Skalierbarkeit von Oracle RAC für DWH-Applikationen

Technical White Paper

Februar 2007

Karol Hajdu und Christian Antognini

Trivadis AG

Ziel dieses White Paper

Applikationen für Data Warehouses (DWHs) stellen im Bezug auf Virtualisierung und Skalierbarkeit sehr spezifische Anforderungen, die sich teilweise erheblich von denen einer OLTP-Applikation unterscheiden.

Dieses White Paper erläutert in groben Zügen diese spezifischen Anforderungen. Viele davon können durch eine Oracle Datenbank – sowohl bei RAC- (Oracle Real Application Clusters) als auch bei Non-RAC-Konfigurationen – gleichermassen gut erfüllt werden. Und wie die Praxis zeigt, *werden* sie auch erfüllt.

Der Kern dieses White Papers widmet sich jenem Teil der DWH-Anforderungen an RAC, für den nur sehr wenige Erfahrungsberichte aus der Praxis vorliegen.

Mit Unterstützung von Sun Microsystems hat Trivadis eine Reihe von Tests entworfen und durchgeführt. Dieses White Paper liefert technische Aussagen dazu, ob, wie und unter welchen Voraussetzungen eine Oracle RAC-Datenbank für eine DWH-Applikation skaliert.

Bei OLTP-Applikationen stellt der Einsatz von RAC eine sinnvolle und bewährte Lösung dar, die Verfügbarkeit der Datenbank zu erhöhen.

Im DWH-Bereich bietet Oracle eine skalierbare Datenbank-Engine an, in der eine Vielzahl von hochleistungsfähigen, auf DWH-Problemstellungen zugeschnittenen Features eingebaut sind – zum Beispiel Star Transformations, Query Rewrite, Partitionierung und Parallel Processing. Der Einsatz dieser Features innerhalb einer Non-RAC-Datenbank fand in den letzten fünf bis sieben Jahren eine breite Anwendung bei vielen DWH-Applikationen. Diese Applikationen wurden meist als zentrale DWH-Rechner aufgestellt, nach Bedarf wurden dann zusätzliche CPUs, Disk-Subsysteme oder Memory-Units integriert (sogenanntes Scale-Up).

In den letzten Jahren sind bei vielen Kunden einerseits die DWH Anforderungen so gestiegen, dass diese die höchsten Ausbaustufen eines zentralen Rechners übersteigen, andererseits liegen die Initial- und Ausbau-Kosten eines solchen High-End-Servers viel höher als diejenigen für mehrere kleindimensionierte Rechnern zusammen (sogenanntes Scale-Out). In dieser Situation liegt der Gedanke nahe, den Oracle RAC auch im DWH-Bereich einzusetzen.

Die Virtualisierung[1] der neu zugefügten Hardware-Ressourcen ist beim Scale-Up-Ansatz die Aufgabe von Hardware und Betriebsystem, denn aus Oracle Sicht handelt es sich um einen einzigen Rechner. Beim Scale-Out-Ansatz ist die Oracle Software – das heisst RAC – für die Virtualisierung der Hardware Ressourcen zuständig.

Neben den technologischen Überlegungen spielen beim Thema Datenbank-Skalierung auch die finanziellen Überlegungen – Stichwort Total Cost of Ownership – eine zentrale Rolle. Solche Überlegungen wurden in diesem White Paper bewusst ausgeschlossen, denn sie hängen sehr stark vom konkreten Projekt, der Preisgestaltung und -flexibilität des jeweiligen Hardware- und Software-Anbieters ab.

[1] Mit Virtualisierung ist gemeint: einzelne Ressourcen werden nach Typ zusammengefasst und via einem Abstraktions-Layer einer Applikation zur Verfügung gestellt.

Neben den gesammelten Erkenntnissen gibt Trivadis in diesem White Paper auch Empfehlungen für das Design, die Implementation und die Konfiguration von DWH-Systemen.

Dieses White Paper bezieht sich auf die Oracle Database 10g Enterprise Edition Release 2.

2 Einführung

2.1 Performance in DWHs

Unter **Performance in DWHs** versteht Trivadis folgendes:

- **ETL-Jobs werden in ihren definierten Zeitfenstern beendet.**
- **Abfragen, die oft von Anwendern ausgeführt werden, sollten nur wenige Hardware-Ressourcen beanspruchen.**
- Zur Tageszeit wird die Ressourcenvergabe auf solche Weise priorisiert und gegebenenfalls provisioniert, dass das Erzielen einer optimalen Performance[2] bei einer bestimmten Anwendergruppe[3] nicht zum Verlust der garantierten Performance[4] bei einer anderen Anwendergruppe führt.

Der Oracle Datenbank-Engine gehört im Bezug auf DWH-Performance klar und deutlich zu den Marktführern. Er bietet sehr viele nützliche Features, die sich in der Praxis bei Design und Implementierung grosser DWH-Applikationen erfolgreich bewährt haben. Dazu gehören:

- Eine fest eingebaute und verlässlich funktionierende Parallelisierung von umfangreichen Datenbank-Operationen: SELECTs, DMLs, Anlegen von Indexen und Constraints, Stored Aggregates und Berechnung der Objekt-Statistiken für Query Optimizer.
- Anlegen, Pflege sowie insbesondere sinnvolle Verwendung von Speicherstrukturen (zum Beispiel Indexe, Materialized Views und partitionierte Objekte), die zu einer wesentlich verminderten Verwendung von Hardware-Ressourcen und dadurch zu kürzeren Antwortzeiten führen (beispielsweise Star Transformation, Query Rewrite, intelligentes Partition Pruning).

2.2 Skalierbarkeit in DWHs

Unter **Skalierbarkeit in DWHs** versteht Trivadis die Fähigkeit der DWH-Applikation und der darunter liegenden Infrastruktur (Hardware, Betriebsystem und Datenbank), die bestehenden Ressourcen gezielt so auszubauen, dass:

- die bestehenden ETL-Jobs kürzere Verarbeitungsfenster brauchen (Motivation: kürzere Latenzzeit), **oder**

[2] Optimale Performance: kurzmöglichste Antwortzeiten
[3] Anwender-Gruppe: ETL-Jobs werden auch als Anwendergruppe betrachtet
[4] Garantierte Performance: maximal zulässige Antwortzeiten

- in bestehenden Verarbeitungsfenster zusätzliche ETL-Jobs terminiert werden können (Motivation: mehr Transformationen),
  oder

- bei erhöhter Simultanität (Anzahl gleichzeitig ausgeführten Abfragen) die garantierte Performance (maximale Antwortzeiten) und die optimale Performance (Antwortzeiten) unverändert bleibt,
  oder

- die Antwortzeiten für Abfragen der bestehenden Anwender mit bestehendem Abfrageprofil (Häufigkeiten, Datenraum) verkürzt werden, ohne dass dafür weitere redundante Strukturen aufgebaut wurden.

Bei der Skalierbarkeit in DWHs geht es also um:

- **Skalierbarkeit der Infrastruktur**

  Unter Infrastruktur werden Hardware, Betriebsystem und Datenbank-Engine als Einheit verstanden. Es geht hier zunächst um die Ausbaufähigkeit der Ressourcen: Diese sollte nicht beliebig sein, sondern zur Durchsatzerhöhung, respektive zu kürzeren Antwortzeiten führen. Dabei muss genau die Ressource ausgebaut werden können, die in der aktuell bestehenden Ausbaustufe und in dem gegebenen Workload den Bottleneck (Engpass) darstellt.

  Zudem geht es um die Virtualisierung der Ressourcen: Das Datenbank-Engine stellt den Applikationen ein passendes API zur Verfügung (SQL-Befehle mit passenden Features), so dass die Applikationen von der Ressourcen-Zuordnung und -Verwaltung möglichst abgeschirmt bleiben.

- **Skalierbarkeit der Applikation**

  Die meisten DWHs bestehen aus Applikationen folgender Typen:

  - **ETL-Applikation** – Batch-orientierte Datenbewirtschaftung und Informationsverteilung.
  - **Online-Abfrage-Applikationen** – OLAP und Online-Reporting.
  - **Komplexe Ad-hoc-Abfragen** mit Antwortzeit-Toleranz.

  Abgesehen von Online-Abfrage-Applikationen werden in DWHs meistens viele Daten gelesen und verarbeitet – das heisst „gejoint“ oder sortiert. In skalierbaren Applikationen sind die Verarbeitungsschritte mit vielen Daten so implementiert, dass Entscheidungen über die Reihenfolge der Teilschritte und über deren Parallelisierungsgrad an die DWH-Infrastruktur beziehungsweise an das Datenbank-Engine delegiert werden. Denn hier werden die Informationen über Datenvolumen und -verteilung sowie die verfügbaren Hardware-Ressourcen automatisiert ausgewertet. Sollte sich künftig das Verhältnis zwischen Datenvolumen und Hardware-Ressourcen ändern, entsteht auf der Applikationsseite nur minimaler Änderungsaufwand. Diese geeignete Implementierung der Verarbeitungsschritte lässt sich unserer Meinung nach nicht einfach durch die Verwendung eines Tools erreichen. Hierfür ist vielmehr ein gezieltes „Application Design for Scalability“ notwendig.

Aus unserer Projekterfahrung geht hervor, dass die Features der Oracle Datenbank eine sehr solide Basis für ein „Application Design for Scalability“ darstellen:

- Oracle stellt durch seine Features diejenigen geeigneten Interfaces zur Verfügung, mit denen eine Ressourcen-Virtualisierung des Datenbank-Engine möglich ist.

- Die Ressourcen-Virtualisierung in einer Single-Node-Konfiguration (Non-RAC) funktioniert gut.

2.3 Ziele des Trivadis Testvorhabens

Wenn eine DWH-Applikation statt auf einem einzigen Rechner künftig mit Hilfe eines RACs betrieben werden soll, sehen sich die technischen Entscheidungsträger meist mit folgenden Fragestellungen konfrontiert:

- Falls ich später ausbauen muss, wo sind dann die häufigsten Bottlenecks zu erwarten?

- Wie muss die DWH-Applikation aufgebaut werden, damit sie sich im Bezug auf Systemskalierbarkeit nicht selbst im Weg steht?

- Gegenüber Single-Node ist noch eine Gruppierungsebene für Ressourcen dazugekommen – die Knoten. Funktioniert die Ressourcen-Virtualisierung cluster-weit – das heisst instanz-übergreifend – immer noch gleich gut? Beim Thema Ressourcen-Virtualisierung geht es weiter auch um Load-Balancing.

Trivadis ist diesen Fragestellungen nachgegangen, hat geeignete Testfälle entworfen bzw. „designed", durchgeführt und ausgewertet. Die wesentlichen Erkenntnisse wurden in diesem White Paper zusammengefasst.

