Dr. rer. nat. Sven Groppe

Übungen zur Vorlesung

Mobile und Verteilte Datenbanken

WS 2008/2009

Blatt 5

Lösung

Aufgabe 1: Parallele Join-Berechnung

Relation R (1 Million Tupel) sei an 4 Rechnern (n=4), Relation S (100.000 Tupel) an 2 Rechnern (m=2) gespeichert. Wie hoch ist der Kommunikationsumfang (in MB) für die Datenumverteilung zur parallelen Join-Berechnung

- für dynamische Replikation
- für dynamische Partitionierung und 5 Join-Rechner (p=5)
- für dynamische Partitionierung, wenn für S das Verteil- mit dem Join-Attribut übereinstimmt, nicht jedoch für R?

Die Tupelgröße betrage 100 B.

Lösung:

R: 1 Mill. • 100 B = 100 MB;
S: 100.000 • 100 B = 10 MB

Dynamische Replikation (Verschicken der S-Relation an alle R-Knoten):
10 MB • 4 = 40 MB

Dynamische Partitionierung (Umverteilung von R und S):
(100 + 10) MB = 110 MB

Dynamische Partitionierung ohne Umverteilung von S:
100 MB

Die dynamische Replikation verursacht in diesem Beispiel den geringsten Kommunikationsumfang !

Aufgabe 2: Mindest-Hauptspeichergröße für partitionierten Hash-Join

Der GRACE-Hash-Join partitioniert S und R zunächst in q Partitionen, falls die kleinere Relation S nicht in den Hauptspeicher paßt. (Es werden also genau zwei Hash-Funktionen verwendet!). Zeigen Sie, daß bei einer Größe der S-Relation von b Seiten, eine Hauptspeichergröße von wenigstens $\sqrt{b}$ + 1 Seiten erforderlich ist.

Lösung:

Für q Partitionen werden zur Partitionierung wenigstens q+1 Seiten im Hauptspeicher benötigt (1 Eingabepuffer, q Ausgabepuffer). Für die Hash-Tabelle jeder S-Partition werden wenigstens b/q Seiten notwendig. Setzt man q=b/q, ergibt sich q=$\sqrt{b}$, so daß der minimale Hauptspeicherumfang dann $\sqrt{b}$+ 1 Seiten beträgt. Für q>b/q wäre der minimale Hauptspeicherbedarf größer als q+1=$\sqrt{b}$+ 1 *für die Partitionierungsrunde*. Für q<b/q wäre der minimale Hauptspeicherbedarf größer als $\sqrt{b}$+ 1 *beim anschließenden Halten der Hash-Tabelle im Hauptspeicher*, da 1+b/q>1+ b/$\sqrt{b}$ =1+$\sqrt{b}$.

Aufgabe 3: TID-Hash-Join

Eine Reduzierung des Speicherbedarfs von Hash-Joins ergibt sich, wenn in der Hash-Tabelle anstelle der vollständigen Sätze nur die Schlüsselwerte für das Join-Attribut sowie die Verweise (tuple identifiers, TID) auf die Sätze gespeichert werden. Diskutieren Sie Vor- und Nachteile eines solchen Ansatzes. Welche Auswirkungen ergeben sich für die parallele Join-Bearbeitung in Shared-Nothing-Systemen ?

Lösung:

Der TID-Hash-Join zahlt sich vor allem aus, wenn damit die E/A-Vorgänge zur Überlaufbehandlung vollkommen umgangen werden. Die Einsparungen sind bei kurzen Join-Attributen bzw. langen Sätzen am höchsten (Bsp.: Komprimierung der Hash-Tabelle um Faktor 10 für Satzlänge 100 B gegenüber Schlüssel+TID von 10 B). Nachteilig ist, daß eine Nachbearbeitung notwendig wird, um die im Verbundergebnis enthaltenen Tupel der kleineren Relation über die TIDs zu rekonstruieren. Der Aufwand für die damit verbundenen Externspeicherzugriffe ist umso höher, je mehr Tupel sich qualifizieren. Im verteilten Fall (Shared-Nothing) ist der Ansatz weniger attraktiv, da bei einer Join-Berechnung außerhalb der Datenknoten keine direkte Auflösung von TID-Verweisen möglich ist (-> Kommunikation).

Aufgabe 4: Paralleler Hash-Join mit Überlaufbehandlung an Datenknoten

Geben Sie einen Algorithmus für einen parallelen Hash-Join in Shared-Nothing-Systemen an, bei dem die Überlaufbehandlung gemäß dem GRACE-Ansatz an den Datenknoten erfolgt.

Lösung:

1. *Paralleles Lesen und Partitionieren von S*
   an jedem S-Knoten j (j=1 .. m) führe parallel durch:
   lies lokale S-Partition S_j und partitioniere sie in q Sub-Partitionen zur Überlaufbehandlung S_{j1} bis S_{jq} durch Anwendung einer Partitionierungsfunktion g auf dem Join-Attribut
2. *Paralleles Lesen und Partitionieren von R*
   an jedem R-Knoten i (i=1..n) führe parallel durch:
   lies lokale R-Partition R_i und partitioniere sie in q Sub-Partitionen R_{i1} bis R_{iq} (Partitionierungsfunktion g)
3. *Parallele Umverteilung und Join-Berechnung*
   für *l* = 1, 2 bis q führe nacheinander aus:
   - an jedem S-Knoten j (j=1 .. m) lies Sub-Partition Sjl und sende jedes S-Tupel an zuständigen Join-Rechner (Verteilungsfunktion f)

- in jedem Join-Knoten k (k=1...p) füge eingehende S-Tupel in Hash-Tabelle ein
- an jedem R-Knoten i (i=1 .. m) lies Sub-Partition Ril und sende jedes R-Tupel an zuständigen Join-Rechner (Verteilungsfunktion f)
- in jedem Join-Knoten k (k=1...p) überprüfe Hash-Tabelle für eingehende R-Tupel und bilde das Join-Ergebnis

Aufgabe 5: Paralleler Hash-Join bei Shared-Everything

Wie können die Algorithmen zur parallelen Hash-Join-Berechnung für Shared-Everything abgewandelt werden?

Lösung:

Voraussetzung für die Nutzung von Datenparallelität ist auch bei SE die verteilte Speicherung der Eingaberelationen sowie ggf. temporärer Dateien auf mehreren Platten. Weiterhin muß eine Aufteilung des gemeinsamen Hauptspeichers in mehrere Bereiche erfolgen, in denen parallel die Join-Berechnung durch unterschiedliche Join-Prozesse erfolgen kann. Das Einlesen der ersten Relation erfolgt parallel durch mehrere Leseprozesse, wobei jeder Prozeß auf disjunkte Plattenmengen zugreift. Die eingelesenen Tupel werden dann - ggf. durch separate Prozesse - in die zugehörige Hash-Tabelle eingetragen, wobei eine Synchronisation (über Semaphore) notwendig wird. Das Lesen der zweiten Relation erfolgt analog, wobei die Join-Berechnung nur Lesezugriffe auf die Hash-Tabellen und somit keine Synchronisation erfordert. Die Join-Ergebnisse werden durch spezielle Schreibprozesse (asynchron) geschrieben. Der Datenaustausch zwischen Lese-, Join- und Schreibprozessen erfordert synchronisierte Zugriffe auf gemeinsame Pufferbereiche; dafür entfällt der Kommunikations-Overhead zur Datenumverteilung von SN-Systemen.

Beispiellösungen aus:

Rahm, E.: Mehrrechner-Datenbanksysteme, Addison-Wesley 1994.