in: Proc. 12. GI/ITG-Fachtagung "Architektur von Rechensystemen", Springer-Verlag, Informatik aktuell, Kiel, März 1992, S. 106-118

Architekturansätze zur Unterstützung heterogener Datenbanken

Karl-Ludwig Butsch
Erhard Rahm

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
6750 Kaiserslautern

Abstract:

Die ständig zunehmende Verbreitung von Datenbanken verlangt eine geeignete Unterstützung eines koordinierten Zugriffs auf heterogen strukturierte Datenbestände. Trotz der Notwendigkeit einer Kooperation beim Zugriff auf mehrere Datenbanken soll die Autonomie der beteiligten Datenbanken weitgehend erhalten bleiben. Verteilte Datenbanksysteme bieten hierfür keinen geeigneten Ansatz, da sie Homogenität und eine enge Zusammenarbeit der Datenbankverwaltungssysteme verlangen. Wir klassifizieren für den Zugriff auf heterogene Datenbanken besser geeignete Systemarchitekturen und stellen ihre kennzeichnenden Eigenschaften heraus. Insbesondere vergleichen wir verschiedene Arten von verteilten DC-Systemen sowie föderativen Mehrrechner-Datenbanksystemen. Als Vetreter zweier wichtiger Architekturansätze stellen wir die Systeme von Sybase und Ingres vor, die den derzeitigen Stand der Technik in kommerziell erhältlichen Produkten kennzeichnen.

1. Einführung

Traditionellerweise erfolgt die Datenbankverarbeitung mittels zentralisierter Datenbanksysteme (DBS), bestehend aus einer Datenbank und einem Datenbankverwaltungssystem (DBVS). Ein “direkter” Zugriff auf die Datenbank wird möglich durch Verwendung der jeweiligen Anfragesprache des DBVS (z.B. SQL), wobei auf die im DB-Schema definierten Objekte Bezug zu nehmen ist. Da diese Schnittstelle für den Endbenutzer i.a. zu komplex ist, werden in der kommerziellen Datenverarbeitung meist sogenannte Transaktionssysteme eingesetzt, welche eine hohe (maskenorientierte) Schnittstelle anbieten und bei denen Anwendungsfunktionen durch vordefinierte Transaktionsprogramme realisiert werden [Me88]. Die Datenbankzugriffe erfolgen in diesem Fall über die Transaktionsprogramme, welche von dem Endbenutzer lediglich mit den aktuellen Parametern zu versorgen sind. Die Verwaltung der Programme erfolgt durch eine eigene Systemkomponente des Transaktionssystems, dem sogenannten TP-Monitor. Der TP-Monitor kontrolliert die Ausführung der Programme und realisiert die Kommunikation von Programmen mit Terminals sowie mit dem DBVS. TP-Monitor und die Menge der Transaktionsprogramme werden auch als DC-System bezeichnet.

Der ständig wachsende Einsatz von Datenbanksystemen führte dazu, daß vor allem in größeren Unternehmen und Institutionen eine Vielzahl von Datenbanken vorliegt, auf die ein gemeinsamer Zugriff zunehmend unterstützt werden muß [LMR90, SL90, Th90, SW91]. Diese Datenbanken wurden in der Regel unabhängig voneinander entwickelt und können auf verschiedenen Rechnern vorliegen. Die Datenbanken sind i.a. heterogen, d.h. es können unterschiedliche Datenmodelle (z.B. relationales, CODASYL- oder hierarchisches Datenmodell) und unterschiedliche DBVS vorliegen, so daß bei den Anfragesprachen, Integritätsbedingungen oder den verwendeten Verfahren zur Transaktionsverwaltung (Synchronisation, Recovery) große Abweichungen möglich sind. Selbst wenn alle beteiligten DBVS identisch sind, kann eine *semantische Heterogenität* vorliegen, wenn dieselbe Information in mehreren Datenbanken in verschiedener Weise repräsentiert ist (z.B. könnten die Adressen von Personen teilweise redundant in zwei relationalen Datenbanken gespeichert sein, wobei die Speicherung in Relationen verschiedenen Namens erfolgt und unterschiedliche Attribute mit abweichenden Datentypen verwendet werden). Daneben ist Heterogenität natürlich auch bezüglich der Hardware, der Betriebssysteme, Kommunikationsprotokolle sowie der TP-Monitore möglich.

Eine geeignete Unterstützung heterogener Datenbanken sollte es einem Benutzer ermöglichen, innerhalb einer Transaktion auf mehrere Datenbanken mit einer einheitlichen Anfragesprache zuzugreifen. Insbesondere sollten dabei trotz der Verteilung und Heterogenität die üblichen Transaktionseigenschaften [HR83] der Atomarität (Alles-oder-Nichts-Eigenschaft), der Serialisierbarkeit und der Permanenz erfolgreicher Änderungen durch entsprechende Commit-, Synchronisations- und Recovery-Protokolle gewährleistet werden. Weiterhin ist es wünschenswert, daß die Involvierung mehrerer Datenbanken nicht nur dem Endbenutzer, sondern auch dem Anwendungsprogrammierer verborgen bleibt, so daß der Datenbankzugriff und die Programmierung wie im zentralen Fall möglich ist (Verteilungstransparenz, "single system image"). Sofern Replikation von Daten vorliegt, sollte auch diese möglichst automatisch durch die beteiligten Datenbanksysteme gewartet werden (Replikationstransparenz). Eine wichtige Rahmenbedingung, die die Erfüllung dieser Anforderungen erheblich erschwert, ist die Forderung nach einer hohen Autonomie der beteiligten Datenbanksysteme. Insbesondere wird der Aufbau der Datenbanken in lokalen Schemata beschrieben, die unabhängig voneinander entwickelt wurden und die jederzeit ohne Rücksichtnahme auf andere Datenbanken angepaßt werden können (Design-Autonomie). Semantische Heterogenität entsteht zum großen Teil durch diese in der Praxis wichtige Art der Autonomie. Weitere Autonomieforderungen betreffen die lokale Administrierbarkeit (Vergabe von Zugriffsrechten, Festlegung von Speicherungsstrukturen, u.ä.) sowie die lokale Entscheidung darüber, in welchem Umfang externe Zugriffe auf die lokale Datenbank zugelassen werden sollen [SL90].
Von eher untergeordneter Wichtigkeit - im Gegensatz zu homogenen verteilten Transaktionssystemen - sind Anforderungen hinsichtlich einer hohen Durchsatzleistung sowie einer hohen Verfügbarkeit [HR86].

Verteilte Datenbanksysteme (VDBS) [BEKK84, CP84, ÖV91] stellen keinen geeigneten Ansatz zur Unterstützung von Heterogenität und Autonomie der lokalen Datenbanken dar. VDBS gehen davon aus, daß eine einzige logische Datenbank, welche durch ein einziges konzeptuelles DB-Schema beschrieben ist, vorliegt und auf mehrere Rechner verteilt wird. Da dieses gemeinsame Schema von allen Rechnern unterstützt wird, kann volle Verteiltransparenz (und Replikationstransparenz) gegenüber den Benutzern und Anwendungsprogrammen erreicht werden. Insbesondere kann eine DB-Operation praktisch auf jedem Rechner gleichermaßen gestartet werden, so daß die beteiligten DBVS bei der Abarbeitung eng zusammenarbeiten müssen, was Heterogenität kaum zuläßt und die Autonomie erheblich einschränkt. Weiterhin ist die Autonomie der einzelnen Rechner/DBVS stark beeinträchtigt, da Schemaänderungen, Vergabe von Zugriffsrechten, etc. global koordiniert werden müssen.

