

Online Analytical Processing with a Cluster of Databases

Uwe Röhm
roehm@inf.ethz.ch

Abstract: Eine attraktive Plattform für große Informationssysteme sind Datenbankcluster. Sie bieten hohe Leistung, so genannte „scale-out“ Skalierbarkeit, Fehlertoleranz und ein sehr gutes Preis-/Leistungsverhältnis. Die Herausforderung ist dabei, ein System zu entwickeln, das sowohl Skalierbarkeit und Leistungsfähigkeit, als auch transaktionelle Garantien in sich vereint.

Diese Arbeit untersucht zentrale Aspekte von *Datenbankclustern* und ihrer Leistungsfähigkeit bei der Verwendung für Online Analytical Processing. Das Ziel ist eine skalierbare Infrastruktur für online Decision Support Systeme, die insbesondere Benutzern die Analyse aktueller Daten erlaubt. Die Arbeit verfolgt dabei einen Ansatz, der auf einer *Koordinationsmiddleware* basiert. Es werden innovative Algorithmen zur leistungsfähigen Anfrageverteilung basierend auf approximierten Cache-Zuständen sowie ein neuartiger Ansatz zur koordinierten Replikationsverwaltung für große Cluster vorgestellt. Die Kombination dieser Techniken ermöglicht effizientes Online Analytical Processing mit einem Datenbankcluster, wobei Klienten „Resultataktualität“ gegen Anfragegeschwindigkeit eintauschen können und darüber hinaus sogar in die Lage versetzt werden, Daten vom neusten Stand zu analysieren.

1 Einführung

Online analytical processing (OLAP) zielt auf die Analyse der gewaltigen Datenmengen, die sich in einem Unternehmens oder einer Organisation durch das Tagesgeschäft ansammeln. Das Ziel ist, strategische Entscheidungen vorzubereiten und genauere Einblicke in die Beschaffenheit der Geschäfte, an der eine Organisation teilnimmt, zu erlangen. Entsprechende Anwendungen reichen von online Shops, die auf Grund des Kundenverhaltens beliebte Produkte vorschlagen, bis zu internationalen Konzernen, die versuchen, über Ihre Kundenbeziehungsdaten neue profitable Märkte zu identifizieren.

Solche OLAP Systeme müssen sehr große Datenbestände bewältigen und zugleich kurze Antwortzeiten für eine interaktive Nutzung erlauben. Sie müssen außerdem skalieren können, das heisst, bezüglich der anwachsenden Datenmenge einfach erweiterbar sein. Außerdem gewinnt die Anforderung, dass die analysierten Daten „up-to-date“ sein sollten, mehr und mehr an Bedeutung. Nun sind dies nicht nur gegensätzliche Anforderungen, sie laufen zudem auch den Leistungsanforderungen des Tagesgeschäfts zuwider.

Deswegen sind die meisten heutigen OLAP Systeme von den geschäftskritischen operationellen Systemen getrennt. Das bedeutet, dass man bewusst einen Kompromiss zwischen Datenaktualität, oder auch Datenfrische genannt, und Antwortzeiten eingeht. Die benötigten Daten werden periodisch aus den operationellen Datenbanken extrahiert und in

ein separates *Data Warehouse* geladen (vgl. Abbildung 1). Dies geschieht dann, wenn das Tagesgeschäft möglichst nicht beeinträchtigt wird, das heisst bevorzugt Nachts oder am Wochenende. Diese Architekturentscheidungen ziehen eine ganze Reihe von Problemen

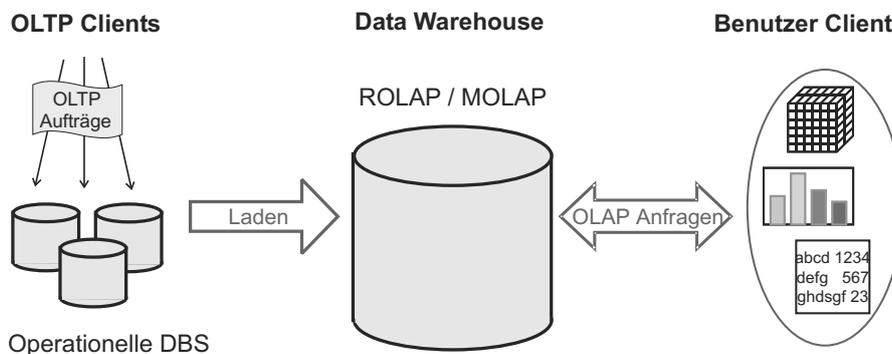


Abbildung 1: Klassischer Aufbau eines kommerziellen Data Warehouses.

bei der Pflege eines Data Warehouses nach sich. Insbesondere können OLAP Nutzer nur (ver-)altete Daten analysieren und Entscheidungsträger, die auf aktuelle Daten angewiesen sind, werden gar nicht unterstützt.

