Seminar Datenbanken und Informationssysteme

Verlässliche, adaptive Informationssysteme
(Dependable Adaptive Information Systems, DAIS)

Thema: Self-tuning Databases

Thomas Sommer
Betreuer: Christian Mathis
19.01.2006

Inhaltsverzeichnis

Abstract - Automatisierte Tuning-Möglichkeiten für Datenbanksysteme (DBS) nehmen einen immer größeren Stellenwert auf dem Gebiet der Datenbanken ein. Weiter wachsende und komplexer werdende Systeme verlangen eine bessere Administration um konkurrenzfähig zu bleiben. Doch auch die Administration wird komplizierter, sodass diese meist nur mit Hilfe von Tuning-Programmen sinnvoll gelöst werden kann. In dieser Ausarbeitung wird ein Verfahren zur Automatisierung von beliebigen Tuning-Parametern mithilfe einer rückgekoppelten Schleife vorgestellt. Darüberhinaus werden Algorithmen vorgestellt, die die Wahl eines geeigneten Multi-Programming-Level ermöglichen. Ein weiteres Thema ist die Selektion von sinnvollen Indizes und materialisierten Sichten.

1. Einleitung

Datenbanksysteme werden immer größer und komplexer. Das liegt zum einen an der größeren Datenmenge, zum anderen aber auch an komplizierteren Anfragen (z. B. mit Rekursion). So fordern vor allem E-Services eine Unterstützung für eine breitere Menge von Anfrageklassen [EPB+03, Härd05]. Je komplizierter jedoch das System, desto mehr Arbeit kommt auf den Datenbankadministrator (DBA) zu. Deshalb werden in vielen Systemen heute hochbezahlte DBA zur Administration eingesetzt. Die Kosten für das Personal übersteigen jedoch in vielen Fällen die Kosten für Hardware und Software des Systems [BROW00]. Um Kosten zu sparen und um neue Einsatzgebiete für DBS zu erschließen, sollten die Aufgaben des DBA automatisiert werden.

Weiterhin wird von aktuellen Systemen, eine gute Performance erwartet, wenn möglich in allen Situationen. Eine schlechte Leistung des DBS, wenn sie auch nur von kurzer Dauer ist, kann ein Unternehmen womöglich viel Geld kosten. Das System muss sich deshalb während der Laufzeit auftretenden Veränderungen anpassen. Darunter fallen z. B. die Anzahl der Benutzer, eine Laständerung oder Änderungen bei der Verfügbarkeit von benutzten Ressourcen. Da sich die genannten Faktoren unter Umständen sehr schnell ändern können, ist nur eine Automatisierung der Tuning-Parameter dazu in der Lage, eine dauerhaft gute Leistung zu garantieren.

Auch in vorhandenen Systemen wurde dieses Problem erkannt und zum Teil schon durch Tuning-Programme gelöst. So gibt es bei IBM z. B. das Projekt SMART (eng., „self-managing and resource tuning“) [LoLi02] oder bei Microsoft den AutoAdmin. Beide Programme übernehmen mehr und mehr die Aufgaben von Datenbankadministratoren, um die Systeme einfacher benutzbar zu machen.

In dieser Ausarbeitung wird zunächst in Kapitel 2 ein Verfahren vorgestellt, im System bereits vorhandene Tuning-Parameter ohne großen Aufwand nachträglich zu automatisieren. Dies wird mit Hilfe einer rückgekoppelten Kontrollschleife erreicht. In den nachfolgenden Kapiteln werden darüber hinaus verschiedene Tuning-Möglichkeiten präsentiert.

Kapitel 3 behandelt die automatische Wahl des Multi-Programming-Levels (MPL). Um einen möglichst hohen Durchsatz zu erreichen, sollten möglichst viele Transaktionen (TA) in einem System aktiv sein. Dies erhöht jedoch die Antwortzeit, aufgrund der höheren Last. Eine Erweiterung dieses Verfahrens, ist das in Betracht ziehen von verschiedenen Anfrageklassen bei der Wahl des MPL.

Typische Design-Entscheidungen, wie der Einsatz Indizes und materialisierten Sichten, werden in Kapitel 4 und 5 näher betrachtet. Um eine schnelle Abarbeitung von Anfragen zu ermöglichen, müssen solche Elemente zusätzlich in das System integriert werden.

In Kapitel 6 werden zwei weitere Möglichkeiten des Auto-Tunings, die dynamische Daten-Verteilung und das multidimensionale Clustering von Tabellen vorgestellt. Eine kurze Schlussfolgerung findet sich in Kapitel 7.

2. Observation, Prediction, Reaction (OPR)

In einem Datenbanksystem gibt es, je nach Komplexität der Software und Größe des Datenbestandes, einige wenige bis viele hundert Tuning-Möglichkeiten (Knöpfe, eng., „knobs“). Heutzutage ist es meist üblich, viele dieser Möglichkeiten „von Hand“ festzulegen. D. h., der Datenbankadministrator muss mit Hilfe seines Wissens und seiner Erfahrung versuchen, eine möglichst hohe Leistung zu erreichen. Es ist oft der Fall, dass dem DBA dabei keine technischen Hilfen zu Verfügung stehen.

Um die vorhandenen Einstellungs-Möglichkeiten zumindest teilweise automatisieren zu können, sollte das DBS diese Knöpfe selbstständig den aktuellen Gegebenheiten anpassen. Ein allgemeiner Ansatz, ist eine rückgekoppelte Kontrollschleife (eng., „feedback control loop“) [WMH+02, WHM+94] zu benutzen (Bild 1). D. h., das System beobachtet sich, bzw. seine Leistungswerte ständig selbst, um bei einem Leistungsabfall oder Fehlverhalten eigenständig und so schnell wie möglich reagieren zu können.

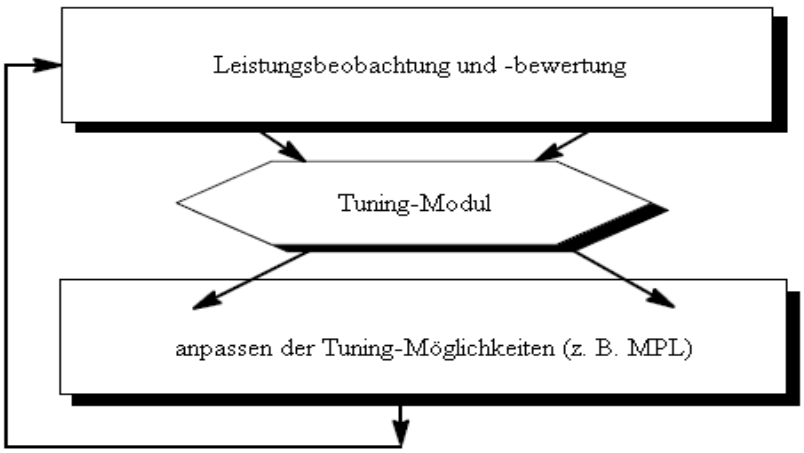

Bild 1. *Rückgekoppelte Kontrollschleife innerhalb eines Datenbanksystems.*

Eine oft verwendete Realisierung der Kontrollschleife ist das OPR-Verfahren (Beobachtung, eng., „observation“, Vorhersage, eng., „prediction“, Reaktion, eng., „reaction“), welches in den folgenden Unterabschnitten näher beschrieben wird. Das OPR-Verfahren kann auf die meisten, vorgestellten Tuning-Möglichkeiten dieser Ausarbeitung angewendet werden.

2.1 Observation

In der Beobachtungsphase überwacht ein Tuning-Programm das Datenbanksystem bzw. seine einzelnen Komponenten. Darunter fällt z. B. der Durchsatz und die Antwortzeit bei der automatischen Anpassung des Multi-Program-

ming-Level (Kapitel 3), oder auch die Leistungsbeinflussung des Systems durch Indizes und materialisierte Sichten, wenn diese Elemente im System automatisiert sind (Kapitel 4 und 5).

Die Beobachtungsphase muss Veränderungen an den zu messenden Werten erkennen und, falls kritische Werte über- bzw. unterschritten werden, das Programm, welches die zugehörige Tuning-Möglichkeit realisiert, über die Veränderung benachrichtigen. Alle weiteren Aktionen bleiben den nachfolgenden Schritten überlassen.

