in: Informatik-Fachberichte 264, Springer, 1990.

Realisierungskonzepte für die parallele Bearbeitung von Anfragen auf komplexen Objekten

Harald Schöning

Sonderforschungsbereich 124, Teilprojekt D2
Universität Kaiserslautern
Postfach 3049
6750 Kaiserslautern

Überblick

Datenbanksysteme für Nicht-Standard-Anwendungsbereiche müssen besonderen Anforderungscharakteristika gerecht werden: Es ist nicht mit einer extrem hohen Anzahl paralleler Transaktionen zu rechnen, aber die auftretenden Anfragen sind sehr komplex und müssen möglichst schnell beantwortet werden. Dies soll durch Ausnutzung von Parallelität unterstützt werden. Anhand von vier verschiedenen Arten von Parallelität bei der Bearbeitung einer Anfrage wird ein Verarbeitungsmodell diskutiert, das bei geringem Zusatzaufwand eine hohe Parallelität verspricht. Das hier vorgestellte Remote-Cooperation-System unterstützt dieses Verarbeitungsmodell und erlaubt zudem die Kopplung heterogener Architekturen und Betriebssysteme. Es handelt sich dabei um ein von Rechnerarchitekturen und Betriebssystemen unabhängiges System, das es ermöglicht, ein Datenbanksystem über mehrere heterogene Rechner zu verteilen, ohne daß die Programme des Datenbanksystems Kenntnis über die zugrundeliegende Architektur haben müssen um miteinander zu kommunizieren. Dadurch wird das Datenbanksystem portabel, und es ist möglich, die Zuordnung von Datenbanksystemkomponenten zu Prozessoren flexibel zu gestalten, auch wenn die Prozessoren nicht homogen sind. Die Implementierung dieses Remote-Cooperation-System sollte natürlich unterstützende Betriebssystemkonzepte möglichst umfassend ausnutzen

1. Einleitung

Datenbanksysteme haben sich zur Unterstützung konventioneller Anwendungen (wie Bankwesen, Lohnbuchhaltung etc.) als geeignete Werkzeuge erwiesen, die eine effiziente Bearbeitung der anfallenden Aufgaben ermöglichen. Für die sogenannten Nicht-Standard-Anwendungsbereiche, wie z.B. CAD oder VLSI-Design, weisen die herkömmlichen Datenbanksysteme eine Reihe von Mängeln auf, die durch die andersartige Verarbeitungscharakteristik dieser Bereiche aufgedeckt werden. Typischerweise werden hier komplexe Objekte verarbeitet, die mit den "klassischen" Datenmodellen nur schwer modellierbar sind. Eine allgemein akzeptierte Architektur für die Verarbeitung in Nicht-Standard-Bereichen ist das Workstation-Server-Modell. Das Anwendungsprogramm (z.B. ein CAD-System) läuft auf einer Workstation mit graphikfähigem Terminal und kommuniziert mit dem Datenbanksystem, das auf einem sogenannten Server arbeitet und von vielen verschiedenen Anwendungen genutzt wird, über "Checkout"- und "Checkin"-Aufrufe. Zu Beginn der Verarbeitung, die im allgemeinen "lange" (Stunden, Tage) dauert, werden Daten aus der Datenbank extrahiert ("Checkout"). Dies wird ggf. im Laufe der Bearbeitung wiederholt. Am Ende der Verarbeitung werden gegebenenfalls neue (veränderte) Daten wieder in die Datenbank eingebracht ("Checkin"). Jeder dieser im Vergleich zu konventionellen Anwendungen relativ seltenen Checkout- bzw. Checkin-Aufrufe dient der Übertragung großer und komplex strukturierter Datenmengen. Während der Bearbeitung eines solchen Aufrufes wartet der Benutzer des Anwendungsprogramm am Bildschirm. Daher ist es unabdingbar, die entsprechenden Antwortzeiten so kurz wie möglich zu halten: Antwortzeiten, die deutlich länger als wenige Sekunden sind, verhindern die Akzeptanz des Systems. Andererseits lassen sowohl die Komplexität der Checkout- und Checkin-Operationen als auch die Größe der Datenmengen einen hohen Bearbeitungsaufwand erwarten. Neben dem Einsatz eines geeigneten Datenmodells und den bekannten Techniken zur Anfrageoptimierung kann eine Parallelisierung der Auftragsbearbeitung zur Reduzierung der Antwortzeit beitragen [HHMM87].

Ein Checkin- oder Checkout-Auftrag besteht im allgemeinen aus einer oder mehreren Operationen des Datenmodells [LP83, HHMM87, Re88]. Um in jedem Fall eine Parallelverarbeitung zu erreichen, muß also die Verarbeitung innerhalb einer Datenmodelloperation parallelisiert werden. Natürlich ist es dazu notwendig, daß die zugrundeliegende Hardware-Architektur des Servers Parallelität ermöglicht. Dies kann auf verschiedene Arten geschehen:

- Herkömmliche Rechner haben oft einen eigenen Ein-/Ausgabeprozessor. Während eines Sekundärspeicherzugriffs kann also eine andere Aufgabe bearbeitet werden. Der Parallelitätsgrad ist hier jedoch gering.
- Ein Netz aus eigenständigen Rechnern (ein lose gekoppeltes System) ermöglicht prinzipiell einen Parallelitätsgrad, der nur durch die Zahl der verfügbaren Rechner begrenzt ist. Diese Architektur wird jedoch problematisch, wenn ein umfangreicher Datenaustausch zwischen den Rechnern stattfindet, da Kommunikation über ein externes Nachrichtenmedium (z.B. LAN) zeitaufwendig ist.
- Ein Rechner mit mehreren Prozessoren, die auf einen gemeinsamen Hauptspeicher zugreifen (eng gekoppeltes System) vermeidet die Kommunikationsproblematik, ist dafür aber im Wachstum (in der Zahl der Prozessoren) beschränkt [HSS89].

In diesem Beitrag werden wir uns nicht auf einen dieser drei Typen festlegen, sondern versuchen, Parallelisierungskonzepte auszuarbeiten, die für alle drei skizzierten Architekturen gleichermaßen anwendbar sind (Kapitel 3). Wir gehen allerdings davon aus, daß wir es nicht mit Spezialprozessoren zu tun haben. Ziel unserer Untersuchungen ist es, ein parallel arbeitendes Datenbanksystem zu erhalten, dessen Komponenten - ohne jegliche Änderung in ihrer Programmierung - auf Rechnern unterschiedlicher Hardware und unter unterschiedlichen Betriebssystemen ablaufen können. Voraussetzung für eine solche transparente Kooperation zwischen Datenbanksystem-Komponenten ist eine betriebssystemunabhängige Kommunikationskomponente (deren Implementierung natürlich nicht unabhängig von unterliegender Hardware und unterliegendem Betriebssystem sein kann). Zur Illustration dieser Ansätze verwenden wir das Nicht-Standard-Datenbanksystem PRIMA mit seinem Molekül-Atom-Datenmodell, das in Kapitel 2 kurz vorgestellt wird. Kapitel 4 präsentiert dann ein Verarbeitungsmodell, in dem diese Ansätze verwirklicht werden können. Kapitel 5 enthält eine kurze Beschreibung des Remote-Cooperation-Systems, das zum einen dieses Verarbeitungsmodell unterstützt, zum anderen aber auch die unterliegende Serverarchitektur weitgehend verbirgt, und so eine einfache Portierung zwischen den oben genannten Architekturtypen und deren gemischte Verwendung erlaubt. In Kapitel 6 wird gezeigt, wie man durch Anwendung dieses Systems aufwendige Prozeßwechsel vermeiden kann. Kapitel 7 faßt die vorgestellten Aspekte kurz zusammen.

2. Kurze Beschreibung von PRIMA

Zur Verdeutlichung der nachfolgenden Diskussion verschiedener Parallelisierungsansätze werden wir deren Anwendung auf das Datensystem des Nicht-Standard-Datenbanksystems PRIMA erläutern. PRIMA wurde als Datenbankkernsystem [HR85, Pa87] entworfen. Neben einem anwendungsunabhängigen Datenbanksystemkern, der ein allgemeines Datenmodell unterstützt und auf dem Server abläuft, gehören also noch anwendungsspezifische Modellabbildungsschichten zum Datenbanksystem, die auf Workstations eine der jeweiligen Anwendung angepaßte Schnittstelle zum Datenbanksystem bereitstellen. Im folgenden betrachten wir nur den Kern, da er diejenige Komponente ist, die Datenmodelloperationen ausführt. Der PRIMA-Kern implementiert das Molekül-Atom-Datenmodell (MAD-Modell), das als Erweiterung des Relationenmodells im Hinblick auf Komplexobjektverarbeitung angesehen werden kann [Mi88].