Diese Arbeit präsentiert einen neuen Ansatz für online Decision Support Systeme, der es ermöglicht, aktuelle „up-to-date“ Daten zu analysieren. Der Ansatz basiert auf einem *Datenbankcluster*: das ist ein Cluster von gewöhnlichen PCs als Hardware-Infrastruktur und Standard Datenbankmanagementsystemen „von-der-Stange“ als transaktionale Speicherschicht. Eine darüber liegende Koordinationsmiddleware kapselt die Details und bietet eine einheitliche, universelle Anfrageschnittstelle. Das Ergebnis ist eine „Datenbank von Datenbanken“ entsprechend der Vision von *Hyperdatenbanken* [SBG⁺00]. Ein wichtiges Entwurfsprinzip eines Datenbankclusters ist seine Komponentenarchitektur. Insbesondere wollen wir den Cluster Dank der Standard Hard- und Softwarekomponenten einfach aufbauen und erweitern können.

Der Beitrag der hier vorgestellten Dissertation [Rö02] ist der Entwurf, die Implementierung und die Evaluation einer Koordinationsmiddleware für eine solche Datenbankcluster-Architektur. Im Zentrum stehen drei neu entwickelte Protokolle zur effizienten Anfrageverteilung — *Cache Approximation Query Routing* — und zur skalierbaren Replikationskontrolle — *Coordinated Replication Management* — sowie ihr Kombination zu *Freshness-Aware Scheduling*.

Dieser Artikel ist wie folgt aufgebaut: Abschnitt 2 stellt die Systemarchitektur vor und Abschnitt 3 gibt einen kurzen Überblick über unseren Ansatz zum Query Routing. Im darauf folgenden Abschnitt 4 kommen wir zum Kern der Arbeit, den neu entwickelten Ansätzen zur Replikationskontrolle und dem Frische-basierten Scheduling. Abschnitt 5 gibt einen kurzen Einblick in die quantitative Evaluation der Verfahren und die wichtigsten Erkenntnisse. Den Abschluss bildet eine Zusammenfassung.

2 Systemmodell und Architektur

Transaktionsarten. Wir unterscheiden die Transaktionen, die von Clients gestellt werden (so genannte *Client-Transaktionen*) in *reine Lese- bzw. OLAP-Transaktionen* und in *Update-Transaktionen*. Eine Lesetransaktion besteht ausschließlich aus Anfragen. Eine Update-Transaktion umfasst mindestens einen Insert, Delete oder Update Befehl – im folgenden kurz unter Updates zusammengefasst – sowie eine beliebige Anzahl weiterer SQL Befehle. Entkoppelte *Refresh-Subtransaktionen* propagieren die Änderungen im Cluster.

Architektur. Ein *Datenbankcluster* ist ein Cluster von Standard PCs, jeder mit seinem eigenen vorinstallierten Datenbankmanagementsystem (DBMS) als transaktionelle Speicherschicht. Wir nehmen an, dass alle Clusterknoten homogen sind, das heisst, dass sie alle das gleiche DBMS mit demselben Datenbankschema verwenden. Jeder Knoten speichert dabei eine vollständige Kopie der Datenbank. Wir bezeichnen eine Datenbank eines Clusterknotens als eine *Komponentendatenbank*. Weiterhin unterscheiden wir zwischen einem ausgezeichneten *Master-Knoten* und den übrigen n *OLAP-Knoten*. Es gibt eine *Koordinationsmiddleware*, kurz *Koordinator* genannt, die den gesamten Cluster verwaltet. Sie ist für das Scheduling, Routing und Logging eintreffender Anfragen verantwortlich. Während der Cluster aus Standard Hard- und Softwarekomponenten „von der Stange“ besteht, ist der Cluster Koordinator eine Eigenentwicklung. Er umfasst eine *Eingabeschlange*, einen *Scheduler*, einen *Router*, einen *Auffrischer* sowie einen *Logger* (vgl. Abbildung 2).

