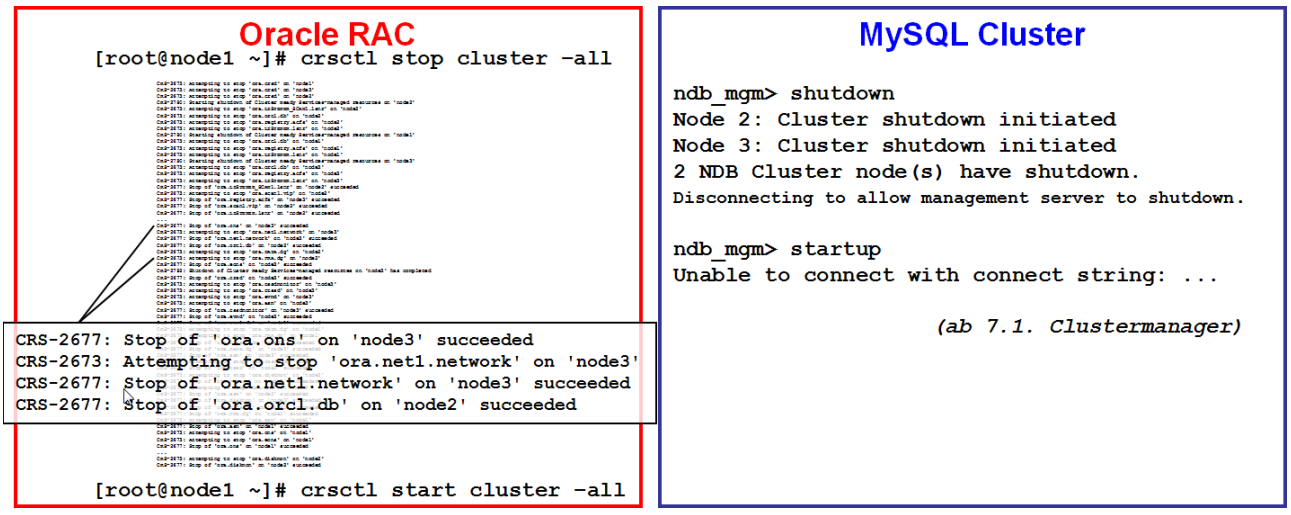
Das Anforderungsprofil des MySQL-Cluster ist deutlich kostensparender. Für die Datenknoten und Verwaltungsknoten gibt es keine ausgewiesene Mindestanforderung an RAM oder Plattenplatz. Klar ist, dass die Größe der MySQL-Cluster-Datenbank direkt vom verfügbaren RAM abhängig ist. Um jedoch einen MySQL Cluster zu installieren und zu starten, reichen ein paar Dutzend MByte RAM erst einmal aus. Sowohl Daten- als auch Verwaltungsknoten benötigen nur eine einzige Netzwerkkarte, die nicht im öffentlich Netz hängen soll, sondern nur für die Cluster-interne Kommunikation dient. Die SQL-Knoten benötigen dagegen zwei Netzwerkkarten, eine für den Zugang zum Cluster und eine für den Zugang zu den Clients. Auch für die SQL-Knoten gibt es keine Mindestanforderungen an Plattenplatz oder RAM. Mit ein paar Dutzend MByte von beiden lässt sich bereits der Betrieb aufnehmen.

Über 20 Prozesse sind beim RAC im Dienste der Clusterware tätig. Etwa 30 Prozesse werden von der ASM-Instanz gestartet, über 40 Prozesse sind notwendig um eine Datenbankinstanz auf einem 11gR2-RAC Knoten zu betreiben. Um die 100 Prozesse sind dies bei einer Standardinstallation pro Knoten, ohne dass spezielle Funkionen aktiviert sind. Auf den Datenknoten des MySQL Clusters laufen im Vergleich dazu

lediglich zwei *ndbd*-Prozesse, einer von beiden ist ein Wächterprozess, der den anderen, den eigentlichen Datenbankprozess nachstarten kann, falls der sich beendet. Auf dem Verwaltungsknoten läuft ein einzelner Managementprozess (*ndb_mgmd*) und auf den SQL-Knoten ein MySQL-Serverprozess (*mysql*) und ein Wächterprozess (*mysqd_safe*).