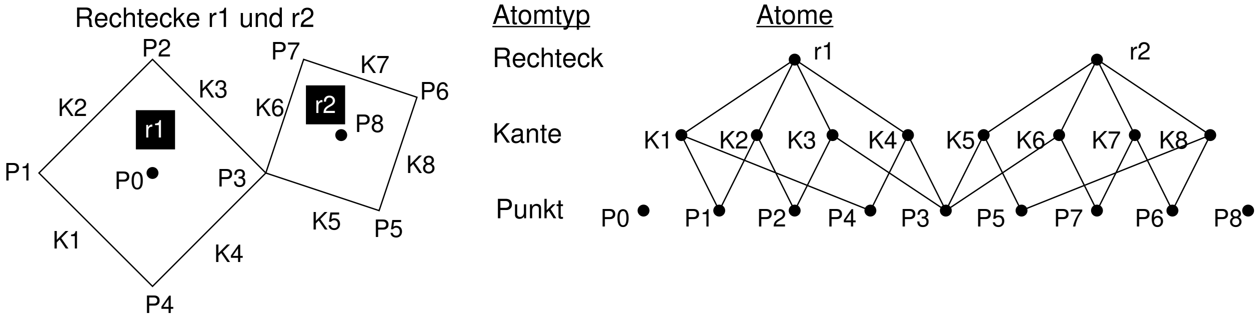
Bevor wir die Architektur von PRIMA skizzieren, müssen wir zur Untersuchung der Parallelisierungsmöglichkeiten innerhalb einer Operation des MAD-Modells zunächst einige seiner Grundzüge darstellen.

Das MAD-Modell erlaubt die dynamische Definition von komplexen Objekten (*Molekülen*) aus Grundbausteinen (*Atomen*). Wie ein Tupel im Relationenmodell besteht ein Atom aus Attributen verschiedener Datentypen und gehört zu genau einem Atomtyp. Jedes Atom besitzt einen systemvergebenen Primärschlüssel (Surrogat). Zwischen Atomtypen können ungerichtete Beziehungen definiert werden (die sogenannten Linktypen), die somit ein Atomtypnetzwerk aufbauen. Entsprechend dieser Linktypdefinition können Atome der betreffenden Typen untereinander durch bidirektionale, ungerichtete Links miteinander verbunden sein und so ein Atomnetzwerk bilden.

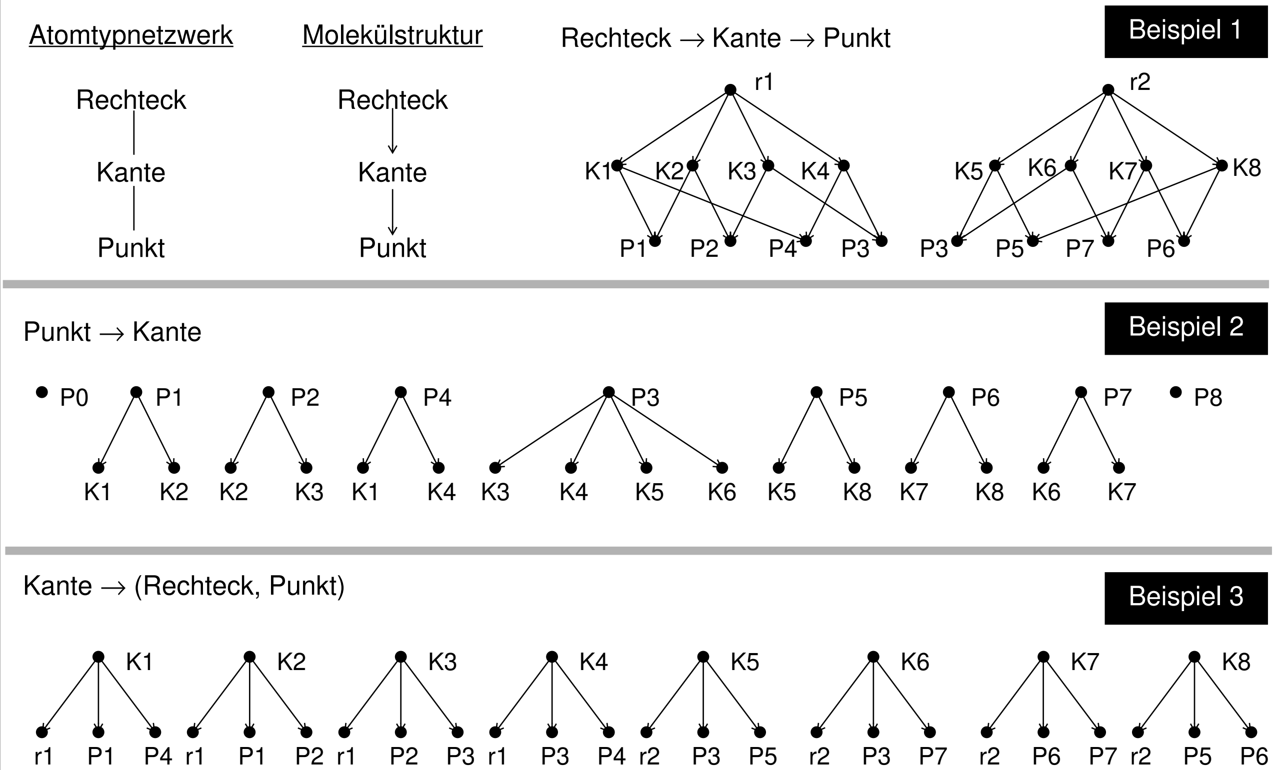
Ein Linktyp wird durch ein Paar von Attributen des speziellen Typs REFERENCE implementiert, eines in jedem der beteiligten Atomtypen. REFERENCE-Attribute enthalten als Wert eine Liste von Surrogaten des entsprechenden anderen Atomtyps. Ein Link zwischen den Atomen a und b wird also dadurch implementiert, daß der Wert eines REFERENCE-Attributes von a das Surrogat von b enthält, und der Wert des entsprechenden REFERENCE-Attributes von b das Surrogat von a enthält. Mit dem Zugriff auf das REFERENCE-Attribut eines Atomes sind deshalb alle Surrogate der Atome verfügbar, die über den betreffenden Linktyp erreicht werden können. Diese Eigenschaft, durch geeignete Operationen unterstützt, läßt sich effektiv dazu ausnutzen, alle Links eines Atomes parallel zu verfolgen.

Jede MAD-Operation bildet aus dem Atomnetzwerk dynamisch Moleküle (komplexe Objekte) eines bestimmten Typs (Molekültyps). Die Struktur dieses Molekültyps wird durch einen gerichteten Ausschnitt des Atomtypnetzwerks (die *Molekülstruktur*) beschrieben, der einen einzigen Startpunkt (den Wurzelatomtyp) hat. Die Moleküle zu dieser Molekülstruktur werden dadurch bestimmt, daß im Atomnetzwerk ausgehend von den Atomen des Wurzelatomtyps alle Atome in ein Molekül aufgenommen werden, die durch Verfolgen der in der Molekülstruktur enthaltenen, gerichteten Linktypen erreichbar sind. Jedes Atom des Wurzelatomtyps (Wurzelatom) bildet also die Wurzel eines Moleküls des spezifizierten Molekültyps. Diese Molekülmenge kann durch zusätzliche Bedingungen eingeschränkt werden.

Üblicherweise werden die komplexen Objekte, mit denen die Anwendung arbeitet, in MAD als Moleküle modelliert, also kann z.B. in einer CAD-Umgebung ein Werkstück einem Molekül entsprechen. Typischerweise setzt sich ein solches Komplexobjekt aus sehr vielen (z.B. 1000) Elementarobjekten zusammen, die als Atome modelliert werden. Daher hat ein typisches Molekül zwar nur eine überschaubare Tiefe (in den meisten Anwendungen < 10), aber eine große Breite. Die in den folgenden Beispielen gezeigten Moleküle sind daher Vereinfachungen, was die Tiefe, und besonders was die Breite angeht. Bild 2.1a zeigt als Beispiel ein Atomnetzwerk (Attribute sind nicht dargestellt), das die Begrenzungsdarstellung zweier sich berührender Rechtecke repräsentiert. Bild 2.1b zeigt drei verschiedene Molekülmengen, die aus diesem Atomnetzwerk durch verschiedene Molekülstrukturen gebildet werden.



a) Darstellung zweier Rechtecke und ihrer Mittelpunkte als Atomnetzwerk



b) Drei verschiedene Molekülstrukturdefinitionen und die entsprechenden Molekülmengen

Bild 2.1: Beispiel zum MAD-Modell

Aus Bild 2.1b wird deutlich, wie dynamisch die Definition von Molekülstrukturen im MAD-Modell ist. A priori ist keine Richtung der Betrachtung vorgegeben. Neben streng hierarchischen Strukturen lassen sich auch netzwerkartige und rekursive definieren, auf die hier jedoch nicht weiter eingegangen werden soll. Die Molekülstrukturdefinitionen bilden die Grundlage der MAD-Operationen. Auf den so spezifizierten Molekülmengen können weitergehende Operationen wie Projektion, Aggregation, oder kartesisches Produkt durchgeführt werden. Zur Beschreibung der Molekülverarbeitung dient die Anfragesprache MQL, die an SQL angelehnt wurde. Datenmanipulationsanweisungen in MQL haben folgenden Aufbau:

{ **SELECT** / **INSERT** / **DELETE** / **UPDATE** } Projektionsklausel
FROM Moleküldefinition
WHERE Selektionsklausel

In der FROM-Klausel wird die Struktur der zu verarbeitenden Molekülmenge(n) spezifiziert. Diese Menge kann durch die Selektionsklausel eingeschränkt werden. In der Projektionsklausel können die daraus zu projizierenden Atome und Attributwerte beschrieben werden.