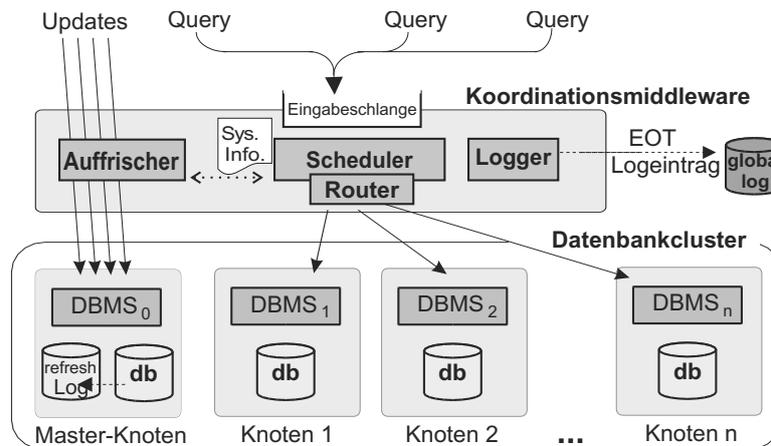


Abbildung 2: Systemarchitektur.

Klienten senden an die Koordinationsmiddleware Lese- und Update-Transaktionen. Die Middleware steuert und leitet Updates und Anfragen zu den Clusterknoten. Dabei erzeugt der *Scheduler* eine korrekt verzahnte Ausführungsreihenfolge. Der Master-Knoten hat die Rolle des Primärknotens, auf dem alle Updates zuerst ausgeführt werden. Diese Arbeit geht von nur einem Master-Knoten aus. Er könnte aber auch selbst geclustert sein. Seine interne Organisation ist aber für den Koordinator ohne Belang. Jener muss lediglich die Serialisierungsordnung kennen und darüber ein Log auf oberer Ebene führen.

Anfragen gelangen in das System über die *Eingabeschlange*. Diese Eingabeschlange wird nicht in „first-in-first-out“ Manier abgearbeitet. Vielmehr entscheidet der *Scheduler*, in welcher Reihenfolge die eintreffenden Aufträge abgearbeitet werden. Dabei wird das Verhungern von Aufträgen mittels einer maximalen Wartezeit vermieden. Grundsätzlich kann es mehrere OLAP-Knoten geben, auf denen eine Anfrage einer Lesetransaktion ausgeführt werden könnte. Der *Router* wählt für jede Anfrage einen der möglichen Knoten aus. Zu diesem Zweck verwaltet die Koordinationsmiddleware eigene Informationen über den globalen *Systemzustand*, z.B. die Version eines jeden Knotens.

Die Transaktionsverwaltung der Koordinationsmiddleware garantiert globale Korrektheit und Konsistenz. Dazu wird eine einfache Form eines zweischichtigen *offen-geschachtelten Transaktionsmodells* [WS92] verwendet: Die Anfragen einer Lesetransaktion werden in den Komponentendatenbanken als separate Subtransaktionen ausgeführt und committed. Die Koordinationsmiddleware beinhaltet einen globalen *Logger*. Dieser führt Buch über die Update-Subtransaktionen auf dem Master-Knoten und die entkoppelten Update-Subtransaktionen auf den OLAP-Knoten. Letztere werden vom so genannten *Auffrischer* kontrolliert. Dieses Vorgehen ermöglicht globale Korrektheit ohne ein verteiltes Commit-Protokoll wie zum Beispiel das Zwei-Phasen-Commit-Protokoll (2PC). Dies zu vermeiden ist gerade für große Cluster besonders wichtig...

3 Cache Approximation Query Routing

Betrachten wir zunächst den einfachen Fall mit rein lesenden Zugriffen ohne Datenänderungen und konzentrieren uns auf die Leistungssteigerung mittels geeigneter *Query Routing* Verfahren. Indem man Daten über alle Clusterknoten repliziert, wird es möglich, mehrere Analyseanfragen unabhängig voneinander und parallel auf verschiedenen Knoten auszuführen. Die Frage ist allerdings, welches der am besten geeignete Knoten für die Ausführung einer gegebenen Anfrage ist.

Das Ziel von Query Routing ist es, die Antwortzeiten von Anfragen zu reduzieren. Die Ausführzeiten von Anfragen – insbesondere von OLAP Anfragen – werden dabei von den I/O Kosten dominiert. Obwohl Caching eine wichtige Rolle für die Geschwindigkeit der Anfrageauswertung spielt, gehen bisherige Routing-Algorithmen darauf nicht ein.

Diese Arbeit stellt nun eine Reihe von neuen Query Routing Algorithmen für den Einsatz über eingekapselten Komponenten entwickelt. Diese Verfahren approximieren den Inhalt der Datenbankpuffer in den Clusterknoten aufgrund der zuletzt dort ausgeführten Anfragen. Sie verwenden diese Approximationen, um Anfragen zu solchen Knoten zu schicken, die eine besonders schnelle Ausführung durch bereits gepufferte Daten erwarten lassen.