Verwaltung

Beide Clusterdatenbanken geben dem Administrator Verwaltungswerkzeuge für die Clusterprozesse an die Hand. Beim RAC sind dies mächtige Konsolenwerkzeuge wie das *Server Control* (srvctl) oder das *Cluster Control* (crsctl) mit denen Datenbankservices konfiguriert oder Cluster-Ressourcen deaktiviert werden. Bild 4 gibt einen Eindruck darüber, wie umfangreich sich ein simples Kommando wie *crsctl stop cluster -all* für einen RAC mit drei Knoten gestaltet. Wie ein Filmabspann laufen die Meldungen der *Cluster Ready Services* (CRS) an der Konsole vorbei. Es werden auf allen drei Knoten (Option *-all*) des Clusters die Clusterressourcen, die ASM-Instanz und die Datenbankinstanz gestoppt. Nur rudimentäre Prozesse der Clusterware (u.a. CRS selbst) laufen weiter, so dass mit dem Umkehrkommando *crsctl start cluster -all* der ganze Zauber in umgekehrter Reihenfolge abläuft, bis alle Datenbankinstanzen wieder geöffnet sind. Ein äußerst beeindruckendes Schauspiel, während dessen man kurze Zeit in Ehrfurcht erstarrt und hofft, das keine Fehler auftreten. Denn wehe dem, der hier debuggen muss.



Komplexität beim Stoppen/Starten des Clusters (Bild 4)

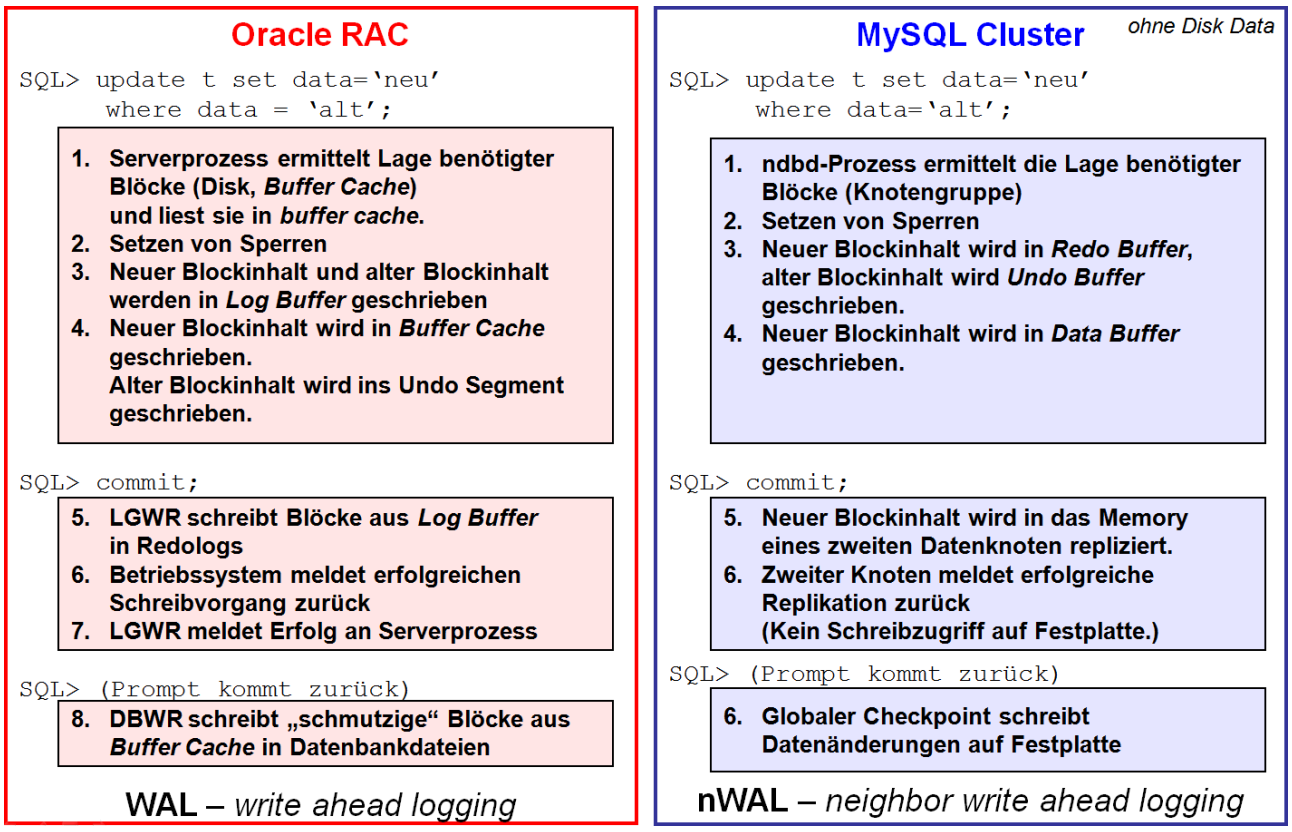
Auch der MySQL-Cluster bietet mit *ndb_mgm* ein Management-Interface, über das sich der Cluster einheitlich verwalten lässt. Mit dem Kommando *shutdown* werden die Clusterprozesse herunterzufahren. Da hier allerdings der Clusterwarestack fehlt, lässt sich nach einem Shutdown nicht einfach ein *startup* eingeben, denn der Managementdienst auf dem Verwaltungsknoten (*ndb_mgmd*), der hierzu notwendig wäre, wurde ebenfalls durch das Shutdown-Kommando gestoppt. Aber auch wenn der *ndb_mgmd*-Prozess wieder gestartet worden ist, reicht dies noch nicht, um den Cluster wieder zu starten. Die *ndbd*-Prozesse auf