2.2 Prediction

Falls in der Beobachtungsphase problematische Werte gemessen wurden, muss in dieser Phase die Art und die (voraussichtliche) Wirkung der Reaktion bestimmt werden. Wie die eigentliche Reaktion dann ausgeführt wird, bestimmt die nachfolgende Phase.

Im ersten Schritt wird die Tuning-Möglichkeit identifiziert, welche das Problem möglichst gut löst. D. h., es ist nötig, den Parameter im DBS zu finden, der das Problem verursacht hat oder der geändert werden muss, um das System wieder in einen akzeptablen Zustand zu überführen. Nachdem der verantwortliche Parameter ermittelt wurde, wird eine Vorhersage getroffen, wie die Änderung des Parameters das Verhalten des Datenbanksystems beeinflusst. Dies geschieht meist über mathematische Modelle, wie zum Beispiel bei der automatischen Anpassung des Multi-Programming-Levels (Kapitel 3.1). Leider ist es nicht immer realisierbar, dass Verhalten der Datenbank exakt vorherzusagen, sodass oftmals Heuristiken oder Erfahrungswerte eine große Rolle spielen. Deshalb ist es auch wichtig eine rückgekoppelte Schleife zu verwenden, um eventuelle Vorhersage-Fehler erkennen zu können.

2.3 Reaction

Wenn in der Prediction-Phase ein Tuning-Parameter ausgewählt wurde und ein neuer Wert für einen Knopf bestimmt wurde, ist es Aufgabe dieser Phase, die neuen Werte in das DBS zu übernehmen.

Da alle Berechnungen und Auswahlverfahren bereits in den vorangegangenen Schritten ausgeführt wurden, erscheint diese Phase als die einfachste. In manchen Fällen, wie der Anpassung des MPL (Kapitel 3) ist dies auch der Fall, da nur die Anzahl der aktiven Transaktionen im System geändert werden muss. In den meisten Fällen ist es jedoch nicht möglich, so einfach zum Ziel zu gelangen. Im schlimmsten Fall muss sogar das Datenbanksystem neu gestartet werden, damit die Änderungen wirksam werden. In vielen Systemen (z. B. bei Banken), ist das nicht so ohne weiteres möglich, da die Ausführung von Transaktionen zu jeder Zeit gewährleistet sein muss. Das System kann also nicht zur Veränderung eines Tuning-Parameters abgeschaltet werden. Vielfach ist auch eine Behinderung bzw. Laufzeitverlängerung aktiver Transaktionen nicht akzeptabel, sodass in der Reaktionsphase darauf geachtet werden muss, dass das System durch die Anpassung der Knöpfe nicht behindert wird.

3. Multi-Programming-Level (MPL)

Eine verhältnismäßig allgemeine Einstellung ist der Parallelitätsgrad (Multi-Programming-Level) des DBS. Durch den MPL wird die Anzahl der theoretisch möglichen aktiven Transaktionen im DBS beschrieben. D. h. es dürfen nur maximal genau so viele Transaktionen gleichzeitig auf das Datenbanksystem zugreifen, wie vom MPL angegeben. Falls neue Transaktionen gestartet werden, die maximale Anzahl an parallelen Transaktionen jedoch schon erreicht ist, müssen diese warten bis ältere Transaktionen beendet sind. Die neu ankommenden Transaktionen werden dann in einer externen Queue (siehe Bild 2) verwaltet, die meistens als FIFO-Schlange (eng., „first in, first out“) realisiert ist. Diese Vorgehensweise garantiert immer eine gleichmäßige Ausnutzung des Systems und eine faire Behandlung der Transaktionen.

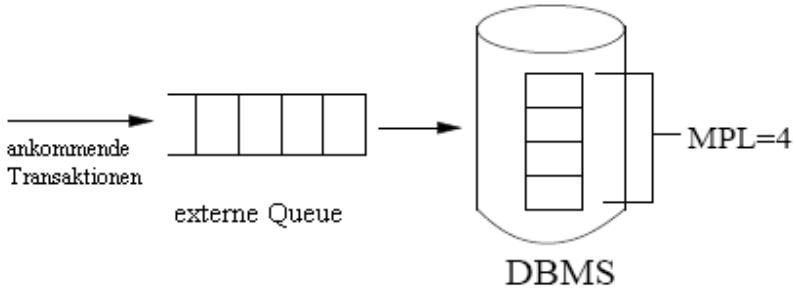

Bild 2. *Funktionsweise eines externen Schedulers in einem Transaktionssytem. [SHW+06].*

Es ist sofort ersichtlich [SHW+06, WMH+02], dass die Wahl des MPL direkt den Durchsatz, wie auch die Antwortzeit beeinflusst. Ein hoher MPL-Wert steigert automatisch auch den Durchsatz an Transaktionen. Ebenso wie der Durchsatz, steigt aber auch die Konfliktgefahr unter den TA, sodass es zu Thrashing-Effekten kommen kann. Von Thrashing spricht man, wenn der Durchsatz aufgrund von zu vielen TA im System wieder sinkt. Dies tritt vor allem auf, wenn die Laufzeit der Transaktionen durch Blockierungen signifikant verlängert wird. Je länger die Laufzeit der TA, desto mehr Behinderungen treten auf, was zur Folge hat, dass zu viele TA im System den Durchsatz (und die Antwortzeit) erheblich beeinträchtigen können. Auf der anderen Seite, hat ein niedriger MPL-Wert den Vorteil, dass die Transaktionen weniger oft in Blockierungen verwickelt sind, was zur Folge hat, dass die Antwortzeit sinkt. Ab einem gewissen Punkt kann jedoch die Antwortzeit nicht mehr verbessert werden, was bei einem weiteren Absenken des Parallelitätsgrades nur noch das Verschlechtern des Durchsatzes nach sich zieht. Ein zu niedriger MPL hat also ebenfalls negative Auswirkungen auf das DBS, auch wenn die Folgen nicht so gravierend erscheinen, wie bei einem zu ho-

hen MPL.

Aufgrund dieser Beobachtungen ist es das Ziel, einen ausgewogenen Wert für den MPL zu finden, sodass einerseits der Durchsatz hoch und andererseits die Antwortzeit gering ist. Die richtige Wahl des MPL hängt also von zwei entgegengesetzten Zielen ab. Eine mögliche Vorgehensweise, um jeweils eines dieser Ziele zu erreichen, wird in den nächsten Abschnitten erläutert. Eine Verfeinerung des Verfahrens für verschiedene Anfrageklassen ist Thema in Kapitel 3.4.

3.1 Ermitteln der oberen Grenze

Wie oben beschrieben, beeinflusst der Multi-Programming-Level auf direktem Weg den Durchsatz an Transaktionen. Je höher der Wert des MPL ist desto höher ist – zumindest theoretisch – der Durchsatz an Transaktionen. Da jedoch von den meisten Transaktionen Sperren angefordert werden, kann es dazu kommen, dass Blockierungen zwischen den Transaktionen auftreten. Es muss also eine Transaktion auf eine andere warten. Im schlimmsten Fall kann sogar eine Deadlock-Situation auftreten, was zur Folge hat, dass mindestens eine Transaktion neu gestartet werden muss.

Durch das Erhöhen des MPL-Wertes kommt es also automatisch auch zu einer Erhöhung der Wahrscheinlichkeit von Blockierungen und Deadlocks im System. Infolgedessen sinkt der Durchsatz an Transaktionen wieder. Natürlich beeinflusst dies auch die Antwortzeit, sodass diese ebenfalls deutlich steigt. Es muss also auf jeden Fall vermieden werden, in ein Thrashing-Verhalten zu geraten. Andererseits muss auch ein möglichst großer Durchsatz gewährleistet sein. Der MPL-Wert sollte also knapp an der (oberen) Grenze zum Thrashing gewählt werden.