Idealerweise würde man dazu den Cache-Zustand in Datenbankseiten ausdrücken wollen. Der Koordinator hat aber keinerlei Zugriff auf das Cache-Directory der Komponentendatenbanken. Also muss er sich diese Informationen ableiten. Die Grundidee ist, die Cache-Zustände mit der Tupelmenge, die zur Evaluation einer Anfrage benötigt wird, zu approximieren. Dies kann nun in verschiedenen Qualitätsstufen geschehen [RBS01]:

Cache Approximation mittels FROM-Klausel. Der erste Ansatz ist, die benötigte Tupelmengemenge mittels der verwendeten Relationen zu approximieren. Diese können ganz einfach aus den FROM Klauseln von SQL-Anfragen abgelesen werden. Die Approximation der Cache-Zustände ist dann die Menge der Relationen, auf die von den letzten n dort ausgeführten Anfragen zugegriffen wurde. Der Nutzen dieses Caches für neue Anfragen ergibt sich aus der Größe der Überschneidung zwischen der Cache Approximation und der FROM Klausel der Anfrage. Es ergibt sich ein einfaches, aber durchaus schon effektives Verfahren, das regelmässig über 10% Laufzeitgewinn erreicht.

Cache Approximation mittels Bitstring-Signaturen. Will man genauer bestimmen, welche Teile einer Relation für die Auswertung einer Anfrage benötigt werden, muss man die Anfrageprädikate betrachten. Allerdings ist es sehr schwierig, die Schnittmenge zweier Prädikate mit gemeinsamen Attributen zu bestimmen. Deshalb wurde in [Rö02] ein weiteres Routingverfahren entwickelt, das die ein Prädikat erfüllende Tupelmengemenge mittels Bitstring-Signaturen approximiert. Auf diese Weise ist die Schnittmengenabschätzung nicht nur recht genau, sondern kann Dank einfacher Bitstringvergleiche auch sehr effizient durchgeführt werden. Dieses Verfahren ermöglicht, die mittleren Antwortzeiten von OLAP Anfragen signifikant zu reduzieren. Normalisiert man die Approximationsvergleiche zusätzlich mit den Ausführkosten, so können die Antwortzeiten im Vergleich zu nicht approximativen Routingverfahren mehr als halbiert werden.

4 Replikationskontrolle mittels Mehrschichtiger Transaktionen

Im nächsten Schritt betrachten wir nun zusätzlich auch Datenänderungen. Replikationsverwaltung ist für grosse Cluster nach wie vor ein offenes Problem — insbesondere wenn das Ziel ein Verfahren ist, das gleichzeitig Skalierbarkeit, Korrektheit und Zugriff auf die zuletzt geänderten Daten garantiert! Dazu wurde ein neues Replikationsverfahren entwickelt, das in sich Prinzipien eines offen-geschachtelten Transaktionsmodells mit asynchroner Replikationskontrolle vereint.

Das neu entwickelte Replikationsverfahren heisst *Coordinated Replication Management* (CRM). Es garantiert allen Clients den Zugriff auf konsistente und aktuelle Daten. Es verhält sich dahingehend wie synchrone Replikation. Gleichzeitig hat das Verfahren aber die Leistungscharakteristiken von asynchroner Replikation. Der folgenden Abschnitt gibt einen Überblick.

4.1 Coordinated Replication Management

Das Ziel von CRM ist, die Ausführung von Lese- und Update-Transaktionen so zu verzahnen, dass Anfragen garantiert immer auf „up-to-date“ Daten zugreifen. Gleichzeitig geben wir aber die Forderung nach Korrektheit nicht auf — das Protokoll soll „one-copy Serialisierbarkeit“ garantieren.