Das Thema der Zugriffsprovisionierung und -priorisierung im Bezug auf die Ressourcen – Stichwort Resource Plans – bildet zwar einen wichtigen Bestandteil der DWH-Performance, wurde in diesem White Paper über Skalierbarkeit bewusst nicht behandelt.

3 Management-Summary

Die Ergebnisse unserer praxisnahen Tests haben wir wie folgt zusammengefasst:

- Die **Erweiterung einer Oracle Umgebung um neue RAC-Knoten** kann **erfolgreich eingesetzt werden**, um die folgenden Zielsetzungen zu erreichen:

Zielsetzung der Skalierung	Ergebnis	Bemerkung
Bei einer höheren Anzahl gleichzeitig ausgeführter **Abfragen** sollen die	☺	

Antwortzeiten unverändert bleiben.		
Zusätzliche, unabhängige[5] **ETL-Jobs** sollen **im bestehenden Verarbeitungs-zeitfenster** beendet werden.	☺	
Bei gleicher Anzahl parallel ausgeführter **Abfragen** sollen mit bestehenden Indizes beziehungsweise Aggregaten die **Antwortzeiten verkürzt** werden.	☺	Dazu müssen jedoch einige wichtige Voraussetzungen im Bezug auf das physische Design des Datenbank-Schemas[6] erfüllt sein. Diese werden im Abschnitt 7.1.2 erläutert.

Bei der Hardware-Dimensionierung sollte daran gedacht werden, dass den neu zugefügten Knoten immer auch eine **entsprechend dimensionierte** Leistung des Disk-Subsystems und des Interconnect-Netzwerks gegenüber stehen muss. Die Praxiserfahrung von Trivadis zeigt, dass dieser logischen Voraussetzung oft wenig Aufmerksamkeit geschenkt wird.

- Die Erweiterung um neue RAC-Knoten kann bei der folgenden Zielsetzung leider **nur einen unwesentlichen Beitrag zur Performance-Steigerung leisten**. Unserer Meinung nach steht dieser Beitrag in einem sehr ungünstigen Verhältnis zu der damit verbundenen Komplexitätssteigerung:

Zielsetzung der Skalierung	Ergebnis	Bemerkung
Der bestehende ETL-Job soll eine kürzere Verarbeitungsdauer (Latency) vorweisen.	☹	Im Gegensatz zum Abfragebereich lässt sich im ETL-Bereich die PX-Kommunikation der umfangreichen SQL-Befehle durch ein geeignetes physisches Modell generell nicht wesentlich vermindern. Umfangreiche SQL-Befehle sind oft mit sehr viel PX-Kommunikation verbunden, bei welcher in der aktuellen RAC-Implementation wegen des erhöhten Synchronisations-Overheads ein Bottleneck entsteht. Dieses Thema wird im Abschnitt 7.1.3 erläutert.

In der aktuellen Implementation der Oracle Datenbank wird keine physische oder logische Ressource[7] so verwendet, dass sie bei der Ausführung einer typischen DWH-Last[8] einen Bottleneck

[5] Unabhängig bedeutet, dass die ETL-Jobs weder kausal von einander abhängig sind (ein Zweiter liest, was der Erste geschrieben hat), noch in eine gemeinsame Zieltabelle modifiziert.
[6] Wenn bei DWH-Applikationen absehbar ist, dass diese Voraussetzungen nicht erfüllbar sind, wird empfohlen, diese Zielsetzung nicht durch das Addieren neuer Knoten, sonder durch das Aufstocken der Knoten selbst zu erreichen.
[7] Zum Beispiel der Eintrag im Global Resource Directory
[8] Typische DWH-Last: Last, welche im Normalfall anfällt, wenn die DWH-Problemstellungen so implementiert werden, dass sie den Faktor Performance (Durchsatz) berücksichtigen.

mit globaler Auswirkung darstellt – und dass diese Verwendung grundsätzlich eine Skalierungsblockierung bedeuten würde.

Trivadis hat jedoch einen starken, nichtlinearen Zuwachs an Durchlaufzeiten bei bestimmten Situationen messen können. Dieser tritt bei paralleler Ausführung auf, wenn die Slave-Prozesse über die Knotengrenzen hinaus intensiv miteinander kommunizieren müssen. Die Ursache des Durchlaufzeitenanstiegs ist ein nichtlinear steigender Synchronisation-Overhead bei dieser Kommunikation. Dieses Thema wird im Abschnitt 6.1 erläutert.

Die Ergebnisse wurden der Firma Oracle zur Verfügung gestellt. Gemäss Aussage von Oracle sind bei vielen Kunden einzelne ETL-Jobs erfolgreich über Knoten hinweg skaliert worden. Zu den Ergebnissen der Tests meint Oracle: „In den hier beschriebenen Fällen begrenzt die Hardware des Interconnects und die Implementierung der Message Protokolle im Betriebssystem die Skalierung. Gegenüber den getesteten Gigabit Ethernet und IP-Protokoll skaliert Infiniband als Interconnect in Verbindung mit einem Low-Latency Protokollen wie RDS deutlich besser“.

Trivadis ist überzeugt, dass Oracle in den nächsten Releases die bestehende, solide Basis ausbauen und auch die oben beschriebenen Lücken schliessen wird. Mehr Details hierzu finden Sie im Abschnitt 8.

Mit RAC stellt Oracle seinen Kunden eine hochleistungsfähige Technologie zur Verfügung. Wie bei jeder Technologie **kommt es auch hier auf die richtige Anwendung an. Dies beginnt mit der richtigen Positionierung der Technologie in Hinblick auf die verfolgten Ziele, was einerseits eine gute Kenntnisse der Stärken und Schwächen dieser Technologie und andererseits ein klar definiertes Einsatzziel voraussetzt.** Die Abschnitte 6 und 7 sollen hier als Einstiegspunkt in diese Thematik dienen. Für weitergehende Überlegungen sind zwingend die projektspezifischen Aspekte in Betracht zu ziehen.

4 Design der Testfälle

Beim Design der Testfälle sind wir von unseren Erfahrungen bezüglich „DWH Design for Performance“ ausgegangen:

- Ein hoher und skalierbarer ETL-Durchsatz wird erreicht, wenn **Transformations- und Datenbank-Engine eine Einheit bilden**, das heisst, die ETL-Transformationen werden vom Datenbank-Engine ausgeführt. Falls die Transformationen mit einem ETL-Tool entwickelt werden, sollte dieses Tool über eine Generator-Komponente verfügen, welche die Transformationslogik in die SQL-Befehle umwandelt.

- Ein hoher und skalierbarer ETL-Durchsatz wird erreicht, wenn die **Transformationslogik bei der Umwandlung in die Datenbank-Befehle mengenorientiert und nicht zeilenorientiert** (also mit Iteration) formuliert ist.

- Im ETL-Bereich werden bestimmte Zwischenresultate vorübergehend in so genannte transiente Strukturen (Stage-Tabellen) geschrieben[9]. Der Inhalt dieser transienten Strukturen ist nur während eines ETL-Schedules von Bedeutung. Je nach dem, wie dieses Abspeichern der Zwischenresultate (Staging) implementiert ist, kann es zu einem wesentlichen Teil der ETL-Laufzeiten beitragen. Einen wichtigen Bestandteil des „Design For High ETL Throughput" bildet der Grundsatz, dass alle Zusatzoperationen, die einerseits für die Ausfall- und Konsistenz-Sicherung (Redo-Logging, Undo-Data) und andererseits für die Komprimierung (Data Segment Compression, Conventional Insert unterhalb High-Water-Marke) notwendig sind, explizit ausgeschaltet werden.

- Im ETL ist es nicht die Aufgabe der Datenbank, bei simultanen DML-Operationen über logisch getrennten Bereichen **einer** Zieltabelle, die Serialisierung der darunter liegenden Kleinstoperationen zu übernehmen. Es ist die Aufgabe vom Design der ETL-Applikation, den Aufbau der Jobketten so zu wählen, dass der ETL-Scheduler diese Serialisierungsaufgabe effizient übernehmen kann. Das bedeutet:

    - Jobs mit „Konkurrenzpotenzial" bei der Zieltabelle sollten eine möglichst kurze Ausführungsdauer haben[10].

    - In den Jobplänen (Schedules) sollten diese Jobs mit „Konkurrenzpotenzial" mit Scheduler-Mitteln in eine passende Hülle der gemeinsamen „kritischen Region umklammert" werden, so dass nur einer von ihnen gleichzeitig ausgeführt werden kann.

Basierend auf den oben ausgeführten Überlegungen lässt sich eine DWH-Applikation aus der Sicht der Skalierungsbeurteilung auf eine Menge von SQL-Befehle reduzieren. In diesem Fall lässt sich die Skalierungsproblematik auf folgende Fragestellungen aufteilen:

- Welche umfangreichen SQL-Befehle werden bei der DWH-Implementierung sehr häufig verwendet?

- Können diese SQL-Befehle vom Ausbau der Hardware-Ressourcen automatisch profitieren? Das heisst, skalieren diese SQL-Befehle als Einzeleinheiten?

- Wird bei der Ausführung dieser SQL-Befehle oft auf eine globale Ressource (kritische Region) zugegriffen, so dass es ab einer bestimmten Prozessanzahl zu einer globalen „Contention" kommen kann?

- Ist für diese SQL-Befehle eine Virtualisierung der Hardware-Ressourcen möglich?

- Falls es sinnvoll oder nötig ist, die Last der SQL-Befehle nicht auf alle Hardware-Ressourcen (vor allem Knoten) gleichmässig zu verteilen: Funktioniert in diesem Fall für solche SQL-Befehle, die während der Ausführung von anderen SQL-Befehlen starten, ein geeignetes Load-Balancing?

Beim Design der Testfälle sind wir zudem von unseren Erfahrungen bezüglich Oracle Datenbank in Non-RAC-Konfiguration ausgegangen:

[9] Der Zweck: Entweder, um die zu integrierenden Datenflüsse zu synchronisieren oder um Statistiken über die Datenverteilung in den Zwischenresultaten zu erhalten.
[10] Sie sollten also nur das DML-Statement enthalten, möglichst ohne aufwändige Joins/Sorts und Sub-Queries.

- Die parallele Ausführung der SQL-Befehle funktioniert „gut und verlässlich“[11]. Es geht dabei vor allem um:
  - SELECTs mit jeglicher Art von Joins, GROUP BYs, und INDEX-Zugriffe
  - INSERT APPEND, UPDATE, DELETE, MERGE
  - Erstellen von Indexen
  - Enablen/Überprüfen der Foreign Key Constraints
  - Berechnen der Objekt-Statistiken

- Das Space Management funktioniert gut und beansprucht kaum globale Operationen, wie zum Beispiel Object Checkpoints bei Direct Path Reads, usw. Voraussetzung hierbei ist, dass:
  - die geeigneten Features verwendet werden, wie zum Beispiel Locally Managed Tablespaces,
  - all diese Features zusammen richtig konfiguriert sind.

