Seminararbeit AW1

Niels-Christian Möller-Freuck

Abfragestrategien in verteilten Datenbanksystemen

Niels-Christian Möller-Freuck

Abfragestrategien in verteilten Datenbanksystemen

Seminararbeit eingereicht im Rahmen des Seminars AW1
im Studiengang Master Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Kai von Luck

Abgegeben am 20. Juli 2006

Niels-Christian Möller-Freuck

Thema der Seminararbeit

Abfragestrategien in verteilten Datenbanksystemen (vDBMS)

Stichworte

Abfragestrategien in vDBMS, homogene vDBMS, heterogene vDBMS, Partitionierungsstrategien, Queryshipping, Datashipping, Join Operatoren

Kurzzusammenfassung

Die Seminararbeit befasst sich mit der Verteilung von Daten in homogenen und heterogenen vDBMS, sowie den Strategien Abfragen gegen solche zu optimieren. Hierbei wird kurz auf die Vorgehensweise in klassischen DBMS eingegangen und einzelne Vorgehensweisen exemplarisch betrachtet, um einen Vergleich mit verteilten DBMS zu ermöglichen. Abschließend werden homogene und heterogene vDBMS kurz miteinander verglichen.

Niels-Christian Möller-Freuck

Title of the paper

Queryprocessing in distributed database management systems (dDBMS)

Keywords

Queryprocessing in dDBMS, homogeneous dDBMS, heterogeneous dDBMS, partitioning strategies, queryshipping, datashipping, join operators

Abstract

This report describes partitioning strategies in homogeneous and heterogeneous dDBMS. Further queryprocessing strategies such as different join operators are described in comparison to classic DBMS strategies. A brief look on the differences between homogeneous and heterogeneous dDBMS concludes the report.

1 Einführung

1.1 Überblick

Datenbanksysteme (DBMS) erreichen heute Größenordnungen im Terrabytebereich [STU00]. Auch wenn die Preise für IT-Technologie in den vergangenen Jahren kontinuierlich gesunken sind, meist im zweistelligen Prozentbereich [SCH02], ist der Durchsatz solch großer Datenbanksysteme ein potentieller Flaschenhals für Nutzer der Daten. Diese zentralen DBMS unterliegen diversen Risiken, wie Netzproblemen, Platten- und sonstigen Hardwarecrashes. Auch wenn immer größere Festplattenspeicher verfügbar sind, unterliegen die Arbeitsspeicher weiterhin gewissen Grenzen, was die performante Verarbeitung der Datenbestände in DBMS behindern kann. Gerade für die weit verteilte Nutzung zentraler Datenbestände, beispielsweise in großen weltweit vertretenen Unternehmen ist der Zugriff über das Netzwerke ein Risiko, nicht nur, dass Daten nicht erreichbar sind, durch einen kompletten Ausfall des Netzwerks, sondern auch, dass Daten nicht in geforderten Höchstzeiten aus dem zentralen Datenbestand extrahiert werden können. Hier setzten verteilte Datenbankmanagementsysteme (vDBMS) an, die je nach Bedarf die Daten auf unterschiedliche Sites verteilen und so den Zugriff auf die Daten erheblich verbessern. Diese Seminararbeit befasst sich mit der Fragestellung der Verteilung und Extraktion von Daten in vDBMS, wobei der Unterschied zwischen homogenen und heterogenen vDBMS einer gesonderten Betrachtung unterzogen wird.

1.2 Themenabgrenzung

Die Seminararbeit befasst sich ausschließlich mit der Extraktion von Daten aus verteilten DBMS. Um eine Analogie zu herkömmlichen DBMS aufzubauen werden auch diese kurz betrachtet. Keine Betrachtung finden hingegen Fragen der Transaktionssteuerung in verteilten Systemen. Heterogene vDBMS werden nur im Ansatz betrachtet, was die Verarbeitung von Abfragen anbelangt, da die theoretischen Grundkonzepte verteilter DBMS identisch sind und homogene und heterogene vDBMS sich nur in der Art der Abfrageaufbereitung grundlegend unterscheiden. Bei der Frage der Verteilung der Daten wird die Arbeit nur kurz auf die Umverteilung von Daten im laufenden Betrieb eingehen, lediglich Strategien zur anfänglichen Partitionierung der Daten werden genauer betrachtet.