Die Darstellung des MAD-Modells mußte sich hier auf die Aspekte beschränken, die für die nachfolgende Diskussion wichtig sind. Der näher am MAD-Modell interessierte Leser sei auf [Mi88, Schö89] verwiesen.

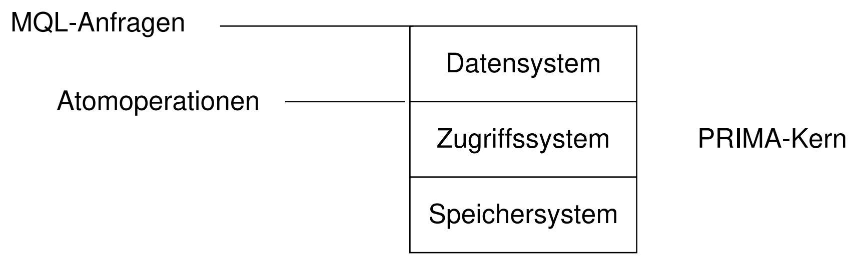


Bild 2.2: Die Schichtenarchitektur des PRIMA-Kerns

Die Verarbeitung von Datenmodelloperationen innerhalb von PRIMA orientiert sich an einem Schichtenmodell [HMMS87]. Die unterste Schicht, das Speichersystem, realisiert die Externspeicherabbildung und stellt an seiner Schnittstelle Seiten und Seitenfolgen zur Verfügung. Diese werden von der nächsten Schicht, dem Zugriffssystem, benutzt, um Atome abzuspeichern und aufzufinden. An seiner Schnittstelle bietet das Zugriffssystem Operationen auf einzelnen Atomen an (Manipulation und Zugriff), wobei das jeweilige Atom über sein Surrogat angesprochen wird. Außerdem bietet das Zugriffssystem die Möglichkeit, einen ganzen Atomtyp atomweise sequentiell zu durchsuchen (Scan). Dabei kann eine Suchbedingung angegeben werden. Ferner können bei einem Scan vorhandenen Zugriffspfade genutzt werden.

Die oberste Schicht (Bild 2.2), das Datensystem, verarbeitet MQL-Anfragen. Diese sind molekülmengen-orientiert, d.h., jede Änderungsoperation arbeitet auf Mengen von Molekülen (desselben Typs) und jede Zugriffsoperation liefert eine Menge von Molekülen. Das Datensystem benutzt zur Auswertung von Anfragen die Operationen des Zugriffssystems. Bei Anfrageübersetzung wird aus der Anfrage ein Abarbeitungsplan (query evaluation plan, QEP) erzeugt, der als gerichteter Operatorgraph dargestellt wird. Die Knoten dieses Graphen bezeichnen Operatoren (wie z.B. Aggregation, Kartesisches Produkt), die Kanten repräsentieren den Datenfluß zwischen den Operatoren. Wir unterscheiden dabei zwei Klassen von Operatoren: die erste Klasse arbeitet auf Molekülmengen, die von untergeordneten Operatoren erzeugt wurden. Die zweite Klasse besteht nur aus dem CSM-Operator* [HSS88], dem einzigen Operator, der das Zugriffssystem aufruft. Er baut aus Atomen eine Menge von Molekülen auf, die durch eine hierarchische Molekülstruktur beschrieben werden (sogenannte "einfache Moleküle"). Diese Molekülmenge kann durch eine Selektionsbedingung eingeschränkt sein, die auf einzelnen Molekülen entscheidbar sein muß **. Bild 2.3 zeigt einen exemplarischen Operatorgraphen.

Zur Entwicklung von Anforderungen an eine Verarbeitungsumgebung und zur Diskussion entsprechender Lösungsmöglichkeiten genügt die Untersuchung von Parallelisierungsmöglichkeiten innerhalb des Datensystems, also bei der Abbildung von MQL-Anfragen auf Atomoperationen des Zugriffssystems.

* construction of simple molecules

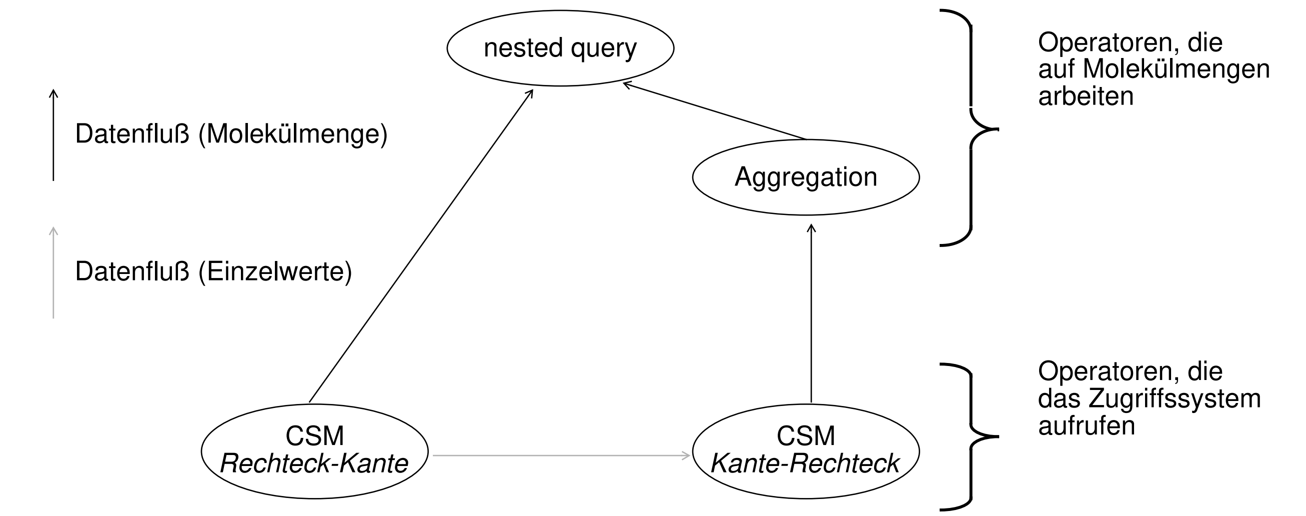
** Komplexere Bedingungen werden durch andere Operatoren ausgewertet.

Beispielanfrage:

Finde alle Rechtecke, bei denen eine Kante länger ist als alle Kanten größerer Rechtecke.

```
SELECT   R.Name
FROM     R (Rechteck) - K (Kante)             (*Umbenennung für späteren Gebrauch*)
WHERE    EXISTS K: K.Länge > MAX (VALUE (   SELECT   Kante.Länge
                                             FROM     Kante - Rechteck
  (* VALUE dient der Typkonvertierung [Schö89] *)  WHERE    EXISTS Rechteck.Fläche>R.Fläche))
```

Zugehöriger Operatorgraph:



Der linke *CSM*-Knoten beschreibt den Zugriff auf alle Moleküle vom Typ Rechteck→Kante. Der zweite *CSM*-Operator erhält den Wert von R.Fläche als Eingabe und berechnet dann alle entsprechenden Kantenlängen, deren Maximum der *Aggregations*-Operator an den *nested-query*-Operator weitergibt.

Bild 2.3: Beispiel für einen Operatorgraphen

3. Parallelisierung einer MQL-Anfrage

In diesem Kapitel sollen verschiedene Möglichkeiten der Parallelisierung bei der Abarbeitung einer MQL-Anfrage, d.h. bei der Auswertung eines Operatorgraphen aufgezeigt werden. Dabei wird jeweils auch über den erwarteten Parallelitätsgrad gesprochen, also über die Anzahl von parallelen Aktivitäten, die durch den jeweiligen Ansatz entstehen können.

Horizontale Parallelität im Operatorgraphen

Zwei Knoten im Operatorgraphen sind voneinander unabhängig, wenn die Ausgabe keines der beiden Operatoren direkt oder indirekt zur Eingabe des anderen beiträgt. Man kann die Knoten eines Operatorgraphen in Klassen von paarweise unabhängigen Operatoren einteilen. Alle Operatoren einer Klasse können dann parallel ausgewertet werden. In Bild 2.3 gibt es keine solchen Knoten. Gäbe es dort mehrere voneinander unabhängige Unteranfragen, so könnten diese parallel zueinander berechnet werden. Die Art der Operatoren in den MAD-Operatorgraphen läßt nicht erwarten, daß auf diese Weise eine erhebliche Parallelität entsteht. Das liegt daran, daß unabhängige Operatoren nur dann im Operatorgraphen auftreten können, wenn mehrere CSM-Operatoren im Operatorgraphen enthalten sind, die voneinander unabhängig sind, da alle anderen Operatoren eine Molekülmenge als Eingabe benötigen, und nur *eine* Molekülmenge als Ausgabe produzieren. Mehrere voneinander unabhängige CSM-Operatoren sind nur dann im Operatorgraphen enthalten, wenn dieser sehr komplexe Operationen enthält, wie z.B. ein kartesisches Produkt über Molekülen, was eine selten verwendete Operation ist. Die Parallelisierungsmöglichkeiten wären größer, wenn statt CSM weniger mächtige Basisoperatoren verwendet würden. Aspekte der Optimierung lassen die Wahl von CSM jedoch geeignet erscheinen.