Im COMFORT-Projekt [WHM+94, WMH+02] wurde ein Verfahren vorgestellt welches es erlaubt, mittels einer rückgekoppelten Kontrollschleife (Kapitel 2) einen annähernd optimalen Parallelitätsgrad zu erreichen, ohne zum Thrashing zu neigen. Zu diesem Zweck wurde der Begriff der Konfliktrate von Transaktionen eingeführt, mit der es möglich ist, Thrashing-Effekte zu erkennen. Die Konfliktrate lässt sich vergleichsweise einfach durch das Zählen von gehaltenen Sperren ermitteln und wird wie folgt berechnet:

$$Konfliktrate = \frac{(Anzahl\ gehaltener\ Sperren)}{(Anzahl\ Sperren\ nicht\ blockierter\ TA)}$$

In aufwändigen Versuchen mit verschiedenen Datenbanksystemen und Transaktions-Mixen wurde ein kritischer Wert von 1,3 ermittelt, ab dem das System in Thrashing-Gefahr gerät. Die Beobachtungsphase gestaltet sich bei diesem Verfahren also, aufgrund der simplen Berechnung, recht einfach. Es muss lediglich die Anzahl der aktuell aktiven Transaktionen und die Konfliktrate gemessen werden. Danach wird eine Vorhersage für jede neu ankommende Transaktion durchgeführt, die berechnet, ob durch Ausführung dieser

Transaktion die Konfliktrate überschritten wird. Natürlich ist es nicht möglich, dies mit hundertprozentiger Wahrscheinlichkeit zu bestimmen. Deswegen ist es nötig, eine Heuristik dafür einzuführen. In der Reaktionsphase wird nun jede neue ankommende Transaktion anhand der Vorhersage entweder ausgeführt oder solange blockiert, bis die Konfliktrate durch die Ausführung nicht mehr gefährdet ist. Es ist eventuell auch nötig schon aktive Transaktionen wieder zu beenden falls in der Beobachtungsphase festgestellt wird, dass ohne das Starten neuer TA, die Konfliktrate überschritten wurde. Die Auswahl der zu beendenden Transaktion geschieht entweder zufällig, oder durch Anwendung von vorher definierten Regeln.

3.2 Ermitteln der unteren Grenze

Neben einem möglichst hohen Durchsatz ist in vielen Anwendungsgebieten von Datenbanksystemen auch die Antwortzeit entscheidend. In immer mehr Bereichen wird z. B. verlangt, dass die Antwortzeit immer unter einem vorher festgelegten kritischen Wert bleibt. Solche Systeme finden sich unter anderem bei Banken und Versicherungen, wo die Verzögerung einer Transaktion einen hohen finanziellen Verlust nach sich ziehen kann.

Je geringer der MPL ist, desto geringer – siehe oben – ist auch die Antwortzeit. Denn einerseits gibt es weniger Blockierungen zwischen den einzelnen Transaktionen, andererseits sind auch die Systemkomponenten weniger ausgelastet, sodass die Transaktionen nicht auf die Freigabe von Ressourcen warten müssen. Zu tief darf der MPL-Wert aber auch nicht gewählt werden. Ein zu niedriger MPL führt zu einem starken Abfall des Durchsatzes dadurch, dass die Transaktionen lange warten müssen bis sie in das System dürfen. Der Benutzer hat dann den Eindruck einer langen Antwortzeit. Eine zu geringe Parallelität wirkt sich also genau gegenteilig zu dem gewünschten Effekt, einer geringen Antwortzeit, aus.

Das Ziel sollte es demnach sein, den MPL-Wert soweit wie möglich abzusenken, jedoch einen gewissen Durchsatzanteil zu garantieren. Wenn ein bestimmter Durchsatz erzielt wird, normalisiert sich automatisch die Antwortzeit auf ein für den Benutzer akzeptables Niveau. In [SHW+06] wurden Versuche durchgeführt, den MPL-Wert soweit wie möglich zu senken, jedoch immer noch 80 % bzw. 95 % des ursprünglichen Durchsatzes (ohne Anpassung der Parallelität) zu erhalten. Hauptergebnis der Untersuchung ist, dass unterschiedliche Transaktions-Typen einen unterschiedlichen Einfluss auf die Antwortzeit haben. So muss z. B. bei Ausführung von vielen verschiedenen Transaktions-Typen die Parallelität erhöht werden, um die Durchsatz-Vorgabe zu erreichen. Wie die Behandlung verschiedenen Anfrageklassen optimiert werden kann, findet sich in Kapitel 3.4.

3.3 Automatische Anpassung

In den letzten beiden Abschnitten wurde gezeigt, dass für einen hohen Durchsatz, auch ein möglichst hoher MPL nötig ist. Fast genauso wichtig ist jedoch auch eine akzeptable Antwortzeit des Systems, weshalb der MPL-Wert nicht

zu hoch gewählt werden darf. In früheren Datenbanksystemen war es meist die Aufgabe des Administrators den Wert für die Parallelität festzulegen und anzupassen. Dies geschah jedoch oft nur bei der Installation des Systems, oder wenn signifikante Leistungseinbußen festzustellen waren. Kurzfristige Veränderungen der Lastsituation oder andere Einflüsse auf die Leistung werden jedoch nicht beachtet und es kann somit auch keine Garantie gegeben werden, dass das System zu jeder Zeit die gewünschte Leistung erfüllt. Für aktuelle und zukünftige Datenbanksysteme ist dieses Vorgehen somit nicht mehr akzeptabel.

Die Anpassung bzw. Optimierung des MPL-Wertes muss demnach direkt durch das DBS erfolgen. Das hier vorgestellte Verfahren benötigt als einziges zusätzliches Element eine externe Warteschlange (Bild 2). Somit können bestehende Datenbanksysteme vergleichsweise einfach erweitert werden [SHW+06]. Die Abbildung auf das OPR-Verfahren (Kapitel 2) fällt ebenfalls recht leicht. In der Beobachtungsphase müssen die Systemparameter, wie die Konfliktrate oder die Antwortzeit der Transaktionen, überwacht werden. Tritt eine Abweichung der überwachten Werte vom Standard auf, muss der MPL-Wert der neuen Situation angepasst werden. Wie die Parallelität angepasst wird, hängt von den gemessenen Werten ab. Ist z. B. die Konfliktrate sehr hoch (siehe Kapitel 3.1), müssen aktive TA beendet werden, um den MPL zu senken. Wenn die Antwortzeit und Konfliktrate verhältnismäßig niedrig sind, muss der MPL erhöht werden, d. h. Transaktionen aus der Warteschlange werden aktiviert. Wie für die beiden beispielhaft genannten Fälle, existiert für jede mögliche Situation eine passende Reaktion des Tuning-Programms. Wichtig bei der Automatisierung des MPL ist auch die ständige Überwachnung der Systemparameter. Es kann z. B. durchaus vorkommen, dass die Konfliktrate in den kritischen Bereich gerät, ohne dass neue TA gestartet wurden, da sich bereits aktive Transaktionen gegenseitig blockieren.

Ein Modul für die Automatisierung des MPL findet sich heutzutage in fast jeder kommerziellen Datenbank (u. a. IBM DB2, Microsoft SQL-Server). Diese Tuning-Option ist somit, im Gegensatz zu den meisten anderen Möglichkeiten (siehe Kapitel 4 und 5), relativ weit entwickelt.

3.4 Multi-Programming-Level für verschiedene Anfrageklassen

In modernen Datenbanksystemen gibt es eine Vielzahl von verschiedenen Anfrage-Typen. Früher wurde nur eine Unterscheidung zwischen kurz dauernden Anfragen und lang laufenden Anfragen aus DSS (eng., „decision-support system“) getroffen. Aufgrund dieser einfachen Trennung war ein MPL für das gesamte System ausreichend. Dazu kommen jedoch bei aktuellen DBS z. B. neue Datentypen, wie Bilder und Videos oder komplexere Anfragen (z. B. mit Rekursion). Eine Unterteilung der Anfragen in mehrere verschiedene Klassen erscheint deshalb sinnvoll. Um eine schnelle Ausführung aller Anfragen zu ermöglichen, muss es aber auch unterschiedliche Leistungs-Ziele für die unterschiedlichen Klassen geben. D. h. die angestrebte Antwortzeit einer Anfrage darf nicht global, sondern muss pro Klasse definiert werden. Auf-

grund dessen sollte auch ein eigener MPL für jede Klasse eingeführt werden [BMC+94].