allen Datenknoten müssen vom Administrator gestartet werden. Dieser Bereich der MySQL-Cluster-Verwaltung wirkt ein wenig vernachlässigt, offensichtlich hatten die Entwickler andere Prioritäten. Allerdings ist dabei zu bedenken, dass die Produktivsysteme in der Telekom-Branche so gut wie nie gestoppt werden und daher das Shutdown-Kommando somit von geringer Bedeutung ist. Ab Version 7.1 kann diese Funktionslücke mit dem lizenzpflichtigen Clustermanager geschlossen werden.

Beide Clusterdatenbanken erreichen ihr Ziel, die simultane, transaktionssichere Datenhaltung über mehrere Clusterkonten mit sehr großen Unterschieden in Installationsaufwand und Komplexitätsgrad. Von dieser eher oberflächlichen Betrachtung abgesehen zeichnen sich RAC und MySQL Cluster durch eine Reihe von Eigenheiten aus, deren sich ein Administrator bewusst sein sollte. Zu diesen zählen unter anderem das Transaktionsverhalten, Indizes, Sicherheitskonzepte, Kapazitätsgrenzen und der mitgelieferte Funktionsumfang.

Transaktionsverhalten

RAC und MySQL-Cluster sind transaktionssichere Datenbanken und genügen den ACID-Kriterien. Nach diesen müssen Transaktionen atomar (A), konsistent (C) und pro Session isoliert (I) behandelt werden. Außerdem müssen Datenänderungen nach einem Commit so dauerhaft (D) gespeichert werden, dass sie einen Systemabsturz überstehen (Bild 5).



Transaktionsverhalten und ACID-Konformität (Bild 5)

Wird in einer RAC-Instanz eine Tabelle aktualisiert, geschieht dies in folgenden Schritten: Der für die Clientsitzung gestartete Serverprozess ermittelt die Lage der für das Update benötigten Blöcke. Diese liegen auf der Festplatte oder bereits im Buffer Cache. Liegen sie im Buffer Cache, ist die besitzende Instanz zu ermitteln. Nach dem erfolgreichen Setzen von Sperren werden zunächst alter und neuer Blockinhalt in den Log Buffer geschrieben. Erst jetzt werden die Datenblöcke im Buffer Cache aktualisiert. Der alte Inhalt wird im Undo-Segment gespeichert. Hiermit ist das Update erfolgt. Innerhalb der Session kann jetzt mit den aktualisierten Daten gearbeitet werden. Dem Anwender steht es frei, die Änderungen mit Rollback rückgängig zu machen oder mit einem Commit zu erhalten und auch für andere Anwender sichtbar zu machen. Gibt er den Befehl *commit* ein, dauert es einen unmerklichen Moment bis der SQL-Prompt zurückkommt. In diesem kurzen Augenblick weist der Serverprozess den Logwriter (LGWR) an, den Inhalt des Log Buffer auf die Festplatte zu schreiben. Das macht er nicht selbst, sondern weist seinerseits das Betriebssystem an, die Transaktionslogs zu öffnen und zu beschreiben. Sobald das Betriebssystem den vollendeten Schreibvorgang zurückmeldet, gibt der Logwriter die Erfolgsmeldung an den Serverprozess weiter. Dieser wiederum signalisiert dem Anwender den erfolgreichen Commit durch die Rückkehr des SQL-Prompts. In diesem Moment könnte das System abstürzen ohne Transaktionen zu verlieren. Zu diesem Zeitpunkt sind die Datendateien selbst jedoch noch nicht vollständig aktualisiert. Dies erledigt der Database Writer (DBWR) einige Sekunden später, indem er die sogenannten „schmutzigen Blöcke“ in die Datendateien schreibt. Dieses Verfahren zur Gewährleistung der Transaktionssicherheit – zuerst in die Transaktionslogs zu schreiben und erst dann in die Datendateien – ist keine Erfindung von Oracle, sondern Standardverhalten für transaktionssichere Datenbankmanagementsysteme. Das Protokoll wird als *write ahead logging* (WAL) bezeichnet.