Vertikale Parallelität im Operatorgraphen

Eine weitere Parallelisierungsmöglickeit ergibt sich aus der Mengenorientierung von MAD: Da jedes Anfrageergebnis aus einer Menge von Molekülen besteht, können die einzelnen Elemente der Menge verzahnt, d.h. mittels "pipelining" zwischen den Operatoren verarbeitet werden. Hier ist ein Parallelitätsgrad erreichbar, der der Höhe des Operatorgraphen entspricht. Besonders erwähnenswert ist die Tatsache, daß die Berechnungszeiten in den einzelnen Operatorknoten stark variieren können und daß die Anzahl der Daten in der Pipeline von Stufe zu Stufe abnehmen kann (z.B., wenn in einer Stufe festgestellt wird, daß ein Molekül die Selektionsbedingung nicht erfüllt). Es handelt sich also nicht um eine "klassische" Pipeline, bei der die Daten alle Stufen durchlaufen und in jeder Stufe gleich lange verweilen.

Vertikale Parallelität innerhalb des CSM-Operators

Betrachten wir als nächstes die Arbeitsweise des CSM-Operators genauer. Die einfachste Art, diesen Operator zu realisieren ist, mit einem Scan atomweise alle Wurzelatome zu durchlaufen, für jedes von ihnen die zugehörigen Nachfolger per Direktzugriff zu lesen, dann deren Nachfolger usw. Wenn ein Atom gelesen wurde, sind damit schon alle Surrogate der direkten Nachfolger bekannt, denn sie sind im Wert der entsprechenden REFERENCE-Attribute enthalten. Somit kann der Zugriff auf diese Nachfolgermenge über die zugehörige Surrogatmenge parallel erfolgen, natürlich auch dann, wenn die Atome zu verschiedenen Atomtypen gehören (s. Beispiel 3 aus Bild 2.1b). Der auf diese Weise erzielbare Parallelitätsgrad hängt von der Anzahl der Atome auf jeder Ebene des Moleküls ab. Für Beispiel 2 aus Bild 2.1b ist er 2, für Beispiel 3 ist er 3 und für das erste Beispiel 4. Neben dieser starken Datenabhängigkeit hat diese Methode noch weitere Nachteile:

- Beispiel 1 aus Bild 2.1b zeigt, daß zu demselben Atom (z.B. P1) mehrere Wege führen können. Dieser Umstand muß berücksichtigt werden, um Mehrfachzugriffe auf ein Atom zu vermeiden. Es ist also eine Duplikateliminierung innerhalb der Surrogatmenge erforderlich, bevor sie zum Zugriff verwendet werden kann.
- Wenn Atome nur zur Auswertung der WHERE-Klausel benötigt werden (wie K in Bild 2.3), ist es oft nicht erforderlich, alle diese Atome zu lesen. Wenn bei der Auswertung der Anfrage aus Bild 2.3 eine Kante gefunden wurde, die die Bedingung erfüllt, müssen die anderen Kanten des Rechtecks nicht mehr betrachtet werden. Wenn jedoch alle Kanten parallel betrachtet werden, entfällt diese Möglichkeit, Atomzugriffe einzusparen, denn wenn festgestellt wird, daß keine weiteren Atome mehr benötigt werden, ist der Zugriff auf diese bereits initiiert.

Horizontale Parallelität innerhalb des CSM-Operators

Auch innerhalb des CSM-Operators ist horizontale Parallelität möglich. Während bei der vertikalen Parallelität der Aufbau eines einzigen Moleküls betrachtet wurde, entsteht hier Parallelität zwischen den Molekülen einer Ergebnismenge. Der Zugriff auf die Wurzelatome muß zwar sequentiell erfolgen, da er ein Durchlaufen des gesamten Atomtyps mittels eines Scan bedeutet*. Der Rest jedes Moleküls kann jedoch jeweils eigenständig aufgebaut werden, parallel für jedes Molekül. Eine gewisse Abstimmung beim Aufbau mehrerer Moleküle ist zur Vermeidung von Mehrfachzugriffen dann erforderlich, wenn ein Atom zu mehreren Molekülen gehören kann (z.B. K_2 in Beispiel 2 aus Bild 2.1b). Der erreichbare Parallelitätsgrad entspricht der Zahl der Ankeratome, die vom Zugriffssystem geliefert werden**.

---

* Dies gilt nur, wenn die oben skizzierte Vorgehensweise zur Auswertung von CSM-Operationen Anwendung findet. In Spezialfällen sind jedoch andere Algorithmen vorzuziehen, die dann nach einer Vorberechnungsphase einen parallelen Zugriff auch auf die Ankeratome erlaubt [HSS88, SS89]. Die folgenden Überlegungen gelten jedoch auch für diesen Fall.

** Diese ist im allgemeinen größer als die Zahl der Ergebnismoleküle, da Bedingungen der WHERE-Klausel, die sich nicht ausschließlich auf das Ankeratom beziehen, zu diesem Zeitpunkt noch nicht ausgewertet werden können.

Bewertung der Parallelisierungsmöglichkeiten

Eine Betrachtung der vier diskutierten Parallelitätsarten zeigt, daß diese voneinander unabhängig eingesetzt werden können. Obwohl die größte Parallelität von der Ausnutzung der horizontalen Parallelität im CSM-Operator zu erwarten ist, können die anderen Verfahren zusätzlich angewandt werden. Die vertikale Parallelität innerhalb des CSM-Operators sollte zur Vermeidung unnötiger Atomzugriffe nur dann eingesetzt werden, wenn sicher ist, daß das betreffende Molekül nicht aus der Ergebnismenge ausgeschlossen werden kann, ohne daß alle Ergebnisse der parallel durchgeführten Zugriffe bekannt sind. Hierzu kann man als Heuristik den Molekülaufbau in zwei Phasen einteilen. In der ersten Phase wird der Teil des Moleküls aufgebaut, aus dem die Selektionsbedingung entschieden werden kann. In einer zweiten Phase wird das Restmolekül aufgebaut. Nur in der zweiten Phase wird dann die vertikale Parallelität eingesetzt. Für Duplikateliminierung bei vertikaler Parallelität und Auflösung gemeinsamer Benutzung bei horizontaler Parallelität innerhalb des CSM-Operators benötigt man eine Abstimmung zwischen den parallel ablaufenden Aktivitäten. Dieser Aspekt wird in Kapitel 4 nochmals aufgegriffen. Die Parallelisierungsstrategien für den CSM-Operator sind offensichtlich datenabhängig. Auch die horizontale Parallelität im Operatorgraphen ist wegen der unregelmäßigen Pipeline-Struktur schlecht vorplanbar. Daher benötigen wir für die Durchführung der Parallelisierung eine Ablaufumgebung, die die flexible Erzeugung und Behandlung paralleler Abarbeitungsfolgen zur Laufzeit unterstützt.

Verwandte Arbeiten

Eine Klassifikation von Parallelität in relationalen Datenbanksystemen gibt Chang in [Ch76]. Die dort aufgeführte Anwendung verschiedener Operationen auf verschiedene Daten entspricht der horizontalen Parallelität im Operatorgraphen, die Pipeline-Anwendung eines Operators auf einen Datenstrom entspricht der vertikalen Parallelität im Operatorgraphen. Die von Chang weiterhin angeführte parallele Ausführung einer Operation auf allen Daten findet sich im Datensystem nicht. Sie ist in PRIMA im Zugriffssystem angesiedelt. Andererseits hat die Parallelität innerhalb des CSM-Operators keine Entsprechung in relationalen System, da sie erst durch die Einführung von Komplexobjekten möglich ist.