1.3 Motivation

Einige der vorgestellten Strategien, insbesondere die Frage der Verteilung von Daten, könnte in einer späteren Arbeit einer besonderen Betrachtung unterzogen werden, da hier einer der Schlüsselindikatoren für die Performanz des Gesamtsystems liegt. Interessant ist hier der exemplarische Vergleich einzelner Partitionierungsverfahren, was die Performanz des Gesamtsystems anbelangt.
Auch der praktische Einsatz der Datenverteilung im laufenden Betrieb ist von einigem Interesse, Da hier neben Anfragestrategien auch Transaktionsprobleme in den Vordergrund rücken, die in dieser Arbeit keine Betrachtung gefunden haben.

2 Abfragestrategien in verteilten DBMS

2.1 Aufbau relationaler DBMS

Um die Besonderheiten verteilter Datenbanken bezüglich der Extraktion von Daten zu untersuchen, soll zunächst auf den Aufbau herkömmlicher relationaler DBMS eingegangen werden. Hierbei wird zunächst die Speicherung der Daten betrachtet, und anschließend die Extraktion

2.1.1 Tabellen

Relationale DBMS speichern die Daten nach ihrer Zerlegung in Relationen, in Tabellen, die auf der Festplatte abgelegt werden. Hierbei unterscheidet man folgende drei Speicherungsarten:

- Heapspeicherung, bei der die Datensätze in der Reihenfolge ihrer Eingabe gespeichert werden.
- Sequentielle Speicherung, die Speicherung erfolgt sortiert entsprechend einer vorgegebenen Sortierung.
- Indexsequentielle Speicherung, entsprechend eines Indexes werden die Daten sortiert auf der Festplatte abgelegt.

2.1.2 Indizes

Um einen performateren Zugriff auf die Daten zu gewährleisten werden zusätzlich zu diesen Indizes aufgebaut, die über Verweise auf bestimmte Attribute den Zugriff auf die Datentupel der Relationen verbessern. Indizes werden meistens in Bäumen gehalten, um die Suche einzelner Datentupel zu beschleunigen. Hierbei wird unterschieden zwischen:

- Primärindizes: Das Attribute, über welches eine Tupel in der Datenbank eindeutig repräsentiert wird. Primärindizes verweisen direkt auf den Speicherplatz eines Datentupels, er enthält somit die jeweiligen Datenseiten der Tupel als Blätter des Indexbaumes. [SAA05]
- Sekundärindizes: Indizes auf weitere Attribute der Datentupel, die jedoch im Unterschied zu Primärindizes nicht auf den Speicherplatz selbst sondern auf den Eintrag des Primärindexes verweisen.[DEL01]
- B-Bäume: B-Bäume sind breite Bäume geringer Tiefe, die die Identifikation eines Datensatzes über n Schritte (n = Tiefe des Baumes) ermöglichen.

2.1.3 Zugriff auf Datensätze

Der Zugriff auf die Daten erfolgt über unterschiedliche Strategien. Sollen beispielsweise nur Tupel einer Relation selektiert werden oder liegen auf einzelnen Relationen keine Indizes, so erfolgt der Zugriff sequentiell auf der Festplatte über Scans oder im Falle von Join-Operationen nicht indizierter Relationen über Full Table Scans, bei denen für den Join jeweils mindestens eine der beteiligten Relationen komplett nach passenden Werten durchsucht werden müssen.
Liegen auf den einzelnen Relationen Indizes, so werden diese per Index-Scan oder Index Seek durchsucht, bis die passenden Tupel gefunden sind.
Bei der Ausführung von Join-Operationen wird je zwischen folgenden Ausführungsarten unterschieden:

- Nested Loops: Die Join Kandidaten aus Relation A werden einzeln mit den Kandidaten aus Relation B verglichen, als in einer Schleife durchlaufen.
- Merge Joins: Aus den beteiligten Relationen A und B werden die Kandidaten in Gänze ausgelesen und beide Mengen dann miteinander abgeglichen. Hierbei werden zu-

nächst die Daten entsprechend des Joinattributs sortiert, falls sie es noch nicht sind.[SAA05]