Genau in diesem Punkt besteht ein fundamentaler Unterschied zwischen Oracle- und MySQL-Cluster. Wird hier das gleiche Update durchgeführt, allerdings gegen eine NDB-Tabelle, dann geschieht folgendes: Anders als bei der Oracle-Instanz wird bei MySQL kein dedizierter Serverprozess gestartet. Der eine *ndbd*-Prozess, der auf den Datenknoten läuft übernimmt die Aufgabe, die Lage der für das Update benötigten Blöcke zu orten. Dabei ist (Disk-Daten außen vorgelassen) nicht zu prüfen, ob die Blöcke noch auf der Festplatte liegen oder im Buffer Cache. Da es sich um einen Hauptspeicherdatenbank handelt, liegen die Böcke in jedem Fall bereits im Speicher. Die Frage ist dabei allerdings, in welcher Knotengruppe im Speicher. Ab vier Knoten werden Tabellen automatisch nach ihrem Primärschlüssel partitioniert und auf die vorhandenen Kontengruppen aufgeteilt (Bild 2). Das Update wird parallelisiert auf den Knotengruppen ausgeführt. Nach dem Setzen von Sperren (im Übrigen wie bei Oracle auf Datensatzebene), wird der neue Blockinhalt in den Redo Buffer, der alte Blockinhalt in den Undo-Buffer geschrieben. Erst jetzt werden die eigentlichen Daten im Data Buffer aktualisiert. Damit ist das Update vollzogen. Auch hier kann der Anwender noch zurückrollen. Will er die Änderungen mit Commit festschreiben, dann vergeht auch beim MySQL Cluster ein kaum wahrnehmbarer Moment, bis das SQL-Prompt sich zurückmeldet. Anders als bei der Oracle-Datenbank wird hier in dem kurzen Moment des Commits nicht in Transaktionslogs geschrieben.