Die Ausgangsposition ist eine Menge von Anfrageklassen und, für jede Klasse ein Antwortzeit-Ziel, also eine obere Grenze für die Ausführungszeit einer Anfrage der Klasse. Das Ziel ist, für jede Klasse einen MPL-Wert zu finden, der diese Beschränkung erfüllt. Die MPL-Wahl darf jedoch nicht vollkommen unabhängig betrachtet werden. Wenn z. B. ein MPL für eine Klasse so gewählt wird, dass das Ziel für diese Klasse erreicht wird, kann das dazu führen, dass andere Klassen ihr Ziel nicht mehr erreichen können. Der Grund dafür sind die beschränkten Ressourcen eines DBS. Wenn eine Klasse (aufgrund von sehr vielen aktiven Transaktionen) mehr Ressourcen in Anspruch nimmt, bleibt dementsprechend weniger Kapazität für den Rest der Anfragen übrig. Die Wahl des MPL einer Klasse sollte also keiner anderen Klasse die Möglichkeit nehmen, ihre angestrebte Antwortzeit zu erreichen.

Generell eignen sich zwei Verfahren zum Ermitteln geeigneter Parallelitätsgrade. Ein gemeinsames Berechnen der MPL aller Klassen, oder ein isoliertes Berechnen der MPL. Der erste Ansatz hat den Vorteil, dass die angesprochenen Abhängigkeiten zwischen den verschiedenen Klassen mit in die Berechnung einfließen. Beim zweiten Ansatz erfolgt jede MPL-Anpassung weitgehend isoliert. Der Vorteil ist vor allem die einfachere Berechnung. Deshalb kann auch viel schneller auf mögliche Änderungen in einer Klasse oder im System reagiert werden. Natürlich bleiben Wechselwirkungen mit anderen Klassen unbeachtet. Der zweite Ansatz hat sich aber, aufgrund seiner größeren Vorteile, als besser erwiesen.

4. Indizes

Eine weitere, für die Leistung des Datenbanksystems wichtige Einstellung, ist die Menge an Indizes die im System vorhanden sind. Jeder Index unterstützt und beschleunigt den Zugriff auf eine Tabelle. Ein Index selbst umfasst eine oder mehrere Spalten der beteiligten Tabelle. Im einfachsten Fall gestattet ein Index einen sequentiellen Zugriff auf alle Werte einer Spalte. So können zum Beispiel Bereichsanfragen (SELECT ... FROM ... WHERE x > 10 AND x < 20) durch Indizes besonders effizient ausgeführt werden. Bei mehrdimensionalen Indizes werden auf analoge Weise auch Anfragen über mehrere Tabellen (eng., „joins“) beschleunigt.

Bisher war es oft eine der Hauptaufgaben des DBA eine für das System und die Umgebung geeignete Menge von Indizes zu ermitteln. Diese Aufgabe ist sehr aufwändig da es notwendig ist, einen Index „von Hand“ zu erstellen und durch Testanfragen oder den im normalen Datenbankbetrieb anfallenden Anfragen auf seine Tauglichkeit zu testen. Falls sich herausstellt, dass der Index eine Verbesserung der Systemleistung nach sich zieht, wird er dauerhaft übernommen. Erfahrene Datenbankadministratoren konnten aufgrund ihres Fachwissens meist noch relativ gute Ergebnisse erzielen. Dennoch ist es notwendig dieses Problem zu automatisieren, um dauerhaft und konstant gute

Ergebnisse zu erzielen. Um zu Beginn eine einigermaßen akzeptable Leistung zu erzielen ist eine erste Vorgabe der heutigen DBS deshalb, dass zumindest auf dem Primärschlüssel und meist auch auf Fremdschlüsseln ein Index angelegt werden muss bzw. automatisch angelegt wird. Heutige Systeme versuchen dieses Problem weiter zu vereinfachen und stellen Programme bereit, die eine geeignete Menge an Indizes vorschlagen. Solche Anwendungen werden anhand von den Datenbanksystemen IBM DB2 (Kapitel 4.1) und Microsoft SQL-Server (Kapitel 4.2) vorgestellt. Allerdings ist bis heute kein DBS in der Lage die Verwaltung der Indizes völlig autonom durchzuführen. Auf diesem Gebiet des Tunings ist also, im Gegensatz zum MPL (Kapitel 3), noch viel Platz für Verbesserungen gegeben.

Ein entscheidender Anteil an der Leistung des Systems hat – wie oben beschrieben – die Auswahl einer geeigneten Menge von Indizes. Auch wenn es verlockend erscheint, kann nicht auf allen Spalten ein Index angelegt werden. Dies würde zum einen den Speicherplatz und die Antwortzeit der Systeme zu stark belasten, da jeder Index einen gewissen Platz beansprucht und ständig gewartet werden muss. Zum anderen sind es i. A. exponentiell viele [VZZ+00] , wie folgende Formel vedeutlicht (*n:* Anzahl der Spalten der Tabelle, *k:* Anzahl der im Index vorhandenen Spalten):

$$\sum \left(\frac{n!}{(n-k)!}\right)$$

Eine weitere Eigenschaft der Indizes ist, dass sich diese zum einen gegenseitig (siehe Kapitel 4.1 und 4.2) und zum anderen auch andere Design-Entscheidungen wie materalisierte Sichten (siehe Kapitel 5) beeinflussen. Falls es zum Beispiel eine Tabelle mit den Spalten A und B gibt, so kann ein vorhandener Index auf Spalte A, einen Index über A und B positiv beeinflussen. Bei materalisierten Sichten können (und sollten) auf den angelegten Sichten ebenfalls Indizes erzeugt werden. Die Einflüsse der Indizes auf materialisierte Sichten (und umgekehrt) werden in Kapitel 5 genauer behandelt.

4.1 Index-Selektion in DB2

In dem Datenbanksystem DB2 von IBM ist ab der Version 6.1 der DB2 Advisor [VZZ+00] im Einsatz. Dieser hat die Aufgabe eine geeignete Menge von Indizes auszuwählen. Ab Version 8.2 ist der DB2 Advisor im DB2 Design Advisor [ZRL+04] enthalten, welcher sich nicht ausschließlich um Indizes, sondern auch um andere wichtige Design-Entscheidungen (wie z. B. materialisierte Sichten) kümmert. Der Design Advisor wird in Kapitel 5 näher betrachtet.

Bei IBM wurde, um den DB2 Advisor zu realisieren, hauptsächlich der bereits bestehende Anfrageplan-Optimierer [SLV+01] erweitert. Der Optimierer berechnet den kostengünstigsten Anfrageplan für eine SQL-Anfrage. Ein im System vorhandener Index kann die Kosten für einen Anfrageplan erheblich

senken. Der Optimierer kann also beurteilen, ob ein vorhandener Index eine Verbesserung für einen Anfrageplan bringt. Die Erweiterung besteht nun darin, den Optimierer selbst entscheiden zu lassen, welche Indizes eine Verbesserung erzielen. Diese Indizes werden dann dauerhaft in das System übernommen. Um während der Auswahl keine echten Indizes anlegen zu müssen, wird dem Optimierer eine Menge von virtuellen Indizes übergeben, aus denen er dann die zu installierenden Indizes auswählt. Das Vorgehen des DB2 Advisor gliedert sich in die folgenden vier Schritte (siehe Bild 3).

