Heterogene Datenbanksysteme

Motivation:

- Es gibt auf dieser Welt 100.000 Datenbanken.
  Jedem Tierchen sein Plaisirchen.
- Die Technologie ist da, diese Datenbanken zu verknüpfen.
  - Kommunikation: Internet, RPC (Client-Server), TPC/IP
  - Interoperabilitätsstandards: http, CORBA, OLE, Java
  - Datenbankinteroperabilitätsstandards: ODBC, JDBC
- Es gibt bereits die ersten, verteilten Anwendungen
  - WWW
  - firmeninterne Informationssysteme im "Intranet"

Bemerkung

Wenn Sie mit Verteilung oder verteilten Datenbanken in Berührung kommen, erwarten Sie stets *Heterogenität*.

Der Kampf der Giganten

Was tun, wenn die Welt aus 1000 Inseln besteht?

1. Man baut sich eine neue Welt.
   Beispiel: SAP (bis Version 3)
   Pro: die neue Welt ist *homogen* und unter Kontrolle; man kann die beste verfügbare Technologie einsetzen
   Contra: riesige Investitionen; man rennt der Technologie hinterher; was passiert, wenn Columbus neue Inseln entdeckt.

2. Man baut sich einen Palast auf Stelzen.
   Beispiel: WWW
   Pro: man läßt die Ameisen für sich arbeiten; billig.
   Contra: Reibungsverluste durch *Heterogenität*. Schlechte Performance und gewisse Sachen kann man einfach nicht machen. Chaos!

3. Kolonialismus: Man bringt alle Schätze nach Rom.
   Beispiel: Data Warehouse
   Pro: wie in (1) ist die Welt in Rom *homogen*, unter Kontrolle und man kann in Rom alle verfügbare Technologie einsetzen
   Contra: Die Schätze in den Kolonien herauszupicken ist teuer. Rom braucht große Lager und Zeug wird evtl. in Rom gar nicht benutzt. Was machen die Leute in den Kolonien?

Der Palast auf Stelzen

Formen von Heterogenität:

1. heterogene Hardware/heterogene Plattform
   Installiere IBM DB2 auf RS 6000 Machinen (AIX) und auf PCs (Windows NT).
2. heterogene (Standard-) Software
   Installiere IBM DB2 in Passau, installiere Oracle in Rom und Informix in Paris.
3. heterogene Software mit unterschiedlichen Datenmodellen
   IBM DB2 in Passau, O_2 in Paris und Lotus 1-2-3 in Warschau.

Heterogene Software mit unterschiedlichen Modellen ist die Herausforderung.

Beispiel 1 für einen Palast auf Stelzen

System 1: Personalmanagement.

- verwaltet Id, Name, Adresse, Gehalt, Bildung, Status, Job, Vorgesetzter von allen Mitarbeitern
- beantwortet Fragen alla: gib mir alle Information des Mitarbeiters mit $id = xyz$.
- API: Funktion, die id als Eingabe kriegt und Info liefert.
  `EmpInfo pm(int id)`

System 2: Lagerverwaltung.

- verwaltet Id, Name, Bestand, Einkäufer für jede Schraube
- organisiert als relationale Datenbank;
- beantwortet alle SQL Anfragen auf die `Lager` Tabellen
- also API: ODBC

Herausforderung: Gib mir das Gehalt aller Mitarbeiter, die für den Einkauf von 6-er Dübeln zuständig sind.

Beispiel 2 für einen Palast auf Stelzen

System 1: Hotelbuchungssystem.

- verwaltet Name, Stadt, Land, Adresse, Tel., Preise von Hotels
- beantwortet so typische Reisebüroanfragen
- führt Hotelbuchungen durch (i.e., Updatetransaktionen!)
- API: schmutziges SQL (d.h. ein bißchen anders und ein bißchen weniger)

System 2: Bilderdatenbank.

- riesige Kollektion von Photos mit Annotation, aus welcher Stadt/Land sie kommen.
- beantwortet Fragen alla: Gib mir die "blauesten Photos" und sag mir wo sie aufgenommen worden.