Kim [Ki90] untersucht Parallelisierungsmöglichkeiten bei der Verarbeitung objektorientierter Anfragen. Dabei werden sogennannte "Pfade" betrachtet, über die ausgehend von bestimmten Objekten (Klassen, Instanzen) andere damit assoziierte Objekte erreicht werden können. Eine Anfrage wird durch Verfolgen einer Menge von Pfaden ausgewertet. Somit besteht eine Ähnlichkeit zum Linkkonzept in MAD. Mehrere Parallelisierungsmöglichkeiten werden in [Ki90] betrachtet; zunächst die Parallelität zwischen unabhängigen Pfaden. Diese entspricht einem Teilaspekt der vertikalen Parallelität im CSM-Operator, nämlich dem parallelen Zugriff auf Atome verschiedener Atomtypen. Dagegen wird ein paralleler Zugriff auf alle Instanzen derselben Klasse (was in MAD Atomen desselben Atomtyps entspricht) in [Ki90] nicht untersucht. Unter der Bezeichnung "Klassen-Hierarchie-Parallelität" diskutiert Kim allerdings den parallelen Zugriff auf Instanzen von verschiedenen Subklassen einer Klasse. Da MAD keine Klassenhierarchie unterstützt, läßt sich hierzu kein direktes Äquivalent in der Verarbeitung von MQL-Anfragen finden. Kim setzt die Parallelität zwischen Pfaden unabhängig von den zu den entsprechenden Instanzen spezifizierten Prädikaten ein, so daß bei seinem Ansatz unnötige Zugriffe entstehen (vgl. Diskussion "Vertikale Parallelität im CSM-Operator"). Ferner betrachtet Kim die sogenannte "Knoten-Parallelität". Diese entspricht in MAD einem Molekülaufbau durch CSM, der parallel bei verschiedenen Atomen startet, an die jeweils ein Prädikat gebunden ist. Diese Strategie ist in MAD prinzipiell auch möglich und würde nach unserer Aufzählung unter "vertikale Parallelität im CSM-Operator" fallen. Sie ist jedoch nur in Spezialfällen, in denen die entsprechenden Bedingungen sehr selektiv sind, sinnvoll und wird daher im folgenden nicht gesondert diskutiert. Als "pipelining" skizziert Kim schließlich eine Strategie, die unserer "horizontalen Parallelität im CSM-Operator" entspricht. Eine Entsprechung zu den Parallelisierungen im Operatorgraphen gibt es in [Ki90] nicht. Wie wir plädiert auch Kim für den gleichzeitigen Einsatz aller aufgezeigten Strategien. Kim sieht allerdings einen hohen seriellen Anteil in seinem Verarbeitungsmodell, der durch die sequentielle Abarbeitung von Knoten innerhalb eines Pfades entsteht. Dieser ist für MAD sicherlich geringer, da im allgemeinen die Breite eines Molekülgraphen deutlich größer ist als seine Höhe.

Nachdem wir nun einige Möglichkeiten der Parallelisierung aufgezeigt haben, stellt sich die Frage, wie man die verschiedenen Arten der Parallelität realisieren kann. Wir werden im nächsten Kapitel zunächst ein Verarbeitungsmodell entwickeln, zu dem wir dann (in Kapitel 5) eine geeignete Realisierung vorstellen.

4. Ein Verarbeitungsmodell für die Verarbeitung von MQL-Anweisungen

Wir modellieren die Verarbeitung von Anfragen im Datensystem mit einem Auftraggeber-Auftragnehmer (Client-Server)-Modell: Das Datensystem ist ein Service, der die Funktion "Ausführen von MQL-Anweisungen" anbietet. Dazu benutzt er andere Services. So gibt es zum Beispiel einen Service für jeden Operatortyp. Gegenüber diesen Services tritt das Datensystem als Client auf. Konkret bedeutet das für das Beispiel aus Bild 2.3, daß der Datensystem-Service den Service für die Verarbeitung von "nested query" aufruft, dieser wieder die Services "CSM" und "Aggregation" usw. Das Zugriffssystem bildet ebenfalls einen solchen Service, der vom Service "CSM" aufgerufen wird. Ein Aufruf an einen Service (Auftrag) erfolgt asynchron, d.h., er besteht aus der Auftragserteilung und der davon getrennten Resultatsentgegennahme. Dies ermöglicht die Erteilung paralleler Aufträge. Ein Resultat kann dabei entweder als Ganzes oder in Form von Teilergebnissen (in PRIMA im allgemeinen in Form von einzelnen Molekülen) übergeben werden. Zwischen den Interaktionen mit dem Service (Auftragserteilung, Annahme eines Teilresultats, Annahme eines Resultats) kann der Auftraggeber beliebige Berechnungen durchführen, insbesondere also weitere Aufträge an denselben oder andere Auftragnehmer erteilen. Dieses Verarbeitungsmodell erlaubt die Unterstützung aller vier diskutierten Parallelisierungsmöglichkeiten:

Die horizontale Parallelität im Operatorgraphen kann dadurch erreicht werden, daß ein Operator-Service parallel Aufträge an alle Services absetzt, von deren Ergebnissen seine eigene Berechnung abhängt. Dies ist durch die Asynchronität der Auftragserteilung leicht möglich. Die vertikale Parallelität im Operatorgraphen könnte man dadurch erreichen, daß jeder Auftrag nur ein einzelnes Ergebnis (ein Molekül aus der Ergebnismenge) liefert. Bei Erhalt des Ergebnisses kann der Auftraggeber dann gleich das nächste Molekül anfordern. In Übereinstimmung mit Graefe [Gr90] haben wir uns jedoch für eine "eager evaluation" entschieden, bei der ein Auftrag, der einmal abgesetzt wurde, solange Teilresultate (einzelne Moleküle) versendet, bis er sein komplettes Resultat berechnet hat. Jedes einzelne Molekül kann so in Pipeline-Art von Operator zu Operator weitergegeben werden. Diese Vorgehensweise erspart den Zusatzaufwand, der mit einer erneuten Auftragsvergabe verbunden ist, und läßt eine bessere Überlappung der Berechnungen zu. Vertikale Parallelität innerhalb des CSM-Operators wird durch mehrfachen parallelen Aufruf des Zugriffssystem-Service erzeugt, horizontale Parallelität innerhalb des CSM-Operators kann z.B. durch Auftragserteilung dieses Operators an sich selbst (zur Erzeugung eines Moleküls zu einem gegebenen Wurzelatom) erreicht werden.

Wie kann nun dieses Verarbeitungsmodell realisiert werden? Wir gehen zunächst einmal davon aus, daß es <u>für jeden Service mindestens eine eigene Verarbeitungseinheit</u> gibt, die diesen realisiert. Diese Annahme hilft uns in der folgenden Diskussion, stellt aber noch keine Aussage über die tatsächliche Abbildung von Services auf (Betriebssystem-) Prozesse dar. Dieser Aspekt wird erst in Kapitel 6 behandelt. Offensichtlich kann es beliebig viele Aufträge an einen Service gleichzeitig geben. Da Selbstaufrufe möglich sind, ist es nicht möglich, nur eine Verarbeitungseinheit pro Service zu haben (**Single-Process**), die immer einen Auftrag vollständig bearbeitet (**Single-Tasking**), also auch auf Ergebnisse aufgerufener Services wartet, bevor sie den nächsten Auftrag bearbeitet. Es gibt folgende prinzipielle Möglichkeiten, die Beschränkungen eines solchen **Single-Process** / **Single-Tasking*** Konzepts zu umgehen:

- Für jeden Auftrag wird ein eigener Prozeß kreiert, der nur diesen Auftrag bearbeitet (**Multi-Process** / **Single-Tasking**). Der Prozeß wartet synchron, bis alle zur Weiterarbeit benötigten Ergebnisse vorliegen. Das Betriebssystem sorgt für die Aktivierung des nächsten rechenbereiten Prozesses. Wenn allerdings ein "statisches" Multi-Process-Konzept verwendet wird, bei dem die Anzahl der gleichartigen Prozesse beschränkt (und vorausbestimmt) ist, kann der Fall des beliebig tief rekursiven Selbstaufrufes nicht behandelt werden. Multi-Process ist also nur in einer "dynamischen" Variante akzeptabel, die die jeweils benötigte Anzahl gleichartiger Prozesse erzeugt. Gegebenenfalls entsteht ein Lastbalancierungsproblem dadurch, daß man einen Prozessor auswählen muß, auf dem der Prozeß gestartet werden soll.

- Ein einziger Prozeß nimmt alle Aufträge entgegen (**Single-Process** / **Multi-Tasking**). Wenn er einen Auftrag nicht weiter bearbeiten kann, weil ihm dazu noch Ergebnisse von Unteraufträgen fehlen, stellt er die Bearbeitung eines Auftrages zurück. Er bearbeitet dann ggf. vorhandene andere (bearbeitbare) Aufträge. Treffen Ergebnisse

* Die Verwendung der Begriffe "Task", "Single-/Multi-Process" und "Single-/Multi-Tasking" ist an [Me88] angeglichen.

eines Auftrags ein, so wird der entsprechende Auftrag (Task) wieder in die Menge der bearbeitbaren Aufträge eingereiht. Nur wenn kein bearbeitbarer Auftrag vorliegt, wartet der Prozeß synchron.

- Die Kombination beider Konzepte ist möglich (**Multi-Process / Multi-Tasking**). In diesem Falle muß jedoch entschieden werden können, welchem (ggf. neu zu erzeugenden) Prozeß ein konkreter Auftrag als (weiterer) Task zugeordnet werden soll. Es muß also eine Lastverteilung durchgeführt werden.

Ein Vergleich der Realisierungen mittels Multi-Process und Multi-Tasking führt zu folgenden Ergebnissen:

- Die bei Multi-Process notwendige Prozeßerzeugung ist relativ "teuer", wie auch in [Gr90] festgestellt wird. Diese Kosten können bei der in PRIMA vorkommenden Häufigkeit der Auftragserteilung nicht vernachlässigt werden. Die Erzeugung eines neuen Task dagegen ist wesentlich weniger aufwendig, da es sich um eine sehr spezielle Operation (im allgemeinen Anlegen einer Datenstruktur zur Repräsentation des Task) handelt, bei der typische Aktionen der Prozeßerzeugung wie Laden des Programmcodes in den Hauptspeicher, Accounting usw. entfallen.