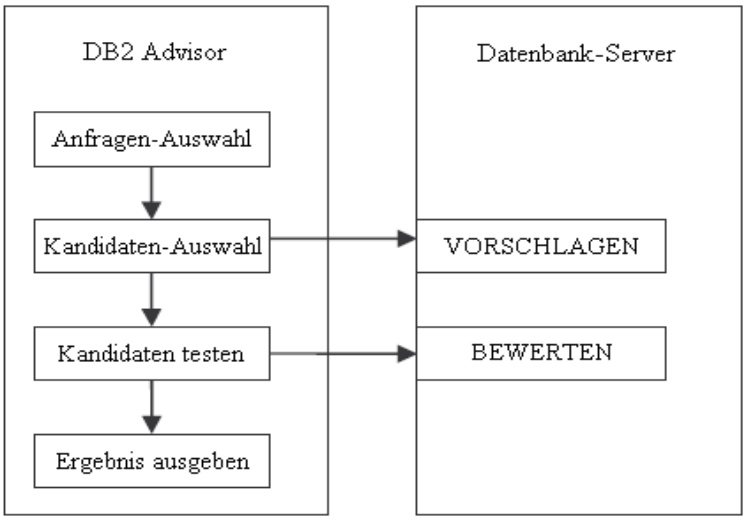

Bild 3. *Allgemeine Architektur des DB2 Advisor [DVZZ+00].*

Auswahl der zu optimierenden SQL-Anfragen. Ausgangspunkt ist die Angabe einer Menge von SQL-Anfragen. Diese Menge kann entweder aus gesammelten Anfragen des Optimierers (zum Beispiel der vergangenen Tage) oder aus selbst gewählten Anfragen bestehen. Im weiteren Verlauf der Index-Auswahl ist es nun das Ziel, eine Menge von Indizes zu finden, die eine möglichst kostengünstige, also schnelle, Ausführung für die gewählte Menge von Anfragen zu ermöglichen.
Kandidaten-Indizes bestimmen. Die optimale Lösung wäre es, jeden möglichen Index als virtuellen Index an den Optimierer zu übergeben. So könnte garantiert werden, dass die optimale Menge an Indizes ausgewählt wird. Dies ist jedoch aus den oben genannten Gründen nicht realisierbar. Deshalb kann nur eine gewisse Teilmenge an den Optimierer übergeben werden. Für die Kandidaten-Auswahl wird der Server in einem speziellen Modus (hier: VORSCHLAGEN) angesprochen, der es ermöglicht eine Anfrage zu kompilieren jedoch nicht auszuführen. So kann ein Anfrageplan berechnet werden, ohne

den Datenbankserver zu belasten. Um die Kandidaten-Indizes auszuwählen, wird nun jede Anfrage kompiliert und die besten Indizes in die Menge der Kandidaten aufgenommen.

Kandidaten-Indizes testen. Nachdem nun eine Menge von SQL-Anfragen und eine Menge von Kandidaten-Indizes bestimmt wurde, wird in diesem Schritt die beste Teilmenge von Indizes ausgewählt. Dazu wird ein Anfrageplan für jede Anfrage berechnet, wobei die Kandidaten-Indizes dem Optimierer als virtuelle Indizes übergeben werden. Dabei wird der Datenbank-Server im BEWERTEN-Modus benutzt, welcher es erlaubt die virtuellen Indizes so zu betrachten, als seien diese bereits im DBS vorhanden. Die Kandidaten-Indizes, die bei der Optimierung im billigsten Anfrageplan benutzt wurden, werden dann dem DBA vorgeschlagen.

Ergebnis ausgeben. Die im vorherigen Schritt bestimmte Teilmenge wird dem Datenbankadministrator als Ergebnis zurückgegeben.

Der DB2 Advisor ist also ein Hilfsprogramm für den Datenbankadministrator um eine möglichst optimale Menge an Indizes zu erhalten. Eine automatische Anpassung während der Laufzeit ist jedoch (noch) nicht möglich. Der DB2 Advisor muss jedesmal „von Hand“ vom DBA gestartet und bedient werden. Auch der Nachfolger DB2 Design Advisor (siehe Kapitel 5) schließt diese Lücke nicht.

4.2 Index-Selektion im Microsoft SQL-Server

Auch im SQL-Server von Microsoft ist ein Modul (der AutoAdmin) integriert, welches es den Datenbankadministratoren erleichtern soll, eine geeignete Menge an Indizes zu finden. Analog zu IBM existierte auch bei Microsoft zu Beginn nur die Unterstützung für Indizes (seit Version 7.0 [ChNa97, ChNa98]), ab dem SQL-Server 2000 wird auch die Auswahl von materialisierten Sichten (siehe Kapitel 5) und deren Wechselwirkungen mit Indizes betrachtet [AgCN00].

Eine Übersicht über die Funktionsweise der Index-Auswahl findet sich in Bild 4. Der Vorgang lässt sich in drei Teilschritte unterteilen, die alle unabhängig voneinander implementiert sind. Wie in Bild 4 zu sehen ist, werden die drei Teilprogramme zyklisch ausgeführt. Im ersten Durchlauf werden nur Indizes über einer Spalte betrachtet, dann über zwei Spalten, usw. Um eine Endlosschleife zu verhindern, wurde eine Grenze für die Anzahl der Indizes festgelegt, die vom Programm vorgeschlagen werden sollen. D. h. wenn eine bestimmte Anzahl an sinnvollen Indizes gefunden wurde, wird das Programm beendet. Auf diese Weise ist gesichert, dass einfache, also Indizes über wenigen Spalten, vor komplexen Indizes ausgewählt werden. Die Funktionsweise der einzelnen Teilprogramme wird in den folgenden Abschnitten erklärt.

Kandidaten-Auswahl. Die Eingabe für den ersten Schritt ist eine Menge von SQL-Anfragen. Die Aufgabe besteht darin zu jeder einzelnen Anfrage der übergebenen Menge die Indizes zu bestimmen, welche für die Ausführung der Anfrage von Vorteil sind. Dazu ruft sich das Programm selbst auf, diesmal jedoch nur mit der gerade betrachteten, einzelnen Anfrage als Parameter.

Wenn das Programm mit nur einer Anfrage gestartet wird, muss dennoch eine initiale Menge von Kandidaten (für den ersten Durchlauf des Zyklus) bestimmt werden. Zu diesem Zweck wird die Menge an indexierbaren Spalten der Relationen der Anfrage bestimmt. In den weiteren Iterationen hat dieses Teilprogramm keine Aufgaben mehr, da es die Kandidaten direkt vom letzten Schritt (Mehrspalten-Indizes auswählen) bekommt. Ist für jede einzelne Anfrage eine geeignete Menge an Indizes (Konfiguration) gefunden, können die Kandidaten-Indizes für die gesamte Menge an den nächsten Schritt übergeben werden. Diese bildet sich aus den Vereinigungen der ermittelten Konfigurationen der einzelnen Anfragen.

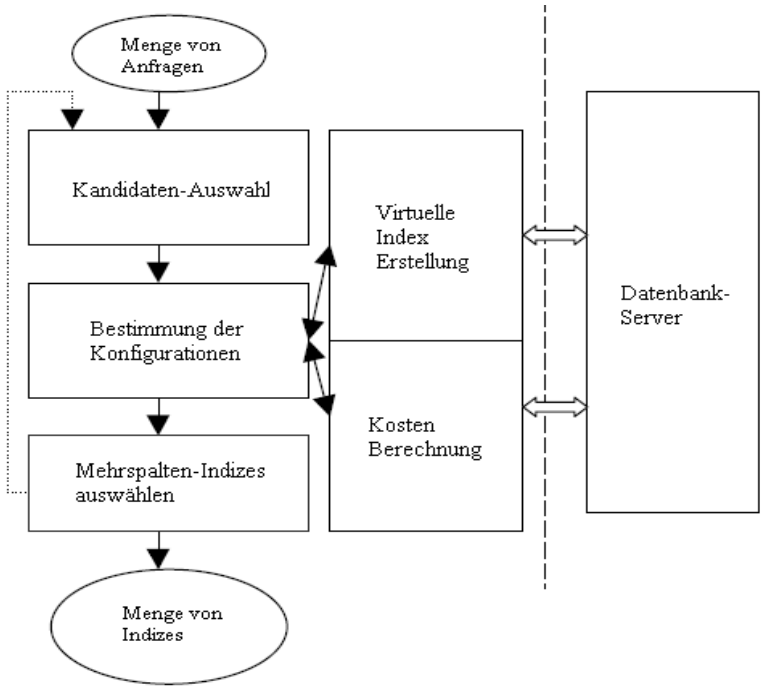

Bild 4. *Allgemeine Architektur der Index-Auswahl im Microsoft SQL-Server. [ChNa97].*

Bestimmung der Konfigurationen. Nachdem eine Menge von Kandidaten gefunden wurde müssen hieraus diejenigen extrahiert werden, welche eine Verbesserung der Systemleistung nach sich ziehen. Dazu wurde im DBS ein Algorithmus mit Greedy-Eigenschaften implementiert. Der Algorithmus ermittelt zuerst eine verhältnismäßig kleine Menge von Indizes als Basis, um diese dann solange zu erweitern, bis entweder die maximale Anzahl an Indi-

zes erreicht wurde oder keine weitere Kostensenkung möglich ist. Um festzustellen, ob ein Kandidat die Leistung verbessert, muss der Optimierer des SQL-Servers den Anfrageplan der einzelnen Anfragen berechnen (Kosten-Berechnung). Voraussetzung für dieses Verfahren ist jedoch, dass die virtuellen Kandidaten-Indizes dem System bekannt gemacht werden (Virtuelle Index-Erstellung). Der Optimierer behandelt die Kandidaten dann so, als wären diese physisch im DBS vorhanden. Senkt nun ein Index die Kosten von mindestens einer Anfrage, wird dieser der Basis hinzugefügt. Die Menge der als sinnvoll betrachteten Indizes wird wieder als Konfiguration bezeichnet.