Ziel dieses Aufsatzes ist es, die zur Erfüllung der genannten Anforderungen besser geeigneten Systemarchitekturen einzuordnen und gegenüberzustellen. Eine Klassifikation verteilter Transaktionssysteme [Ra91], welche in Kap. 2 kurz zusammengefaßt wird, läßt erkennen, daß verschiedene Arten von verteilten DC-Systemen und sogenannter föderativer Datenbanksysteme als vielversprechendste Alternativen anzusehen sind. Diese Ansätze werden auch in Kap. 2 beschrieben und qualitativ bewertet. Zur Verdeutlichung des "State-of-the-Art" in kommerziell verfügbaren Systemen werden die Ansätze von Sybase und Ingres zur Unterstützung heterogener Datenbanken in den Kapiteln 3 und 4 dargestellt. Dabei repräsentiert der Ingres-Ansatz ein Beispiel eines föderativen DBS, während die Sybase-Funktionalität der eines verteilten DC-Systems entspricht.

2. Systemarchitekturen für den Zugriff auf heterogene Datenbanken

In Unterkapitel 2.1 fassen wir zunächst die in [Ra91] vorgestellte Klassifikation verteilter Transaktionssysteme zusammen. Danach werden in 2.2 und 2.3 verschiedene Arten von verteilten DC-Systemen sowie föderativer DBS mit Hinblick auf die Unterstützung heterogener Datenbanken gegenübergestellt.

2.1 Klassifikation verteilter Transaktionssysteme

Als primäres Klassifikationsmerkmal wurde in [Ra91] die Rechnerzuordnung von Systemfunktionen, also der DBVS und TP-Monitore, verwendet. Hierzu wurde zwischen horizontal und vertikal verteilten Transaktionssystemen unterschieden. Bei *horizontal verteilten Transaktionssystemen* liegt eine Replikation von Systemfunktionen vor, so daß jeder Rechner die gleiche Funktionalität hinsichtlich der Transaktionsbearbeitung aufweist. Dies bedeutet, daß in jedem der beteiligten Rechner ein TP-Monitor und ein DBVS vorliegt und an der Transaktionsverarbeitung teilnimmt. Allerdings wird keine Homogenität der Rechner gefordert; vielmehr können verschiedene TP-Monitore und DBVS auf den einzelnen Rechnern laufen und heterogene Betriebssysteme und Rechner-Hardware vorliegen. Dagegen erfolgt bei den *vertikal verteilten Transaktionssystemen* eine Partitionierung der Systemfunktionen auf Front-End- und Back-End-Rechner (Server), welche zu einer funktionalen Spezialisierung der Rechner führt. Beispiele dieser Systemklasse sind Workstation-/Server-Systeme, bei denen gewisse Funktionen zur Transaktionsverarbeitung auf Workstations vorgelagert werden, sowie DB-Maschinen, bei denen eine Auslagerung der DB-Verarbeitung auf Back-End-Rechner erfolgt.

Sowohl bei horizontal als auch bei vertikal verteilten Transaktionssystemen kann die Verteilung auf mehreren Ebenen des DC-Systems oder des DBVS erfolgen, ebenso lassen sich beide Verteilformen kombinieren. Wird die Verteilung im DC-System realisiert, z.B. durch Aufruf entfernter Unterprogramme oder Weiterleiten von DB-Operationen, sprechen wir von einem *verteilten DC-System*. Die Verteilformen auf DBVS-Ebene werden kollektiv als *Mehrrechner-Datenbanksysteme* bezeichnet (da der Begriff “Verteilte Datenbanksysteme” schon mit einer eingeschränkteren Bedeutung belegt ist, s.o.). Weiterhin wird zwischen integrierten und föderativen Mehrrechner-DBS unterschieden. Integrierte Mehrrechner-DBS realisieren eine verteilte DB-Verarbeitung auf einer einzigen logischen Datenbank, die durch ein gemeinsames DB-Schema beschrieben ist. Föderative DBS dagegen bestehen aus einer Menge weitgehend unabhängiger DBVS, die jeweils eine eigene logische Datenbank verwalten, die in einem lokalen (privaten) DB-Schema beschrieben ist. Es soll eine begrenzte Kooperation ("Interoperabilität" [LMR90]) unterstützt werden, jedoch unter Wahrung einer möglichst hohen Autonomie der beteiligten DBVS.

Verteilte Datenbanksysteme zählen zur Klasse der horizontal verteilten, integrierten Mehrrechner-DBS. Zu dieser Klasse gehören auch sogenannte DB-Sharing-Systeme [Ra89], bei denen keine Partitionierung der Datenbank vorgenommen wird, sondern jeder Rechner direkt auf alle Platten und damit die Datenbank zugreifen kann ("Shared Disk"). Eine weitergehende Behandlung integrierter Mehrrechner-DBS findet sich in [HR86]; hier werden sie nicht weiter betrachtet, da sie zur Unterstützung heterogener Datenbanken nicht geeignet sind.

2.2 Verteilte DC-Systeme

Bild 1 zeigt den Grobaufbau von horizontal und vertikal verteilten DC-Systemen. In beiden Fällen erfolgt die rechnerübergreifende Kommunikation innerhalb einer Transaktion ausschließlich zwischen den jeweiligen TP-Monitoren. Die DBVS dagegen sind voneinander unabhängig, so daß ein hohes Maß an Heterogenität und Autonomie bei den Datenbanksystemen in einfacher Weise erreicht wird. Bei den DBVS sind nahezu keine Erweiterungen im Vergleich zum zentralisierten Fall erforderlich, da sämtliche DB-Aufrufe von der lokalen DC-Komponente gestellt werden und im DBVS keine weitere Aufteilung vorgenommen wird.

Diese Einschränkung impliziert, daß jede DB-Operation eines Transaktionsprogrammes vollständig auf einer der Datenbanken abzuarbeiten ist, d.h. die DB-Operation ist das feinste Verteilgranulat das von verteilten DC-Systemen unterstützt wird. Operationen, welche Daten verschiedener Rechner betreffen, müssen explizit ausprogrammiert werden (z.B. Join-Berechnung zwischen Relationen mehrerer Rechner). Desweiteren sieht der Anwendungsprogrammierer i.d.R. mehrere DB-Schemata, d.h. es wird keine Verteilungstransparenz geboten (s.u.). Darüber hinaus bieten die DBVS keine Unterstützung zur Wartung von Daten-Replikation oder semantischer Heterogenität; diese Probleme müssen daher von dem DB-Benutzer (Programmierer) gelöst werden.
Die TP-Monitore müssen ein gemeinsames Zweiphasen-Commit-Protokoll [BEKK84, CP84] unterstützen, um die Alles-oder-Nichts-Eigenschaft einer verteilten Transaktion zu garantieren. Die DBVS sind an dem verteilten Zweiphasen-Commit über die TP-Monitore indirekt beteiligt. Eine mögliche Erweiterung gegenüber zentralisierten DBS besteht also darin, daß ein lokales DBVS eine Commit-Initiierung "von außen" unterstützen muß. Viele zentralisierte DBVS besitzen jedoch bereits diese Funktionalität, da i.d.R. ein lokal verteiltes Commit mit dem TP-Monitor vorgenommen wird, um das Datenbank-Commit mit dem DC-Commit (Sichern der Ausgabenachricht) zu koordinieren. Im verteilten Fall wird damit jedoch eine Verringerung der lokalen Autonomie in Kauf genommen, da der Commit-Koordinator auf einem anderen Rechner residieren kann, dessen Ausfall den Zugriff auf lokale Daten möglicherweise für unbestimmte Zeit blockiert.
Im Falle von vertikal verteilten DC-Systemen sollte die Commit-Koordinierung in keinem Fall auf den Front-End-Rechnern (PCs, Workstations) erfolgen, da diese als inhärent unzuverlässig einzustufen sind (z.B. Abschalten der Front-End-Station durch den Benutzer während einer Transaktion, u.ä.). In diesem Fall sollte die Aufgabe der Commit-Koordinierung auf einen der Server-TP-Monitore übertragen werden.