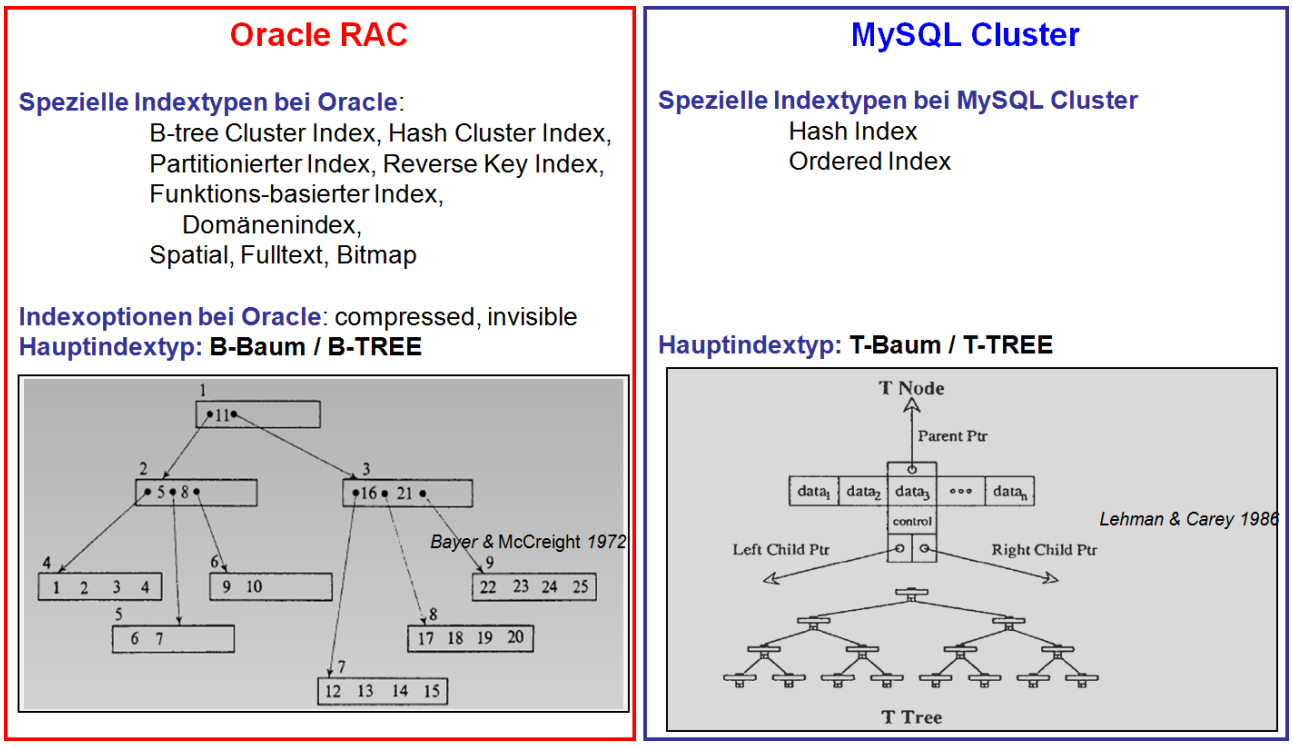
Statt dessen ergeht ein Schreibbefehl an die Nachbarknoten in den beteiligten Knotengruppen, die Synchronisierung der Datenänderung nachzuvollziehen. Erst nachdem die Nachbarknoten die erfolgreiche Replikation zurückmelden, kommt der SQL-Prompt des Anwenders zurück. Zu diesem Zeitpunkt erfolgt kein Schreibzugriff auf die Festplatte. Das System dürfte jetzt abstürzen, denn die Transaktionen sind im Memory des Nachbarknoten gespeichert und würden den Systemabsturz überleben. Nach einigen Sekunden werden bei MySQL-Cluster die Transaktionen in Form so genannter globaler Checkpoints auf die Festplatte geschrieben, so dass der Cluster gefahrlos herunterfahrbar ist. Im Gegensatz zur Oracle-Datenbank wird hier beim Commit also nicht in Transaktionslogs, sondern in das Memory des Nachbarknoten geschrieben, bevor die Daten in Datendateien verewigt werden. Dieses Verfahren ermöglicht es dem MySQL-Cluster nicht nur bei Lesezugriffen, sondern auch bei Schreiboperationen extrem performant zu sein. Das Protokoll wird als *neighbor write ahead logging* (nWAL) bezeichnet und wurde speziell für Hauptspeicherdatenbanken entwickelt.

Sind Sie sich dieses Unterschieds im Transaktionsverhalten nicht bewusst, können Sie unwissentlich in die Situation kommen, die Transaktionssicherheit zu gefährden, nämlich dann, wenn mehrere Datenknoten auf ein und demselben Host installiert werden. Dieses Szenario war vor der Version 7.0 durchaus üblich. Ab Version 7.0 kann der *ndbd*-Prozess in bis zu 8 Threads laufen. Davor konnte er nur als einfacher Thread ausgeführt werden, was den Administrator nach dem Kauf eines Multiprozessorsystems noch Anfang 2009 in Verlegenheit brachte. Statt auf einem Host mit vier CPUs nur einen einzigen *ndbd*-Thread laufen zu lassen, wurden vier Prozesse gestartet, was technisch vier individuellen Datenknoten entspricht. Achtet der Administrator hierbei nicht auf die Zusammensetzung der Knotengruppen und legt alle Mitglieder einer Knotengruppe auf ein und denselben Host, gefährdet er die Transaktionssicherheit. Stürzt der Server unmittelbar nach dem Commit ab und mit ihm alle Mitglieder einer Knotengruppe, gehen Transaktionen verloren. Eine ähnliche Gefährdung der Transaktionssicherheit erreicht man bei Oracle, wenn die Redologs zusammen mit den Datendateien auf die selbe Festplatte gelegt werden oder man nur ein Member pro Redologgruppe konfiguriert oder alle Member einer Redolog-Gruppe auf die selbe Festplatte legt. In beiden Fällen ist das D der ACID-Konformität betroffen. Die Kriterien A, C und I lassen sich bei Oracle durch Ändern des Transaktions-Isolationslevels manipulieren (Vorgabe ist READ COMMITED). Die MySQL-NDB-Engine kennt dagegen nur den Isolationslevel READ COMMITED, der sich nicht ändern lässt.