Mehrspalten-Indizes auswählen. Wie oben erwähnt, werden im ersten Durchlauf der Schleife nur Indizes über einer Spalte betrachtet. Um für die nächste Iteration eine Menge von Kandidaten-Indizes zur Verfügung stellen zu können, werden in diesem Schritt alle vernünftigen Indizes ermittelt, welche über eine Spalte mehr definiert sind, als die im letzten Schritt betrachten Kandidaten. Wenn z. B. im letzten Schritt einspaltige Indizes berechnet wurden, müssen jetzt solche Indizes gefunden werden, die über zwei Spalten definiert sind. Um nicht zu viele Kandidaten zu erhalten, wird ein mehrspaltiger Index jedoch nur als sinnvoll angesehen, wenn über mindestens einer seiner Spalten ein „nieder klassigerer“ Index vorhanden ist. D. h., ein Index über zwei Spalten A und B kann nur ausgewählt werden, wenn in der Konfiguration ein Index über A oder B vorhanden ist. Die ermittelten „höher klassigen“ Indizes bilden die Eingabe für die Kandidaten-Auswahl im nächsten Zyklus-Schritt.

Wie bei IBM, ist dies nur ein Hilfsprogramm für den Datenbankadministrator. Eine Anpassung der Indizes während der Laufzeit wird auch bei Microsoft nicht unterstützt, ist aber Teil aktueller Forschungen.

5. Materialisierte Sichten

Analog zu Indizes (Kapitel 4), sind materialisierte Sichten dazu geeignet, Anfragepläne an ein Datenbanksystem zu optimieren. Sie sind ebenfalls zusätzliche physische Strukturen [AgCN00]. Indizes jedoch beschränken sich bei ihrer Definition auf eine Tabelle und lassen nur einfache Anfragen (z. B. ohne Selektion und GROUP BY) zu. Eine materialisierte Sicht kann jedoch auch aus mehreren Relationen (JOIN) aufgebaut sein. Weiterhin sind auch komplexere Definitionen, z. B. mit Selektionen möglich. Ein Index kann somit als Spezialfall einer materialisierten Sicht angesehen werden. Theoretisch ist sogar die gesamte SQL-Syntax in materialisierten Sichten nutzbar. Von den meisten zur Zeit im Betrieb befindlichen Systemen, wird jedoch nur eine Teilmenge davon unterstützt. Eine weitere besondere Eigenschaft einer Sicht ist, dass sie wie eine zusätzliche Relation im DBS genutzt werden kann. Somit sind also auch Sichten auf Sichten möglich.

Wie in Kapitel 4 beschrieben, ist die Auswahl von Indizes, obwohl sie eine sehr einfache Struktur haben, schon eine schwere Aufgabe. Es ist aufgrund der oben genannten Eigenschaften daher umso schwerer, materialisierte

Sichten zu finden, die eine Kostenreduzierung für Anfragen zur Folge haben. Ein weiteres Problem ergibt sich daraus, dass es Beziehungen zwischen Indizes und materialisierten Sichten gibt [AgCN00, ZRL+04]. So kann eine materialisierte Sicht einen Index (und umgekehrt) überflüssig machen. Oder die Existenz einer materialisierten Sicht erfordert das Anlegen eines Index darauf, um die Sicht effektiv genug zu machen. Wenn z. B. ein Index auf einer Spalte A existiert und eine materalisierte Sicht über den Spalten A und B angelegt werden soll, ist der ursprüngliche Index nicht mehr nötig, es sei denn es handelt sich um einen Index mit Clusterbildung (siehe auch Kapitel 6.2).

Moderne Programme in DBSn sollten daher nicht materialisierte Sichten und Indizes isoliert betrachten sondern ein Verfahren bereitstellen, welches es ermöglicht die bestehenden Beziehungen auszunutzen. Diese Beobachtung wurde auch von den beiden großen DBS gemacht, weshalb im Microsoft SQL-Server (Kapitel 5.1) und in DB2 von IBM (Kapitel 5.2) Programme integriert wurden, die eine gemeinsame Auswahl von Indizes und materialisierten Sichten unterstützen. Auch diese Programme sind allerdings nur als Hilfe für Datenbankadministratoren gedacht. Die endgültige Wahl bleibt nach wie vor dem DBA überlassen. Aufgrund der wachsenden Komplexität und Qualität der berechneten Ergebnisse, ist es jedoch meist ratsam die vorgeschlagenen Indizes und Sichten genauso ins System zu übernehmen.

5.1 Microsoft SQL-Server und materalisierte Sichten

Seit dem SQL-Server 2000, stellt das Datenbanksystem von Microsoft ein Programm für Datenbankadministratoren bereit, welches die Wahl von geeigneten Indizes und materialisierten Sichten vereinfacht. Wichtigste Neuerung des Programms ist, dass Indizes und materialisierte Sichten nicht isoliert betrachtet werden. Somit können die bestehenden Abhängigkeiten zwischen Indizes und Sichten in die Berechnung mit einbezogen werden. Das Verfahren ist eine Erweiterung des in Kapitel 4.2 vorgestellten Verfahrens zur alleinigen Berechnung von Indizes. Eine Übersicht findet sich in Bild 5. Die einzelnen Teilschritte werden in den nachfolgenden Abschnitten behandelt.

Kandidaten-Auswahl. In der Vorgänger-Version des Tuning-Moduls war es die Aufgabe dieses Schrittes, die Gesamtmenge von theoretisch möglichen Indizes auf eine sinnvolle Menge von Kandidaten zu verkleinern. Wie eine sinnvolle Menge von Kandidaten-Indizes berechnet werden kann, wurde in Kapitel 4.2 ausführlich beschrieben. Jetzt müssen zusätzlich noch Kandidaten für materialisierte Sichten, sowie Kandidaten für Indizes auf Sichten gefunden werden.

Die Wahl von Sichten-Kandidaten gestaltet sich jedoch als noch kniffliger, da aufgrund der erweiterten Syntax mehr materialisierte Sichten als Indizes gebildet werden können. Die einfachste Möglichkeit wäre es, für jede Anfrage, aus der gegebenen Menge, genau eine Sicht anzulegen. Dies ist jedoch meist nicht möglich, da die DBS nicht die volle SQL-Syntax zur Bildung von materialisierten Sichten zulassen. Weiterhin haben Sichten die zu sehr an eine einzelne Anfrage angelehnt sind, zwar eine große Kostenreduktion für diese

Anfrage zur Folge, für alle anderen Anfragen jedoch nicht. Es muss daher nach alternativen Sichten gesucht werden, die für eine möglichst große Menge von Anfragen Verbesserungen bringen.