Basierend auf diesen Überlegungen wurden Fragestellungen identifiziert, die wir durch eigene praxisnahe Tests beantwortet haben wollten:

- Werden bestimmte Data Flow Operationen (wie Joins, GroupBys etc.) von mehreren Slave-Prozessen parallel durchgeführt, dann müssen die Input-Daten nach speziellen Aufteilungsregeln (Distribution Method) auf die Menge der Slave-Prozesse aufgeteilt werden. Diese Datenverteilung geschieht über das Versenden so genannter Parallel Execution Messages. Der Versand findet über die PX-Channels (Buffers im bestimmten Bereich der SGA) statt. Bei Single-Instanz kommunizieren alle Slave-Prozesse über ein und dieselbe SGA. Sind beim RAC die Slave-Prozesse über mehrere Instanzen verteilt, stellen sich folgende Fragen:
  - Welcher Hardware-Ressourcentyp (CPU, Interconnect, Memory) werden bei dieser Art der Message-Übertragung (so genannte Inter-Node-PX-Kommunikation) gebraucht – und wie intensiv?
  - Wie verhält sich die Struktur dieses Ressourcenverbrauchs bei steigendem Durchsatz des Input-Streams (steigende Anzahl der PX-Messages pro Zeiteinheit)?
  - Wird bei der Übertragung eventuell auf eine globale logische Ressource zugegriffen – zum Beispiel auf Einträge im Global Resource Directory, welche ab einem bestimmten Durchsatz zur vollständigen Blockierung der Skalierung führen würde?

- Kurz nach dem Absetzen eines SQL-Befehls werden dem „Parallel Execution Coordinator“ die Slave-Prozesse zugewiesen. Solange es auf der Oracle Instanz, auf der die Koordinator-Session läuft, noch ausreichend freie Slave-Prozesse gibt, werden diese aus dem Slave-Pool dieser Instanz genommen. Ist dies nicht der Fall, stellt sich folgende Frage:

[11] „gut und verlässlich“ bezieht sich auf die parallele Ausführung. Im Bereich der Optimierung für die parallele Ausführung sind uns noch Probleme bekannt, und zwar bei der Wahl der passenden Distributionsmethode und bei der Festlegung der Reihenfolge der Rowsource-Operationen im Execution Plan. Dies kann zu suboptimalen Execution Plans führen.

- Wie werden die Slave-Prozesse aus den verschiedenen Slave-Pools der Instanzen alloziert? Bei der Allozierung müssen zwei Ziele verfolgt werden: Einerseits sollte das Volumen der Internode-PX-Communication minimiert werden. Andererseits sollte eine gleichmässige Benutzung der Knoten (Load-Balancing[12]) erzielt werden.

Für diese Problemstellungen haben wir geeignete Test-Cases erstellt. Die Beschreibung dieser Test-Cases würde den Umfang dieses White Papers allerdings sprengen.

Beim Design der Test-Cases wurden die folgenden Rahmenbedingungen berücksichtigt:

- Das Ziel der Test-Cases liegt nicht im Benchmarking, sondern in der Gewinnung neuer Erkenntnisse über die Zusammenhänge innerhalb von RAC, die wir auf andere Weise – zum Beispiel Dokumentationsrecherchen, Praxiserfahrung – nicht erwerben konnten.

- Die SQL-Befehle der Test-Cases arbeiten mit einem einigermassen realitätsnahen Datenvolumen.

- Die Elapsed Times der SQL-Befehle sind lang genug – ein bis zwei Minuten – um in der Waits-Struktur systematische Zusammenhänge ausfindig machen zu können

- Der Workload (Art der SQL-Befehle) ist einfach parametrisierbar, um die Umverteilungen in der Waits-Struktur analysieren zu können.

- Die Daten sowie der Workload sind zwar „synthetisch“, sie orientieren sich aber an den Trivadis Kundenprojekten und sich daher praxisnah.

- Die Test Cases sollten auf einer realitätsnahen, ausbalanciert dimensionierten Hardware-Infrastruktur durchgeführt werden. Unmittelbar nach der Installation und Konfiguration wurde daher die Genauigkeit der Ausbalancierung zusätzlich durch geeignete Prüfskripte verifiziert.

5 Testinfrastruktur

Sun Microsystems hat uns in seinem Solution Center in Frankfurt eine den Zielen und Rahmenbedingungen unserer Test Cases entsprechende Hardware-Infrastruktur zur Verfügung gestellt.

Abbildung 1 zeigt in groben Zügen die Architektur und Dimensionierung unserer Test-Umgebung.

[12] Das Load-Balancing anhand der Allozierung der Koordinator-Session (Load-Balancing via Listener oder Client-Konfiguration) bildet hier keine richtige, sondern nur eine unterstützende Lösung. Der überwiegende Ressourcenverbrauch findet erfahrungsgemäss nicht beim Coordinator, sondern in der Regel in den Slave-Prozessen statt.

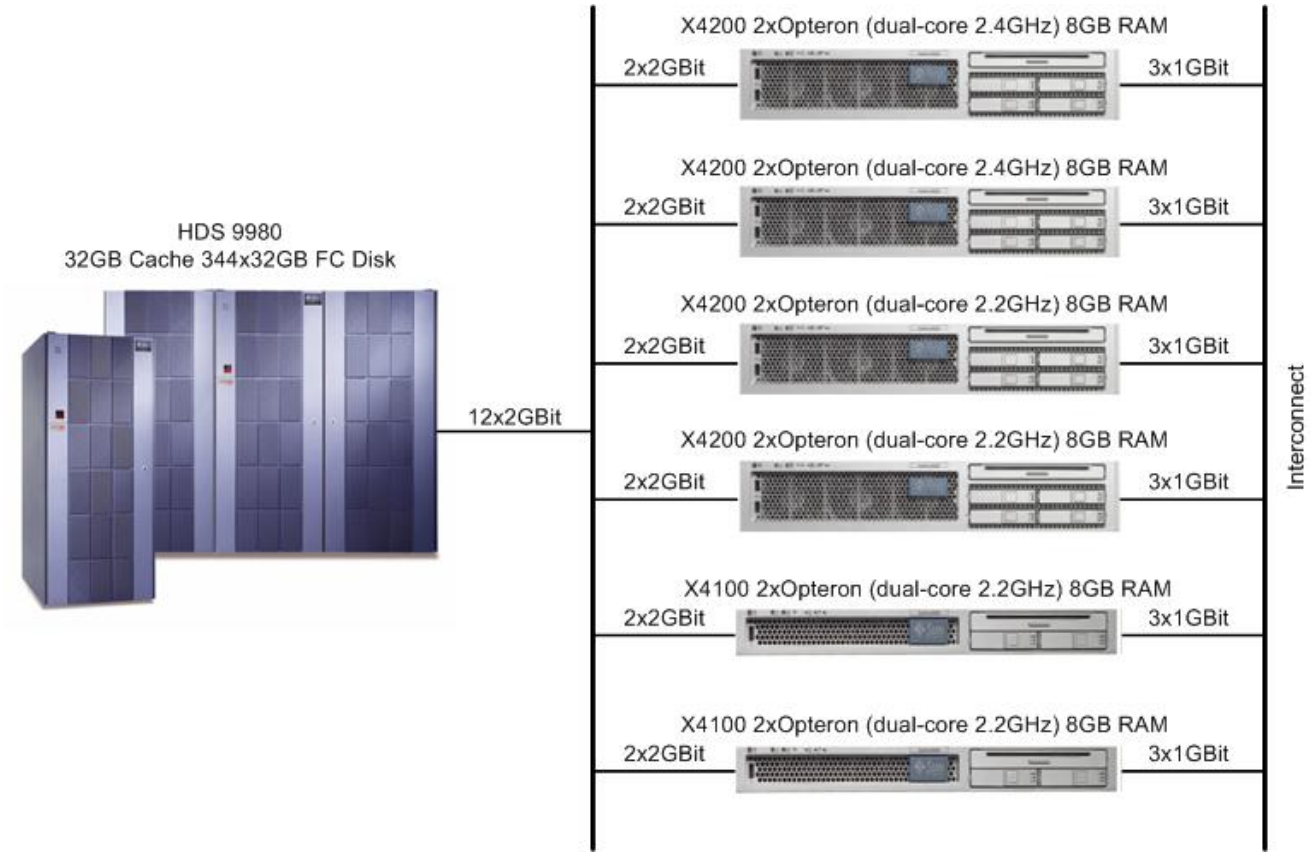

Abbildung 1: Übersicht der Hardware-Architektur.

Installierte Software:

- Sun Solaris 10 x64
- Sun Cluster 3.1
- QFS 4.5
- Oracle Database 10g Enterprise Edition Release 2 (10.2.0.2.0)

Nach der Installation und Konfiguration wurden Prüfmessungen durchgeführt, aus denen folgenden Eckdaten über den maximalen I/O-Durchsatz (aus Oracle Perspektive gemessen: Direct Path Read) der Hardware-Infrastruktur hervorgingen:

- Maximaler Knoten-Durchsatz: 280 MB/s
- Maximaler Durchsatz vom Cluster (d.h. alle Knoten): 1'260 MB/s

In unseren Tests haben wir Tabellen mit Datenvolumen in Bereich von etwa 160 GB pro Tabelle verwendet.

6 Interpretation der Resultate

In diesem Kapitel werden die technischen Erkenntnisse über die Zusammenhänge erläutert. Die Implikationen für das Design der DWH-Applikationen auf Oracle RAC und für die Hardware-Dimensionierung werden im Kapitel „Leasons learned“ behandelt.

6.1 Kommunikation der PX-Slaves via Interconnect (Inter-Node-Kommunikation)

Der Datenfluss zwischen dem Producer-Slave und dem Consumer-Slave findet in „Einheiten“ – so genannten Messages statt. Diese Messages fliessen durch die so genannten Message-Buffers. Beim Befüllen und beim Lesen gibt es je einen Synchronisationspunkt:

- Ein Synchronisationspunkt, dass eine Message vollständig in den Message-Buffer geschrieben wurde. Bedeutet: Eine Message wurde vom Buffer des Producers in den Buffer des Consumers „kopiert“ und der Buffer steht somit zum Auslesen bereit.

- Ein Synchronisationspunkt, dass diese Message vom Consumer vollständig gelesen wurde. Der bestehende Inhalt eines Message Buffers wurde erfolgreich ausgelesen und dieser Buffer steht somit für die weiteren Kopiervorgänge zur Verfügung.

Abbildung 2 verdeutlicht die Aufgabe dieser Synchronisation.