Die Serialisierbarkeit der Transaktionsverarbeitung ist gewährleistet, wenn jedes der beteiligten DBVS ein Sperrverfahren zur Synchronisation einsetzt und die Sperren bis zum Commit hält (Zweiphasen-Sperrprotokoll). Da die Verwendung solcher Sperrverfahren Standard in existierenden DBVS ist, ergeben sich hier i.d.R. keine neuen Probleme. Allerdings kann es zu globalen Deadlocks kommen, die nur mit einem Timeout-Verfahren aufgelöst werden können, wenn keine Erweiterung der DBVS zur Mitteilung von Wartebeziehungen eingeführt werden soll. Die Unterstützung des Timeout-Ansatzes erfordert eine (einfache) Erweiterung der lokalen DBVS, wenn zur Auflösung lokaler Deadlocks ein andereres Verfahren verwendet wurde (z.B. explizite Erkennung von Deadlocks).

Da der TP-Monitor für die Kommunikation von Transaktionsprogrammen mit den Endbenutzern sowie dem DBVS verantwortlich ist, werden sämtliche Eigenschaften der eingesetzten Kommunikationsmechanismen dem Programmierer verborgen (Kommunikationsunabhängigkeit). Dies gilt in verteilten DC-Systemen auch für die Kommunikation zwischen TP-Monitoren. Wenn unterschiedliche TP-Monitore eingesetzt werden

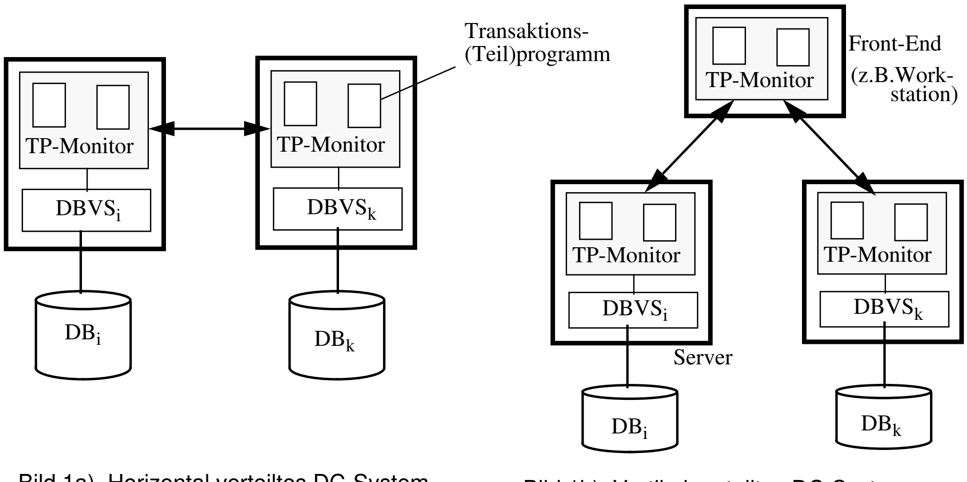

Bild 1a) Horizontal verteiltes DC-System

Bild 1b) Vertikal verteiltes DC-System

sollen, ist die Bereitstellung einer gemeinsamen (normierten) Anwendungsschnittstelle (Application Program Interface, API) wünschenswert, um die Erstellung von Anwendungen zu erleichtern bzw. deren Portabilität zu erhöhen. Darüber hinaus erfordert die Kooperation zwischen heterogenen TP-Monitoren die Verwendung gemeinsamer Protokolle, insbesondere zur Transaktionsverwaltung (Zweiphasen-Commit). Dazu wird in existierenden Systemen meist auf das SNA-Protokoll LU 6.2 von IBM zurückgegriffen [Gr83, Du89]. Es bietet u.a. Funktionen für die Kommunikation zwischen Programmen sowie zur Transaktionsverwaltung. Das verteilte Commit-Protokoll, das sich über mehrere Aufrufstufen erstrecken kann, wird automatisch ausgeführt und entweder implizit (durch Beendigung der Transaktion) oder explizit gestartet. Eine sehr ähnliche Funktionalität wie LU6.2 wird von dem künftigen OSI-Standard TP (Transaction Processing) angeboten [Up91], jedoch stehen reale Implementierungen noch aus.

Zur Realisierung einer verteilten Transaktionsbearbeitung im DC-System kommen im wesentlichen zwei Alternativen in Betracht [Me87, HM90, Ra91]: eine Verteilung auf Ebene der Transaktionsprogramme mit Aufruf von entfernten Teilprogrammen (“verteilte Programmierung”) sowie eine Verteilung auf Ebene der DB-Operationen.

Verteilte Programmierung (Remote Procedure Call, RPC)

In diesem Fall kann auf eine entfernte Datenbank durch Aufruf eines Unter- bzw. Teilprogrammes an dem entsprechenden Rechner zugegriffen werden. Jedes Teilprogramm kann nur lokale Daten eines Rechners referenzieren, wenn keine weitere Verteilung auf DBVS-Ebene erfolgt. Verteilungstransparenz kann für den Anwendungsprogrammierer i.a. nicht erreicht werden, da er die verfügbaren Teilprogramme anderer Rechner kennen muß, um sie korrekt aufzurufen. Gegebenenfalls sind zur Realisierung einer verteilten Anwendung auch eigens Teilprogramme für entfernte Datenbanken zu entwickeln, wobei dann die jeweilige Anfragesprache und Schemainformationen benutzt werden müssen. Zudem werden in existierenden DC-Systemen (UTM-D von Siemens, CICS von IBM) entfernte Unterprogramme (via RPC) zum Teil anders als lokale aufgerufen. Aufgabe der beteiligten TP-Monitore ist die Weiterleitung der entfernten Aufrufe/Antworten sowie die Unterstützung eines verteilten Commit-Protokolles.

Im vertikal verteilten Fall ergibt sich bei dieser Kooperationsform eine Unterscheidung zwischen Front-End- und Server-Transaktionsprogrammen (Bild 1b). Für Programme auf der Front-End-Seite besteht eine eingeschränkte Flexibilität, da sie nicht direkt mit den Server-DBVS kommunizieren, sondern nur Teilprogramme auf den Servern aufrufen können. DB-bezogene Schemainformationen sind nur auf den Back-End-Rechnern sichtbar.

Die Realisierung der RPCs kann prinzipiell vom TP-Monitor ins Betriebssystem verlagert werden. Dieser sich abzeichnende Entwicklungstrend wird durch die zunehmende Verbreitung des DCE-Standards (Distributed Computing Environment) der OSF (Open Software Foundation) unterstützt. OSF DCE definiert eine Reihe von Betriebssystem-Basisdiensten zur Kooperation in heterogenen Systemen, darunter die Realisierung einer verteilten Dateiverwaltung, Directory-Funktionen und von RPCs [Cy91, Se91]. Einige neuere TP-Monitore wie Encina nutzen bereits die DCE-Dienste zur RPC-Abwicklung, allerdings erweitert mit einer Transaktionssemantik ("transaktionsgeschützter RPC") [Sp91]. Im MIA-Projekt (Multivendor Integration Architecture) soll eine herstellerunabhängige Architektur zur verteilten Transaktionsbearbeitung über transaktionsgeschützte RPCs realisiert werden, wobei auch die Prozeduraufrufe im Rahmen eines normierten APIs vereinheitlicht werden.