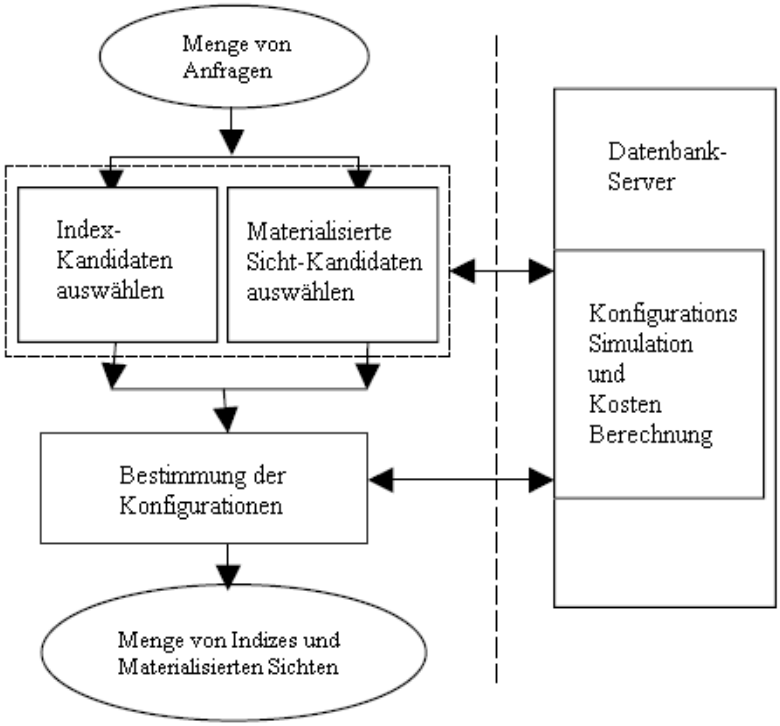

Bild 5. *Allgemeine Architektur der Index- und Sichten-Auswahl im Microsoft SQL-Server [AgCN00].*

Der erste Schritt muss demnach sein, die ursprünglich extrem große Menge an möglichen materialisierten Sichten auf eine akzeptable Menge zu reduzieren. Dazu werden geeignete Teilmengen der Basistabellen (Tabellen, die in der Menge von Anfragen referenziert werden) ermittelt. Im weiteren Selektions-Verfahren wird dann nur noch nach Sichten auf den gefundenen Teilmengen gesucht.

Im zweiten Schritt müssen aus den ermittelten Mengen von Basistabellen materialisierte Sichten gefunden werden. Dazu wird zuerst für jede Teilmenge eine eigene Sicht definiert. Die gefundene Menge an materialisierten Sichten ist die initiale Menge der Kandidaten für die weiteren Schritte. Um eine erste Vorauswahl zu treffen wird eine Sicht wieder aus der Menge der Kandidaten gelöscht, wenn sie nicht im besten Anfrageplan mindestens einer Anfrage zu

finden ist.
Damit eine Sicht für möglichst viele Anfragen eine Verbesserung bringt, werden im dritten Schritt verschiedene, zuvor ermittelte Sichten miteinander gemischt. Diese ersetzen die bisherigen Kandidaten nicht. Die Menge der Kandidaten wird jedoch um die gemischten Sichten erweitert. Damit die neuen Sichten als sinnvoll erachtet werden können, müssen sie bestimmte Eigenschaften erfüllen. Zum einen müssen sie (im Anfrageplan) von allen Anfragen benutzt werden können, die auch ihre Eltern-Sichten (Sichten aus denen die gemischte Sicht entstanden ist) benutzt haben. Zum anderen sollten die Anfragepläne nicht viel teurer sein, als die Pläne mit den ursprünglichen Sichten.

Mithilfe der drei vorgestellten Schritte, wird eine qualitativ hochwertige Menge von Kandidaten ermittelt.

Bestimmung der Konfigurationen. Dieser Schritt hat den Sinn aus der Menge der Kandidaten eine, für die Menge der Anfragen optimale Konfiguration zu finden. Die Vorgehensweise ist jedoch identisch mit der bei der isolierten Wahl von Indizes. Für eine detaillierte Beschreibung dieses Schrittes, wird deshalb auf das vorige Kapitel verwiesen.

5.2 IBM DB2 und materalisierte Sichten

Auch bei IBM hat man erkannt, dass es notwendig ist das physische Datenbankdesign nicht allein dem Administrator zu überlassen. In dem Datenbanksystem DB2 ist man ab der Version 8.2 sogar noch einen Schritt weiter gegangen als bei Microsoft. So unterstützt der DB2 Design Advisor nicht nur die automatische Wahl von Indizes und materialisierten Sichten, sondern auch noch die Partitionierung und das Problem des multidemensionalen Clustering von Tabellen (siehe Kapitel 6.2).

Um die bestehenden Abhängigkeiten zwischen Indizes und materialisierten Sichten (siehe oben) nicht außer Acht zu lassen, wurde auch bei IBM ein Algorithmus entwickelt, der die vorhandenen Wechselwirkungen mit in Betracht zieht. Dazu wurde der im DBS vorhandene DB2 Advisor (Kapitel 4.1) erweitert. Die allgemeine Vorgehensweise ist jedoch identisch mit dem Verfahren zur isolierten Index-Selektion (siehe Bild 3). Einzig die Berechnung der Kandidaten-Sichten beruht auf einem etwas anderen Verfahren [Zili04].

6. Weitere Self-Tuning Möglichkeiten

Neben den, in den vorangegangenen Kapiteln vorgestellten Möglichkeiten, ein Datenbanksystem zu automatisieren, gibt es noch viele weitere. Einige davon hängen direkt mit den bisher behandelten Themen zusammen. So ist z. B. das multidimensionale Clustering von Tabellen, welches in Kapitel 6.2 behandelt wird, verwandt mit der Selektion von Indizes (Kapitel 4) und materialisierten Sichten (Kapitel 5). Weiterhin wird in Kapitel 6.1 die automatisierte Verteilung der Daten eines DBS behandelt.

6.1 Dynamische Daten-Verteilung

Bei fast allen DBS ist es heutzutage (aufgrund der Größe der Datenbank) notwendig, die Daten auf verschiedene Festplatten zu verteilen. Die Verteilung bietet aber auch entscheidende Vorteile. So ist ein Verbund von mehreren Platten wesentlich schneller, da alle Festplatten parallel angesprochen werden können. Auch sind die Kosten für mehrere kleine Speicher meist niedriger, als die Kosten für einen großen Datenträger. Natürlich bringt die Verteilung auch Probleme mit sich. Eines davon ist, dass aufgrund einer unglücklichen oder ungeschickten Verteilung, einige Festplatten stärker in Anspruch genommen werden, als andere. Damit das System jedoch eine hohe Leistung erreichen kann, sollten alle Speicherelemente möglichst ausgeglichen angesprochen werden. Doch die Auslastung einer Festplatte ist nicht immer gleich. Denn die Art der Transaktionen und deren Typ ändert sich ständig und somit kann sich die Auslastung der Platten ebenfalls verändern. Das Ziel muss demnach sein, zu jeder Zeit eine annähernd gleiche Auslastung aller Datenträger zu garantieren [WMH+02].

Wie eben beschrieben, gibt es Daten, die (zu einem bestimmten Zeitpunkt) hoch frequentiert sind und solche, die gar nicht oder nur sehr wenig benutzt werden. Um eine Charakterisierung der Datenbenutzung zu ermöglichen, wird zwischen sogenannten heißen (Daten die sehr oft genutzt werden) und kalten (solche die selten genutzt werden) Daten unterschieden. Dazu existiert eine Formel, die es ermöglicht, die Anzahl der Zugriffe pro Zeiteinheit zu berechnen:

$$heat = \frac{k}{(t_k - t_1)}$$

Die Formel bezieht sich auf die letzten k Zugriffe (t_1 bis t_k) auf ein Objekt. Je kleiner der Zeitabstand zwischen dem letzten und dem k-letzten Zugriff ist, desto heißer ist ein Datenobjekt. Die Hitze eines Datenträger kann jetzt durch die Addition der Hitze aller Daten berechnet werden. Da sich die Werte sehr einfach berechnen lassen, ist eine ständige Überwachung möglich, um eine gleichmäßige Auslastung der Festplatten zu garantieren.

Übersteigt die Differenz der heißesten und der kältesten Platte einen festgelegten Wert, müssen Daten zwischen den Festplatten verschoben werden. Dazu werden Datenobjekte der heißesten Platte ausgewählt, um diese auf den kältesten Datenträger zu verschieben. In einigen Fällen kann eine solche Verschiebung zu unüberwindbaren Problemen führen. Ist z. B. auf der kältesten Festplatte kein Platz mehr frei, muss eine alternative, ebenfalls möglichst kalte Platte gefunden werden. Ein weiteres, nicht zu unterschätzendes Problem ist der zusätzliche Aufwand, der beim Verschieben von Daten ensteht. So kann es vorkommen, dass die Verschiebung eines Objekts die heiße Platte (aufgrund noch höherer Wartezeiten) endgültig unbrauchbar macht. Es muss daher vor der Auswahl der beteiligten Platten und Datenobjekte sehr

gründlich überprüft werden, ob keine Probleme beim Verschieben auftreten. Ist eine akzeptable Kombination gefunden, bleibt als letzter Schritt das Bewegen der Daten zwischen den Festplatten.