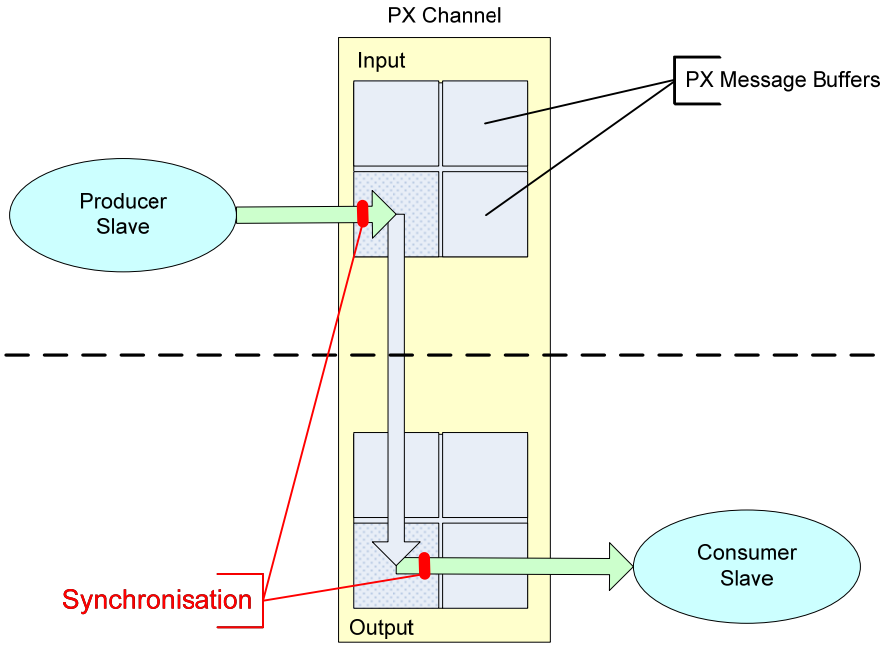

Abbildung 2: Aufgabe der Synchronisation über den PX-Message-Buffer.

Die Datenübertragung vom I/O-Subsystem ins Memory ist bei paralleler Ausführung in Oracle bekanntlich mit sehr wenig Synchronisations-Overhead verbunden. Dies wird mit einer Direct Path Read Operation realisiert, mit der durch einen einzigen Zugriff das Datenvolumen der `db_block_size * db_file_multiblock_read_count` übertragen wird (bei einer Block-Grösse von 16KB und einem Multi-Block-Read-Count von 64 sind es also 1 MB Daten). Zusätzlich kann diese Übertragung auch asynchron abgewickelt werden.

Unsere Tests[13] haben gezeigt, dass die Datenübertragung aus der SGA einer gegebenen Instanz in die SGA einer anderen Instanz mit deutlich mehr Synchronisations-Overhead verbunden ist, als bei einer Übertragung innerhalb derselben SGA.

Hier die wichtigsten Erkenntnisse:

- Die Inter-Node-Kommunikation zwischen zwei PX-Slave-Prozessen – Producer-Slave und Consumer-Slave – bedient sich keiner globalen Synchronisationslogik, für die alle Knoten beteiligt werden müssten. Es werden vielmehr Synchronisationsobjekte verwendet, die nur für diese zwei miteinander kommunizierenden Prozesse relevant sind. Die Synchronisation der PX-Kommunikation ist also eine „lokale" Angelegenheit der zwei Slave-Prozesse – Producer- und Consumer-Slave – und stellt konzeptionell gesehen bei steigender Anzahl der Prozesspaare keinen Engpass dar.

- Der implementierte Synchronisationsmechanismus geht davon aus, dass die Dauer des Kopiervorgangs aus einem Input-Buffer in den Output-Buffer viel kürzer ist als die Zeit, die ein Producer braucht, um diesen Input-Buffer zu füllen – beziehungsweise die Zeit, die ein Consumer braucht, um diesen Output-Buffer zu verarbeiten.

  - Bei der Intra-Node-PX-Kommunikation (alle Slaves laufen auf dem gleichen Knoten) ist dies sicherlich der Fall.

  - Bei der Inter-Node-PX-Kommunikation (Slaves laufen auf mehreren Knoten) muss beim Kopiervorgang die Message von einem Knoten in den SGA eines anderen Knotens übertragen werden. Die Dauer dieses Kopiervorgangs ist um ein Vielfaches höher als die Dauer eines Intra-Node-Kopiervorgangs.

- Der implementierte Synchronisationsmechanismus ist für eine Inter-Node-PX-Kommunikation nur bedingt geeignet. Fallen beispielsweise bei einer I/O-Operation mehr als nur sehr wenige Message-Übertragungen an – konkrete Zahlen werden weiter aufgeführt – kann diese Message-Übertragung zum „Bottleneck" werden:

  - Ein günstigeres Verhältnis kann man erzielen, indem man die Grösse der Message Buffers vergrössert (systemweiter Initialisierungsparameter `parallel_execution_message_size`) Der höchste Wert ist Betriebssystem-abhängig, maximal jedoch 64KB. Auf unserem Testsystem ist uns nicht gelungen, den Wert höher als 16KB zu setzen[14].

  - Wie günstig oder ungünstig dieses Verhältnis ist, hängt auch von der Art der SQL-Befehle ab. Wird nur ein geringer Teil der gelesenen Daten[15] an die Consumer-Slaves weitergeschickt, kann ein günstiges Verhältnis erreicht werden: Es fallen nur wenige Kopiervorgänge für einen I/O an[16]. Werden aber viele der gelesenen Daten verschickt, sind sehr viele Kopiervorgänge (Messages) pro I/O erforderlich. Die Art der SQL-Befehle ist

[13] Der Testcase wird im Anhang A dokumentiert.
[14] Beim Versuch, den Wert auf 64KB zu setzen, kam zwar keine Fehlermeldung, der Wert wurde jedoch auf 16KB zurückgestuft, mit folgendem Eintrag im alert.log „kxfpnfy: Adjusting parallel execution message size to 16384 bytes to conform to IPC OSD message size limit".
[15] „Geringer Teil der gelesenden Daten" bedeutet: Ein kleiner Anteil aller Rows oder ein kleiner Anteil aller Spalten.
[16] In 10gR2 hat Oracle hier ein weiteres nützliches Feature bereitgestellt: den so genannten Bloom Filter.

massgeblich durch die Anforderungen an die Transformationslogik bestimmt und lässt sich nicht durch Umformulierung beeinflussen.

- Vergleicht man also die Laufzeiten ein und derselben DWH-Applikation (Menge der SQL-Befehle) auf einer Non-RAC-Datenbank mit denen einer RAC-Datenbank, so fällt der **Overhead für die PX-Kommunikation** sehr unterschiedlich aus. Bei einer Non-RAC-Datenbank (Intra-Node-Parallelisierung) ist der Overhead vernachlässigbar**. Bei eine RAC-Datenbank (Inter-Node-Parallelisierung)** kann sich dieser jedoch sehr schnell **zu einem zentralen Faktor für die Laufzeitenverlängerung** entwickeln.

- Ein gewisser Overhead bei Inter-Node-Parallelisierung wäre eigentlich vertretbar – sozusagen als „Preis" für die Skalierungsmöglichkeit. Um die Frage der Skalierung in diesem Fall zu beantworten, ist es wichtig, die Overheadstruktur näher zu untersuchen. Der Synchronisations-Overhead entsteht dann, wenn:

  - ein Producer die zu verschickenden Daten in einen Message-Buffer hineinschreiben möchte, jedoch alle Message-Buffers noch „besetzt" (nicht frei) sind **und**

  - der Grund dafür nicht die Langsamkeit des Consumers beim Abholen ist, sondern die Tatsache, dass aus Consumer-Sicht alle Buffer noch „leer" sind (das heisst, aufgrund eines verhältnismässig langen Delays, der für den Kopiergang einer Message anfällt – lange Latenzzeit des Kopiervorgangs pro Message).

  Der erhöhte Synchronisations-Overhead hat die folgende Ressourcen-Struktur:

  - **CPU-Aufwand** für sehr oft wiederholte Abfragen. Abfragen des Producers, ob bereits ein Buffer frei ist, respektive Abfragen des Consumers, ob bereits ein Buffer gefüllt wurde.

  - **Wartezeit (non-busy) zwischen den Wiederholungen.**

  Die Wartezeit fällt pro Message an. Werden mehrere Messages gleichzeitig übertragen, zum Beispiel durch eine DOP-Erhöhung, wird dadurch auch die Elapsed Time verkürzt[17]. Dies gilt aber nicht für die Skalierung im Falle des CPU-Aufwands, denn die CPU-Ressourcen pro Knoten sind limitiert[18].

Fazit: Führt eine lange Latenzzeit beim Kopiervorgang einer PX-Message zur Situation, dass der Producer weitere PX-Messages nicht verschicken kann, dann verursacht die PX-Kommunikation einen überhöhten CPU-Verbrauch (sehr CPU-lastig). **Der Durchsatz der PX-Kommunikation wird in diesem Fall durch die Anzahl/Stärke der CPUs in dem Knoten selbst limitiert**. Diese Limitierung lässt sich durch die Erweiterung um weitere Knoten nicht eliminieren. **Es handelt sich also um eine Limitierung, welche die Skalierbarkeit des RAC-Clusters blockiert**.

[17] Vorausgesetzt, der Interconnect Durchsatz ist ausreichend.
[18] Wir untersuchen nicht die Skalierbarkeit in Richtung Scale-Up (mehr CPUs je Knoten), sondern Scale-Out (mehr Knoten).

Wie tief ist dieser Limitierungswert der Skalierbarkeit bei ausreichender Bandbreite vom Interconnect-Netzwerk? Und wovon ist er abhängig?

- Je schneller das I/O-Subsystem im Vergleich zur CPU-Leistung pro Node ist, desto früher trifft die Limitierung ein.

- Je grösser der Anteil der gelesenen Daten ist, die via PX-Channels weiter verschickt werden müssen, desto früher trifft die Limitierung ein.

Und jetzt zu den konkreten Zahlen.

Eines der Ziele unserer Tests war es, aufzuzeigen, wie hoch die Elapsed Time für den Synchronisations-Overhead bei PX-Kommunikation ist – beziehungsweise wovon sie massgeblich abhängig ist.

Mit einem geeignetem „Workload“[19] haben wir einige PX-Message-Streams mit unterschiedlichem Durchsatz konstruiert, diese als Input für die PX-Kommunikation verwendet und anschliessend den Output-Durchsatz gemessen.