Verteilung von DB-Operationen

Hierbei werden DB-Befehle von den TP-Monitoren zwischen Rechnern ausgetauscht, mit der Einschränkung, daß jede DB-Operation nur Daten eines Rechners referenzieren darf (keine Kooperation zwischen den DBVS). Das feinere Verteilgranulat gestattet mehr Flexibilität für externe Datenzugriffe als bei der verteilten Programmierung, jedoch auf Kosten eines potentiell höheren Kommunikationsaufwandes. Verteilungstransparenz kann nicht erzielt werden, da bei der Formulierung der DB-Operationen explizit auf das entsprechende DB-Schema Bezug genommen werden muß (der tatsächliche Ort der Datenbanken kann durch Verwendung logischer Namen verborgen werden, nicht jedoch die Unterscheidung mehrerer Datenbanken und ihrer Schemata). Bei einer vertikalen Verteilung werden die Transaktionsprogramme vollständig auf Front-End-Seite ausgeführt; die DB-Operationen können auf Server-Seite über einen TP-Monitor an die DBVS weitergereicht werden.

Die Weiterleitung von DB-Operationen muß jedoch nicht notwendigerweise durch den TP-Monitor erfolgen, sondern kann auch durch andere Kommunikationskomponenten oder (im Falle einer horizontalen Verteilung) direkt durch die beteiligten DBVS realisiert werden. Entscheidend ist lediglich das Verteilgranulat "DB-Operation" und die damit verbundene Restriktion, daß pro Operation nur Daten eines DBVS/Rechners referenziert werden können. Diese Funktionalität wird auch vom DBVS DB2 von IBM unterstützt, wobei jedoch Änderungen in einer Transaktion auf einen Knoten beschränkt sind und nur andere IBM-DBVS mit SQL-Unterstützung beteiligt sein können.

Im Fall heterogener DBVS müssen Unterschiede in den Anfragesprachen, Datenmodellen, Zeichendarstellungen, etc. überbrückt werden, da ansonsten u.a. mehrere Anfragesprachen in einem Programm verwendet werden müßten. Das bereitzustellende normierte API sollte daher insbesondere auch eine einheitliche DB-Anfragesprache zur Formulierung der DB-Operationen umfassen, wozu derzeit nur SQL in Betracht kommt. Eine bestimmte SQL-Version ist bereits Teil des API (CPI genannt) der IBM Systemarchitektur SAA, das eine Interoperabilität zwischen verschiedenen IBM-Plattformen unterstützt. Daneben liegt bereits seit 1987 ein erster ISO-Standard für SQL vor, wobei existierende Implementierungen allerdings eine Vielzahl von Abweichungen zum Standard aufweisen. Dies liegt zum Teil an Unzulänglichkeiten des derzeitigen Standards [Da89]; für die geplanten Folgeversionen SQL2 und SQL3 ist eher mit einer Verschärfung des Problems zu rechnen. Die Herstellervereinigung X/Open definiert ein API, in dem u.a. eine Teilmenge von SQL vorgesehen ist, die von den meisten SQL-Implementierungen unterstützt wird. Trotz dieser Standardisierungsbemühungen werden auch künftig noch DB-Gateways benötigt werden, z.B. um den Zugang zu Nicht-SQL-DBS wie IMS zu ermöglichen.

Um die Übertragung von DB-Operationen/Anfrageergebnissen zwischen heterogenen Rechnern zu erleichtern, wurde der ISO-Standard RDA (Remote Database Access) entwickelt [Pa91, La91]. Damit soll insbesondere die Notwendigkeit einer großen Anzahl von Gateways zur Konvertierung von Daten- und Nachrichtenformaten zwischen heterogenen Systemen umgangen werden. RDA unterstützt dabei eine einheitliche Schnittstelle zur Übertragung von DB-Operationen sowie für Funktionen wie Verbindungskontrolle und Transaktionsverwaltung. Das RDA-Kommunikationsprotokoll spezifiziert ferner die genauen Nachrichtenformate und die erlaubten Reihenfolgen aller Nachrichten. RDA setzt die Verwendung eines einheitlichen APIs (SQL) voraus und verwendet die Dienste anderer ISO-Protokolle (TP) zur Durchführung der Transaktionsverwaltung.

2.3 Föderative Mehrrechner-Datenbanksysteme

In diesem Fall erfolgt die Kooperation zwischen den DBVS der Rechner. Die DC-Systeme der Rechner können unabhängig voneinander arbeiten und verschiedene TP-Monitore benutzen.

Föderative Mehrrechner-DBS streben nach größerer Knotenautonomie im Vergleich zu den integrierten Systemen, wobei die beteiligten DBS entweder homogen oder heterogen sein können. Ähnlich wie bei den verteilten DC-Systemen verwaltet jedes DBVS eine eigene logische Datenbank, die durch ein lokales konzeptuelles Schema beschrieben ist. Allerdings soll jetzt der Zugriff auf externe Daten durch eine begrenzte Kooperation der DBVS unterstützt werden, falls dies von dem “Besitzer” der Daten zugelassen wird. Die Menge kooperierender DBS wird auch als *Föderation* bezeichnet; prinzipiell kann ein DBS an mehreren Föderationen beteiligt sein, falls diese anwendungsbezogen definiert werden. Im Vergleich zu verteilten DC-Systemen soll eine höhere Integrationsstufe erreicht werden, indem ein einheitliches Datenmodell bzw. eine gemeinsame Anfragesprache “nach außen” angeboten werden. Damit soll es auch möglich werden, innerhalb einer DB-Operation auf Daten verschiedener Datenbanken zuzugreifen. Wenn möglich sollen ferner Verteilungs- und Replikationstransparenz sowie Mechanismen zur Behandlung semantischer Heterogenität angeboten werden.

Nach [SL90] kann man grob zwischen eng und lose gekoppelten föderativen DBS unterscheiden, welche unterschiedliche Integritätsstufen anstreben. In beiden Fällen wird davon ausgegangen, daß jedes lokale konzeptuelle DB-Schema in ein Schema eines kanonischen Datenmodells überführt wird, das von allen beteiligten DBVS unterstützt wird (diese *Schema-Translation* ist natürlich nur im Falle heterogener Datenmodelle erforderlich). Für das kanonische Datenmodell bietet sich der relationale Ansatz mit SQL oder ein sogenanntes semantisches Datenmodell mit größerer Ausdrucksmächtigkeit an (z.B. erweitertes Entity-Relationship-Modell). Aufbauend auf dem kanonischen DB-Schema legt jedes DBVS innerhalb eines *Export-Schemas* fest, welche Datenbereiche in welcher Weise von externen Benutzern einer Föderation referenziert werden dürfen (Bild 2). In *eng gekoppelten föderativen DBS* wird durch eine sogenannte Schemaintegration [BLN86] aus den Export-Schemata für jede Föderation ein föderatives DB-Schema abgeleitet (die Anzahl zugelassener Föderationen kann auf 1 begrenzt sein). Der Benutzer des föderativen DBS arbeitet entweder auf dem föderativen Schema oder auf darauf abgeleiteten externen Schemata (Sichten), die nur die ihn betreffenden DB-Bereiche umfassen. Bei *lose gekoppelten föderativen DBS* wird auf eine Schemaintegration sowie föderative Schemata verzichtet, sondern die externen Schemata werden direkt auf den Export-Schemata definiert. Dabei wird i.a. die Unterscheidung mehrerer Datenbanken aufrechterhalten.