- Hash Joins: beginnen mit der kleiner der beteiligten Relationen und bilden über diese einen Hash, der mit einem Hash über der zweiten Relation abgeglichen wird. Anschließend werden die ermittelten Werte aus den Relationen ausgelesen. Hashin hat trotz des Mehraufwands für das doppelte Lesen auf den Daten den Vorteil, dass nur die Speicherseiten wirklich selektiert werden, die tatsächlich zu dem Verbund gehören. [KEM01]

Die Verarbeitung der Daten kann in Client-Server DBMS-Umgebungen nicht nur auf der Datenbankmaschine erfolgen, sondern kann auch auf dem Client stattfinden. Hier wird zwischen Query Shipping und Data Shipping Verfahren unterschieden. Das Query Shipping bezeichnet die Übertragung des Ausführungsbefehls vom Client zum Server, wo die Selektion der Daten stattfindet; beim Data Shipping werden die kompletten Daten an den Client übergeben, auf welchem dann die Selektion der relevanten Daten stattfindet. Neben diesen Grundverfahren existiert noch das sog. Hybrid Shipping bei dem einzelne Teile der Daten oder Hashes auf den Client übertragen werden, beispielsweise Lookup-Tabellen, über die anschließend Hashes gebildet werden und an den Server zur Ausführung von Join-Operationen übergeben werden können.

2.1.4 Optimierungskriterien für Abfrageoperationen

In klassischen DBMS hängt die Güte der Abfrageausführung stark von dem Aufbau der Relationen und den auf diese angewandten Indizes ab. Über vorgefertigte Abfragen, für die das DBMS Queryexecutionplans ablegen kann, kann weiterhin eine Optimierung der Datenzugriffe bewirkt werden. [DEL01] Diese Pläne liegen meist im Arbeitsspeicher des Servers vor und erhöhen den Datendurchsatz. Sie verhindern jedoch die Beteiligung des Clients an der Ausführung der Abfrageoperation, wodurch heutige sehr mächtige PC-Systeme auf dem Stand von früheren Terminals gehalten werden.
Dieses Phänomen wird noch zusätzlich durch den Trend verstärkt, ganze Teile der Geschäftslogic in die DBMS einzubringen, z.B. in Form von Stored Procedures auf dem MS SQL-Server oder ORACLE DBMS. Neuere Versionen dieser Produkte bieten nicht nur die Möglichkeit SQL-Prozeduren abzulegen, sondern gleich ganze Programmpakete in Java oder .net Sprachen auf dem Server zum Laufen zu bringen. [STU00] [DEL01] So wird das DBMS immer weiter in Richtung Mainframe ausgebaut was wiederum die Kosten für die Server und den Aufwand für deren Betreuung steigert. Bei unternehmenskritischen Anwendungen sind zudem Failoverserver oder ganze Serverfarmen und Plattencluster zu betreiben, um eine Ausfallsicherheit des Systems zu gewährleisten.

2.2 Verteilte Datenbanken

Verteilte Datenbanken haben zwei Ziele: zum einen sind sie interessant für Unternehmen, die an diversen weit entfernten Standorten auf die Daten zugreifen müssen, zum anderen ermöglicht die Verteilung der Daten auf viele kleine Maschinen einen wesentlich höhere Performanz und Reduzierung der Kosten gegenüber teuren und aufwendig zu wartenden Servern. Hierbei stellen sich jedoch diverse andere Probleme ein, die bei monolitischen DBMS nicht anzutreffen sind.

2.2.1 Anforderungen an verteilte Datenbanksysteme

Folgende Forderungen werden in der Literatur an vDBMS gestellt:[SAA05]

- vDBMS stellen sich dem Benutzer gegenüber wie integrierte zentrale DBMS dar
- Lokale Autonomie: die lokalen Knoten haben maximale Kontrolle über ihre Daten und hängen nicht von anderen Knoten ab
- Unabhängigkeit von zentralen Strukturen um Flaschenhälse zu vermeiden

- Hohe Verfügbarkeit ist bei Ausfällen einzelner Knoten zu gewährleisten
- Ortstransparenz bedeutet, dass der Benutzer ohne Kenntnis des Speicherortes von Daten auf diese zugreifen kann
- Fragmentierungstransparenz: die Verteilung einzelner Relationen ist transparent zu halten
- Replikationstransparenz: die Verteilung von redundanten Daten zur Gewährleistung der Verfügbarkeit ist vor dem Benutzer zu verbergen
- Verteilte Anfragebearbeitung: Anfragen sind durch Optimierer zu verteilen
- Hardwareunabhängigkeit
- Netzwerunabhängigkeit
- Datenbanksystemunabhängigkeit

Die letzteren drei Forderungen treffen jedoch nur auf heterogene vDBMS zu.

