UNIVERSITÄT DES SAARLANDES
Fachbereich 14– Informatik
Prof. Dr.–Ing. G. Weikum
Dipl.–Inform. A. Kraiß

Datenbanksysteme WS 1998/99

Übung 9 — Beispiellösung

Aufgabe 1: Verteilte Ausführung von Joins

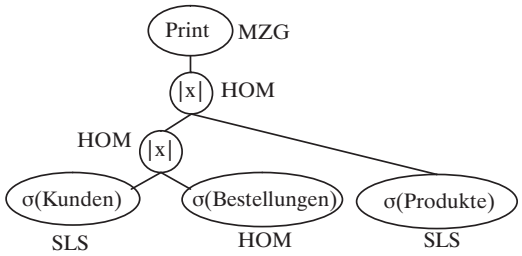
Eine Firma habe ein verteiltes Datenbanksystem mit den Standorten Merzig (MZG), Homburg (HOM) und Saarlouis (SLS). Gegeben ist die folgende Query in Merzig:

π[PNr,Bez,KNr,Name,Gewicht,Menge]
(σ[Ort='Bexbach']{Kunden) |x| σ[Monat=10](Bestellungen) |x| σ[Gewicht>100](Produkte))

Es sollen die folgenden Annahmen über Kardinalitäten und Attributlängen gelten:

card(Kunden) = *40000 Tupel*
card(Produkte) = *10000 Tupel*
card(Bestellungen) = *240000 Tupel*
card(σ[Ort='Bexbach']{Kunden))= *4000 Tupel*
card(σ[Monat=10](Bestellungen))= *20000 Tupel*
card(σ[Gewicht>100](Produkte))= *1000 Tupel*
length(Bez)= length(Name)= *50 Bytes*
length(PNr)= length(KNr)= length(Gewicht)= length(Menge)= *4 Bytes*

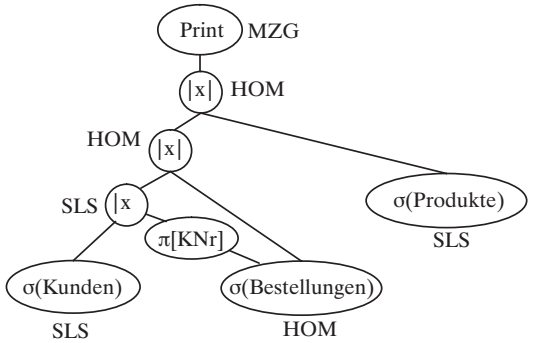
Werteverteilungen verschiedener Attribute seien unabhängig voneinander und jeweils gleichverteilt. Die Relationen *Kunden* und *Produkte* seien in Saarlouis gespeichert, und die Relation *Bestellungen* sei in Homburg gespeichert. Der Query-Optimierer habe als Join-Reihenfolge festgelegt, daß zunächst *σ(Kunden)|x|σ(Bestellungen)* berechnet werden soll, und zwar in Homburg, und der Join mit *σ(Produkte)* anschließend in Homburg erfolgen soll. Es ergibt sich also folgender Ausführungsplan:



Schätzen Sie die notwendigen Kommunikationskosten (in Anzahl zu versendender Bytes)

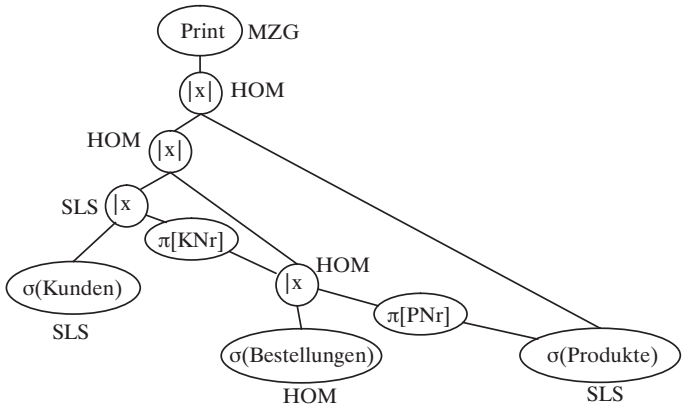
a) für diesen Ausführungsplan,

b) für den folgenden um einen Semijoin (|x) erweiterten Ausführungsplan



und

c) für den folgenden um ein Semijoin-Programm erweiterten Ausführungsplan



Lösung:

Allgemeine Annahme:

Die 40.000 Kunden haben alle 10.000 Produkte in 240.000 Bestellungen, über 12 Monate verteilt, bestellt. Wegen der Gleichverteilung und Unabhängigkeit gilt insbesondere:

(a) Ein Kunde hat 6 Bestellungen, und damit hat ein Kunde im Mittel jeden zweiten Monat ein Produkt bestellt. In einem Monat hat also die Hälfte der Kunden bestellt (im Sinne eines Erwartungswertes).

(b) Pro Monat wird ein Produkt 2x bestellt.

(c) Ein Produkt wurde insgesamt 24x bestellt.

Lösung zu Aufgabe 1a)

Zunächst werden die 4000 Tupel aus der Selektion von SLS nach HOM gesendet:

SLS → HOM: 4000 Tupel * 54 Bytes = 216’000 Bytes

Die Selektion [Monat=10] liefert die Bestellungen eines Monats, d.h. die Bestellungen der Hälfte der Kunden (wegen (a)). Wegen (b) sind dies je zwei Bestellungen für jedes Produkt.

Auf HOM werden die 4000 empfangenen Kunden-Tupel mit den Bestellungen von der Hälfte der Kunden mittels des Joins verbunden. Das liefert also 2000 Tupel auf HOM.

Anschließend werden 1000 Produkte-Tupelauf HOM übertragen:

SLS → HOM: 1000 Tupel * 58 Bytes = 58’000 Bytes

Auf HOM werden die 2000 Joinergebnis-Tupel (über alle Produkte) nun mit den 1000 Produkte-Tupeln (1/10 aller Produkte) mittels Join verbunden, wodurch die Ergebnismenge auf 200 Tupel reduziert wird. Diese 200 Tupel werden nun nach MZG gesendet.

HOM → MZG: 200 Tupel * 116 Bytes = 23’200 Bytes

In der Summe werden **297’200 Bytes** übertragen.

Man kann sich die Kardinalität der Ergebnismenge (200 Tupel) auch herleiten, indem man den Join als kartesisches Produkt der drei Relationen mit entsprechenden Selektionen betrachtet, d.h. als

$$\sigma[K.KNr=B.KNr]\sigma[B.PNr=P.PNr]\sigma[Monat=10]\sigma[Ort=..]\sigma[Gewicht>100](KxBxP)$$

Die Selektivitäten sind durch

$$SEL(\sigma[K.KNr=B.KNr](KxBxP)) = |\sigma[K.KNr=B.KNr](KxBxP)| / |(KxBxP)| = (240’000*10’000)/(40’000*10’000*240’000) = 1/40’000$$

$$SEL(\sigma[B.PNr=P.PNr](KxBxP)) = |\sigma[B.PNr=P.PNr](KxBxP)| / |(KxBxP)| = (240’000*40’000)/(40’000*10’000*240’000) = 1/10’000$$

$$SEL(\sigma[Monat=10](KxBxP)) = 1/12$$

$$SEL(\sigma[Ort=..](KxBxP)) = 1/10$$

$$SEL(\sigma[Gewicht>100](KxBxP)) = 1/10$$

wodurch aus der Unabhängigkeit der Selektionen für die Kardinalität des Ergebnisses folgt:

$$(1/40’000)*(1/10’000)*(1/12)*(1/10)*(1/10)*(40’000*10’000*240’000)=200 \text{ Tupel}$$

Lösung zu Aufgabe 1b)

Wegen des Semi-Joins werden zunächst von HOM aus die Kundennummern für alle Bestellungen eines Monats (Monat=10) nach SLS übertragen. Wegen (a) sind dies die Hälfte aller Kunden, d.h. 20’000 verschiedene Kundennummern.