Indizes

Oracle bietet verschiedenste Indizes an (Bild 6). Trotz dieser Vielfalt bleibt der häufigste und wichtigste Index bei Oracle der B-Baum (B-Tree), ein Indexveteran aus der Pionierzeit relationaler Datenbanksysteme. Das „B“ deutet auf die Fähigkeit dieses Indextyps hin, sich selbst auszubalancieren. Der typische Aufbau ist anhand der originalen Veröffentlichung von Bayer und McCreight in Bild 6 gezeigt. Der Einstieg in den Index erfolgt über den Wurzelknoten, von dort gelangt man über die Zweige zu den Indexblättern, auf denen die Indexeinträge stehen. Erstellt man beispielsweise einen Index für eine Telefonnummernsuche, würde man die Namen der Telefonbesitzer indizieren. Bei einer Suche über Nachnamen würde es mindestens drei

Schritte benötigen (über Wurzelknoten, Zweig und Indexblatt), um an den gesuchten Datensatz zu gelangen. Hier stünde ein Verweis auf die Adresse des Datensatzes im Dateisystem, aus der die Telefonnummer gezogen werden könnte. Jeder Schritt bei einer solchen Indexsuche bedeutet im allgemeinen einen Zugriff auf die Festplatte und somit einen E/A-Vorgang. Lediglich den letzten Schritt können Sie wegrationalisieren, indem Sie die Telefonnummer mit in den Index legt, so dass die Anfrage nach einer Telefonnummer direkt aus dem Index beantwortet werden kann. Für B-Bäume ist es deshalb wichtig, dass sie flach bleiben und nicht in die Tiefe gehen; B-Bäume fächern auf und gehen in die Breite.



Indextypen im Vergleich (Bild 6)

Der MySQL-Cluster kennt nur zwei Indextypen: Den Hash Index für Primär- und Unique-Schlüssel, sowie den Sortierten Index (*ordered index*) für selbst erstellte Indizes (z.B. einen Index zur Telefonnummernsuche). Beim sortierten Index von MySQL-Cluster handelt es sich im Gegensatz zur Oracle-Datenbank nicht um einen B-Baum, sondern den speziell für Hauptspeicherdatenbanken entwickelten T-Baum, so genannt, weil die Indexknoten in der originalen Veröffentlichung von Lehman und Carey wie ein „T“ gezeichnet worden sind (Bild 6). Auch der T-Baum ist ein balancierter Index, das heißt er versucht die Zweige gleich tief zu halten. Er unterscheidet sich jedoch in zwei Eigenschaften vom B-Baum: Er ist ein binärer Baum, verzweigt also an jedem Zweig nur zweifach. Dadurch wird er nicht breit, sondern tief, was für eine Standarddatenbank negative Auswirkungen auf die Performance hätte, weil dort die E/A-Last direkt von der Indextiefe abhängt. Da der T-Baum aber komplett im Speicher liegt, spielt seine Tiefe keine Rolle. Der zweite wesentliche Unterschied besteht darin, dass die Indexeinträge des T-Baums extrem schlank sind.

Bei B-Bäumen versucht man nach Möglichkeit Anfragen direkt aus dem Index zu beantworten, um E/A-Vorgänge einzusparen. Deshalb wird die Telefonnummer mit indiziert, obwohl die Suche über den Namen erfolgt. Beim T-Baum sind solche Maßnahmen überflüssig, da die Daten ohnehin im Speicher liegen, es handelt sich schließlich um eine Hauptspeicherdatenbank. Alles was im Indexblatt steht, ist ein Zeiger auf den kompletten Datensatz im Memory. Diese Eigenart der MySQL Cluster Indizierung sollte dem Administrator bewusst sein: alle Indizes werden in Gänze in den Hauptspeicher geladen. Auch dann, wenn man festplattenbasierte Tabellen anlegt. Hier gilt, das zusätzlich jede indizierte Spalte einer festplattenbasierten Tabelle ebenfalls in den Hauptspeicher geladen werden muss, denn der sortierte Index verlinkt zum Datensatz im Hauptspeicher und nicht auf die Festplatte. Werden bei einer 10 GByte großen Disk-Data-Tabelle mit 5 Spalten alle 5 Spalten indiziert, dann müssen diese 10 GByte im Hauptspeicher Platz finden.