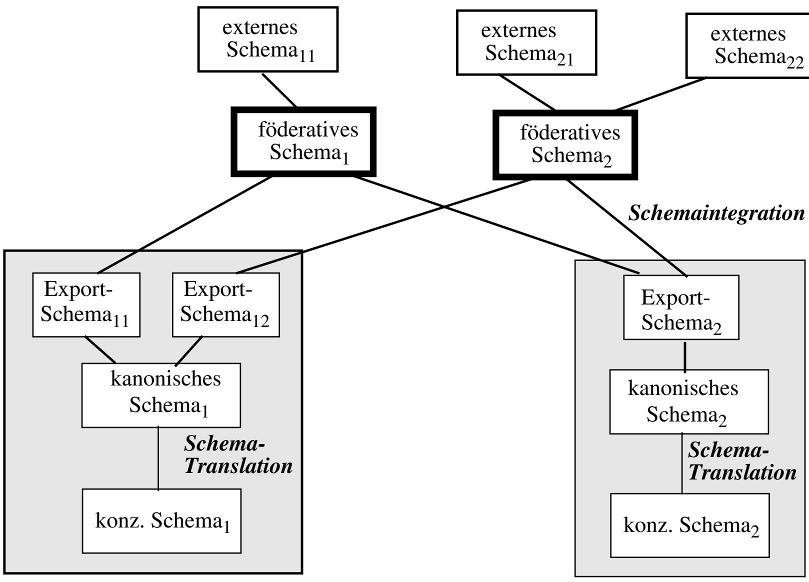

Bild 2: Schemaarchitektur eines eng gekoppelten föderativen DBS (nach [SL91])

Ziel der Schemaintegration ist, die Unterscheidung mehrerer Datenbanken sowie eventuell vorhandener Redundanz für den Benutzer zu verbergen und damit Verteilungs- und Replikationstransparenz zu erzielen. Weiterhin soll semantische Heterogenität eliminiert werden, indem die unterschiedliche Repräsentation derselben Information umgangen wird. Hierzu müssen u.a. Namens- und Typkonflikte erkannt und durch Wahl einer einzigen Repräsentation im föderativen Schema behoben werden; zur Abbildung zwischen föderativem und lokalen Schemata sind geeignete Transformationen zu spezifizieren. Obwohl die Unterstützung eines kanonischen Datenmodells die Schemaintegration erleichtert, kann dieser Prozeß höchstens teilweise automatisiert werden, da derzeitige Datenmodelle keine vollständige Spezifikation der Semantik zulassen. Die Schemaintegration ist daher weitgehend manuell (unterstützt durch geeignete Werkzeuge) seitens eines Datenbank-Administrators vorzunehmen. Ähnliches gilt für die Ableitung der kanonischen DB-Schemata aus den lokalen konzeptuellen Schemata (Schema-Translation).

Die Verfechter der lose gekoppelten föderativen DBS (auch Multi-DBS genannt) vertreten die Ansicht, daß eine vollständige Schemaintegration i.a. nicht möglich ist und semantische Konflikte am besten durch den DB-Benutzer aufgelöst werden [LMR90]. Außerdem reduziert die Definition föderativer Schemata die Autonomie der lokalen DBVS, da lokale Schemaänderungen ggf. wieder global zu koordinieren sind. Daher wird an der Benutzerschnittstelle die Unterscheidung mehrerer Datenbanken beibehalten, jedoch eine Anfragesprache (“multidatabase language”) bereitgestellt, die u.a. DB-übergreifende Operationen zuläßt sowie Hilfsmittel zur Konversion zwischen verschiedenen Datentypen anbietet.

Die Diskussion zeigt, daß föderative DBS erhebliche Erweiterungen in den DBVS erfordern, insbesondere zur Unterstützung von Schema-Translation und Schemaintegration. Weiterhin sind Erweiterungen bei Query-Verarbeitung und -Optimierung notwendig, um globale Anfragen in mehrere lokal ausführbare Teiloperationen zu zerlegen und die Ergebnisse zu kombinieren. Ein aktives Forschungsgebiet ist auch die globale Transaktionsverwaltung, wenn in den beteiligten DBVS unterschiedliche Synchronisations- und Recovery-Verfahren verwendet werden sollen. Derzeitige Systementwicklungen gestatten oft nur lesenden Zugriff auf externe Daten.

3. Systembeispiel 1: Sybase

Die Systemarchitektur von Sybase beruht auf dem allgemeinen Client/Server-Ansatz (Bild 3). Transaktionsprogramme (Clients) und DBVS (Server) können dabei auf verschiedenen Rechnern vorliegen. Insbesondere ist es möglich, die Anwendungen auf Workstations laufen zu lassen, so daß eine vertikale Verteilung der Transaktionsverarbeitung unterstützt wird. Innerhalb einer Transaktion können mehrere Sybase-DBVS (und DBVS anderer Hersteller, s.u.) aufgerufen und auf die von ihnen verwalteten Datenbanken zugegriffen werden. Das Werkzeug "Sybase Tools" bietet eine interaktive Benutzerschnittstelle, welche die Formulierung von Ad-Hoc-Queries, Reportgenerierung etc. unterstützt [Sy89a].

