

## Übungen zur Vorlesung

# Mobile und Verteilte Datenbanken

WS 2010/2011

Aufgabenstellung zur Abgabe

Dies ist das einzige Aufgabenblatt, dessen Lösung abgegeben werden muß. Das Bestehen dieses Aufgabenblattes ( $\geq 50\%$ ) entscheidet über die Teilnahme an der Prüfung. Die Lösung muß spätestens bis zum **2.2.2011** elektronisch per Email an [groppe@ifis.uni-luebeck.de](mailto:groppe@ifis.uni-luebeck.de) erfolgen. Die Lösung sollte aus einer schriftlichen Ausarbeitung, in der auf die aufgeführten Fragen eingegangen und eine Installationsanweisung gegeben wird, sowie sämtlicher Quelltext (versehen mit sinnvollen Kommentaren) bestehen. Abgaben im 2er Team sind erlaubt. Die Programmiersprache ist Java und zur verteilten Kommunikation sollte Java Remote Method Invocation (RMI) verwendet werden.

### Aufgabe 1 (Verteiltes Sortieren):

**20 Punkte**

Es soll ein Basisdienst zum verteilten Sortieren entwickelt werden. Als Implementierung für das *lokale* Sortieren können Sortiermethoden von den Standard Java Bibliotheken verwendet werden. Es soll dabei eine Infrastruktur geschaffen werden, wobei sich zunächst Client-Rechner für das lokale Sortieren bei einem Server registrieren. Anschließend kann der Server die Clients für das verteilte Sortieren verwenden, in dem der Server die Daten an die einzelnen Clients verteilt, diese dort lokal sortiert werden und anschließend die lokal sortierten Daten wieder an den Server übermittelt und auf den Server zu global sortierten Daten vereinigt werden. Zur Optimierung der Kommunikation sollten die Daten vom Server zu den Clients und von den Clients zum Server immer blockweise (anstatt einzeln) geschehen.

Die zu sortierenden Elemente sollen der Einfachheit halber Instanzen von `String` sein. Ein Programm zum Erzeugen von zufälligen Datensätzen mit einer anzugebenden Anzahl von Elementen ist zu implementieren *oder* ein frei verfügbarer Datensatz aus der realen Welt ist zu verwenden.

Der Server sollte mindestens die Methoden `void add(String element)` zum Einfügen von einzelnen zu sortierenden Elementen, `void sort()` zum Anstoßen der verteilten Sortierung und `java.util.Iterator<String> iterator()` zum Ermitteln eines Iterators, der nach und nach die global sortierten Daten in der sortierten Reihenfolge zurückliefert, bereit stellen.

Es sollen zwei Alternativen für das verteilte Sortieren implementiert werden:

- a) *Verteiltes Merge Sort*: Die Daten sollen der Reihe nach umverteilt werden zu den einzelnen Clients. Die lokal sortierten Daten der Clients sollen auf dem Server gemergt werden, in dem die lokal sortierten Daten blockweise von den Clients angefragt werden.
- b) *Verteiltes Distribution Sort*: Hierbei sollte dem Sortierobjekt auf dem Server bei der Initialisierung eine Menge von zufällig gewählten Elementen  $m_0, \dots, m_k$  aus dem zu sortierenden Datensatz übergeben werden. Im Folgenden sei  $m_0, \dots, m_k$  aufsteigend sortiert. Dann wird ein Element  $e$  auf den Rechner  $i$  verteilt, falls  $m_{i-1} < e \leq m_i$  gilt (bzw.  $e \leq m_i$  für  $i = 0$  und  $e > m_k$  für  $i = k+1$ ). Die lokal sortierten Daten der Clients

sollen auf dem Server nacheinander zurückgegeben werden. Dabei braucht nicht gemergt zu werden, sondern es müssen zuerst alle lokal sortierten Daten von Rechner 0, dann von Rechner 1, ... bis zum Rechner  $k + 1$  nacheinander zurückgegeben werden. Die lokal sortierten Daten von den Clients sollen dabei blockweise zu dem Server übertragen werden. Die zufällig gewählten Elementen  $m_0, \dots, m_k$  müssen in dem Test- und Messprogramm ermittelt werden bei einer zusätzlichen Lese-Phase des zu sortierenden Datensatzes. **Hinweis:** Die Güte der Verteilung hängt stark von der Wahl von  $m_0, \dots, m_k$  ab und könnte durch Berechnung von Histogrammen verbessert werden, dies ist jedoch nicht Teil der Aufgabenstellung.

Durch Messreihen soll zusätzlich ermittelt werden, ob und ab wann sich das verteilte Sortieren mit den Ansätzen a) und b) gegenüber dem lokalen Sortieren lohnt. Weiterhin soll auch die Blockgröße für die zu übermittelnden Blöcke zwischen Client und Server variiert werden, um zu ermitteln, wie die optimale Blockgröße beschaffen ist. Die Ergebnisse sollen analysiert werden.

### **Aufgabe 2 (Verteilte Join-Berechnung):**

**40 Punkte**

In dieser Aufgabe sollen verteilte Join-Verfahren implementiert werden. In der Vorlesung werden verschiedene Ansätze zur verteilten Join-Berechnung behandelt:

- a) *Ship-Whole/Fetch-as-needed*
- b) *Semi-Join*: Hierbei sollten alle 4 Verfahrensvarianten (Join-Ausführung an Knoten  $KR/KS$ , Join-Ausführung an Knoten  $K$  (parallele Variante und sequentielle Variante über  $KS$ )) betrachtet werden.
- c) *Bitvektor-Join (Hash-Filter-Join)*: Hierbei soll die Größe des Bitvektors variiert werden.

Die einzelnen Verfahren a), b) und c) und deren Varianten sollen untereinander und mit einem lokalen Join-Verfahren durch Messreihen verglichen werden. Der lokale Join muss ebenfalls implementiert werden. Als Algorithmus für den lokalen Join kann der Nested-Loop Join *oder* ein Merge Join mit vorherigem Sortieren implementiert werden. Dabei sollen insbesondere die Kommunikationskosten, die Ausführungszeiten auf den beteiligten Rechnern und die Gesamtauswertungszeit gemessen, dargestellt und analysiert werden.