- Weiterhin verursacht die oben skizzierte Verarbeitungscharakteristik wegen des häufigen Wartens auf Ergebnisse anderer Services eine große Anzahl von Wechseln zwischen den verarbeiteten Aufträgen. Diese resultieren bei Multi-Process in Prozeßwechseln und sind hier wiederum teurer als die entsprechenden Taskwechsel bei Multi-Tasking.

- Die große Anzahl von Prozessen, die bei Verwendung von Multi-Process entstehen können, bereitet manchem älteren Betriebssystem ggf. Schwierigkeiten.

- Bei der horizontalen Parallelisierung des CSM-Operators wurde erwähnt, daß es möglich sein muß festzustellen, ob ein bestimmtes Atom schon beim Aufbau eines anderen Moleküls verwendet wurde. Da hier jedem Molekül ein eigener Auftrag entspricht, ist die Abstimmung zwischen verschiedenen Aufträgen nötig. Bei Single-Process / Multi-Tasking ist dies durch den gemeinsamen Adreßraum, in dem alle Aufträge verarbeitet werden, problemlos möglich. Bei Multi-Process ist zu dieser Abstimmung eine Interprozeß-Kommunikation nötig, die ohne gemeinsamen Speicher zeitaufwendig sein kann.

- Bei Single-Process / Multi-Tasking ist offensichtlich keine "echte" Parallelität zwischen den einzelnen Tasks möglich, da immer nur ein Task gleichzeitig verarbeitet werden kann. Dagegen sorgt das unterliegende Betriebssystem bei Vorhandensein mehrerer Prozessoren für eine echt parallele Auftragsbearbeitung, falls eine Multi-Process-Lösung gewählt wird.

- Bei Multi-Process kann die Programmierung der Verarbeitung eines Auftrages sequentiell und weitgehend ohne Berücksichtigung des Multi-Process-Konzepts geschehen, da Dienste wie Scheduling vom Betriebssystem übernommen werden. Bei Multi-Tasking hingegen muß ein prozeßinterner Mechanismus zum Task-Scheduling benutzt werden. Im allgemeinen wird man das zwar durch Hinzunahme einer für den DB-Kern-Programmierer* weitgehend unsichtbaren Task-Verwaltungskomponente realisieren; der Programmierer muß jedoch immer noch darauf achten, daß der Programmcode zumindest seriell wiederverwendbar ist [Me88].

Eine Bewertung der Unterschiede zwischen der Multi-Tasking- und der Multi-Process-Realisierung zeigt in der betrachteten Umgebung deutlich die Vorteile des Multi-Tasking-Ansatzes. Ein Nachteil von Single-Process / Multi-Tasking ist, daß keine echte Parallelität zwischen den Aufträgen an einen Service erzielt werden kann. Die Vielzahl vorhandener Services und die oft im Vergleich zur Wartezeit auf Ergebnisse relativ kurze Berechnungszeit heben jedoch im allgemeinen diesen Nachteil durch Parallelität zwischen den Service-Prozessen auf. Bei geeigneter Realisierung des Task-Scheduling bleibt auch der Aufwand für den Programmierer von Multi-Tasking-Prozessen in vertretbarem Rahmen. Wir haben uns daher entschieden, für PRIMA jeden Service des Datensystems mit dem Multi-Tasking-Ansatz zu realisieren. Im allgemeinen wird dies mit Single-Process einhergehen, da zusätzliches Multi-Process eine Lastverteilungskomponente bedingt. Diese Option bleibt jedoch erhalten.

Da ein Taskwechsel immer im Zusammenhang mit einem Kommunikationsvorgang steht (Deaktivierung eines Task nach Absetzen von Unteraufträgen, Aktivierung bei Erhalt von Antworten oder Aufträgen), bot es sich an, Task-Sche-

---

* Wir verwenden diesen Begriff für einen Programmierer, der Dienste des RC-Systems nutzt, ohne dessen Realisierung zu kennen.

duling und Kommunikation in PRIMA in einem gemeinsamen Basisdienst zu integrieren, den wir Remote Cooperation System (RC-System) nennen. Er soll im folgenden Kapitel näher beschrieben werden.

5. Das RC-System

Entwurfsziel des RC-Systems [HKS90] war die Realisierung eines Basisdienstes für Kommunikation und Task-Verwaltung, der von Betriebssystem und Architekturtyp (herkömmlicher Rechner, lose oder enge gekoppeltes System, s. Kapitel 1) möglichst gut abstrahiert, um die Portabilität unseres Datenbanksystems zu gewährleisten. Um zu analysieren, welche Operationen unser RC-System für einen Service anbieten sollte, präsentieren wir zunächst einen Zustandsgraphen für einen Task (Bild 5.1). Ausgehend von diesem Zustandsgraphen kann man die folgenden Operationen ableiten, die das RC-System anbieten muß.

Grundlegende Operationen des RC-Systems

Der Service-Prozeß muß einen bereiten Task aktivieren können, wenn gerade kein Task aktiv ist. Dazu gibt es die Operation **Accept_Task**, die Zustandsinformation und Parameter eines Task an den Service-Prozeß übergibt. Die Versendung von Antworten kann ein Task mittels **Reply_Part_Of_Task** vornehmen, wenn er ein Teilergebnis verschicken möchte, oder mit **Reply_Task**, was dann zu einer Beendigung der Task führt. Hat ein Task Aufträge abgesetzt, deren Ergebnis abgewartet werden muß, so kann er bei der Blockierung der Task mittels **Break_Task** eine Liste dieser Aufträge angeben. Trifft zu einem dieser Aufträge eine Antwort ein, geht der Task in den Zustand "bereit" über. Es ist auch möglich, diesen Übergang bis zum Eintreffen aller Antworten der spezifizierten Aufträge zu verzögern (**Break_Task_Until_All_Terminated**). Zur Erteilung von Aufträgen dient **Remote_Service_Initiation**; Ergebnisse können mittels **Get_Service_Result** abgeholt werden. Neben diesen Grundfunktionen gibt es noch Funktionen zum Abbruch bereits erteilter Aufträge, zur Prioritätsvergabe etc. Auf diese soll hier aber nicht weiter eingegangen werden.

Anhand dieser kurzen Beschreibung sollte deutlich werden, daß das RC-System eine geeignete Basis darstellt, um die beschriebene Multi-Tasking-Verarbeitung durchzuführen. Das RC-System unterstützt das Konzept "Multi-Process" nur eingeschränkt: Es können über das RC-System keine neuen Prozesse zur Bearbeitung von Aufträgen erzeugt werden; es ist aber möglich, gezielt die Ausprägung eines Service-Prozesses auf einem bestimmten Prozessor anzusprechen*. Ferner ist es möglich, einen Service mehreren Prozessoren zuzuordnen, die über gemeinsamen Speicher verfügen. In diesem Fall wird nur eine Auftragswarteschlange (im gemeinsamen Speicher) verwaltet, auf die alle Prozessoren zugreifen. Kommunizieren mehrere Aufträge über eine gemeinsame Datenstruktur miteinander, so muß der Zugriff darauf synchronisiert werden. Eine Unterstützung dieser Synchronisation durch das RC-System ist nicht vorgesehen, so daß in diesem Fall der DB-Kern-Programmierer die Bearbeitung durch mehrere Prozessoren vorsehen muß.

* Im allgemeinen ist es nicht sinnvoll, mehrere Multi-Tasking-Prozesse für denselben Service auf einem Prozessor ablaufen zu lassen, da hierbei keine Parallelität erzeugt werden kann. Eventuelle Gewinne durch Überbrückung von Page-Fault-Zeiten oder ähnlichem lassen wir hier zunächst außer acht.

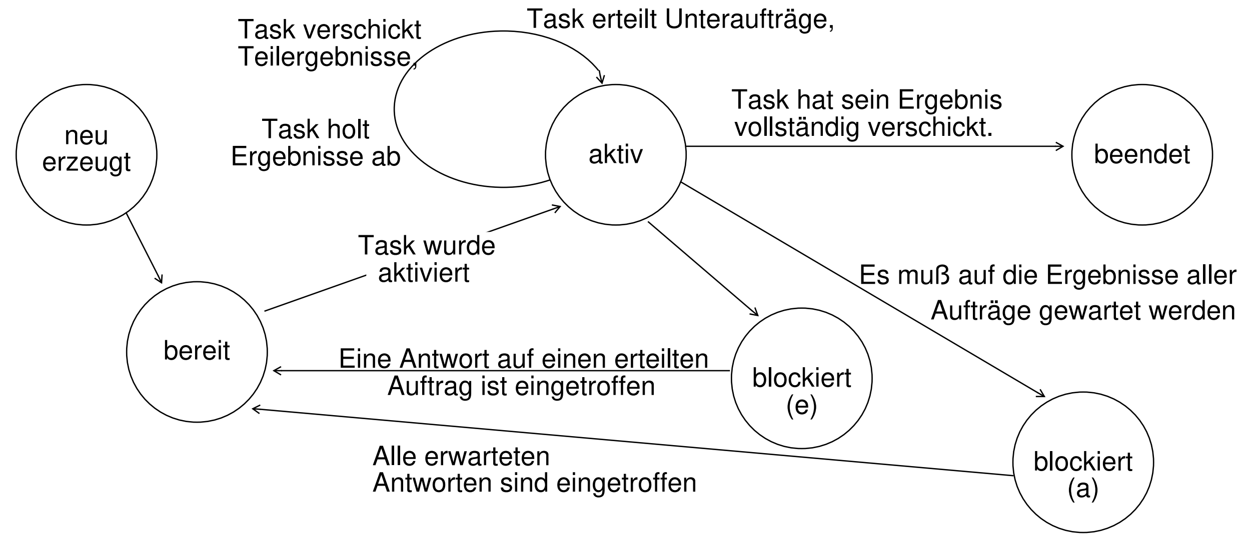


Bild 5.1: Zustandsgraph für einen Task (ohne Abbruchbehandlung)