HOM→SLS: 20’000 Kundennummern * 4 Bytes = 80’000 Bytes

Auf SLS werden die 4000 selektierten Kunden-Tupel mit den empfangenen Kundennummern mittels Join verbunden, und die dadurch auf die Hälfte reduzierte Tupelmenge wird nach HOM gesendet, wo die empfangenen Tupel wieder mit Bestellungen verbunden werden:

SLS→HOM: 2000 Tupel * 54 Bytes = 108’000 Bytes

Die 1000 Produkte-Tupel werden nun nach HOM gesendet:

SLS→HOM: 1000 Tupel * 58 Bytes = 58’000 Bytes

Dort werden sie mit dem ersten Joinergebnis verbunden und die 200 resultierenden Ergebnistupel werden nach MZG gesendet:

SLS→HOM: 200 Tupel * 116 Bytes = 23’200 Bytes

Durch den Semi-Join entstehen leicht niedrigere Transferkosten von **269’200 Bytes.**

Lösung zu Aufgabe 1c)

Beim Semi-Join-Programm werden zunächst die Produktnummern der 1000 selektierten Produkte von SLS nach HOM gesendet:

SLS→HOM: 1000 Tupel * 4 Bytes = 4000 Bytes

Dort werden sie mit den 20’000 selektierten Bestellungs-Tupeln zu 2’000 Ergebnistupeln verbunden, da nur die Bestellungen von 1/10 der Produkte verbunden werden. In den 20’000 Bestellungs-Tupeln waren 20’000 Kundennummern vorhanden, so daß nun nur noch 2000 Kundennummern vorhanden sind. Diese werden nun von HOM nach SLS gesendet:

HOM→SLS: 2000 Tupel * 4 Bytes = 8000 Bytes

Auf SLS werden die 2000 Kundentupel mit 1/10 der Kunden verbunden, wodurch 200 Ergebnistupel entstehen, die wieder nach HOM zurückgesendet werden:

SLS→HOM: 200 Tupel * 54 Bytes = 10’800 Bytes

Nun werden die auf SLS selektierten Produkte-Tupel nach HOM gesendet:

SLS→HOM: 1000 Tupel * 58 Bytes = 58’000 Bytes

und auf HOM zur den 200 Ergebnistupeln verbunden, die schließlich nach MZG gesendet werden:

HOM→MZG: 200 Tupel * 116 Bytes = 23’200 Bytes

Insgesamt wurden hier nur **104’000 Bytes** versendet.

Aufgabe 2: Lastbalancierung bei parallelen Hash-Joins

Bei einem parallelen Hash-Join können Lastunbalanciertheiten (engl.: Skew) auf den Join-Knoten auftreten, die den erzielten Speedup gegenüber dem theoretisch erreichbaren Speedup stark reduzieren können.

Welche Möglichkeiten zur verbesserten Lastbalancierung sehen Sie

a) vor Beginn der eigentlichen Join-Operation, also z.B. beim Versenden von Daten an die Join-Knoten und

b) während der Join-Berechnung, z.B. durch Austauschen von Daten zwischen Join-Knoten.

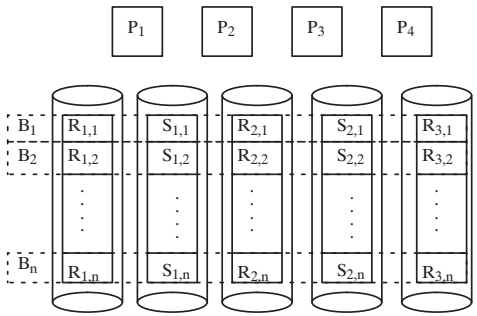
Bei beiden Varianten kann die Organisation der Daten in Hash-Buckets ausgenutzt werden.