Sicherheitskonzepte

Bei der Handhabung von Sicherheitsaspekten haben Oracle und MySQL diametral unterschiedliche Ansätze verwirklicht. Bei Oracle RAC darf man getrost sagen, dass die Datenbank, so wie sie mit Standardeinstellungen (*„out of the box“*) installiert wird, sicher ist. Es gibt keinen Zugriff auf die Daten, ohne dass sich ein Anwender mindestens einmal mit Benutzernamen und Passwort authentisiert hat – und sei es am Betriebssystem. Beispielschemata sind gesperrt, wer mit SCOTT oder HR arbeiten möchte, muss diese Konten erst entsperren und sie mit einem eigenen Passwort belegen. Netzwerkpakete sind gesperrt: Nach der Migration auf 11g funktionieren Email-versendende Pakete oder Apex-Anwendungen solange nicht, bis man Netzwerk-ACLs (*access control lists*) formuliert, in denen einzelnen Datenbankschemata der Zugriff auf das Netzwerk gestattet wird. Wer vertrauliche Daten verwaltet, kann die Datenablage, den Netzwerkverkehr, Backups und Exporte verschlüsseln – dies setzt allerdings die kostenpflichtige *Advanced Security Option* voraus.

Bei MySQL Cluster haben die Entwickler die Verantwortung für die Sicherheit in die Hände des Administrators gelegt. Ein MySQL-Cluster, der mit Standardeinstellungen (*„out of the box“*) installiert wird, ist unsicher. MySQL verheimlicht dies nicht, es wird ausdrücklich in der Dokumentation darauf hingewiesen. Die NDB-Clusterdatenbank kennt keine Benutzerauthentifizierung. Die gesamte Information über Benutzerauthentifizierung und –authorisierung ist in MyISAM-Granttabellen auf den SQL-Knoten abgelegt. Ein großes Sicherheitsloch kann der Administrator aufstoßen, wenn er in der Konfigurationsdatei des Clusters so genannte „leere“ Slots einträgt und dabei den Netzwerkzugang zum Cluster nicht kontrolliert. Jeder Besucher kann dann seinen Laptop in das private Netz des Clusters hängen und mit seinem eigenen SQL-Knoten dem Cluster beitreten. Der Verwaltungsknoten lässt jeden SQL-Knoten in den Cluster, sofern noch ein leerer, anonymer Slot frei ist. Der Besucher hängt dann mit seinem eigenen Laptop und root-Zugriff im Clusterverbund unter Umgehung jedweder Authentifizierung. Dieses Sicherheitsloch muss vom Clusteradministrator bewusst geschlossen werden, entweder durch das Abschaffen von leeren Slots in der Cluster-Konfigurationsdatei oder durch das Abschotten des privaten Clusternetzes über Firewalls. Ferner ist

zu beachten, dass die MySQL-Server, die auf den SQL-Knoten laufen, in der Standardinstallation mit passwortfreien root-Konten ausgerüstet werden. Dies entspricht der Situation, die es bei Oracle-Datenbanken vor einigen Jahren noch gab, als Produktivsysteme mit bekannten Initialpasswörtern für den SYS-Benutzer liefen. Bei Oracle ist dieser Zustand Vergangenheit, bei MySQL noch Gegenwart. Erweiterte Sicherheitsoptionen wie sie Oracle liefert, z.B. zum Verschlüsseln der Datenablage oder des Netzwerkverkehrs, werden für den MySQL Cluster nicht angeboten.

Entscheidungskriterien

Was sollte geklärt werden, bevor man eine bestehende Datenbank auf eine der beschriebenen Clusterlösungen portiert? Die Art der Fragen, die sich der verantwortliche Datenbankadministrator in einer solchen Situation stellt, unterscheidet sich bei RAC und MySQL-Cluster stark voneinander. Zu den wichtigsten Fragen vor der Installation eines *Real Application Clusters* zählen die folgenden fünf:

1) Ist die Hardware zertifiziert?

2) Ist das Betriebssystem zertifiziert?

3) Welche Lizenzen werden benötigt?

4) Wie muss die *Shared Storage* ausgelegt sein, um die erwartete Performance zu liefern?

5) Ist genügend Know How vorhanden, um den RAC zu betreiben?

Bei MySQL Cluster zählen zu den wichtigsten Fragen, die zu klären sind:

1) Werden alle Datentypen unterstützt? Volltextindizes können in der NDB-Engine z.B. nicht abgebildet werden.

2) Werden Fremdschlüssel benötigt? Diese werden von der NDB-Engine nicht unterstützt.

3) Spielen komplexe Joins, Aggregierungen, Tabellenscans eine zentrale Rolle? Für diese ist die NDB-Engine nicht optimiert, sie müssen auf den SQL-Knoten ausgeführt werden.