Herausforderung: Buch mir ein Doppelzimmer, das maximal 300 Schillinge pro Nacht kostet, in einer Stadt mit ganz viel Meer.

Beispielsysteme

Kommerziell: heterogene Software – relationales Datenmodell

- IBM Data Joiner
- Produkte von Sybase, Oracle, UniSQL, ...
- (Microsoft setzt auf Standards alla OLE und ODBC)

Forschung: heterogene Software – unterschiedliche Datenmodelle

- Garlic (IBM)
- Pegasus (HP)
- TSIMMIS (Stanford)
- DISCO (Inria, Frankreich und University of Maryland)
- zig andere Projekte an Unis in XML-Land

Überblick

Das große Ziel:

Verstecke Heterogenität, so dass wir im wesentlichen ein großes homogenes, verteiltes Datenbanksystem vor Augen haben, und alle die tollen Gimmics aus den bisherigen Vorlesungen anwenden können.

Plan-of-Attack:

1. Datenmodell unseres Palastes
2. Architektur unseres Palastes
   (i.e., Stelzen plus Körper)
3. Transaktionsverwaltung
4. Anfragebearbeitung

Bemerkung 1

Wir schaffen das alles in einer Stunde, weil wir das im Grunde ja schon alles können. Wir müssen uns dessen nur immer wieder bewusst werden.

Bemerkung 2

Technischer Begriff für Palast: *Mediator* oder *Middleware*.

Datenmodell von Middleware (1)

- Wir brauchen das kleinste gemeinsame Vielfache aller nur vorstellbaren Datenmodelle auf dieser Welt.
- Wir brauchen ein objektorientiertes Datenmodell.
- Objektorientierte Datenmodelle bieten deklarativen Zugriff alla SQL.
- Objektorientierte Datenmodelle bieten Zugriff via Methoden. Damit kann man alle exotische Funktionen verpacken.
- Objektorientierte Datenmodelle bieten alle möglichen Typkonstruktoren; damit kann man auch exotische Datenbestände einverleiben.

Bemerkung 1

In der Forschung gibt es auch Vorschläge für *logische* und *funktionale* Modelle. Meine persönliche Meinung ist, dass sie keine Chance haben werden.

Bemerkung 2

Dies ist vielleicht der Weg, durch den sich objektorientierte Modelle so richtig in der Datenbankwelt etablieren.

Datenmodell von Middleware (2)

Klappen unsere Beispiele von eben?

JA!

Datenmodell von Middleware (3)

ODMG reicht trotzdem nicht ganz aus:

- Object Identity. Lasse *weak identity* zu, da gewisse Systeme (z.B. relationale Datenbanken) Object Identity alla ODMG nicht zulassen.
- Legacy References. Einige Systeme implementieren Referenzen auf ihre ganz persönliche Art. Hier muß man entweder die Referenzen verpacken oder zusätzliche Einschränkungen bei der Bearbeitung machen. (Siehe Architektur, kommt gleich.)
- Konformität von Typen. Das ist in ODMG nicht vorgesehen aber in einer heterogenen, verteilten DB Welt notwendig.
- Views. Hier muß man ODMG ohnehin ein bißchen Nachhilfe geben.
  Konzept: Object-centered View. Ein View ist immer an ein (Middleware) Objekt gebunden und hat auch die Identity dieses Middleware-Objektes.

Die Rolle von XML

- XML ermöglicht die (semi-) automatische Konstruktion von Stelzen (i.e., Wrapper)
- Der XML Standard garantiert, dass man dieselben Stelzen für viele verschiedene Paläste verwenden kann
- Es gibt besondere XML Datenmodelle, Anfragealgebren und Anfragesprachen (z.B. Quilt), die man verwenden kann
- Wir reden noch über XML!

Hauptidee:

Verwende *Wrapper*.