Den Message-Stream haben wir mit einem parallel ausgeführten „Partial Partition-wise Join“ (PPWJ) erzielt. Als Input diente eine Full-Table-Scan-Operation mit Direct-Path-Read. Es wurden Variationen im Input-Durchsatz des Message-Streams erzielt: Einerseits dadurch, dass unterschiedliche Message-Grössen verwendet wurden (siehe Tabelle 1), andererseits dadurch, dass nicht immer alle der gelesenen Spalten an die zweite PX-Slaves-Reihe geschickt wurden (so genannter Projektionsfaktor, siehe Tabelle. 2).

Der Input-Durchsatz wurde auf die Kenngrösse **Anzahl PX-Messages pro Prozess und Sekunde** normiert. Und zwar wie folgt:

$$\frac{tatsächlichÜbermitteltesPXVolumen}{AvgTotalElapsedTimePerSlaveBeiFPWJ * DOP * PARALLEL\_EXECUTION\_MESSAGE\_SIZE}$$

Daraus liesse sich der Wert für die durchschnittliche Bereitstellungsdauer einer Message auf den PX-Buffer ermitteln. Die Elapsed Time bei Full Partition-wise Join (FPWJ) entspricht einer Zeit, die sich im Faller eines PPWJ erreichen liesse, falls bei der PX-Kommunikation kein Engpass entstehen würde.

Der Output-Durchsatz wurde auf analoge Weise normiert. Dabei wurde von der Elapsed Time des Consumers ausgegangen:

$$\frac{tatsächlichÜbermitteltesPXVolumen}{AvgTotalElapsedTimePerConsumerBeiPPWJ * DOP * PARALLEL\_EXECUTION\_MESSAGE\_SIZE}$$

[19] Workload = Menge von SQL-Statements

Auf der oben aufgeführten Hardware-Infrastruktur haben wir bei Tests auf vier Knoten die folgenden Werte gemessen:

			Durchsatz vom Input Stream für PX-Kommunikation			Durchsatz vom Output Stream für PX-Kommunikation				
Message Size [kB]	PX Volumen in # Msgs	Anzahl Msg pro Prozess	Duration Direct Path Read in FPWJ [s]	Anzahl Msg pro Prozess und Sekunde	Duration for Producing 1 Message [ms]	Duration PX TableQ in PPWJ [s]	Anzahl Msg pro Prozess und Sekunde	Duration for Consuming 1 Message [ms]	Overhead für PX-Transfer von 1 Message [ms]	Overhead im Vergleich zu Input
2	2'516'582	157'286	90	1'748	0.572	181	869	1.151	0.579	101%
4	1'258'291	78'643	89	884	1.132	133	591	1.691	0.559	49%
8	629'146	39'322	90	437	2.289	117	336	2.975	0.687	30%
16	314'573	19'661	87	226	4.425	106	185	5.391	0.966	22%

Tabelle1: Gemessener Durchsatz der PX-Kommunikation in Abhängigkeit der Grösse des Message-Buffers, DOP pro Knoten = 4, Projektionsfaktor = 6% der gelesenen Daten.

			Durchsatz vom Input Stream für PX-Kommunikation			Durchsatz vom Output Stream für PX-Kommunikation				
Anteil verschickter Daten (Projektions-faktor)	PX Volumen in # Msgs	Anzahl Msg pro Prozess	Duration Direct Path Read in FPWJ [s]	Anzahl Msg pro Prozess und Sekunde	Duration for Producing 1 Message [ms]	Duration PX TableQ in PPWJ [s]	Anzahl Msg pro Prozess und Sekunde	Duration for Consuming 1 Message [ms]	Overhead für PX-Transfer von 1 Message [ms]	Overhead im Vergleich zu Input
6%	314'573	19'661	87	226	4.425	106	185	5.391	0.966	22%
21%	1'101'005	68'813	85	810	1.235	139	495	2.020	0.785	64%
42%	2'202'010	137'626	90	1'529	0.654	193	713	1.402	0.748	114%
62%	3'250'586	203'162	93	2'185	0.458	261	778	1.285	0.827	181%
81%	4'246'733	265'421	90	2'949	0.339	323	822	1.217	0.878	259%
100%	5'242'880	327'680	90	3'641	0.275	406	807	1.239	0.964	351%

Tabelle 2: Gemessener Durchsatz der PX-Kommunikation in Abhängigkeit des Projektionsfaktors (Anteil der gelesenen Daten), DOP pro Knoten = 4, Grösse des Message-Buffers = 16KB.

Natürlich haben wir auch die weiteren Kombinationen von Buffer-Grösse und Projektionsfaktor gemessen, dazu noch auch in Kombination mit unterschiedlichen DOP-Werten. Die Werte haben wir wie folgt interpretiert:

- Die Übertragung einer PX-Message ist mit einer nicht vernachlässigbaren Duration verbunden. Diese Duration ist auf die Ausführung der folgenden Aufgaben zurückzuführen:
  - Den eigentlichen Datentransfer.
  - Die „Synchronisierung" zu Beginn des Schreibens in den PX-Message-Buffer und die „Synchronisierung" zu Beginn des Lesens aus dem PX-Message-Buffer.

  Diese Duration fällt für jede PX-Message an. In jedem Prozesspaar (Producer-Consumer) werden die Messages seriell verarbeitet, keine Mengenabfertigung, kein Pipelining. Der Zuwachs an Antwortzeit ensteht daher aus der Multiplikation dieser Durations mit der Anzahl der übertragenen Messages. In unserer Konfiguration lag die Duration pro PX-Message zwischen 0.5 ms und 1 ms.

- Der absolute Wert der Duration hängt nur unwesentlich davon ab, wie gross die Message-Grösse ist[20]. Eine Verdoppelung der Message-Grösse führte nicht zu einer Verdoppelung der Duration (siehe Tabelle 1, Spalte „Overhead für PX-Transfer von 1 Message).

[20] Bei ausreichender Netzwerk-Kapazität des Interconnect-Netzwerks, die wir auf Hardware-Ebene während der Tests überprüft haben.

- Der relative Wert, nämlich das Verhältnis dieser Duration zu der Zeit, in der ein Producer eine Message mit Daten vorbereiten kann, hängt jedoch stark von der Message-Grösse ab. **Der Anteil der Duration für die PX-Synchronisation pro Message sinkt also mit steigender Message-Grösse. Schnelle Producers – also solche, die 1000 oder mehr Messages pro Sekunde vorbereiten können – werden durch die Duration der PX-Synchronisation wesentlich gebremst** (siehe Abbildung 3)**.**

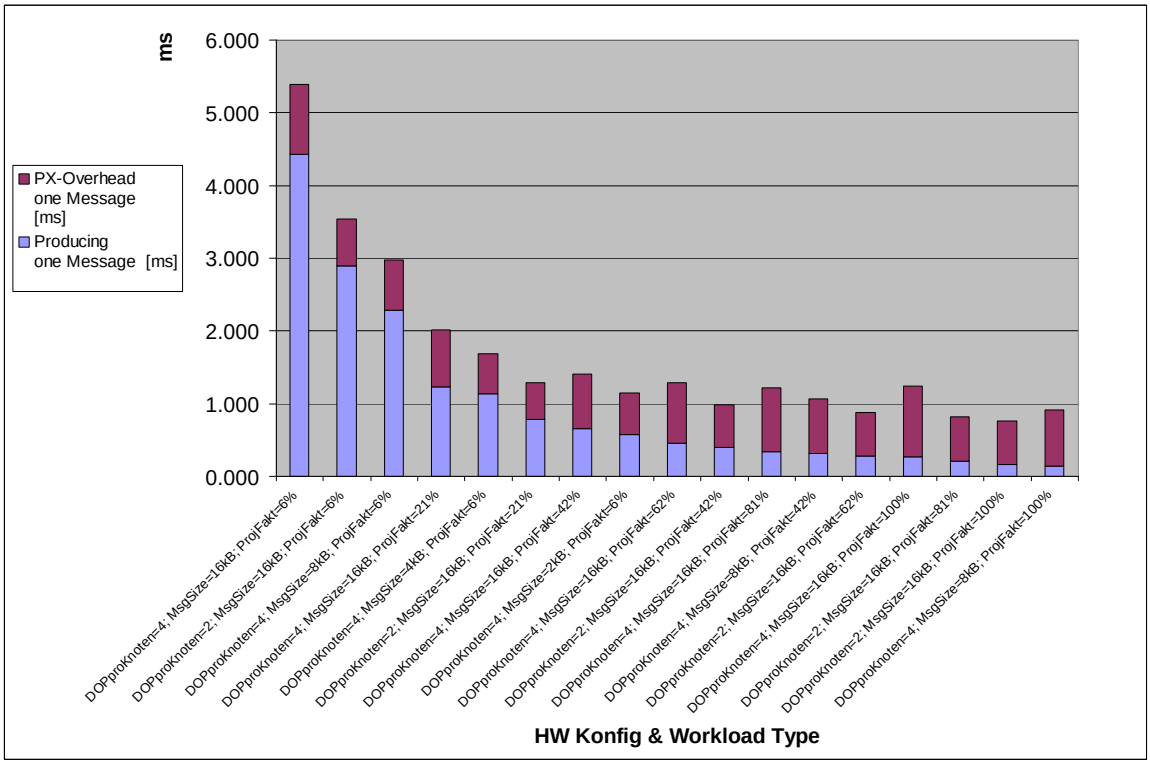

Abbildung 3: Der Anteil der Duration für PX-Synchronisation pro Message steigt mit steigender „Produktivität" des Producers (kürzere Zeit für „Producing one Message").

Solche „schnellen" Producers können bei heutiger Hardware-Infrastruktur und heutigem DWH-Workload sehr leicht entstehen – beispielsweise dadurch, dass

- die Message-Grösse zu klein ist im Verhältnis zum hohen Durchsatz von Direct-Path-I/Os, **oder**
- die SQL-Befehle oft verlangen, mehr als ca. 10 Prozent der gelesenen Daten weiterzuschicken – dies ist vor allem im ETL-Bereich üblich.

Der Query Optimizer geht davon aus, dass die Kosten (Duration) für die PX-Kommunikation vernachlässigbar klein sind: Mit anderen Worten, **der Query Optimizer geht davon aus, dass alle SQL-Befehle nur mit Intra-Node-Parallelisierung ausgeführt werden**. Wenn jedoch im Falle von RAC die SQL-Befehle auch mit Inter-Node-Parallelisierung ausgeführt werden – um die Ressourcen auf anderen Knoten zu verwenden, kann diese Annahme zu suboptimalen Ausführungsplänen führen (zum Beispiel falsche Reihenfolge der Joins und/oder GROUP BYs).

6.2 Load Balancing

Bei einer Inter-Node-PX-Kommunikation werden die Hardware-Ressourcen naturgemäss immer höher beansprucht als bei der Intra-Node-PX-Kommunikation.