Lösung zu a):

Wir nehmen im folgenden an, daß die Relation R in r Partitionen $R_1,..,R_r$ partitioniert ist und die Relation S in s Partitionen $S_1,..,S_s$ partitioniert ist, d.h. daß gilt:

$$R = R_1 \cup ... \cup R_r \text{ und } S = S_1 \cup ... \cup S_s.$$

Für den Join sollen p Joinknoten $P_1,..,P_p$ zur Verfügung stehen. Zur Ausführung des parallelen Hash-Joins müssen die Tupel der Relationen zunächst in einer Verteilungsphase an die Joinknoten verteilt werden, wo sie dann in der Joinphase miteinander verbunden werden. Eine einfache Vorgehensweise zur Verteilung der Tupel wäre, direkt eine Hash-Funktion auf den Joinattributwert jedes Tupels anzuwenden, die die Nummer eines Joinknotens liefert, und das Tupel an diesen Joinknoten zu versenden. Durch ungleichmäßige Verteilungen der Joinattributwerte kann es jedoch zu einer ungleichmäßigen Verteilung der Tupel an die Joinknoten kommen und dadurch zu einer Lastunbalanciertheit bei Ausführung des Joins. Aus diesem Grund werden die Tupel auf sogenannte Buckets verteilt, die wiederum auf Joinknoten verteilt werden. Wir nehmen der Einfachheit halber im folgenden an, daß nur ein einziges Joinattribut A existiere, d.h. daß der Join $R \,|x|_{[R.A=S.A]}\, S$ zu berechnen ist. Durch Verwendung einer Hash-Funktion $h : dom(A) \rightarrow 1..n$ kann jedes Tupel einer Menge von n Buckets B_i, $i=1..n$, $n \geq p$ (häufig n>>p), zugeordnet werden, die das Granulat zur anschließenden Lastbalancierung darstellen. In der folgenden Abbildung ist die Vorgehensweise für den Fall $p=4$, $r=3$, und $s=2$ illustriert.



Für jedes Bucket B_i gilt:

$$B_i = \bigcup_{k=1}^{r} R_{k,i} \cup \bigcup_{k=1}^{s} S_{k,i},$$

wobei

$$R_{k,i} = \{t \mid t \in R_k \wedge h(t.A) = i\}$$

und

$$S_{k,i} = \{t \mid t \in S_k \wedge h(t.A) = i\}$$

gilt.

Die Buckets B_i sollten so gebildet und den Joinknoten zugewiesen werden, daß die Last möglichst gleichmäßig verteilt wird. Liegt ein Bucket B_i an einem Joinknoten vor, so kann der Knoten den Join

$$\left(\bigcup_{k=1}^{r} R_{k,i}\right) |x| \left(\bigcup_{k=1}^{s} S_{k,i}\right)$$

lokal berechnen. Sei *load(B_i)* die durch den Join des Buckets B_i verursachte Last an einem Knoten. Wir nehmen an, daß *load(B_i)* durch

$$load(B_i) = \sum_{k=1}^{r} f(R_{k,i}) + \sum_{k=1}^{s} g(S_{k,i})$$

geschätzt werden kann, wobei die Funktionen f und g die durch die R-Tupel bzw. S-Tupel verursachte Last schätzen und von der Implementierung der Joins abhängig sind. Beispielsweise könnten die Funktionen $f(R_{k,i}) = |R_{k,i}|$ und $g(S_{k,i}) = |S_{k,i}|$ verwendet werden, die lediglich die Anzahl der R- bzw. S-Tupel im Bucket liefern. Dies wäre in etwa proportional zu den wirklichen Kosten eines lokalen Hash-Joins.

Wir nehmen im folgenden an, daß n auf einen Wert gesetzt sei, der sehr viel größer als die Anzahl der Joinknoten p ist (z.B. $n=10*p$). Dadurch kann jedem Joinknoten eine Vielzahl kleiner Buckets zugewiesen werden, und es ergibt sich ein erhöhter Freiheitsgrad zur Lastbalancierung.