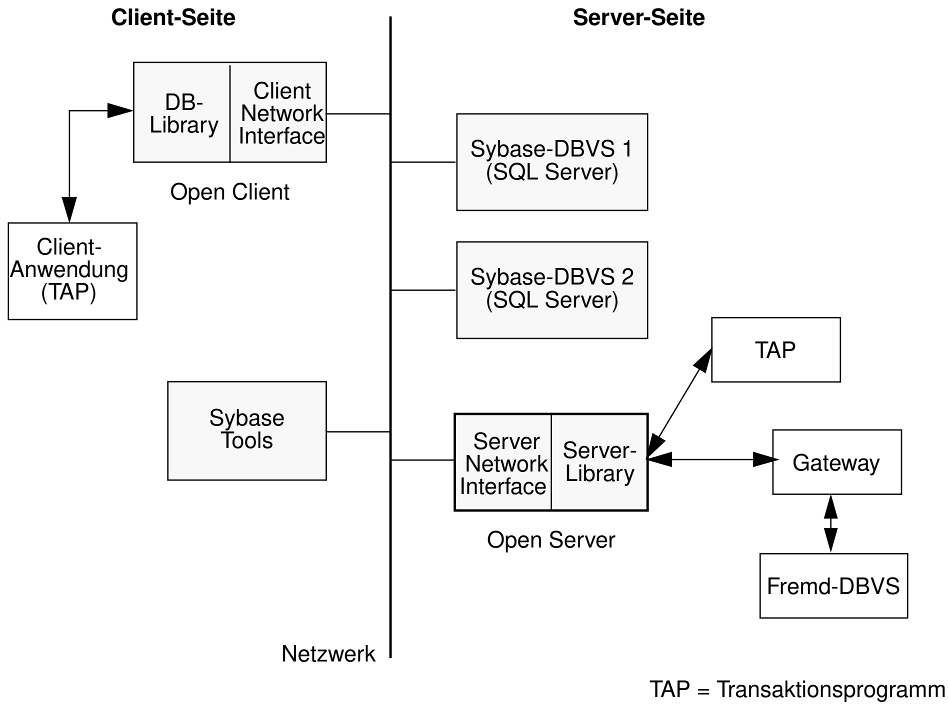

Bild 3: Systemkomponenten beim Einsatz von Sybase

Das DBVS von Sybase ("SQL Server" genannt [Sy89b]) basiert auf dem relationalen Datenmodell und bietet ein Zweiphasen-Commit-Protokoll nach außen hin an. Zur Kommunikation zwischen Anwendungen und dem SQL-Server bietet Sybase zwei Möglichkeiten. Zum einen können wie üblich DML-Befehle an das DBVS übergeben werden, wobei die Einschränkung besteht, daß jede Operation vollständig auf einer Datenbank (von einem DBVS) abzuarbeiten ist. Zum anderen können sogenannte "stored procedures" aufgerufen werden. Diese Prozeduren werden in einer eigenen Programmiersprache (Transact-SQL) definiert und vom Sybase-DBVS verwaltet. Transact-SQL unterstützt neben SQL-Befehlen auch lokale Programmvariablen und allgemeine Kontrollstrukturen (IF, WHILE, etc.). Beim Aufruf der Prozeduren ist es möglich, Parameter zu übergeben; als Ergebnis werden entweder berechnete Werte oder/und eine Menge von Tupeln (Ergebnis der DB-Operationen) zurückgeliefert. Innerhalb der Prozeduren ist es möglich, weitere "stored procedures" aufzurufen, die zu einem anderen SQL-Server gehören können. Diese Kooperation zwischen den DBVS entspricht einer horizontalen Verteilung der Transaktionsverarbeitung durch Aufruf von Teilprogrammen.

Bei Sybase sind Teile der DC-Funktionen im DBVS integriert, wie eine Netzwerkschnittstelle zur Kommunikation und die Verwaltung von Anwendungsprogrammen (stored procedures). Auf Anwenderseite wird die Kommunikation mit Servern über eine Systemkomponente "Open Client" realisiert, die als Front-End-TP-Monitor aufgefaßt werden kann. Open Client enthält ebenfalls eine Netzwerkschnittstelle (client network interface), welche eine Unabhängigkeit von physichen Eigenschaften des Kommunikationsnetzwerkes und Rechner-Hardware gewährleistet. Daneben werden den Anwendungsprogrammen über eine spezielle Bibliothek (DB-Library) höhere Dienste zum Verbindungsaufbau mit Servern, zur Übertragung von DB-Operationen, zum Aufruf von Server-Transaktionsprogrammen (Remote Procedure Call) sowie zur Transaktionsverwaltung (Zweiphasen-Commit, s.u.) angeboten. Die DB-Library repräsentiert somit ein API.

Um den Zugriff auf DBVS anderer Hersteller ("Fremd-DBVS") zu ermöglichen, führte Sybase 1989 die Systemkomponente "Open Server" ein (Bild 3). Diese Komponente ist in der Lage, DB-Operationen und Prozeduraufrufe von Client-Anwendungen (Open Client bzw. Sybase Tools) entgegenzunehmen und an Server-Anwendungen (Transaktionsprogramme) oder Fremd-DBVS weiterzugeben. Aus Sicht der Client-Anwendungen erfolgt die Kooperation mit einem "Open Server" wie mit dem Sybase-DBVS. Server-Anwendungen können die Dienste des Open Server (Server Library) zur Kooperation mit den Client-Anwendungen nutzen. Zum Zugriff auf Fremd-DBVS werden daneben noch Gateways benötigt, die u.a. Unterschiede in den Anfragesprachen ausgleichen und eine Behandlung auftretender Fehler durchführen können. Sybase bietet u.a. ein Gateway zu IBM's DBVS DB2 sowie eine spezielle Version des Open Servers zur Kopplung mit dem TP-Monitor CICS [Du90, Sy90].

Zur Sicherstellung der Alles-oder-Nichts-Eigenschaft verteilter Transaktionen unterstützt Sybase ein verteiltes Zweiphasen-Commit-Protokoll an der Anwendungsschnittstelle (DB-Library). Diese Funktionalität ist jedoch z.Zt. auf verteilte Transaktionen auf Sybase-Datenbanken beschränkt. Außerdem liegt die Kontrolle der Commit-Verarbeitung in der derzeitigen Realisierung [Sy90] weitgehend in der Verantwortung des Anwendungsprogrammierers, so daß Programmfehler zur Inkonsistenz der beteiligten Datenbanken führen können! Insbesondere fungiert die Anwendung als Commit-Koordinator und muß beide Commit-Phasen durch explizite Aufrufe an die an der Transaktion beteiligten Server initiieren. Wird einer der Server bei der Commit-Behandlung "vergessen" oder ihm ein falsches Commit-Ergebnis mitgeteilt, ergeben sich inkonsistente Datenbankzustände.

Ein weiteres Problem ist, daß die Anwendungen i.a. auf Workstations ausgeführt werden, die für die Commit-Koordinierung als zu unsicher einzustufen sind (unbestimmte Ausfalldauer während der Commit-Behandlung möglich). Um zumindest den aktuellen Stand des Commit-Protokolls auf einer "sicheren" Seite zu vermerken, unterstützt Sybase die Einrichtung eines Commit-Services auf einem der Server. Allerdings liegt es wieder in der Verantwortung des Programmierers, diesen Service einzurichten und ihm das Commit-Ergebnis mitzuteilen (der Commit-Service protokolliert daraufhin die Commit-Entscheidung). Nur im Fehler-fall (Server-Ausfall, Timeout) wenden sich die involvierten Datenbank-Server an den Commit-Service, um den Zustand der Transaktion zu erfragen. In einer späteren Version soll das Commit-Protokoll vollständig durch den Commit-Service ausgeführt werden.

Die Diskussion zeigt, daß Sybase die Funktionalität eines verteilten DC-Systems aufweist, wobei beide Aufrufgranulate (DB-Operationen und Teilprogramme) unterstützt werden. Eine Koopera-tion auf DBVS-Ebene erfolgt nur zwischen Sybase-DBVS und ist beschränkt auf den Aufruf entfernter Teilprogramme (stored procedures) sowie die Commit-Behandlung. Eine verteilte Ausführung von DB-Befehlen wird ebensowenig unterstützt wie die Gewährleistung von Verteilungs- oder Replikationstransparenz.

4. Systembeispiel 2: Ingres

Das relationale Datenbanksystem Ingres unterstützt ebenfalls mehrere Formen einer verteilten Transaktionsverarbeitung. Mit der Kommunikationskomponente *Ingres/Net* wird eine vertikale Verteilung möglich, bei der Anwendungen auf Front-End-Rechnern durch Verschicken einzelner DB-Operationen auf das Ingres-DBVS auf einem Server zugreifen können. Ein Zugriff auf Fremd-DBVS ist über Ingres/Net auch möglich, wobei dann Anfragen und Datenformate über ein DB-Gateway anzupassen sind. Seit der Ingres-Version 6.3 kann von einer Anwendung aus auf mehrere Datenbanken zugegriffen werden. Ein verteiltes Zweiphasen-Commit wird dabei nur in rudimentärer Weise unterstützt, da die Anwendung als Commit-Koordinator fungiert und beide Commit-Phasen explizit starten muß. Weiterhin ist die Anwendung in diesem Fall für das Sichern des Commit-Ergebnisses verantwortlich. Damit ist die Konsistenz der beteiligten Datenbanken in noch stärkerem Maße als bei Sybase durch Workstation-Ausfälle und Fehler in den Anwendungsprogrammen gefährdet.