Ist die Knotenanzahl ungefähr gleich oder kleiner als die Anzahl der CPUs (Cores) pro Knoten, dann kann neben den Fällen „keine Parallelisierung“ und „volle Parallelisierung auf alle CPUs aller Knoten“ noch ein weiterer Ansatz in Betracht gezogen werden: die „Parallelisierung auf alle CPUs eines Knotens“. Der Beweggrund hierbei ist, die Ressourcen und Laufzeiten für eine Inter-Node-PX-Kommunikation zu sparen.

Dieser Ansatz setzt voraus, dass

- Die Antwortzeiten der Ausführung auf einem Knoten bereits ausreichend sind.
- Die restlichen, nicht verwendeten Hardware Ressourcen auf anderen Knoten für die garantierte Performance anderer, eventuell gleichzeitig gestarteter SQL-Befehle fest reserviert werden sollen.

Auch wenn gerade die zweite Voraussetzung in der Praxis nur recht schwierig sowie durch Kompromisse erfüllbar sein wird[21], dürfte gerade dieser Ansatz – „Parallelisierung auf alle CPUs eines Knotens“ – eine breite Anwendung finden, nicht zuletzt wegen der hohen Kosten der Inter-Node-PX-Kommunikation.

Da bei diesem Ansatz während der Ausführung des SQL-Befehls nicht alle verfügbaren Ressourcen gleichermassen beansprucht werden, erschien es uns sinnvoll, das in der Oracle Datenbank eingebaute Load-Balancing näher unter die Lupe zu nehmen.

Im Gegensatz zum Load Balancing bei OLTP-Applikationen, wo die Anwender-Session die wichtigste Allozierungs-Einheit für die Zuordnung zu einem bestimmten Knoten darstellt, ist bei DWH-Applikationen die Allozierung der Slave-Prozesse aus den PX-Server-Pools der einzelnen Knoten entscheidend, das heisst auf welchen Knoten sollen die Slave-Prozesse eines SQL-Befehls laufen, damit:

- sie dort auf den einzelnen Knoten ungefähr gleich wahrscheinlich oft zu Hardware-Ressourcen kommen, das heisst mit der Bildung der Zwischenresultate ungefähr gleich schnell fertig sind,
- alle Knoten ungefähr gleich stark ausgelastet sind,
- alle Slaves (sofern möglich) auf dem gleichen Knoten laufen, und dieser Knoten ist (wiederum sofern möglich) auch derselbe, auf dem auch der Koordinator läuft.

[21] Grund für die schwierige Erfüllung: Nur selten ist die Anzahl der Concurrent User Sessions, für die eine predictable Performance (garantierte Antwortzeit) erzielt werden soll, mit der Anzahl der Hardware-Knoten identisch.

Anhand unserer Tests[22] konnten wir die folgenden Erkenntnisse sammeln:

- Bei SQL-Befehlen mit einem DOP bis (2*Anzahl CPUs)-1 wird die Ausführung auf einen Knoten gelegt
- Beim höherem DOP bis (3*Anzahl CPUs)-1 wird die Ausführung auf zwei Knoten gelegt
- Beim noch höherem DOP wird die Ausführung auf alle Knoten gelegt
- Beim Load-Balancing werden die folgenden Angaben berücksichtigt:
  - Anzahl der Running Instances.
  - Anzahl der CPUs (Cores).
  - Auslastung der CPUs pro Knoten aus der nahen Vergangenheit.

Interessant war es, zu erkennen, dass der gegenseitige Informationsaustausch zwischen den Instanzen über ihre eigene Auslastung vermutlich in „längeren" Abschnitten passiert, und zwar im Bereich von ein paar Sekunden.

7 „Leasons learned" für den praktischen Einsatz von RAC

In diesem Kapitel werden die technischen Erkenntnisse aus dem vorherigen Kapitel zu Konsequenzen für die praktische Umsetzung zusammengefasst.

7.1 Design der DWH-Applikationen

Zwecks einfacher Erklärung wurden in diesem Absatz die DWH-Applikationen in folgende drei Bereiche aufgeteilt: OLAP-Bereich, Abfragen auf Basis sehr feiner Einzelaussagen und ETL-Bereich.

7.1.1 OLAP-Bereich

Erfahrungsgemäss kann man die Erwartungshaltung im Bereich Online Analytical Processing sowie Online Reporting so beschreiben, dass die Antwortzeiten in der Regel im Bereich von ein paar wenigen Sekunden bis etwa 10 Sekunden liegen sollen. Meist werden hier die Stored Aggregates (z.B. Materialized Views) eingesetzt, um diese kurzen Antwortzeiten zu erzielen. Stored Aggregates führen dazu, dass die Abfragen weniger Hardware-Ressourcen benötigen. Beim Nachführen (Refresh) der Stored Aggregates ist eine Skalierbarkeit sehr wohl gefragt, dies wird im Abschnitt „ETL-Bereich" behandelt.

Bei einer stark steigender Benutzeranzahl kann es vorkommen, dass die garantierten Antwortzeiten bei der Ausführung einzelner Abfragen erzielt werden können, nicht aber im Falle der Ausführung vieler Abfragen gleichzeitig, das heisst bei hoher Simultanität. Bei steigender Anzahl gleichzeitig ausgeführter Abfragen können hier neue RAC-Knoten zugefügt werden, um die garantierten Antwortzeiten auch bei erhöhter Simultanität zu erreichen.

[22] parallel_threads_per_cpu = 2; parallel_automatic_tuning = TRUE; parallel_adaptive_multiuser = FALSE; DEFAULT Resource Plan, default Werte für INSTANCE_GROUPS und PARALLEL_INSTANCE_GROUP

7.1.2 Abfragen auf Basis sehr feiner Einzelaussagen

Erfahrungsgemäss ist die Performance – und bei steigendem Datenvolumen somit auch die Skalierung – besonders bei komplexen und umfangreichen Abfragen gefragt – zum Beispiel bei Abfragen auf Basis einzelner Aussagen über einen bestimmten Kunden, über sein einzigartiges Produktportfolio oder Nutzungsverhalten. Es heisst also immer dort, wo die Daten auf einer sehr feinen Granularität analysiert werden müssen, und dort, wo die Art dieser Analysen schwierig vorhersehbar ist oder eine hohe Variabilität ausweist, also bspw. in analytischem CRM System, in Applikationen zum Entwurf der Kampagnen für Direkt Marketing, in Applikationen für Erkennung der Kundenkredit-Risiken oder für Erkennung der möglichen Geldwäschereifälle.

Unserer Meinung nach ist es mit RAC – bei Einhaltung gewisser Voraussetzungen – möglich, eine in Richtung Scale-Out (mehrere Knoten) skalierbare Abfrageplattform aufzubauen. Neben den Technologie-Issues, wie zum Beispiel geeignete Datenbank-Features[23] oder die Qualität der Query-Optimizer-Entscheide, ist **für die Skalierung einer solchen Abfrageapplikation das logische und physische Datenmodell entscheidend**. Das physische Datenmodell kann wesentlich dazu beitragen, das Kommunikationsvolumen zwischen den PX-Slaves zu verkleinern. Abfragen mit vergleichsmässig wenig PX-Kommunikationsvolumen:

- haben generell kürzere Ausführungszeiten, denn die Übertragungszeit der Zwischenresultate zu anderen Slaves entfällt,

- skalieren mit der aktuellen Version von RAC viel besser, denn es kommt nicht zum oben beschriebenen Sychronisations-Overhead bei PX-Message-Übertragung.

Das physische Datenmodell ist unserer Meinung nach sehr wichtig. Darum haben wir die geeignete Gestaltung zur Voraussetzung für die Skalierung der Abfrageplattform eingestuft. Das Wesentliche:

- Im Datenmodell der Abfrageapplikation wird zwischen den Tabellen der Fakten, der Subjekte und der Klassifizierungen unterschieden. Die Subjekte sind Akteure aus den operativen Prozessen. Die Anzahl der Einträge in den Subjekttabellen – Kunde, Konto, Vertrag, Registration, etc. – ist sehr viel höher als die Anzahl der Einträge in den Klassifizierungstabellen – Kundensegment, Kontotyp, Nutzungsart, usw., und zwar mindestens um den Faktor 100, meistens sogar noch viel mehr.

- **Die Subjekttabellen und Fakttabellen müssen hash-partitioniert[24] werden. Die Anzahl der Hash-Partitionen sollte gross genug gewählt werden.** Sie sollte mindestens durch die Anzahl der CPUs (Cores) pro Knoten und durch Anzahl der Knoten dividierbar sein. Sie muss auch durch die künftige CPU- und Knoten-Anzahl der einzelnen Ausbaustufen dividierbar sein, weil sonst Aufwand und Platzbedarf für die Tabellenreorganisationen anfallen.

- **Die Fremdschlüssel der Beziehungen zwischen den Einträgen in Fakttabellen und Subjekttabellen – sowie den Subjekttabellen untereinander – müssen den Partition-Key der Hash-Partitionierung enthalten.** Hiermit wird sichergestellt, dass die Joins dieser Tabellen als **Full-Partition-Wise Joins** ausgeführt werden können. Hintergrund der Überlegung: Full-Partition-Wise

[23] Wie zum Beispiel Star Transformation (Empty-Cells-Optimierung), Query Rewrite (Aggregate Awareness) und Partition Prunning.
[24] Bei Fakttabellen ist darüber hinaus auch eine Range-Partitionierung nach Zeit üblich. Diese sind daher composite-partitioniert (range, hash).

Joins verursachen nur minimale PX-Kommunikationsvolumina. Partition-Wise Joins sind generell sehr gut geeignet, die Hash-Table eines Hash-Joins zu verkleinern[25].

7.1.3 ETL-Bereich

Erfahrungsgemäss gibt es im ETL-Zeitfenster einige wenige ETL-Jobs, die einen Grossteil dieses Fensters füllen. Simultane (gleichzeitige) Ausführung dieser Jobs ist meistens nur bedingt möglich, denn zwischen den Jobs bestehen Datenabhängigkeiten, die eine serialisierte Ausführung voraussetzen. Aus Sicht der Durchlaufzeit des ETL-Schedules sind diese Jobs also auf dem „kritischen Pfad".

Bei solchen Jobs sollten die Performance-Aspekte (Durchsatz) bereits beim Design besonders berücksichtigt werden. Die aus unserer Sicht wichtigsten Kriterien haben wir am Kapitelanfang von „Design der Testfälle" aufgeführt. Neben diesen Kriterien sollte hier besonders die Skalierbarkeit der ETL-Jobs in Betracht gezogen werden. Und, falls angefordert, auch die Skalierbarkeit in Richtung Scale-Out (weitere RAC-Knoten).

Wie wir bereits ausgeführt haben, geht es beim **Scale-Out eines einzelnen ETL-Jobs** im aktuellen Release von Oracle RAC darum, in Fällen von umfangreichen SQL-Statements nur solche Operationen zu verwenden, die wenig PX-Kommunikationsvolumen erzeugen.