Kommunikationsmechanismen

Ein Verarbeitungsmodell, das auf einer Vielzahl von (Service-) Prozessen beruht, zwischen denen Nachrichten ausgetauscht werden, ist natürlich nur dann vertretbar, wenn der Nachrichtenaustausch effizient realisiert werden kann. Zum Nachrichtenaustausch soll also ein möglichst schnelles Medium verwendet werden, außerdem soll der Zusatzaufwand möglichst gering sein. Wir betrachten zwei Medien des Nachrichtenaustausches, nämlich gemeinsamen Speicher, der einen sehr schnellen Nachrichtenaustausch ermöglicht, aber nicht unbedingt immer zur Verfügung steht, und einen Nachrichtenversandmechanismus, der immer zur Verfügung steht, aber nicht so schnell ist. Prinzipiell gibt es drei Szenarien für die Auftragserteilung an einen Service (Bild 5.2): Bei Selbstaufruf eines Services (und nur einem Service-Prozeß) liegt immer gemeinsamer Speicher vor, da sich alle Aktionen des Service in demselben Adreßraum abspielen. Hier entstehen auch keine Prozeßwechsel durch Auftragserteilung, Blockierung des Task zum Warten auf Ergebnisse oder Bearbeitung des Unterauftrags. Dies wird jedoch normalerweise der Fall sein, wenn ein Service einen anderen Service (und damit einen anderen Prozeß) beauftragt, der auf demselben Prozessor abläuft. Abhängig vom Betriebssystem (z.B. bei UNIX System V) kann auch in diesem Fall gemeinsamer Speicher benutzt werden. Anderenfalls ist ein Nachrichtenaustausch vorzunehmen. Liegen die Prozesse von Auftraggeber und Auftragnehmer auf verschiedenen Prozessoren, so kann abhängig von der Hardware-Architektur (s. Kapitel 1) ebenfalls über Nachrichten oder gemeinsamen Speicher kommuniziert werden. Im letzteren Fall muß kein Prozeßwechsel erfolgen.

Prozeß des auftragnehmenden Service	in demselben Prozeß	auf demselben Rechner	auf einem anderen Rechner
Kommunikation über gemeinsamen Speicher	immer möglich	möglich je nach Betriebssystem	möglich je nach Hardware-Architektur
Kommunikation über Nachrichtenaustauch	nicht sinnvoll	immer möglich	immer möglich

Bild 5.2: Drei Szenarien für Auftragserteilung

Um zu erreichen, daß der Zusatzaufwand der Auftragserteilung und Ergebnisübermittlung möglichst gering ist, muß man sicherstellen, daß die Anzahl der Kopiervorgänge, die im Laufe einer solchen Kommunikation auftreten, minimal bleibt. Man kann bei gemeinsamem Speicher völlig auf das Kopieren verzichten, falls sich der DB-Kern-Programmierer über diese Tatsache bewußt ist. Er muß nämlich beachten, daß er die Kontrolle über einen Speicherbereich aufgibt, wenn er ihn zur Auftragserteilung oder Resultatsübermittlung verwendet.

Ergebnisübergabe / Auftragserteilung	nicht für Ergebnisübergabe genutzt	Kopie	Kontrolle
nicht für Auftragserteilung benutzt	——	copy-out	control-out
Übergabe als Kopie	copy-in	copy-in-out	*
Überlassen der Kontrolle	control-in	*	exclusive

Bild 5.3: Die verschiedenen Möglichkeiten der Parameterübergabe (* = nicht sinnvoll)

Da es durchaus denkbar ist, daß Auftragserteilung und Resultatsübermittlung sich auf denselben Speicherbereich beziehen, lassen sich verschiedene Optionen der Parameterübergabe unterscheiden (Bild 5.3) je nachdem, ob ein Speicherbereich beiderseits (zur Auftragserteilung und Resultatsübermittlung) oder nur einseitig benutzt wird. Bei "Kontrolle" gibt der jeweilige Prozeß die Kontrolle über den Speicherbereich vollständig an seinen Partner ab. Bei der Nutzungsart "exclusive" arbeiten Auftraggeber und Auftragnehmer auf demselben Speicherbereich, ohne daß dieser kopiert wird. Das bedeutet, daß der Auftraggeber solange nicht auf den Bereich zugreifen darf, wie der Auftrag noch nicht vollständig bearbeitet ist. Bei gemeinsamen Speicher wird bei den Varianten "control-in", "control-out" und "exclusive" nicht kopiert, bei "copy-in" und "copy-out" jeweils einmal, und bei "copy-in-out" zweimal. Liegt kein gemeinsamer Speicher vor, so muß bedingt durch die Übertragung per Netzwerk mindestens ein weiterer Kopiervorgang vorgenommen werden. Diesen Kopiervorgang maskiert das RC-System jedoch für die "Kontrolle"-Varianten [HKS90].

Um eine Portabilität des Gesamtsystems auf beliebige Hardware-Architekturen zu unterstützen, muß für die Services verborgen bleiben, um welche Art der Kommunikation es sich handelt. Das RC-System muß also die jeweils effizienteste Art der Kommunikation wählen können.

Verwandte Arbeiten

Während wir die Entscheidungsfindung über den Kommunikationsmechanismus dem RC-System überlassen, wird die entsprechende Information in Volcano [Gr90] fest in den Operatorbaum der jeweiligen Anfrage integriert. Dort wird ein bestimmter Operator eingesetzt, wenn eine Kommunikation über Prozeß- (Prozessor-) Grenzen hinweg erfolgen soll.

In KARDAMOM [Bü89] gibt es eine ähnliche Komponente wie unser RC-System, die sogenannte "mengenorientierte Speicherverwaltung", die ebenfalls die Art des Datentransfers (gemeinsamer Speicher oder Nachrichtenaustausch) verbirgt. Diese Komponente ist aber eine reine Datentransfereinheit, bietet also keine Taskverwaltung an. Im Gegensatz zum RC-System leitet sie Ergebnisse nicht von sich aus weiter, sondern nur auf Anforderung, so daß die Daten auch erst zum Anforderungszeitpunkt transferiert werden können.

Ein Vorteil des Verbergens der Kommunikationsart im RC-System ist es, daß man dynamisch die Zuordnung von Services zu Prozessen und von Prozessen zu Prozessoren ändern kann. Im folgenden Kapitel wird untersucht, wie eine solche Zuordnung vorteilhaft realisiert werden kann.

6. Prozeß-Prozessor-Zuordnung

Offensichtlich beeinflußt die Verteilung von Service-Prozessen auf Prozessoren die Leistungsfähigkeit des Gesamtsystems erheblich. So sollten zum Beispiel häufig benutzte, rechenzeitintensive Services nicht auf demselben Prozessor angesiedelt sein. Es lassen sich mehrere Heuristiken zur Zuteilung von Prozessen zu Prozessoren aufstellen:

- Services, die große Datenvolumina austauschen, sollten dazu gemeinsamen Speicher nutzen können. Je nach Hardware-Architektur kann das bedeuteten, daß sie auf demselben Prozessor ablaufen müssen.
- Die Services sollten so über die vorhandenen Prozessoren verteilt sein, daß diese gleichmäßig belastet sind.

- Datenbanksystemfremde Prozesse sollten nicht einem Prozessor zugeordnet werden können, auf dem "kritische" Service-Prozesse ablaufen. Es sollte also zumindest einige "reine" Datenbanksystem-Prozessoren in einem System geben.
- Ein Service sollte nur dann durch mehrere Prozesse realisiert werden, wenn eine Lastverteilung "einfach" realisierbar ist (z.B. nach Auftraggebern getrennt).
- Ein Service, der einen Engpaß darstellt, sollte auf mehrere Prozessoren verteilt werden (im gemeinsamen Speicher zwischen diesen wird die Auftragswarteschlange des Service verwaltet).

Natürlich sind diese Heuristiken gegenseitig widersprüchlich - um eine gute Konfiguration zu finden, wird man also ein Mittelmaß suchen müssen. Analysewerkzeuge [HKS90] für diesen Zweck werden bereitgestellt. Selbst wenn man aber nach den oben angeführten Heuristiken vorgeht, kann es dazu kommen, daß mehrere Datenbankprozesse um einen einzigen Prozessor konkurrieren. Das bedeutet, daß es zwischen diesen häufig zu Prozeßwechseln kommt. Die Eigenständigkeit der Services als Prozesse bringt jedoch keinen Vorteil, da auf einem einzigen Prozessor ohnehin keine echte Parallelität möglich ist (von E/A-Aktivitäten abgesehen). Daher würde hier ein Taskwechsel-Mechanismus vollends genügen. Um dieses zu ermöglichen, bietet das RC-System eine bisher noch nicht erwähnte Option an: es kann mehrere Services in einem Prozeß verwalten. Sorgt man dafür, daß dieser Prozeß der einzige auf dem jeweiligen Prozessor ist, so vermeidet man Prozeßwechsel völlig. Es kommt also eine weitere Heuristik hinzu:

- Alle Services, die auf einem Prozessor ablaufen, sollen zu einem Prozeß zusammengefaßt werden.