Alle bisherigen Replikationsverfahren gehen von einem konventionellen flachen Transaktionsmodell aus: Entweder findet die Replikation innerhalb einer verteilten Trans-

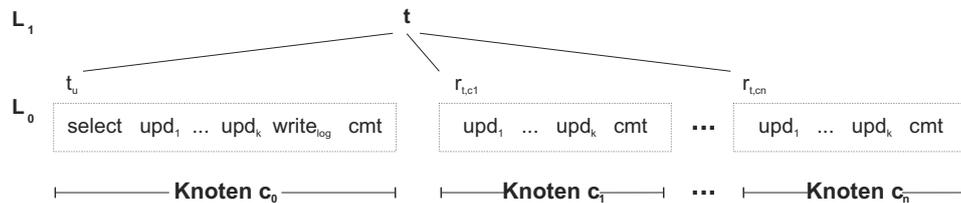


Abbildung 3: Verwendung von geschachtelten Transaktionen mit CRM.

aktion statt oder Transaktionen werden im Zugriff auf einzelne Knoten beschränkt. Im ersteren Fall wird ein verteiltes atomares Commit-Protokoll benötigt, was aber nicht für große Knotenanzahl geeignet ist [GHOS96]. In letzterem Fall kann Serialisierbarkeit nur für bestimmte restriktive Konfigurationen sichergestellt werden [BKR⁺99]. Die letztere Alternative hat auch keine Kontrolle über die entstehende Serialisierungsordnung. Es wird lediglich garantiert, dass die Transaktionen eines Clients generell korrekt mit allen anderen Transaktionen serialisiert wird, aber nicht, *in welcher Reihenfolge genau*.

CRM verfolgt einen anderen Ansatz, in dem es für die Replikationskontrolle ein mehrschichtiges Transaktionsmodell anwendet: Im Falle einer Lesetransaktion wird jede Anfrage zu einer Subtransaktion, einer so genannten *Lese-Subtransaktion*. Bei CRM greift jede einzelne Anfragen auf genau einen Clusterknoten zu, wobei aber jede Anfrage einer Lesetransaktion aus Performanzgründen durchaus zu verschiedenen OLAP-Knoten geschickt werden kann (sofern sie dieselbe Datenversion vorhalten), wie Abschnitt 3 gezeigt hat.

Im Falle einer Update-Transaktion wird für jedes Replikat eine eigene Subtransaktion gestartet. Wir unterscheiden dabei zwischen der ersten *Update-Subtransaktion* und den übrigen *Refresh-Subtransaktionen*: Die Koordinationsmiddleware führt Updates zuerst in einer Update-Subtransaktion auf dem Master-Knoten aus und loggt dabei die ausgeführten Updates. Dabei ist die Anzahl gleichzeitig ausgeführter Update-Subtransaktionen auf dem Master-Knoten in keiner Weise eingeschränkt. Nachdem eine Update-Subtransaktion endete und sobald ein Refresh aktiviert wird, propagiert der Auffrischer die Änderungen mittels entkoppelter Refresh-Subtransaktionen zu den Replikaten. CRM folgt also einem Primary-Copy Replikationsschema mit entkoppelter Auffrischung.

Abbildung 3 verdeutlicht das. Der Client startet eine globale Update-Transaktion t . Als erstes wird auf dem Master-Knoten c_0 die Update-Subtransaktion t_u ausgeführt, einschliesslich zusätzlicher Logoperationen für das Refresh-Log. Nach dem Commit dieser ersten Subtransaktion propagieren entkoppelte Refresh-Subtransaktionen $r_{t,c_1}, \dots, r_{t,c_n}$ die Änderungen zu allen Replikaten. Jene starten typischerweise zu verschiedenen Zeitpunkten, so dass der Cluster nie als Ganzes blockiert ist.

Für den Client, der die Update-Transaktion gestellt hat, ist die Transaktion mit dem erfolgreichen Commit der ersten Update-Subtransaktion auf dem Master-Knoten beendet. Das entspricht dem Vorgehen bei asynchroner Replikation. Insbesondere muss ein Update-Client nicht warten, bis die ganze globale Update-Transaktion beendet wurde. CRM kann dies auf Grund der vollständigen Replikation zulassen: Wenn eine Update-Subtransaktion auf dem Master-Knoten erfolgreich mit Commit beendet wurde, dann kann CRM auch garantieren, dass alle zugehörigen Refresh-Subtransaktionen ebenfalls committen werden.

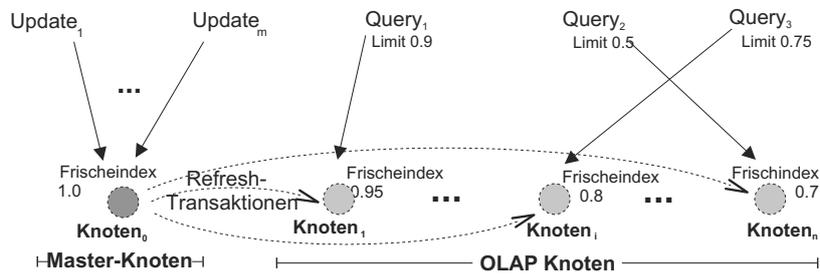


Abbildung 4: Prinzip von Freshness Aware Scheduling.