2.2.2 Partitionierung der Daten

Soll eine bestehende zentrale Datenbank in eine verteilte überführt werden oder wird ein neues vDBMS geplant, so ist zunächst die Modalität der Datenverteilung festzulegen. Im Folgenden wird hier von der Verteilung einer bestehenden Datenbank ausgegangen, die auf diverse Knoten verteilt werden soll. Dieses Szenario kümmert sich zunächst nicht um die spätere laufende Neu- oder Umverteilung von Daten, sondern nur um den aktuellen Datenbestand. Die spätere Partitionierung wird nur oberflächliche Betrachtung finden. Ferner werden nur redundanzfreie Methoden dargestellt werden.
Die Verteilung der Daten ist von elementarer Bedeutung, da sie zum einen über die Berechenbarkeit des Speicherortes entscheidet, was bei späteren Anfragen an das System die Performanz der Zugriffe wesentlich beeinflusst, zum anderen direkten Einfluss auf die Gesamtperformanz der Abfragen hat.
Der gewählte Mechanismus zur Partitionierung der Daten kann ebenfalls entscheidend für die Frage sein, wie der globale Datenkatalog gehalten werden kann. So kann beispielsweise bei einem berechenbaren Verteilungsverfahren der globale Datenkatalog klein gehalten und somit auf sämtlichen Knotenrechnern vorgehalten werden. Ist hingegen der globale Katalog sehr groß, ist über einen zentralen oder einen Netz- oder Clusterkatalog nachzudenken. Diese Varianten haben gegenüber dem vollredundanten Katalog jedoch den Nachteil, dass sie einen Single-Point-of-Failure darstellen können.

2.2.3 Fragmentierungsverfahren

Bei der Aufteilung der Datenbestände wird grundlegend zwischen horizontaler und vertikaler Fragmentierung unterschieden.
Die Horizontale Fragmentierung zerlegt den Datenbestand entsprechend der Verteilungsalgorithmen satzweise, d.h. einzelne Tupel einer Relation werden auf unterschiedliche Knoten aufgeteilt.
Bei der vertikalen Partitionierung werden die einzelnen Tupel zerlegt und als Unterrelationen auf verschiedenen Knoten gespeichert. Hierbei ergibt sich das Problem, das die Rekonstruktion der Tupel nur in Abhängigkeit von einem anderen Knoten erfolgen kann, was der Regel der lokalen Autonomie widerspricht.
Neben diesen beiden Grundprinzipien gibt es auch die Möglichkeit beide Verfahren zu mischen.[SAA05][STU00]
Die jeweilige Fragmentierungsart hängt stark von der Art der Zugriffe auf die Daten ab und entscheidet auch über die Möglichkeit einer Indizierung von Daten. So könnten beispielsweise bei einer vertikalen Fragmentierung ursprüngliche Sekundärindizes auf Teilrelationen in Primärindizes umgewandelt werden, was einen Beschleunigten Zugriff auf die Datenfragmente ermöglicht.

2.2.4 Partitionierungsverfahren

Bei der horizontalen Fragmentierung können unterschiedlichste Verfahren angewandt werden. Die Wahl des eingesetzten Verfahrens hängt stark von den Bedarfen der jeweiligen Anwendung ab. Im folgenden werden die wesentlichen Verfahren dargelegt.
Eindimensionale Verfahren:

- Round Robin Partitioning: Die Datensätze werden nummeriert und MOD n = Anzahl der Knoten auf die einzelnen Knoten verteilt. Dieses Verfahren sorgt für eine sehr balancierte Verteilung der Daten.
- Range Partitioning: Die Datensätze werden in einzelne Bereiche unterteilt, z.B. alphabetisch oder entsprechend Filialnummernkreisen, und auf entsprechende Knoten verteilt
- Hash Partitioning: Über die Schlüsselattribute einer Relation wird ein Hashwert gebildet und die Daten entsprechend des Hashwertes auf die einzelnen Knoten verteilt. [BLA05]