- Ein Wrapper kriegt eine Teilanfrage und übersetzt sie in Aufrufe des APIs der darunterliegenden Datenbanken.
- Wrapper verkapseln die Besonderheiten der einzelnen Datenbanken. Z.B. führen Wrapper Formatskonversionen durch, führen automatisch gewisse Transformationen ($ → DM nach Tageskurs) durch usw.
- Wrapper reichern ggf. Funktionalität der einzelnen Datenbanken an.
- Wrapper implementieren besondere Techniken (z.B. Caching), um Performance zu verbessern.
- Wrapper sind einfach magisch. Es dauert ungefähr 1/2 Jahr (Größenordnung DM 100.000), um einen guten Wrapper zu entwickeln.
- Wichtig: Man muß das System so auslegen, daß es auch mit relativ dämlichen Wrappern (Größenordnung DM 1.000) umgehen kann. Man muß Tools zur Verfügung stellen, die Implementierung erleichtern.

- Wenn es schlecht läuft, dann kann alles mögliche passieren: Verletzung der Serialisierbarkeit, weil lokale DBs autonom sind und sich nicht um globale Schedules kümmern.
- Wenn es gut läuft, dann reichen die lokalen Synchronisationskomponenten der einzelnen Datenbanken aus.
- Alle Middlewaresystem und Prototypen, die ich kenne, gehen davon aus, dass es gut läuft.

Fazit:

Seien Sie wie immer auf der Hut!

Anfragebearbeitung

Bestandsaufnahme:

- Wrapper machen Konvertierung usw. Jeder Wrapper kann bestimmte Menge von OQL Teilanfragen bearbeiten.
- Manche Wrapper (und ihre darunterliegen Datenbanken) können mehr als andere Wrapper.
- Manche Ausführungsstrategien sind nicht möglich. Z.B. kann man nicht voraussetzen, daß ein Server in Nordkorea mit einem Server in Südkorea kooperiert, um ein Semijoin Programm auszuführen.

Fazit:

Im Grunde ist es ganz normale verteilte Anfragebearbeitung – nur dass einiges nicht möglich ist.

Deshalb:

1. Anfrageoptimierung unter Berücksichtigung, was möglich ist.
2. Krücken bauen im Middlewaresystem, um Fehlendes auszugleichen und so nahe wie möglich an das unmögliche Optimum zu kommen.

Anfrageoptimierung (1)

Ziel:

Verwende bisherige Optimierertechnologie (i.e., Rewrite, Dynamic Programming, ...)

Herausforderung:

Teile dem Optimierer mit, was die einzelnen Wrapper können:

1. Jeder Wrapper definiert seine eigenen Regeln und seine eigenen Operatoren.
2. Alle Wrapperregeln werden vom Dynamic Programming Optimierer (wie gehabt) aufgerufen, um Pläne zu erzeugen.
3. Zusätzlich werden noch die Middlewareregeln aufgerufen, um die Pläne zu erzeugen, die das Middlewaresystem kann.
4. Wie gehabt wird pruning durchgeführt und anhand eines Kostenmodells der beste Plan bestimmt.

Anfrageoptimierung (2)

Besonderheiten:

1. Wrapperpläne sind kryptisch. Sie werden vom Middlewaresystem (und anderen Wrappern) nicht interpretiert, da sie Operatoren enthalten, die der Wrapper selber erfunden hat.
   → wichtig für Erweiterbarkeit und Wrapper von exotischen Datenbanken

2. Middleware versteht trotzdem, was los ist, da Wrapperpläne (wie gehabt) durch *Properties* beschrieben werden.
   (Angewandte Prädikate, Spalten, Sortierung, Kosten usw. sind Beispiele von Properties, die man braucht, um einen Plan zu verstehen.)

3. Aus den Properties der Wrapperpläne des insgesamt besten Planes, bastelt das Middlewaresystem am Ende die Teilanfragen zusammen, die jeder Wrapper zur Bearbeitung bekommt.

4. **Wichtige: Pläne mit Wrapper-Wrapper Kommunikation werden nicht aufgezählt.**
   (Krücken: siehe später.)

Übung

Ziehen Sie das mal an einem Beispiel durch.