Der Vorteil der separaten Refresh-Subtransaktionen ist, dass sie Replikatе asynchron auffrischen, dabei aber sowohl globale Korrektheit als auch up-to-date Garantien für Client garantieren. Jede Refresh-Subtransaktionen frischt nur einen Knoten auf und wird dazu getrennt aktiviert. Weiterhin garantiert jeder Knoten lokal serialisierbare Ausführungen. Der Logger der Koordinationsmiddleware verfolgt das Propagieren der Updates, d.h., welche Refresh-Subtransaktionen bereits durchgeführt wurden und welche noch ausstehen.

Außerdem garantiert CRM *Lesekonsistenz*: CRM propagiert Refresh-Subtransaktionen dergestalt, dass Lesetransaktionen während ihrer Laufzeit immer auf dieselbe Version der Daten zugreifen. Dazu sind einige Vorkehrungen notwendig, da mit CRM jede Transaktion beliebig viele verschiedene Replikatе ansprechen darf und zusätzlich der Router jede Anfrage einer Lesetransaktion zu einem anderen OLAP-Knoten leiten kann. Wie wir im vorherigen Abschnitt gesehen haben, ist das Routen von Anfragen derselben Transaktion zu verschiedenen Clusterknoten wegen der Caching-Effekte recht vorteilhaft..

4.2 Freshness-Aware Scheduling

CRM garantiert den Zugriff auf immer topaktuelle Daten. Wir werden es im folgenden verallgemeinern, in dem wir diese Datenaktualität variabel machen. Die Idee hierbei ist, Benutzern zu ermöglichen, Datenaktualität gegen kürzere Antwortzeiten einzutauschen. Zu diesem Zweck führen wir einen Quality-of-Service Parameter für Lesetransaktionen ein: die *Mindestfrische*. Dies ermöglicht „Frische-basiertes“ Scheduling, das die verschiedenen Frischegrade der Clusterknoten verwendet, um Anfragen, die mit weniger frischen Daten zufrieden sind, früher zu bearbeiten als solche, die nach aktuellen Daten fragen. Das entstehende Verfahren heisst *Freshness-Aware Scheduling* (FAS) [RBSS02].

FAS vereint in sich Replikationsverwaltung und Mehrversionen-Concurrency Control. Der Ansatz drückt die Aktualität eines OLAP-Knotens c_i mittels eines so genannten *Frischeindex* $f(c_i) \in [0, 1]$ aus. Der Frischeindex besagt, wie sehr sich der Knoten c_i von dem Master-Knoten unterscheidet. Die verwendete Metrik basiert auf dem Zeitunterschied zwischen dem letzten propagierten Update und der neusten Version der Daten. Frischeindex 1 bedeutet, dass die Daten auf dem aktuellen Stand sind. Dagegen sind Daten mit einem Frischeindex 0 „unendlich“ veraltet.

Die Vorgehensweise, um Serialisierbarkeit und Lesekonsistenz sicher zu stellen, ist bei FAS dieselbe wie bei CRM: Updates werden zuerst auf dem Master-Knoten ausgeführt

und danach mittels entkoppelter Update-Subtransaktionen propagiert. FAS startet Refresh-Subtransaktionen dabei so, dass Lesetransaktionen immer auf dieselbe Version von Daten zugreifen. Aber diese Version muss nun nicht mehr der letzte Stand sein. Vielmehr wird von FAS nur garantiert, dass die Frische der zugegriffenen OLAP-Knoten mindestens so hoch ist wie von den Lesetransaktionen geforderte Mindestfrische.

Abbildung 4 gibt ein Beispiel mit drei Anfragen mit unterschiedlichen Frischelimiten. Die erste Anfrage benötigt Daten mit einem Frischegrad von mindestens 0,9. Da nur beim ersten OLAP-Knoten der Frischeindex höher ist, kann die Anfrage nur dort ausgeführt werden. Anfrage 2 hingegen fragt nach Daten ab Frische 0,5. Diese Frische wird von allen Clusterknoten geboten. Deshalb kann FAS Anfrage 2 zu einem beliebigen OLAP-Knoten schicken. In Abbildung 4 wird der letzte Knoten gewählt, es hätte aber auch irgendein anderer sein dürfen. Der Effekt ist, dass Anfrage 2 in der Tat einen Cluster der Größe n sieht, während für Anfrage 1 nur ein Knoten nutzbar ist.