Anders als bei einer Abfrageapplikation, bei der das Datenmodell und somit auch die SQL-Befehle der Abfragen in der Hand des Designers liegen, sind bei der ETL-Applikation die SQL-Befehle – Joins, GROUP BYs, usw. – zusätzlich durch viele anderen Faktoren bestimmt, zum Beispiel das Datenmodell der Quellsysteme, die Anforderungen an die Business Transformationslogik sowie die Cleansing-Logik, usw.

Im ETL-Bereich ist es unserer Ansicht nach kaum möglich, das physische Datenmodell so zu gestalten, dass beim Grossteil der SQL-Befehle ein Full-Partition-Wise Join[26] verwendet werden kann. Kann dies bei dieser Art von SQL-Befehlen nicht erfüllt werden, ist bei einer Ausführung auf mehreren Knoten mit einem starken Skalierungseinbruch zu rechnen, denn:

- im ETL-Bereich werden die Daten sehr oft auf feinster Granularitätsstufe verarbeitet,
- im ETL-Bereich werden öfters viele oder sogar alle Spalten verarbeitet,
- in solchen Fällen entsteht ein überproportional hohes Synchronisations-Overhead bei der PX-Kommunikation, und dies bereits bei einer Ausführung auf zwei Knoten.

Leider ist es so, dass gerade ein ETL-Job, der lange dauert und dessen Ausführung man auf mehrere Knoten legen möchte, meistens mit vielen Daten auf tiefster Granularität arbeitet, tendenziell eher mehr PX-Kommunikation verlangt, und daher auch tendenziell anfälliger ist für die Entstehung eines Synchronisations-Overheads.

[25] Die Anzahl Einträge in der Hash-Table eines Hash-Joins pro Slave-Prozess ist beim Non-Partition-Wise Join: „Anzahl Keys in der kleineren Tabelle" / DOP. Beim Partition-Wise-Join kann die Anzahl der Einträge durch das Erhöhen der Partition-Anzahl verkleinert werden. Diese ist bestimmt durch: „Anzahl Keys in der kleineren Tabelle" / Anzahl der Partitionen.
[26] Oder aber GROUP BY mit Partition-Key-Attribut.

Es gibt sicherlich Fälle, wo eine Skalierbarkeit durch Design denkbar und möglich ist. Aber es handelt sich hier aus unserer Sicht um einzelne Sonderfälle.

Die ETL-Applikation kann von mehreren RAC-Knoten auch auf solche Weise profitieren, dass **jeder der gleichzeitig (simultan) ausführbaren ETL-Jobs auf einem anderen RAC-Knoten ausgeführt wird**[27]. Über eine Skalierung der ETL-Applikation lässt sich hier nur dann sprechen, wenn bei der Job-Ausführung das Load-Balancing automatisch von einem ETL-Scheduler übernommen wird. Eine wichtige Voraussetzung bildet auch hier das eigentliche Design der ETL-Transformationen – ein Design for Performance, das wir am Anfang vom Kapitel „Design der Testfälle" beschrieben haben.

Erfahrungsgemäss sehen wir bei diesem Ansatz aber konkrete Grenzen, basierend auf der Natur der Aufgabe des ETLs:

- für das ETL ist ein Datenfluss charakteristisch – Änderungen aus den OLTP-Systemen „fliesen" in die integrierten Abfragequellen (Datamarts) ein,
- in vielen Fällen ist die Datenintegration eine der wichtigen Aufgaben von ETL. Die Datenintegration setzt zusätzliche Synchronisationspunkte in ETL-Schedules voraus.

Konsequenz: Die Anzahl gleichzeitig ausführbarer ETL-Jobs variiert während des Ladefensters sehr stark. Werden also für den Peak-Zeitraum – bei mehreren umfangreichen, gleichzeitig ausgeführten ETL-Jobs – mehrere Hardware-Knoten beschafft, dann können diese ausserhalb dieses Zeitraums nicht genügend ausgelastet werden.

7.2 Dimensionierung der Hardware-Infrastruktur: Verhältnis Knoten-Stärke zur Knoten-Anzahl

7.2.1 Generell

Alle Knoten sollen gleich stark dimensioniert werden – sowohl im Bezug auf Stärke und Anzahl der CPUs, den I/O- und Netzwerk-Interfaces-Durchsatz, als auch im Bezug auf das Interconnect und die Memory-Grösse. Unsere Tests haben gezeigt, dass dies eine sehr grosse Rolle spielt. Ist dies nicht der Fall, bestimmt in vielen Fällen der schwächste Knoten den Durchsatz des gesamten Systems.

Die Knoten-übergreifenden Ressourcen wie Storage Subsystem, Switches vom Knoten zum Storage Subsystem, Switches zwischen den Knoten (Interconnect) sollten ausreichend breit dimensioniert werden, damit sie die gleichzeitig erzeugte Knotenlast bedienen können. Dies klingt zwar trivial, wird aber in der Praxis sehr oft übersehen.

7.2.2 Bedarf für Abfrageapplikationen

Aufgrund der vorher beschriebenen Spezifika der PX-Kommunikation ist generell zu empfehlen, dass die Knotenstärke so gewählt wird, dass die angeforderten Antwortzeiten pro End-Anwender-Abfrage mit diesem einem Knoten erreicht werden können – solange diese Abfrage als einzige Last auf dem

[27] Wie auch in den anderen Fällen setzt auch dies eine geeignete Dimensionierung des I/O Subsystems voraus.

Knoten verarbeitet wird. Knoten mit weniger als vier Core-CPUs würden wir für den Praxiseinsatz in der Regel nicht empfehlen.

In den folgenden Fällen ist es empfehlenswert, in die Ausbaufähigkeit der einzelnen Knoten wesentlich zu investieren (Scale-Up Freiraum schaffen):

- es ist zu erwarten, dass die Abfragen immer umfangreicher werden (immer mehr Daten müssen gelesen und zusammenstellt werden),

  und

- die Entwicklungsaufwände für Aufbau weiterer Stored Aggregates (Materialized Views) nicht steigen dürfen oder das ETL-Ladefenster für periodisches Nachführen der Materialized Views bereits voll ist.

In Fällen, in denen absehbar ist, dass die Abfragen aufgrund des Datenwachstums im DWH immer umfangreicher werden – wenn zum Beispiel das DWH immer mehr feingranulierte Aussagen über die Kunden enthält und/oder der Kundenstamm exponentiell wächst, ist ein Ausbau in Richtung „neue Knoten hinzufügen“ (Scale-Out) nur dann eine erfolgreiche Investition, wenn bestimmte Voraussetzungen bezüglich Design der DWH-Applikation vollständig erfüllt sind[28]. Diese Voraussetzungen wurden in den Abschnitten 4 und 7.1.2 grob skizziert.

Sind diese Voraussetzungen erfüllt, dann kann man die Antwortzeiten der End-Anwender-Abfragen mit dem „Zufügen weiterer Knoten“ verkürzen. Wir haben in der aktuellen RAC Implementation keine logische Ressource gefunden, die bei solcher Erhöhung des Verarbeitungsdurchsatzes zum Bottleneck werden kann.

In Fällen, in denen absehbar ist, dass die Anzahl der simultanen Abfragen zunimmt, können die gewohnten Antwortzeiten der End-Anwender-Abfragen beibehalten werden, indem so viele neue Knoten zufügt werden, wie die Simultanität (auch Concurrency genannt) gestiegen ist. Bezüglich dieser Zielsetzung skaliert die RAC Implementation gut.

Der Grossteil des Komplexität-Zuwachses entsteht beim Übergang von einem Knoten (Non-RAC) auf mehrere Knoten. Aus Sicht der Abfragen-Entwicklung wird man vom Komplexitätszuwachs in Oracle leider nicht komplett abgeschirmt, denn der Query Optimizer hat in Bezug auf PX-Kommunikation ein noch unzureichend feines Kostenmodell. Daher empfiehlt sich, eigene Erfahrung mit dieser Problematik in Form eines Prototyps mit mehreren Knoten im Vorfeld zu gewinnen.

7.2.3 Bedarf der ETL Applikation

Wie bereits erwähnt, ist es unserer Ansicht nach generell nicht möglich, die SQL-Befehle der ETL Jobs so zu gestalten, dass ihre Ausführungszeit bei einer Ausführung auf mehreren Knoten nahe-zu-linear verkürzt wird.

[28] Es ist unmöglich, eine Scale-out-Ability-Promise über eine DWH-Applikation (sowohl eine bestehende als auch eine neu gebaute) auf Basis von Oracle RAC abzugeben, solange man das Applikations-Design nicht kritisch unter die Lupe nimmt, das heisst bei den Skalierungsvoraussetzungen nachhakt.

Die Knoten-Mindeststärke soll so gewählt werden, dass die umfangreichen Jobs aus dem kritischen Pfad[29] – falls sie auf diesem Knoten alleine ausgeführt werden – im definierten Zeitfenster beendet werden können. Die Knotenanzahl ist davon abhängig, wie viele dieser umfangreichen Jobs gleichzeitig ausgeführt werden können und welchen Anteil die Zeit der gleichzeitigen Ausführung am gesamten Ladefenster ausmacht.

Diese Überlegungen setzten voraus, dass beim Design der ETL-Applikation die im Abschnitt 7.1.3 beschriebenen Aspekte berücksichtigt wurden.

7.3 Konfiguration der Datenbank sowie der Instanzen

In diesem Abschnitt werden nur solche Empfehlungen abgegeben, welche RAC-spezifischen Hintergrund haben und über die „klassischen" Empfehlungen für eine DWH Applikationen im Non-RAC-Umfeld hinausgehen.

7.3.1 Degree of Parallelism (DOP)

Entsteht der Bedarf, den DOP für einen SQL-Befehl höher zu setzen als die Anzahl der CPUs (Core) des einzelnen Knotens, dann sollte man den DOP gleich auf ein n-faches dieser Anzahl setzen.

7.3.2 Grösse des PX-Message-Buffers (PARALLEL_EXECUTION_MESSAGE_SIZE)

Wir sind überzeugt, dass die optimale Grösse des PX-Message-Buffers vom konkreten SQL-Befehl abhängig ist. Genauer gesagt hängt sie ab vom Datenvolumen, welches als Input für einzelne Row-Source-Operation aus dem Execution Plan des SQL Befehls dient. Je grösser das Volumen, desto grösser sollte der Buffer sein.

Bis es bei Oracle so weit ist, dass eine variable Buffer-Grösse unterstützt wird und diese Grösse durch den Query Optimizer oder die SQL-Engine von Fall zu Fall bestimmt wird, **empfehlen wir, die maximal unterstützte Grösse zu verwenden**. Je nach Betriebssystem sind dies zwischen 16KB und 64KB.