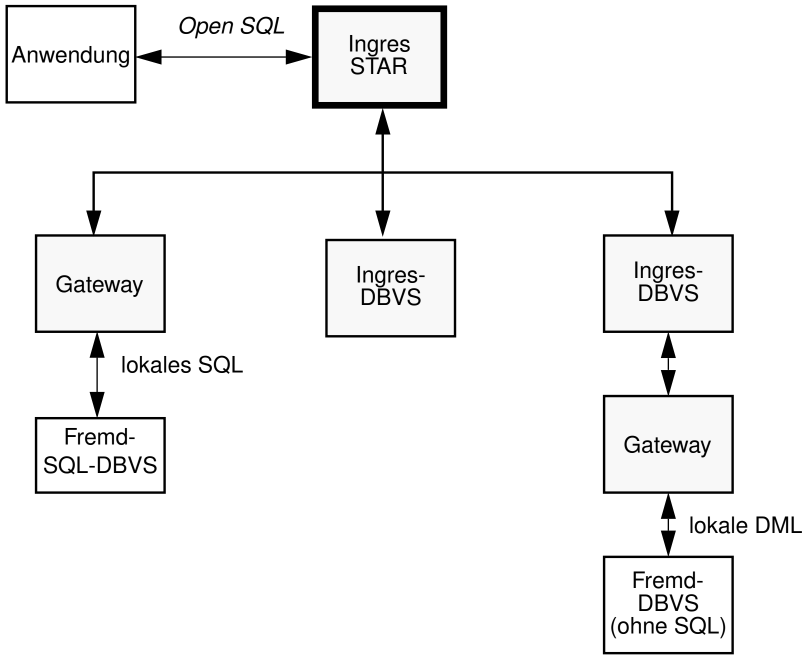

Bild 4: Ingres/Star im Ingres-Systemverbund

Der skizzierte Zugriff auf mehrere, möglicherweise heterogene Datenbanken über Ingres/Net entspricht einer Verteilung im DC-System mit DB-Operationen als Verteilgranulat. Daneben bietet Ingres jedoch auch eine Verteilung auf DBS-Ebene durch die Systemkomponente *Ingres/Star* an. Diese Komponente macht Ingres zu einem (eng gekoppelten) föderativen Mehrrechner-DBS, da sie es ermöglicht, innerhalb einer DB-Operation auf mehrere Datenbanken zuzugreifen. Wie in Bild 4 skizziert kann dabei sowohl auf Ingres-Datenbanken als auf Datenbanken unter der Kontrolle von Fremd-DBVS zugegriffen werden. Die Kommunikation zwischen Ingres/Star und den Fremd-DBVS erfolgt dabei über geeignete Gateways (s.u.). Nicht gezeigt in Bild 4 sind die Ingres/Net-Komponenten, die in jedem der beteiligten Rechner zur Kommunikationsunterstützung vorliegen müssen.

Ingres/Star erlaubt die Definition mehrerer DB-Föderationen, wobei zu jeder Föderation ein föderatives Schema spezifiziert wird. Die Definition der föderativen Schemata setzt eine einheitliche Schemabeschreibung der lokalen Datenbanken als Menge von Relationen voraus (seitens der beteiligten Ingres-DBVS bzw. der Gateways). Das föderative Schema legt dann im wesentlichen nur für jede Datenbank die Relationen fest, die an der Föderation beteiligt sein sollen. Die Existenz mehrerer, möglicherweise heterogener Datenbanken kann somit dem Ingres-Benutzer vollkommen verborgen werden (Verteilungstransparenz). Für den Zugriff auf heterogene Datenbanken (über das föderative Schema) bietet Ingres als gemeinsame Anfragesprache (Programmierschnittstelle) "Open SQL" an, welche aus einer Untermenge der gebräuchlichsten SQL-Dialekte gebildet wurde. Anfragen, die sich auf mehrere lokale Datenbanken beziehen (z.B. Joins), werden von Ingres/Star in mehrere Teiloperationen zerlegt, die auf den lokalen Datenbanken ausführbar sind (globale Query-Optimierung); die Teilergebnisse werden ebenso von Ingres/Star zusammengefaßt und an den Benutzer zurückgegeben. Um den Anwendungen eine einheitliche Fehlerbehandlung auch beim Zugriff auf heterogene Datenbanken zu ermöglichen, wurde eine Menge generischer Fehlercodes festgelegt, die sowohl vom Ingres-DBVS als den von Ingres angebotenen Gateways zu Fremd-DBVS unterstützt werden.

Ingres unterstützt zwei Arten von Gateways in Abhängigkeit davon, ob das Fremd-DBVS SQL unterstützt oder nicht [SB90]. Für SQL-DBVS (z.B. DB2) erfolgt im Gateway eine Umsetzung von "Open SQL" auf das lokale SQL in der einen Richtung und die Versorgung des generischen Fehlercodes in der anderen Richtung. Weiterhin muß, falls erforderlich, eine Typanpassung der Daten in beide Richtungen vorgenommen werden. Der andere Typ von Ingres-Gateways wird für (nicht-relationale) DBVS benötigt, die SQL nicht unterstützen (z.B. für IMS). Hier wird das Fremd-DBVS im wesentlichen nur als Dateisystem verwendet, auf das ein Ingres-DBVS zur Bearbeitung der SQL-Anweisungen aufgesetzt wird (Bild 4). Diese Art der Kopplung kann naturgemäß nur eine geringe Leistungsfähigkeit aufweisen und dürfte auf Lesezugriffe beschränkt sein, da bei Änderungen Modellbedingungen des zugrundeliegenden Datenmodells verletzt werden könnten (z.B. hierarchische Abhängigkeiten zwischen Satztypen).

In der derzeitigen Realisierung von Ingres/Star bestehen noch eine Reihe von Einschränkungen in der Funktionalität. Insbesondere wird nur eine sehr begrenzte Art der Schemaintegration unterstützt, die sich im wesentlichen auf die Vereinigung lokaler Schemafragmente beschränkt (keine Behandlung semantischer Heterogenität). Weiterhin ist die Autonomie der lokalen Datenbanken relativ begrenzt, da eine enge Zusammenarbeit zwischen Ingres/Star und den lokalen Ingres-DBVS (bzw. Gateways) zur verteilten Query-Verarbeitung und für den Zugriff auf lokale Schemainformationen notwendig ist. Andererseits kann Ingres/Star bei Beschränkung auf Ingres-DBVS auch als konventionelles verteiltes Datenbanksystem angesehen werden, bei der die verteilte Datenbank durch mehrere lokale Datenbanken repräsentiert ist. Eine Einschränkung in diesem Fall ist, daß Relationen das feinste Verteilgranulat darstellen und keine Replikation unterstützt wird. Diese Begrenzungen sollen in einer künftigen Version entfallen.

5. Zusammenfassung

Die rapide zunehmende Speicherung von Informationen in immer mehr Datenbanken verlangt eine geeignete Unterstützung eines koordinierten Zugriffs auf heterogen strukturierte Datenbestände. Trotz der Notwendigkeit einer Kooperation beim Zugriff auf mehrere Datenbanken soll die Autonomie der beteiligten Datenbanken weitgehend erhalten bleiben. Verteilte Datenbanksysteme bieten hierfür keinen geeigneten Ansatz, da sie die Beschreibung des gesamten Datenbestandes innerhalb eines konzeptuellen Datenbankschemas und eine enge Zusammenarbeit der Datenbankverwaltungssysteme (DBVS) beim DB-Zugriff verlangen. Wie in Kap. 2 dargestellt kommen im wesentlichen verteilte DC-Systeme sowie föderative Mehrrechner-DBS als System-architekturen für den Zugriff auf heterogene Datenbanken in Betracht.