5 Evaluation

Abschliessend waren wir auch an den Leistungscharakteristiken der neu entwickelten Verfahren interessiert. Dazu wurden die vorgestellten Scheduling Algorithmen prototypisch implementiert und einer umfangreichen Evaluierung mit dem TPC-R Benchmark für Online Analytical Processing unterzogen.

Für die Evaluation stand ein Datenbankcluster mit 128 Pentium III 1 GHz Rechnern zur Verfügung, auf denen jeweils ein SQL Server 2000 unter Windows 2000 Advanced Server installiert war. Die Datenbanken wurden entsprechend dem TPC-R Benchmark mit Skalierungsfaktor 1 generiert (Größe inklusive Indexe circa 2 GBytes). Der Koordinator lief auf einem separaten PC mit zwei 1 GHz Pentium III und 512 MB RAM.

5.1 Leistungsfähigkeit von CRM und FAS

Wir konzentrieren uns im folgenden auf die zentralen Evaluationsergebnisse mit CRM und FAS. Wie man in Abbildung 5 sehen kann, erlaubt es FAS in der Tat, Antwortzeit gegen Datenfrische einzutauschen. Die Clients haben verschiedene mittlere Frischelimiten zwischen 0,6 und 1 angegeben. Je nach gefordertem Frischegrad liegt die Verlangsamung der Anfragen durch parallele Updates gerade einmal zwischen 10% und 60% im Vergleich zur mittleren Antwortzeit ohne Updates (vgl. Abbildung 5(b)). Offensichtlich profitieren Clients, die explizit erlauben, ihre Anfragen auf älteren Daten auszuführen, von mehr verfügbaren Clusterknoten und können daher früher und schneller bedient werden.

Dabei ist der Overhead für Clients, die nach aktuellen Daten mit Frische 1 fragen (das entspricht CRM), nur moderate 60%. Und obwohl dabei vor jeder Anfrage die OLAP-Knoten auf den neusten Stand gebracht werden müssen, wird die OLTP Seite nicht negativ beeinflusst. Das sieht man in Abbildung 5(c), wo der Update-Durchsatz auf dem Master-Knoten für verschiedene Clustergrößen und Frischeanforderungen aufgezeigt ist — jener bleibt konstant und bricht nicht mit der steigenden Refresh-Last ein.

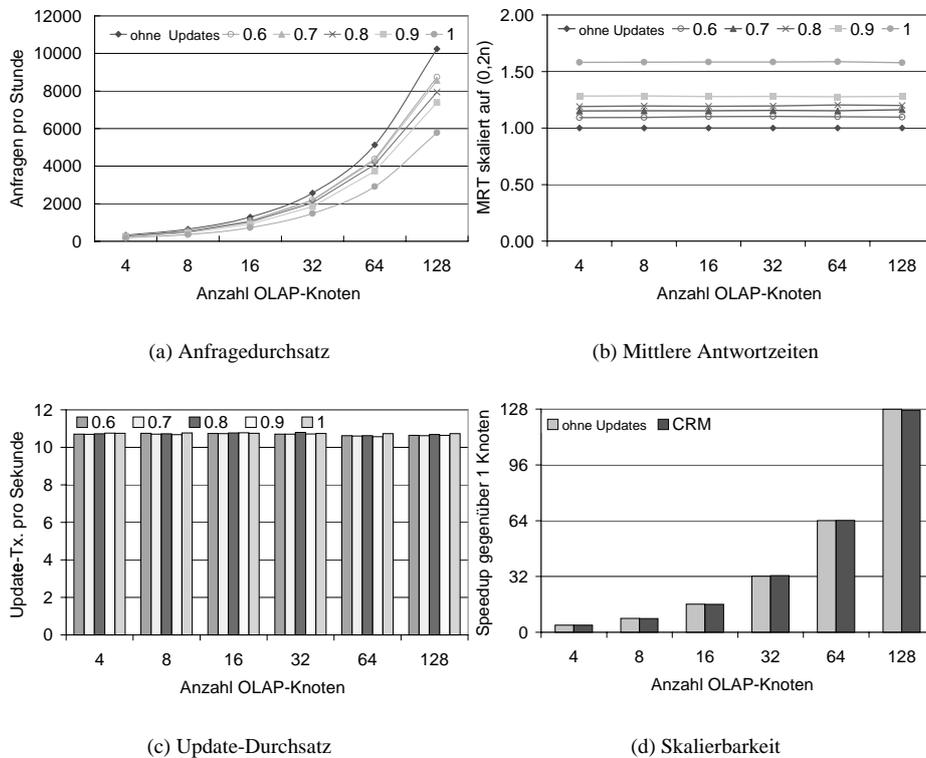


Abbildung 5: Leistungsergebnisse von CRM und FAS.