Anfrageoptimierung (3)

Eine Erfolgsstory

- IBM hat innerhalb des Garlic Projektes den DB2 Optimierer (200 KLOC) auf diese Weise innerhalb von 6 Monaten in einen verteilten, heterogenen Optimierer umgewandelt. [Haas et al.: VLDB 1997]
- Regeln für Wrapper sind sehr einfach: Größenordnung 100 LOC; es gibt ein paar generische Regeln, die man für fast alle Wrapper einsetzen kann.

Bestehendes Problem

Kostenmodelle für die Wrapper:

- Es gibt kein allgemeingültiges Kostenmodell, das immer hinhaut
- Man kann Implementierern von Wrappern unmöglich zumuten, ein Kostenmodell zu implementieren
  (Das sind Leute im Dunstkreis der Anwender.)
- Kostenmodelle müssen kalibriert werden. Das ist teuer, und es gibt hierzu wiederum kein allgemeingültiges Verfahren.
- Ohne gescheites Kostenmodell kann man sich die ganze Optimierung sparen.

Anfrageoptimierung (4)

Kuriosität: Eine syntaktisch korrekte Query, die nicht bearbeitet werden kann.

Beispielanfrage an unsere Personalverwaltung:

```
SELECT Name
FROM   Emp
WHERE  Gehalt = 3000
```

- Aus Sicht der Middleware ist das eine intakte Query.
- Wenn der Personalverwaltung-Wrapper allerdings keine besonderen Anstalten macht, dann kann er nur Queries bearbeiten der Form:
  `SELECT ... FROM Emp WHERE id = xyz`
- Kontrollierter Abbruch. Keine der Regeln des Personalverwaltungswrappers ist anwendbar. Der Optimierer findet keinen Plan, und das Middlewaresystem kann dem Benutzer mitteilen, daß die Query zwar korrekt aber nicht ausführbar ist.

Krücke: Der BindJoin (1)

Bestandsaufnahme:

- Wie gesagt, kann man nicht immer erwarten, daß sich zwei Server (freiwillig) unterhalten, und der Optimierer geht davon aus, daß sie es nicht können.
- Häufig können aber Server Eingaben annehmen (sogenannte Bindings).
  z.B. können das fast alle relationale Datenbanken und der Server unser Personalabteilung kann das auch.

Idee für einen verteilten Join – BindJoin:

Semi-Join durch das Middlewaresystem durch:

1. Wrapper 1 schickt R ans Middlewaresystem
2. Das Middlewaresystem führt die entsprechende Projektion (mit Duplikatelimination) auf R durch.
3. Das Middlewaresystem schickt Spalte von R an Wrapper 2 (mittels Binding).
   je nach Wrapper: tupelweise oder in einem Schwupp
   (in einem Schwupp ist meistens schneller)
4. Wrapper 2 schickt die entsprechenden Tupel von S zurück.
5. Das Middlewaresystem baut das Ergebnis des Joins zusammen.

Krücke: Der BindJoin (2)

Bemerkung

Anwendung von Bloom Hashfiltern sind in der Regel nicht möglich. (Zumindest sieht das Interface der Personalabteilung mir nicht danach aus.)

Krücke: Gateways

- Gateways ermöglichen doch, daß sich zwei Server unterhalten.
- Müssen von den Administratoren der Server eingerichtet und gewartet werden.
- Wrapper müssen Gateways berücksichtigen; i.e., falls Datenbank A ein Gateway zu Datenbank B hat, muß der Wrapper von A wissen, daß er an alle Daten aus B herankommt.
- Insbesondere muß der Wrapper von A Regeln (und Kostenmodell) für den Optimierer des Middlewaresystems zur Verfügung stellen, damit das Gateway auch wirklich eingesetzt wird.
- **Also: Middlewaresystem hilft wenig bei der Einrichtung von Gateways. Das Middlewaresystem steht aber auch nicht im Weg, wenn es ordentlich designed ist.**

Bemerkung

Replikation kann ähnlich wie Gateways in das System integriert werden. Wiederum hilft das Middlewaresystem nicht, aber es steht auch nicht im Weg.