Auf diese Weise wird die Gesamtanzahl der Service-Prozesse der Anzahl der zur Verfügung stehenden Prozessoren entsprechen, wie dies auch [Gr90] vorschlägt. Es muß allerdings erwähnt werden, daß diese Zusammenfassung mehrerer Services zu einem Prozeß für den DB-Kern-Programmierer noch nicht völlig verborgen bleibt. Eine geringfügige Anpassung des Programmcodes ist hierzu erforderlich. Die Zusammenfassung von Services ist eine Optimierungsmaßnahmen, die nur vorgenommen werden sollte, wenn die korrekte Funktion der Services gewährleistet ist. Dann entstehen auch keine Probleme durch die Überlagerung der Adreßräume.

7. Zusammenfassung und Ausblick

Zur Unterstützung von Nicht-Standard-Anwendungen müssen Datenbanksysteme eine möglichst geringe Antwortzeit bei der Bearbeitung einer Datenmodelloperation aufweisen. Dazu stellt die Ausnutzung von Parallelität innerhalb der Bearbeitung einer solchen Operation einen vielversprechenden Ansatz dar. Am Beispiel des Nicht-Standard-Datenbankkernsystems PRIMA wurden vier prinzipielle Arten der Parallelisierung aufgezeigt: vertikale und horizontale Parallelität im Operatorgraphen und die durch Verarbeitung von Komplexobjekten mögliche vertikale und horizontale Parallelität innerhalb eines Operators. Als geeignetes Verarbeitungsmodell wurde ein Multi-Tasking-Ansatz herausgestellt. Das RC-System ist speziell auf die effiziente Unterstützung einer solchen Verarbeitung zugeschnitten. Es verbirgt dabei Verteilung von Prozessen und Eigenschaften der Hardware-Architektur wie das Vorhandensein mehrerer Prozessoren oder gemeinsamen Speichers. Anhand einiger Heuristiken kann eine "gute" Verteilung der Prozesse auf die zur Verfügung stehenden Prozessoren gefunden werden. Meß- und Analysewerkzeuge [HKS90] ermöglichen es, die Eignung einer Verteilung zu untersuchen.

In Zukunft werden wir untersuchen, wie gut sich die Heuristiken, die in Kapitel 6 angeführt wurden, in der Praxis wirklich bewähren. Insbesondere muß sich zeigen, ob das RC-System effizient genug arbeitet, wenn ein Prozeß sich selbst einen Auftrag erteilt. Es bleibt ferner zu untersuchen, ob alle der angeführten Parallelitätsarten bei allen Arten der Prozessorzuordnung sinnvoll sind oder zumindest nicht zu Leistungseinbußen führen.

Ein großer Vorteil des RC-Systems liegt darin, daß man heterogene Hardware-Architekturen über das RC-System koppeln kann, ohne daß dies Einfluß auf die Programmierung der Services hat. Insbesondere lassen sich kombinieren:

- Homogene Rechner, z.B. ein Netz von SUN-3 Workstations.

- Rechner mit unterschiedlicher Hardware-Plattform, aber gleichem Betriebssystem (z.B. Workstations vom Typ SUN-3 und SUN-4). Hier ist die Kopplung völlig unproblematisch.
- Rechner mit unterschiedlichen Hardware-Architekturen und unterschiedlichem Betriebssystem. Hier ist eine Unterstützung eines gemeinsamen Austauschprotokolls nötig. Es lassen sich aber durchaus so unterschiedliche Rechner wie Siemens 7.540 mit BS2000 und SUN-3/80 unter SunOS 4.0 erfolgreich miteinander koppeln. Insbesondere ist natürlich auch die Kopplung von Mehrprozessormaschinen wie SEQUENT Symmetry und SUN-Workstations mit dem RC-System durchführbar. Daher ist das RC-System auch für die Realisierung der Kommunikation zwischen Modellabbildung und Datenbanksystemkern (s. Kapitel 2) geeignet.

Wir haben eine erste Implementierung des RC-System durchgeführt [HKS90], die die Brauchbarkeit der Konzepte gezeigt hat. Insbesondere ist diese Implementierung weitgehend betriebssystemunabhängig. So mußte für die Portierung von UNIX nach BS2000 nur die Interrupt-Behandlung angepaßt werden. Das RC_System wurde nicht als eigenständige Systemkomponente (z.B. Dämon) realisiert, sondern als Bibliothek von Prozeduren, die zu den beteiligten Prozessen dazugebunden werden. Die auf diese Weise realisierte Dezentralisierung erhöht die Fehlertoleranz (RC-System ist kein "single point of failure") und läßt eine hohe Dynamik bei der Zugehörigkeit einzelner Rechner zum Gesamtsystem zu. Eine effiziente Implementierung des RC-Systems wird natürlich soviel wie möglich auf Konzepte zurückgreifen, die vom unterliegenden Betriebssystem angeboten werden (z.B. "lightweight processes"). Wichtig ist jedoch, daß auf allen Plattformen den Anwendern des RC-Systems eine einheitliche Sicht angeboten wird.

Referenzen

Bü89 von Bültzingsloewen, G., et al.: Design and Implementation of KARDAMOM - A Set-Oriented Data Flow Database Machine, in: Proc. 6th Int. Workshop on Database Machines, IWDM '89, Deauville, Frankreich, 1989, pp. 18-33.

Ch76 Chang, P.Y.: Parallel Processing and Data Driven Implementation of a Relational Database System, ACM SIGMOD Int. Conf on Management of Data, Houston, Texas, 1976, pp. 314-318.

Gr90 Graefe, G.: Encapsulation of Parallelism in the Volcano Query Processing System, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 102-111.

HHMM87 Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: Coupling Engineering Workstations to a Database Server, in: Proc. Conf. on Data and Knowledge Systems for Engineering and Manufacturing, Hartford, Connecticut, 1987, pp. 30-39.

HKS90 Hübel, C., Käfer, W., Sutter, B.: Ein Client/Server-System als Basiskomponente für ein kooperierendes Datenbanksystem, Sonderforschungsbereich 124, Universität Kaiserslautern, SFB-Bericht 26/90, Mai 1990.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th VLDB, 1987, pp. 433-442.

HR85 Härder, T., Reuter, A.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen, in: Proc. Datenbank-Systeme für Büro, Technik und Wissenschaft, Karlsruhe, 1985, Springer-Verlag IFB 94, pp. 253-286.

HSS88 Härder, T., Schöning, H., Sikeler, A.: Parallelism in Processing Queries on Complex Objects, in: Proc. Int. Symposium on Databases in Parallel and Distributed Systems, Austin, 1988, pp. 131-143.

HSS89 Härder, T., Schöning, H., Sikeler, A.: Evaluation of Hardware Architectures for Parallel Execution of Complex Database Operations, in: Proc. 3rd Annual Parallel Processing Symposium Vol. 2, Fullerton, Ca, 1989, pp. 564-578.

Ki90 Kim, K.-C.: Parallelism in Object-Oriented Query Processing, in: Proc. 6th Int. Conf. on Data Engineering, 1990, pp. 209-217.

LP83 Lorie, R.A., Plouffe, W.: Complex Objects and Their Use in Design Transactions, in: Proc. Annual Meeting Database Week: Engineering Design Applications, 1983, pp. 115-121.

Me88 Meyer-Wegener, K.: Transaktionssysteme, Leitfäden der angewandten Informatik, B.G. Teubner, Stuttgart, 1988.

Mi88 Mitschang, B.: Ein Molekül-Atom-Datenmodell für Non-Standard-Anwendungen - Anwendungsanalyse, Datenmodellentwurf und Implementierungskonzepte, Springer-Verlag IFB 185, 1988.

Pa87 Paul, H.-B., et al.: Architecture and Implementation of the Darmstadt Database Kernel System, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco, 1987, pp. 196-207.

Re88 Rehm, S., et al.: Support for Design Processes in a Structurally Object-Oriented Database System, in: Dittrich, K. (ed): Advances in Object-Oriented Systems, Springer-Verlag LNCS 334, 1988, pp. 80-97.

Schö89 Schöning, H.: Rekursion im MAD-Modell: Rekursivmoleküle als Objekte des Datenmodells, in: Proc. Datenbanksysteme für Büro, Technik und Wissenschaft, Zürich, 1989, IFB 204, pp. 389-407.

SS89 Schöning, H., Sikeler, A.: Cluster Mechanisms Supporting the Dynamic Construction of Complex Objects, in: Proc. 3rd Int. Conf. on Data Organization and Algorithms, Paris, 1989, Springer-Verlag, LNCS 367, pp. 31-46.