Zu guter Letzt ist auch die Skalierbarkeit der Verfahren sehr positiv zu bewerten. Im Wesentlichen ist die Performanz unabhängig von der Clustergröße. Wie Abbildung 5(d) schön zeigt, skaliert CRM und damit auch FAS perfekt linear, zumindest bis zu der hier möglichen maximalen Clustergröße von 128 Knoten.

6 Zusammenfassung

In dieser Arbeit wurden neue Verfahren zum Query Routing, zur Replikationskontrolle und zu Mehrversionen-Scheduling für OLAP in einem Datenbankcluster vorgestellt. Die Verfahren sind sowohl auf einer formalen Grundlage aufgebaut, als auch vollständig in einem lauffähigen Prototypen implementiert. In einer umfangreichen experimentellen Evaluation wurde außerdem erfolgreich ihre praktische Anwendbarkeit gezeigt.

Es zeigte sich, dass Cache-approximatives Query Routing im Vergleich zu einfacheren Verfahren die Antwortzeiten deutlich beschleunigen kann. Dabei ist es ein dynamisches online Verfahren und für beliebige SQL Anfragen geeignet. Mit CRM wurde ein neuartiges Replikationsverfahren vorgestellt, das selbst bei sehr großen Replikatenmengen (in

unseren Tests bis 128) perfekt linear skaliert und gleichzeitig One-Copy-Serialisierbarkeit bietet. Dabei ist die Verlangsamung der OLAP-Anfragen, die auf aktuelle Daten zugreifen, gerade einmal moderate 60%. Darüber hinaus ermöglicht das Frische-basierte Schedulingprotokoll FAS Benutzern, effektiv Datenfrische gegen Anfragegeschwindigkeit einzutauschen. Mit diesem Verfahren lassen sich die Antwortzeiten für OLAP-Anfragen in einem Datenbankcluster beliebig nahe an das Optimum ohne Updates heranzuführen.

Literatur

- [BKR⁺99] Breitbart, Y., Komondoor, R., Rastogi, R., Seshadri, S., und Silberschatz, A.: Update propagation protocols for replicated databases. In: *Proceedings of SIGMOD 1999, June 1–3, Philadelphia, USA*. 1999.
- [GHOS96] Gray, J., Helland, P., O’Neil, P. E., und Shasha, D.: The dangers of replication and a solution. In: *Proceedings of the 1996 ACM SIGMOD Conference, June 4–6, Montreal, Quebec, Canada*. S. 173–182. 1996.
- [RBS01] Röhm, U., Böhm, K., und Schek, H.-J.: Cache-aware query routing in a cluster of databases. In: *Proceedings of the 17th ICDE Conference, April 2–6, Heidelberg*. 2001.
- [RBSS02] Röhm, U., Böhm, K., Schek, H.-J., und Schuldt, H.: FAS – a freshness-sensitive coordination middleware for a cluster of OLAP components. In: *Proceedings of the 28th VLDB Conference, August 20–23, Hong Kong*. 2002.
- [Rö02] Röhm, U.: *Online Analytical Processing with a Cluster of Databases*. PhD thesis. Diss. ETH No. 14591. DISBIS 80, infix Verlag. 2002.
- [SBG⁺00] Schek, H.-J., Böhm, K., Grabs, T., Röhm, U., Schuldt, H., und Weber, R.: Hyperdatabases. In: *Proceedings of the First Int. Conference on Web Information Systems Engineering (WISE2000), June 19–21, 2000, Hong Kong, China*. S. 14–25. 2000.
- [WS92] Weikum, G. und Schek, H.-J.: Concepts and applications of multilevel transactions and open nested transactions. In: Elmagarmid, A. K. (Hrsg.), *Database Transaction Models for Advanced Applications*. S. 515–553. Morgan Kaufmann. 1992.

Curriculum Vitae

Name	Dr. Uwe Röhm Diplom Informatiker	
Werdegang	seit 2002	IT Consultant, Avinci AG
	1996 – 2002	Institut für Informationssysteme, ETH Zürich Promotion in der Forschungsgruppe von Prof. H.-J. Schek
	1990 – 1996	Informatikstudium an der Universität Passau Vertiefung: Informationssysteme / Datenbanken
	1989 – 1990	Zivildienst als Rettungssanitäter, ASB Taunusstein
	1976 – 1989	Abitur, Gesamtschule „Obere Aar“, Taunusstein