Überlegungen aus diversen Informationsquellen, dass ein allzu hoher Wert zu Performance-Einbussen führen kann, können wir nicht nachvollziehen. Nach diesen Überlegungen soll eine allzu grosse Message-Size eine verschachtelte (interleaved) Arbeit der zwei Slave-Reihen blockieren – vor allem dadurch, dass der Consumer-Slave längere Zeit nichts zu verarbeiten hat. Dies trifft unserer Meinung nach auch zu, aber nur während der Übertragung der ersten und der letzten Message (kein Pipelining). Werden also bei der PX-Kommunikation mit einer Message-Size von 8KB etwa 200 Messages pro Consumer-Prozess benötigt – (1+1)*8KB davon ohne Pipelining im Processing, entspricht 1 Prozent der Datenverarbeitung, dann werden bei einer PX-Kommunikation mit doppelt so grossem Message-Buffer (16KB) etwa 100 Messages benötigt – dabei 2*16KB davon ohne Pipelining, entspricht 2 Prozent der Datenverarbeitung. Mit dem Wechsel auf 16KB ginge der Pipelining-Effekt für 1 Prozent der Datenverarbeitung verloren, allerdings würden 50 Prozent des

[29] Es können mehrere kritische Pfade existieren: Dies soll für die Gesamtheit dieser kritischen Pfade erreicht werden – und zwar nicht zusammen, sondern einzeln.

Synchronisation-Overheads für die PX-Kommunikation gespart! Dies stellt im Falle einer Inter-Node-Parallel-Execution einen bedeutenden Teil der gesamten Ausführungszeit dar!

7.3.3 Load-Balancing

Wie bereits ausgeführt, geht es im DWH primär um Load-Balancing auf der Ebene Allozierung der PX-Slaves. Eine gewisse PX-Kommunikation besteht aber auch zwischen dem PX-Koordinator (Anwender-Session) und den PX-Slaves. Um den Bedarf für eine Inter-Node-PX-Kommunikation auch hier zu reduzieren, empfiehlt sich, dass ein Load-Balancing auch bei der Allozierung der Anwender-Sessions durchgeführt wird. Dies soll mit den „klassischen" SQL-Net-Mitteln erzielt werden.

8 Weiterentwicklung von RAC: Empfehlungen und Erwartungshaltung

Trivadis ist überzeugt, dass Oracle in den nächsten Releases die bestehende, solide Basis ausbauen und auch die oben beschriebenen Lücken schliessen wird. Es geht dabei um folgende Punkte:

- **Die Limitierungen für die Grösse des PX-Message-Buffers heraufsetzen:** Diese sollten im Bereich der I/O-Grösse liegen[30].

- **Die Grösse des PX-Message-Buffers variabel machen**: Tatsächlich variiert das PX-Traffic-Volumen recht stark – nicht nur pro SQL-Befehl, sondern auch pro Operation innerhalb eines SQL-Befehls. Eine dynamisch angepasste Grösse des Message-Buffers soll dieser Realität Rechnung tragen. Die optimale Grösse sollte vor der Ausführung anhand des geschätzten PX-Volumens festgelegt werden.

- In Fällen, in welchen es zwangsläufig zu einer Inter-Node-PX-Kommunikation kommen wird[31], **sollte der Query Optimizer die Kosten für die PX-Kommunikation berücksichtigen**, und zwar nicht nur bei der Wahl der Distributionsmethode, sondern bereits bei der Festlegung der Row-Source-Operationen, die den Execution Plan bilden. Neben dem geschätzten PX-Volumen sollte er auch über System-Statistiken der PX-Message-Übertragung verfügen.

Karol Hajdu, Senior Consultant, Trivadis GmbH, Wien, karol.hajdu@trivadis.com
Christian Antognini, Senior Consultant, Trivadis AG, Zürich, christian.antognini@trivadis.com

[30] Die damit verbundenen zusätzlichen Memory-Ansprüche sind unserer Meinung vertretbar, vor allem auf dem Hintergrund, dass man bei dem Buffer-Cache recht viel einsparen kann.
[31] Angeforderter DOP ist grösser als die Anzahl von CPUs (Core).

Anhang A: Abkürzungsverzeichnis

API	Application Programming Interface
CRM	Customer Relationship Management
DML	Data Modification Language (Insert, Update, Delete und Merge Statement)
DWH	Data Warehouse
DOP	Degree of Parallelism
ETL	Extract, Transform, Load
OLAP	Online Analytic Processing
OLTP	Online Transaction Processing
PX	Parallel Execution
RAC	Real Application Clusters
SGA	System Global Area

Anhang B: Testcase zur Messung des Synchronisation-Overheads bei Inter-Node Parallel Execution

Dieser Anhang liefert die Eckdaten des Testcases, der zur Messung des Synchronisation-Overheads bei Inter-Node Parallel Execution verwendet wurde.

DB Schema

Tabelle	Spalten	Weitere Beschreibung
MASTER	id NUMBER, pad VARCHAR(25)	Tabelle ist hash-partitioniert nach ID. Tabelle enthält ca. 20'000'000 Rows und belegt ca. 600 MB Platz in den Data Files. Tabelle besteht aus <Anzahl Partitionen>[32] Partitionen. Die Werte für Spalte ID wurden so gewählt, dass jede Partition ungefähr gleich viele Rows enthält (Abweichung weniger als 1%). Tabelle hat den DOP in der Höhe von <Anzahl Partitionen>.
DETAIL	id NUMBER, pad01 VARCHAR(25), …, pad20 VARCHAR(25)	Tabelle ist hash-partitioniert nach ID. Tabelle enthält ca. 180'000'000 Rows und belegt ca. 80 GB Platz in den Data Files. Tabelle besteht aus <Anzahl Partitionen>[32] Partitionen. Die Werte für Spalte ID wurden so gewählt, dass jede Partition ungefähr gleich viele Rows enthält (Abweichung weniger als 1%). Tabelle hat den DOP in der Höhe von <Anzahl Partitionen>.

Alle Segmente wurden in locally managed Tablespaces als Segmente mit ASSM erstellt. Jeder Tablespace enthält Files aus allen LUNs.

[32] Während der Tests wurden mehrere „Versionen" dieser Tabelle mit demselben Dateninhalt (Anzahl Rows) aber mit unterschiedlicher Anzahl Partitionen angelegt

Workload

Es wurde die folgenden zwei Statements in verschiedenen Variationen ausgeführt:

- Full-Partition-Wise Join (FPWJ)

```
SELECT /*+ pq_distribute(detail none, none) */
       count(master.id),
     , count(master.pad)
     , count(detail.id)
     , count(detail.pad01)
     , <Liste der projizierten Spalten aus detail>[33]
  FROM master, detail
 WHERE master.id = detail.id;
```

- Partial-Partition-Wise Join (PPWJ)

```
SELECT /*+ pq_distribute(detail none, partition) */
       count(master.id),
     , count(master.pad)
     , count(detail.id)
     , count(detail.pad01)
     , <Liste der projizierten Spalten aus detail>[33]
  FROM master, detail
 WHERE master.id = detail.id;
```

Die verschiedenen Variationen wurden durch unterschiedliche Werte der folgenden Parameter erzielt.

Parameter	Bemerkungen
<Anzahl beteiligter Knoten>	Es wurden Tests mit Werten 1, 2, 3, 4, 5 und 6 gemacht. Dieser Parameter bestimmt die folgenden, abgeleiteten Parameter: <DOP> und <Anzahl Partitionen>.
<DOP pro Knoten>	Es wurden Tests mit Werten 2 und 4 gemacht. Dieser Parameter bestimmt die folgenden, abgeleiteten Parameter: <DOP> und <Anzahl Partitionen>.
<Liste der projizierten Spalten>	Es wurden Tests mit 6%, 21%, 42%, 62%, 81% und 100% der Daten aus Tabelle detail gemacht.
<parallel_execution_message_size>	Es wurden Tests mit Werten 16 KB, 8 KB, 4 KB und 2 KB gemacht.

[33] Diese Liste diente dazu, das Volumen der PX-Kommunikation zu steuern.

Messmethode

Für verschiedene Variationen des Workloads wurden die Elapsed Times, die Durations aus dem Wait Interface und die CPU Usage für sowohl FPWJ als auch PPWJ wie folgt gemessen:

```
SELECT     event,
           sum(total_waits) AS total_waits,
           sum(total_timeouts) AS total_timeouts,
           sum(time_waited_micro) AS time_waited_micro
FROM (
     SELECT event, total_waits, total_timeouts, time_waited_micro
     FROM gv$session_event
     WHERE sid = p.sid AND inst_id = p.inst_id
     UNION ALL
     SELECT cast('CPU used by this session' AS VARCHAR2(64)) AS event,
                cast(0 AS NUMBER) AS total_waits,
                cast(0 AS NUMBER) AS total_timeouts,
                cast(value*10000 AS NUMBER) AS time_waited_micro
     FROM gv$sesstat
     WHERE statistic# = 12 /* CPU used by this session */
     AND sid = p.sid AND inst_id = p.inst_id
     )
GROUP BY rollup(event)
ORDER BY grouping(event), time_waited_micro DESC
```

Anhand der gemessenen Werte wurden die folgenden Rahmenbedingungen geprüft.

- Für FPWJ:
  - Bei jedem PX-Slave bilden die Durations für „direct path read“ und „CPU used by this session“ der Gesamt-Duration aller Events.
  - Die Summe der Durations dieser Events ist in jedem PX-Slave ungefähr gleich.
- Für PPWJ:
  - Bei jedem PX-Slave in der Rolle Consumer bilden die Durations für „PX Deq: Table Q Normal“, „events in waitclass Other[34]“ und „CPU used by this session“ den Grossteil der Gesamt-Duration aller Events.
  - Die Summe der Durations dieser Events ist in jedem PX-Slave der Consumer-Reihe ungefähr gleich.

Ableitungen

Anhand der durchschnittlichen Duration für „direct path read“ + „CPU used by this session“ pro PX-Slave in FPWJ wurde der sog. Input-Durchsatz für PX-Kommunikation ermittelt (siehe 6.1). Der sog. Input-Durchsatz entspricht dem theoretisch erreichbaren PX-Durchsatz im Falle, dass auf den PX-Channels kein Engpass entstehen würde.

Anhand der durchschnittlichen Duration für „PX Deq: Table Q Normal“, „events in waitclass Other[34]“ und „CPU used by this session“ pro PX-Slave in der Rolle Consumer in PPWJ wurde der sog. Output-Durchsatz ermittelt.

[34] Solange das Event “PX Deq: reap credit” einen wichtiger Bestandteil von „events in waitclass Other“ bildet. Eine feinere Aufteilung von „events in waitclass Other“ ist via trace events 10046 ermittelbar.