Das vorgestellte Verfahren ermöglicht es, ohne großen rechnerischen Aufwand die Auslastung aller benutzten Speicher eines DBS (in etwa) auf dem gleichen Level zu halten. Es kann somit, zumindest im speichertechnischen Bereich, eine hohe Leistung des Systems garantiert werden.

6.2 Multidemensionales Clustering von Tabellen

Nach Indizes (Kapitel 4) und materialisierten Sichten (Kapitel 5) ist das multidimensionale Clustering von Relationen eine weitere wichtige Design-Baustelle für Datenbankadministratoren. Die einzige Möglichkeit Daten einer Relation mit den bisher vorgestellten Mitteln physisch zu ordnen, ist das Anlegen eines Index mit Clusterbildung. Die Tupel einer Tabelle werden bei einem solchen Index aufgrund der Werte einer einzigen Spalte auf dem Datenträger angeordnet. Auf einer Tabelle kann nur ein Index mit Clusterbildung angelegt werden, weshalb alle weiteren Indizes auf dieser Relation auch als sekundäre Indizes bezeichnet werden. Bei komplexeren Anfragen, z. B. mit GROUP-BY oder mehrdimensionalen Bereichsanfragen, wie sie häufig bei DSS oder OLAP-Systemen (eng., „online analytical processing“) vorkommen, ist eine solche Ordnung der Daten jedoch nicht optimal. Besser wäre eine gemeinsame Speicherung von Tupeln, welche z. B. in den mehrdimensionalen Bereichsanfragen mit hoher Wahrscheinlichkeit gemeinsam angefragt werden. Das multidimensionale Clustering ermöglicht genau diese Verteilung der Daten. Bei einem solchen Clustering können beliebig viele Dimensionen (maximal so viele wie die Tabelle Spalten hat) mit frei wählbaren Granularitäten definiert werden. Die daraus resultierenden Blöcke werden gemeinsam auf dem Datenträger gespeichert. Um die Performance des Clustering weiter zu verbessern, wird danach auf jedem Block ein Index angelegt. Ist eine Tabelle mit einem multidimensionalen Cluster versehen ist es somit besonders DSS und OLAP-Systemen möglich, kostengünstige Anfragepläne auf der Tabelle zu berechnen.

Die Wahl von geeigneten Clustern ist jedoch wiederum kein einfaches Unterfangen da, ähnlich wie in Kapitel 5, Abhängigkeiten zwischen multidimensionalen Clustern und Indizes bzw. materialisierten Sichten existieren. Die gemeinsame Berechnung aller drei Design-Bereiche ist deshalb von Vorteil [ZRL+04]. Die allgemeine Vorgehensweise bleibt identisch mit der Selektion von Indizes (siehe Kapitel 4.1, Bild 3). Da jedoch bei einem Cluster nicht nur eine Spalte, sondern gleich mehrere betrachtet werden müssen, vergrößert sich der Suchraum noch einmal drastisch. Das Verfahren zur Auswahl geeigneter multidimensionaler Clusterkandidaten gestaltet sich daher sehr kompliziert und wird deshalb nur sehr kurz vorgestellt. Eine detaillierte Beschreibung des kompletten Verfahrens findet sich in [LiBh04]. Im ersten Schritt werden geeignete Dimensionen (d. h. Spalten aufgrund derer die Relation physisch verteilt werden soll) möglicher Cluster bestimmt und ihre Vor-

teile für die Menge der Anfragen geprüft. Dazu wird, wie bisher auch, der Optimierer mit virtuellen Clustern gefüttert, um deren Effizienz ermitteln zu können. Im zweiten Schritt werden für die gefundenen Dimensionen passende Granularitäten (Intervall der Werte eines Blocks einer Dimension im Cluster) der Verteilung gesucht. Die ermittelten Cluster bilden dann die Menge der Kandidaten. In Experimenten [LiBh04] wurde nachgewiesen, dass die Menge der mit diesem Verfahren gefundenen Cluster kostentechnisch vergleichbar ist mit den Vorschlägen erfahrener DBA.

7. Konklusion

Automatisierte Tuning-Möglichkeiten bekommen einen immer größeren Stellenwert in modernen Datenbanksystemen. Je größer und komplexer die Datenbanken werden, desto komplexer werden auch die Systemparameter zur Administration. Datenbankadministratoren sind immer öfter auf die Hilfe von Tuning-Programmen angewiesen.

Die in dieser Ausarbeitung vorgestellten Verfahren wurden dazu entwickelt, dem DBA die Arbeit komplett oder zumindest teilweise abzunehmen. Vor allem auf dem Gebiet des Multi-Programming-Level ist die Automatisierung schon weit fortgeschritten und das vorgestellte Verfahren wird in aktuellen DBS so oder so ähnlich bereits verwendet. Die behandelten Design-Automatisierungen (vor allem für Indizes und materialisierte Sichten) sind zur Zeit nur eine Unterstützung für den DBA. Die von den Programmen vorgeschlagenen Lösungen sind jedoch auf dem Niveau von erfahrenen Datenbankexperten. Eine vollständige Automatisierung existiert auf diesem Gebiet jedoch noch nicht und muss ein Ziel für die Forschung der nächsten Jahre sein.

Literaturverzeichnis

AgCN00 Agrawal, S.; Chaudhuri, S.; Narasayya, V.: Automated Selection of Materialized Views and Indexes for SQL Databases, VLDB 2000

BROW00 Brown, D. H.: DB2 UDB vs. Oracle8i: Total Cost of Ownership, D. H. Brown Associates, Inc. December 2000

BMC+94 Brown, K. P.;Mehta, M.;Carey, M. J.;Livny, M.: Towards Automated Performance Tuning for Complex Workloads, VLDB Conf. 1994

ChNa97 Chaudhuri, S.; Narasayya, V.: An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server, VLDB 1997

ChNa98 Chaudhuri, S.; Narasayya, V.: AutoAdmin “What-If” Index Analysis Utility, ACM SIGMOD 1998

EPB+03 Elnaffar, S.; Powley, W.; Benoit, D.; Martin, P.: Today's DBMSs: How autonomic are they?, ACS 2003

Härd05 Härder, T.: DBMS Architecture – New Challenges Ahead. Datenbank-Spektrum 5. Jahrgang, Heft 13, Mai 2005

LiBh04 Lightstone, S.; Bhattacharjee, B.: Automated design of Multidimensional Clustering tables for relational databases, VLDB 2004

LoLi02 Lohman, G. M.; Lightstone, S.: SMART: Making DB2 (More) Autonomic, VLDB 2002

SHW+06 Schröder, B.; Harchol-Balter M.; Wiermann, A.; Iyengar, A.; Nahum, E.: How to determine a good multi-programming level for external scheduling, ICDCS 2006

SLV+01 Stillger, M.; Lohman, G. M.; Markl, V.; Kandil, M.: LEO: DB2’s LEarning Optimizer, VLDB Conf. 2001

VZZ+00 Valentin, G.; Zuliani, M.; Zilio, D. C.; Lohman, G.; Skelley, A.: DB2 Advisor: An optimizer smart enough to recommend its own indexes, Proceedings of the ICDE Conference, 2000

WHM+94 Weikum, G.; Hasse C.; Mönkeberg, A.; Zabback, P.: The COMFORT Automatic Tuning Project, Information Systems 19(5), 1994

WMH+02 Weikum, G.; Mönkeberg, A.; Hasse, C; Zabback, P.: Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering, VLDB 2002

Zili04 Zilio, D. C. et al: Recommending Materialized Views and Indexes with IBM’s DB2 Design Advisor, International Conference on Autonomic Computing 2004

ZRL+04 Zilio, D. C.; Rao, J.; Lighstone, S.; Lohmann, G.; Storm, A.; Garcia-Arellano, C.; Fadden, S.: DB2 Design Advisor: Integrated Automatic Physical Database Design, VLDB 2004