Verteilte DC-Systeme bieten den Vorteil, daß bestehende DBVS nur geringfügig zu erweitern sind, um innerhalb einer Transaktion auf mehrere heterogene Datenbanken zuzugreifen. Im wesentlichen genügt dazu, daß die DBVS die Initiierung eines verteilten Zweiphasen-Commit von "außen" akzeptieren und einen Timeout-Mechanismus zur globalen Deadlock-Behandlung verwenden. Die Kommunikation innerhalb der Transaktion erfolgt weitgehend durch das DC-System (TP-Monitor), wobei entweder einzelne Teilprogramme (Remote Procedure Call) oder DB-Operationen als Aufrufeinheiten zwischen Rechnern in Betracht kommen. Diese Ansätze können gegenüber dem Anwendungsprogrammierer keine Verteiltransparenz bieten; zudem ist die Ausführung jeder DB-Operation auf eine Datenbank begrenzt. Aufgabe des TP-Monitors ist die Durchführung eines verteilten Commit-Protokolls sowie das Verbergen von Heterogenität in der Rechner-Hardware und in der Kommunikationsumgebung. Soll dem Anwender gegenüber Heterogenität bei den DBVS (Anfragesprache, Datentypen, etc.) verborgen werden, sind geeignete Gateways bereitzustellen, die eine gemeinsame DB-Schnittstelle unterstützen. Eine solche Funktionalität wird z.B. von Sybase geboten (Kap. 3), wenngleich dort die Commit-Durchführung noch zum Großteil in der Verantwortung der Programmierer liegt. Die TP-Monitore CICS von IBM, UTM von Siemens oder Tuxedo von AT&T unterstützen eine verteilte Transaktionsausführung (durch Aufruf entfernter Teilprogramme) und realisieren das Commit-Protokoll bereits vollständig innerhalb der Systemsoftware.

Föderative DBS unterstützen einer weitergehende Kooperation zwischen den DBVS und nehmen damit zwangsweise eine geringere Autonomie der beteiligten DBVS in Kauf. Sie bieten an ihrer Schnittstelle zu den Anwendungen eine einheitliche Anfragesprache zum Zugriff auf heterogene Datenbanken, mit der es insbesondere auch möglich wird, innerhalb einer DB-Operation mehrere Datenbanken zu referenzieren (verteilte Join-Berechnung u.ä.). Externe Transaktionen dürfen nur auf solche Daten zugreifen, die in einem Export-Schema deklariert sind. Während bei lose gekoppelten föderativen DBS die Existenz mehrerer Datenbanken sichtbar bleibt, streben eng gekoppelte föderative DBS eine vollkommene Verteiltransparenz durch Definition von föderativen DB-Schemata an. Ein großes Problem dabei ist das Verbergen semantischer Heterogenität, was durch eine sogenannte Schemaintegration erreicht werden soll. Inwieweit diese Problematik in voller Allgemeinheit gelöst werden kann, ist noch Gegenstand aktueller Forschungsanstrengungen. Existierende Systeme wie Ingres/Star (Kap. 4) bieten nur eine eng begrenzte Unterstützung zur Schemaintegration.

Literatur

BEKK84 Bayer, R., Elhardt, K., Kießling, K., Killar, D.: Verteilte Datenbanksysteme - Eine Übersicht über den heutigen Entwicklungsstand. Informatik-Spektrum 7 (1), 1-19 (1984)

BLN86 Batini, C., Lenzirini, M., Navathe, S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18 (4), 323-364 (1986)

Br91 Braginsky, E.: The X/OPEN DTP Effort. Proc. 4th Int. Workshop on High Performance Transaction Systems, Asilomar, CA (Sep. 1991)

CP84 Ceri, S., Pelagatti, G.: Distributed Databases: Principles and Systems. McGraw-Hill, 1984.

Cy91 Cypser, R.J.: Communications for Cooperative Systems. Addison-Wesley (1991)

Da89 Date, C.J.: A Guide to the SQL Standard, 2nd edition. Addison Wesley (1989)

Du89 Duquaine, W.: LU 6.2 as a Network Standard for Transaction Processing. In: Lecture Notes on Computer Science 359, Springer-Verlag (1989)

Du90 Duquaine, W.: Mainframe DBMS Connectivity via General Client/Server Approach. IEEE Data Engineering 13 (2), 34-39 (1990)

HM90 Härder, T., Meyer-Wegener, K.: Transaktionssysteme in Workstation/Server-Umgebungen. Informatik Forschung und Entwicklung 5, 127-143 (1990)

HR83 Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys 15 (4), 287-317 (1983)

HR86 Härder, T., Rahm, E.: Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit. Informationstechnik 28 (4), 214-225 (1986)

Gr83 Gray, J.P. et al.: Advanced Program-to-Program Communication in SNA. IBM Systems Journal 22 (4), 298-318 (1983)

La91 Lamersdorf, W.: Remote Database Access: Kommunikationsunterstützung für Fernzugriff auf Datenbanken. Informatik-Spektrum (Das aktuelle Schlagwort) 14 (3), 161-162, (1991)

LMR90 Litwin, W., Mark, L., Roussopoulos, N.: Interoperability of Multiple Autonomous Databases. ACM Computing Surveys 22 (3), 267-293 (1990)

Me87 Meyer-Wegener, K.: Transaktionssysteme - verteilte Verarbeitung und verteilte Datenhaltung. Informationstechnik 29 (3), 120-126 (1987)

Me88 Meyer-Wegener, K.: Transaktionssysteme. Teubner-Verlag (1988)

ÖP91 Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Prentice Hall (1991)

Pa91 Pappe, S.: Datenbankzugriff in offenen Rechnernetzen. Springer-Verlag, Reihe "Informationstechnik und Datenverarbeitung" (1991)

Ra89 Rahm, E.: Der Database-Sharing-Ansatz zur Realisierung von Hochleistungs-Transaktionssystemen, Informatik-Spektrum 12 (2), 65-81 (1989)

Ra91 Rahm, E.: Klassifikation und Vergleich verteilter Transaktionssysteme, Univ. Kaiserslautern, Fachbereich Informatik, 1991

SB90 Simonsen, D., Benningfield, D.: INGRES Gateways: Transparent Heterogeneous SQL Access. IEEE Data Engineering 13 (2), 40-45 (1990)

Se91 Semich, W.: The Distributed Connection: DCE. Datamation, 28-30 (Aug. 1, 1991)

SL90 Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys 22 (3), 183-236 (1990)

Sp91 Spector, A.: Open, Distributed Transaction Processing with Encina. Proc. 4th Int. Workshop on High Performance Transaction Systems, Asilomar, CA (Sep. 1991)

SW91 Schek, H.-J., Weikum, G.: Erweiterbarkeit, Kooperation, Föderation von Datenbanksystemen. Proc. BTW-91, Informatik-Fachberichte 270, Springer-Verlag, 38-71 (1991)

Sy89a SYBASE SQL Toolset. Technical Overview. Sybase Inc. (1989)

Sy89b SYBASE SQL Server. Technical Overview. Sybase Inc. (1989)

Sy90 SYBASE Connectivity. Technical Overview. Sybase Inc. (1990)

Ta91 Takagi, A.: Multivendor Integration Architecture and its Transaction Processing. Proc. 4th Int. Workshop on High Performance Transaction Systems, Asilomar, CA (Sep. 1991)

Th90 Thomas, G. et al.: Heterogeneous Distributed Database Systems for Production Use. ACM Computing Surveys 22 (3), 237-266(1990)

Up91 Upton, F.: OSI Distributed Transaction Processing. An Overviw. Proc. 4th Int. Workshop on High Performance Transaction Systems, Asilomar, CA (Sep. 1991)