All diese Verfahren sind sehr gut geeignet um auf Datensätze über ihr Primärattribut zuzugreifen, da diese zu jeweils einem berechenbaren oder festgelegten Knoten führt, auf dem die Abfrage ausgeführt wird. Soll jedoch über Sekundärattribute auf die Daten zugegriffen werden, so sind die Anfragen an alle Knoten zu stellen, was die Gesamtperformanz des Systems stark beeinträchtigt.
Eine Alternative bieten hier mehrdimensionale Verfahren, von denen exemplarisch das Verfahren MAGIC hier vorgestellt werden soll.
Beim MAGIC Verfahren werden die Daten ebenfalls horizontal fragmentiert. Es werden neben den Primärattributen einer Relation aber auch die Sekundärattribute betrachtet. Für jedes betrachtete Attribut wird eine Dimension in einer Matrix erzeugt und die Attribute entsprechend der Anzahl der Knoten analog der Rangepartition unterteilt. Jede Dimension wird in gleich viele Einheiten zerlegt und die Daten anschließend entsprechend der jeweiligen Zelle auf die einzelnen Knoten verteilt. So kann z.B. ein Aktienchart nach Firmenname und nach Aktienpreis zweidimensional zerlegt werden. Wird jede Dimension in sechs Einheiten zerlegt, so ergibt sich eine Verteilung auf 36 Knoten. Der Vorteil dieses Verfahrens ist, dass bei Zugriff auf eines der betrachteten Attribute stets die gleiche Anzahl von Knoten, im Beispiel sechs Knoten, die Anfrage ausführen muss. Dies steigert sehr stark den Gesamtdurchsatz des Systems. [BLA05][GHA01][GHA02]

2.2.5 Partitionierung im laufenden Betrieb

Da die Datenbestände sich mit der Zeit ändern, können Ungleichgewichte im Datenbestand entstehen, die sich auf die Performanz des Gesamtsystems auswirken. Daneben kann im laufenden Betrieb erkannt werden, dass einzelne Knoten stärker belastet sind, als in der ursprünglichen Verteilung vorgesehen. Auch können weitere Knoten in das System eingebracht und bestehende ersetzt werden. Dies führt zu der Notwendigkeit, die Daten umzuverteilen.
Die Umverteilung der Daten wird in drei Phasen aufgeteilt:

- Monitor: die Phase betrachtet den Workload jedes Knotens in Bezug auf seine Datenmenge, den Datendurchsatz und die durchschnittliche Antwortzeit. Hierbei werden feste Zeitintervalle betrachtet.
- Predict: in dieser Phase werden die neu zu verteilenden Datensätze ausgewählt und die Knoten bestimmt, denen sie zugeordnet werden sollen.
  Hierbei können unterschiedliche Verfahren zum Einsatz gebracht werden. Im EVEN Verfahren wird der Knoten mit dem höchsten und der mit dem niedrigsten Workload im Verhältnis zur durchschnittlichen Last identifiziert.
  Alternativ hierzu wird im EVEN C/B Verfahren noch die Berechnung der Kosten des Transfers der Daten im Verhältnis zu der erwarteten Durchsatzsteigerung berechnet und in Abhängigkeit hiervon der Knoten berechnet, der die Daten erhalten soll

- Migrate: In dieser Phase werden die Daten auf den neuen Knoten übertragen. Hierbei ist zu entscheiden, ob Änderungen während dieser Phase zuzulassen sind oder der Datensatz als gesperrt markiert werden soll. Da eine Sperre auf den Daten oft nicht in Frage kommt, wird folgende Strategie vorgeschlagen: Das Lesen der Daten erfolgt stets auf dem Quelldatensatz. Das Schreiben hingegen muss auf beiden Datensätzen erfolgen. Hier ist ein entsprechender Transaktionsmechanismus (2PC oder 3PC) zu implementieren, um die Integrität der Daten zu gewährleisten. [BLA05]

Performanzvergleiche der einzelnen Strategien können in [GHA01] und [BLA05] nachgelesen werden. Die Studien zeigen, dass bereits nach kurzer Zeit die Gesamtsystemperformanz sinkt, wenn die Daten nicht stätig neu verteilt werden. Die Studien zeigen weiterhin, dass es ggf. lohnend sein kann, zu gewissen Zeiten die Verteilung des Gesamtsystem in Gänze neu zu berechnen.