Zur Verteilung der Tupel kann nun folgendermaßen vorgegangen werden.

a) *Ermitteln statistischer Werte für die B_i.* Jeder Knoten, der Partitionen R_i oder S_i beheimatet, bestimmt statistische Werte für die Teilbuckets $R_{i,j}$ bzw. $S_{i,j}$. Beispielsweise kann der Knoten durch Anwendung der Hash-Funktion h auf alle Tupel aus den Relationen R_i bzw. S_i die Anzahl der Tupel in den Teilbuckets oder die Größe der Teilbuckets bestimmen. Um den Aufwand zum Ermitteln der Werte zu reduzieren, können auch Schätzungen für die statistischen Werte durch das zufällige Betrachten weniger Stichprogem-Tupel bestimmt werden (engl.: Sampling).

   Jeder Knoten sendet die von ihm ermittelten statistischen Werte an einen vorher bestimmten Koordinatorknoten, der aus der Vereinigung der Werte für die $R_{i,j}$ bzw. $S_{i,j}$ die Werte für die *load(B_i)* bestimmen kann.

b) *Lastbalancierung*. Der Koordinatorknoten hat nun die Zuweisung der Buckets an die Joinknoten zu bestimmen, d.h. er hat für jeden Knoten P_i eine Menge $W_i \subseteq [1..n]$ zu bestimmen, so daß insgesamt gilt: $\bigcup_{k=1}^{n} W_k = [1..n]$ und $W_k \cap W_l = \emptyset$ für alle

$k, l = 1..n$ *und* $k \neq l$. Ausgehend von den geschätzten Lasten *load(B_i)* bestimmt der Koordinator die W_k so, daß gilt:

$$\max_{i=1..p}\left\{\sum_{j \in W_i} load(B_j)\right\} \text{ ist minimal !}$$

Ein solches Zuordnungsproblem ist NP-hart, weshalb Heuristiken angewendet werden müssen. Insbesondere kann die in der Vorlesung vorgestellte LPT-Heuristik (bzw. Largest-Heat-First-Heuristik) direkt angewandt werden (siehe Vorlesungsunterlagen S.220)

Hat der Koordinator die "optimalen" Zuordnungen W_k bestimmt, so übergibt er jedem involvierten Knoten die Zuordnung der Teilbuckets auf die Joinknoten, damit die Bukkets schließlich an die Joinknoten gesendet werden können.

Lösung zu b):

Dadurch, daß in Teil a) die Lastverteilung a priori auf der Basis von Schätzfunktionen realisiert ist, kann es in der anschließenden Joinphase trotzdem zu Lastungleichgewichten kommen. Aus diesem Grund ist es sinnvoll, zusätzlich eine dynamische Lastbalancierung in der Joinphase durchzuführen. Die Grundidee hierbei ist, daß Knoten, die keine Joins mehr auszuführen haben, ihre Rechenleistung den anderen Knoten anbieten und dann von noch beschäftigten Knoten Buckets übergeben bekommen. Folgendes Verfahren ist denkbar:

Ein Knoten P_s, der die Joinberechnung für sein letztes Bucket beendet hat, sendet an alle anderen am Join beteiligten Knoten eine Nachricht (z.B. per Broadcast), daß er gerne neue Bukkets hätte. Jeder Knoten P_r, der eine solche Nachricht empfängt, geht folgendermaßen vor:

1. Ist P_r ein Knoten, der ebenfalls keinen Join mehr auszuführen hat, so wartet dieser (weiterhin) auf neue Arbeit.
2. Ist P_r ein Knoten, der mindestens ein noch vollständig abzuarbeitendes Bucket besitzt, so versendet er einen Teil der noch vollständig abzuarbeitenden Buckets an P_s (z.B. *1/p,* die Hälfte oder ein Bucket, je nachdem, wie hoch die Latenzzeiten und die Transferzeiten für Buckettransfers sind).
3. Ist P_r ein Knoten, der gerade den Join für sein letztes Bucket durchführt, so kann er alle für das Hash-Building verwendeten Tupel, evtl. inklusive der bereits aufgebauten Hash-Tabelle, an P_s senden (behält sie selbst aber auch). Dann gibt er z.B. die Hälfte der für das Hash-Probing verbleibenden Tupel an P_s ab.