4) Ist der Datenbestand klein genug, um zweifach in den Speicher zu passen oder muss auf Diskdaten ausgewichen werden, was zu Performance-Einbußen führen kann.

5) Wie soll mit Objekten umgegangen werden, die nicht auf den Datenknoten gespeichert werden können wie Views, Methoden, Trigger Grants, Scheduler Events?

Während sich beim RAC die Hauptsorgen um Infrastruktur und Kosten drehen, gilt die Hauptsorge beim Einführen von MySQL Cluster dem eingeschränkten Funktionsumfang der NDB-Engine. Und so viel Zeit wie das Ermitteln der korrekten Lizenzierung bei Oracle beansprucht, wird der Administrator beim MySQL Cluster verwenden, um notwendige Workarounds für fehlende Funktionalität zu recherchieren.

Fazit: Transaktionssicher

Oracle Real Application Cluster und MySQL Cluster sind zwei transaktionssichere, im Produktivbetrieb bewährte Clusterdatenbanken, bei denen alle beteiligten Clusterknoten gleichzeitig lesend und schreibend auf den Datenbestand zugreifen. Mit dieser Eigenschaftenbeschreibung enden auch schon fast die Gemeinsamkeiten. Beide Systeme unterscheiden sich grundlegend in ihrer Architektur: Während es sich

beim RAC um ein extrem komplexes *Shared-Everything*-Sytem handelt, besticht die *Shared-Nothing*-Architektur des MySQL Clusters durch das extrem schlanke Design. Eine Folge der großen Komplexitätsunterschiede ist der stark unterschiedliche Funktionsumfang, der bei den Systemen mitgeliefert wird. Während RAC eine Vollausstattung mitbringt, muss sich der Administrator beim MySQL Cluster verglichen mit den Engines des MySQL Standardservers auf einen stark eingeschränkten Funktionsumfang einstellen. Die Clustervariante von Oracle ist aus der selben Software entwickelt worden, mit der auch die Einzelinstanz betrieben wird, eine Portierung einer Anwendung von einer Einzelinstanz auf einen *Real Application Cluster* ist immer möglich und wird in der Regel von den Anwendern gar nicht bemerkt. Beim MySQL Cluster funktioniert so eine Portierung nur in Ausnahmefällen. Der MySQL Standardserver bietet Funktionen, die von der NDB-Engine nicht 1:1 umgesetzt werden können. Workarounds müssen gesucht werden (ein guter Starter hierzu ist das Oracle Whitepaper *„MySQL Cluster Evaluation Guide“* von April 2010). In den Disziplinen Kosten und Verfügbarkeit spielt der MySQL Cluster seine Stärken aus. Mit geringem Budget können 5 Neunen Verfügbarkeit erreicht werden. Beim RAC erfordert alles jenseits von 99% Verfügbarkeit das Einrichten eines Standbysystems, wodurch Kosten und Komplexität nochmals erhöht werden.

Als Konkurrenten zu einander kann man die beiden Systeme kaum bezeichnen, zu unterschiedlich sind sie in ihren Eigenschaften. Für beide gibt es Anwendungen, in denen sie jeweils herausragen. So ist ein RAC eine ideale Lösung, um Datenbanken in Fachabteilungen zu konsolidieren. Zusammen mit dem Servicekonzept und dem *Resource Manager* lassen sich hier ausgeklügelte Lösungen entwickeln, die mit einem MySQL Cluster nicht umsetzbar wären. MySQL Cluster andererseits ist immer dort eine Idealbesetzung, wo einfache Transaktionen mit Primärschlüssel-Lookups im Vordergrund stehen, Latenz kritisch ist, die Transaktionslast hoch und die Anforderungen an die Verfügbarkeit ambitioniert sind. Anwendungsbeispiele, in denen MySQL Cluster brilliert, sind Backends für LDAP, Webshops oder für eine neue Generation von Handy-Applikationen, bei denen der Standort des Handys berücksichtigt werden muss (*Location Based Services*). Es ist kein Zufall das die meisten Referenzprojekte für MySQL Cluster auch heute noch aus der Telekombranche kommen. (am)

[1] Ronströms Veröffentlichung; http://user.it.uu.se/~udbl/Theses/MikaelRonstromPhD.pdf
[2] Oracle RAC Download;
http://www.oracle.com/technology/software/products/database/index.html
[3] MySQL Cluster Download; http://dev.mysql.com/downloads/cluster
[4] MySQL Cluster Evaluation Guide; http://www.mysql.com/why-mysql/white-papers/mysql_cluster_eval_guide.php