2.3 Join Operationen in vDBMS

Um den Verbundoperator auf verteilten Datenbanken auszuführen sind unterschiedliche Ansätze möglich. Hierbei ist jeweils die Menge der beteiligten Knoten sowie die Ergebnismenge zu betrachten.

Sort-Merge Join: Bei diesem Verfahren wird die kleinere der beteiligten Relationen mittels eines Hashverfahrens, das das Vergleichsattribut betrachtet in eine temporäre Datei verpackt und diese wird an alle Knoten übertragen, wo dann der Join-Operator ausgeführt wird.

Grace Hash-Join: Bei diesem Verfahren werden die Daten einer Relation entsprechend des Vergleichsattributes mittels eines Hashverfahren aufgeteilt und in sog. Buckets abgelegt. Anschließend wird der gleiche Hashwert auf die zweite Relation angewandt und die Daten entsprechend auf die einzelnen Buckets verteilt. In einem letzten Schritt werden die einzelnen Buckets untersucht und die Ergebnismenge extrahiert. [BLA05]

Beide Verfahren bewegen jeweils die gesamten Daten der beteiligten Relationen über das Netz. Bei sehr großen Datenmengen kann es von Vorteil sein mit Semi-Join- oder Hashfilter-Join-Verfahren zu arbeiten

Hierbei wird im ersten Fall zunächst berechnet, welche Tupel der ersten Relation überhaupt als mögliche Kandidaten der Ergebnismenge in Betracht kommen. Nur diese Tupel werden übertragen.Der Empfängerknoten führt über dieser Teilmenge seinen Join aus und übergibt das Ergebnis an den ersten Knoten, der nun die gesamte Ergebnismenge zusammenstellt. Als Alternativ können auf beiden Knoten nur die Unterergebnismengen berechnet werden und an einen weiteren Knoten übertragen werden. [SAA05][KOS01]

Beim Hashfilter-Join werden von einer der beiden Relationen nur die Nummern der Datensätze und ggf. das Verbundattribut übertragen, die von der ersten Relation zum Join beitragen. Die empfangende Seite selektiert entsprechende Datensätze und überträgt diese zurück an den Sender. Somit ist der Netzwerkverkehr möglichst gering gehalten. [SAA05]

Eine Alternative ist es in Analogie zum Loop-Join Verfahren Datensätze einzeln oder in kleinen Portionen über das Netz zu versenden. Hierbei sinkt zwar nicht die Gesamtnetzlast, die Abarbeitung kann jedoch bereits auf dem entfernten Knoten beginnen, bevor der sendende Knoten die Daten in Gänze berechnet hat. Somit steigt der Gesamtdurchsatz des Systems an.

2.4 Heterogene vDBMS

Die bisher betrachteten Zugriffstrategien sind zwar allgemeingültig für vDBMS, lassen sich jedoch sehr unproblematisch auf homogene vDBMS anwenden.

Im Unterschied hierzu bedürfen heterogene vDBMS einer besonderen Betrachtung, da sie anders als homogene Systeme neben unterschiedlichen Betriebssystemen und Hardwareausstattungen auch völlig unterschiedliche Datensysteme darstellen können. So können Daten beispielsweise auf einzelnen Knoten im Filesystem als XML-Dateien oder in nichtrelationalen DBMS abgelegt sein.

Deshalb sind heterogene vDBMS auf sog. Förderierungsdiensten aufgebaut, die zentral oder dezentral die Datenkataloge in globalen Schemata halten. Ein Mediator übersetzt die Anfragen gegen den Datenbestand in ein allgemeingültiges Anfrageformat.
Jeder beteiligte Knoten verfügt über einen Wraper Dienste, der die vom Mediator versandte Anfrage übersetzt und für die Ausführung der Anfrage zuständig ist. Weiterhin übersetzt der Wraper Dienst das globale in das lokales Schema.
Die Ergebnisse müssen abschließend wieder vom Wraper Dienst im allgemeingültigen Format überstellt werden.[SAA05]
Als Beispiel hierfür kann das MOCHA Projekt (Middleware based on a Code Shipping Architecture) gelten. Die Daten werden als XML-Dateien von einer Java basierte Mittelschicht übertragen. [MOC01]
Ein zentraler Query Processing Coordinator (QPC) übernimmt folgende Aufgaben:

- Query Parsing
- Ressource Discovery
- Metadata and Control Exchange
- Code Deployment Auswahl der Java Klassen
- Query Execution

Jeder Knoten verfügt über einen Data Access Provider (DAP) der für folgende Aufgaben zuständig ist:

- Data Access
- Data Translation
- Query Execution

3 Zusammenfassung

In den letzten Jahren sind die Netzwerke beständig ausgebaut worden und die Kosten gleichzeitig stark gesunken. Gleichzeitig ist die Rechenleistung von Arbeitsplatzrechnern heute auf einem Stand, auf dem vor wenigen Jahren Mainframes waren. Diese Kombination läst die Möglichkeit zum Einsatz verteilter DBMS immer interessanter werden, da im Gegenzug die Kosten für große Server bei weitem nicht so stark gesunken sind wie diese an Rechenleistung zugelegt haben. Als Beispiel für diese Erkenntnis und deren Umsetzung mag die Firma Google gelten, die ihren Datenbestand auf einer täglich wachsenden Zahl von PC verwaltet. Auch im Bereich des mobilen Computings ist der Ansatz verteilter Datenbanken zu betrachten, da der Trend zu überall verfügbaren Daten mit der zunehmenden Mobilität zunehmen wird. Hier kann ein zentrales System schnell zum Flaschenhals werden.
Interessant sind hier besonders die Ansätze heterogener vDBMS, da sie die Möglichkeit bieten beliebige Datenformate zu bearbeiten und so jegliche Geräteklasse vom Server bis zum Smartphone als Datenträger in Frage kommt.

4 Literaturverzeichnis

[BLA03] Jacek Blazewicz, Wielaw Kubiak, Tadeusz Morzy, Marek Rusinkiewicz (Herausgeber), Handbook on Data Management in Information Systems, ISBN 3-540-43893-9

[DEL01] Kalen Delaney, Ron Soukup, Inside Microsoft SQL Server 2000, ISBN 3-86063-619-7

[GHA01] Shahram Ghandeharizadeh, Shan Gao, Chris Gahagan, and Russ Krauss, High Performance Parallel DBMS, http://dblab.usc.edu/Users/shkim/papers/HandBookofDB.pdf

[GHA90] Shahram Ghandeharizadeh, David J. Dewitt, Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines, http://www.vldb.org/conf/1990/P481.PDF

[GHA92] Shahram Ghandeharizadeh David J. DeWitt, A Performance Analysis of Alternative Multi-Attribute Declustering Strategies, http://170.210.130.9/root/public/SMOD7592/SIGMOD92/P029.PDF

[KEM01] Alfons Kemper, André Eickler, Datenbanksysteme, ISBN 3-486-25706-4

[KOS01] Donald Kossmann, The State of the Art in Distributed Query Processing, http://www.db.fmi.uni-passau/~kossman

[LAM94] Winfried Lamersdorf, Datenbanken in verteilten Systemen, ISBN 3-528-05467-0

[MOC01] MOCHA Self Extensible Database Middleware, http://www.cs.umd.edu/projects/mocha/mocha.html

[LAW01] Christopher Lawson, Oracle Performance Tuning, ISBN 1-904347-01-0

[SAA05] Gunter Saake, Andreas Heuer, Kai-Uwe Sattler, Datenbanken, ISBN 3-8266-1438-0

[STU00] Günther Stürmer, Oracle8i, ISBN 3-930124-01-7

[SCH02] André Schlüter, Technischer Fortschritt durch Informations- und Kommunikationstechnologien http://hsr-trans.zhsf.uni-koeln.de/hsrretro/docs/artikel/hsr/hsr2002_538.pdf