データベース・システム 68－5
（１９８８．１１．１７）

分散データベースにおける基本方式の開発

神田基博、　根岸和義、　石川博道、　山中治
（株）日立製作所

大型オンラインシステム間の水平分散型データベースシステムの開発を（１）負荷分散による処理能力の増大、（２）危険分散による信頼性の向上、を主な目標として実施した。本システムでは既存ユーザプログラム、およびデータベースの移行性を重視して、従来のデータベースシステムに分散機能を付加する方式を採用した。つまり、従来のデータベース管理システムとそのスキーマやデータベースはそのまま使い、ユーザプログラムにはそれら全てを統合した単一のデータベースシステムという見方を提供した。本システムではデータベースモデルはＮＤＬとし、ＯＳＩのＲＤＡ，ＣＣＲをもとにしたプロトコルを採用した。

IMPLEMENTATION OF DISTRIBUTED DATABASE SYSTEM

MOTOHIRO KANDA, KAZUYOSHI NEGISHI, HIROMICHI ISHIKAWA and OSAMU YAMANAKA
Hitachi Ltd.
1099, Ozenji, Asao, Kawasaki, Kanagawa, 215 Japan

We developed a distributed database system for a large scale online system. Main objects of this system are (1) larger processing power by load distribution, and (2) higher availability by autonomous distribution. Distribution feature is implemented as an independent software product from the database management system we already have, in order that user application programs and databases can be moved to the distributed environment easily. Database management systems with thier schemas and databases are kept unchanged. User programs see them as one integrated database system. We adopted NDL database model and protocols similar to the OSI RDA and CCR.

１．まえがき

近年のオンライントランザクション処理分野の発達には、目をみはるものがあり、計算機システムに要求される処理能力も急激に増大している。一般に、処理能力の不足は、より高性能な計算機ハードウェアの導入によって解決されるが、一部のユーザにおいては、単一のプロセッサからなる計算機システムでは対応しきれない処理能力が要求されるために、プロセッサ間データベース共用、分散データベースなどの技術を用いた、高度で大規模な複合計算機システムが構築されている。

この場合の分散データベースシステムとは、複数の計算機システム（ノード）を特殊なバスやＬＡＮなどの高速な通信パスで連結し、互いのデータベースを相互にアクセス可能としたものであり、その利点として、（１）負荷を分散することにより単一システムの処理能力を越えた負荷を処理することが可能、（２）処理の分散により一台のシステムが障害となっても残りのシステムで処理を続けることができる、という高性能、高信頼の二点があげられる。

分散データベースシステムに関しては、いくつかの商用システム[1)]が発表されている。本論文では、増大するオンライントランザクション処理に対応するための大型水平分散型データベースシステムについて述べる。

２．分散システムの導入に伴う問題点

一般に、新しいシステムへの移行は大変な作業であり、分散システムへの移行も大きなユーザの負担を伴うことが予想される。それには、新しいハードウェア機器の設置や、システム運用、保守の負担増加もあるが、最も大きいのは、ユーザのデータベースや、アプリケーションプログラムなどの既存のソフトウェア資産の移行のための作業である。ユーザに、スムースな計算機システムのグレードアップを提供するためには、分散システムは、以下の条件を満足するのが望ましい。

（１）データベースとユーザプログラムインタフェースの保証

業務を分散システムへ移行する場合に、ユーザにとって必要な作業は、業務を各ノード（計算機システム）へ分担する方法を決め、それに従ってデータベースの配置を定義することだけとする。すなわち、データベース、そのスキーマ定義及びサブスキーマはそのまま使用でき（本論文でデータベースはＮＤＬモデル[3)]とする。）、ＤＭＬ（データ操作言語）他のユーザインタフェースを既存のものと同じにすることで、ユーザプログラムは再コンパイルするだけで済むこととする。

またユーザプログラムにはレコードが分散配置されていることは全く意識させない（位置透過性を実現する）。

（２）非分散処理の混在

業務のなかには、分散システムへの移行が不必要なものもあり、分散システムと既存の非分散システムとは共存しなければならない。従来の非分散システムのためのユーザプログラムは、何の修正も無く今までどおりの性能で動作するのが望ましい。

これらの条件を満たし、かつ高性能、高信頼な分散データベースシステムの開発のため、我々は以下に示す設計方針を定めた。

（１）ユーザプログラムには全ノードのデータベースの総和を、一つのデータベースとして見せる。これを定義したものをグローバルスキーマと呼び、各ノードのスキーマと全く同じ論理的構造が定義できる。一方、各ノードのスキーマ、格納スキーマの定義は従来と同一とし、データベースの管理は既存のＤＢＭＳ（データベース管理システム）が行う。

（２）分散データベース管理システム（分散ＤＢＭＳ）のプログラムと既存のＤＢＭＳ（以下、ローカルＤＢＭＳと呼ぶ。）とのインタフェースは、従来のユーザプログラムとローカルＤＢＭＳとのインタフェースをほとんどそのまま使い、互いのシステムの依存度を減らしたソフトウェア構成とする。

（３）分散システムの導入の一つの目的が処理能力の向上であるため、その導入による性能の低下は最小限に抑える必要がある。通常、分散データベースシステムでは、ローカル（ユーザプログラムが存在するノードへの）アクセスが大部分であるようにデータベース設計をおこなう。従って、ローカルアクセス性能を従来のものと同程度に保つことが重要となる。

（４）通信パスや分散ＤＢＭＳに障害が発生しても、非分散処理は続行できることとする。さらに上記の場合、分散データベース処理を行うユーザプログラ

ムであっても、アクセスする目的ノードが障害ノード以外のノードであれば、正常に処理ができなければならない。また、各ノードは完全に対等とし、あるノードのプロセッサに障害が発生してもそれが他ノードへ与える影響は、そのノードのデータベースがアクセスできなくなることだけに限る。

上記方針に基づいて開発した分散データベースシステム[2)3)]について、次章で述べる。

3．分散データベースシステムの処理方式

3．1　システム構成

本システムの構成を図１に示す。

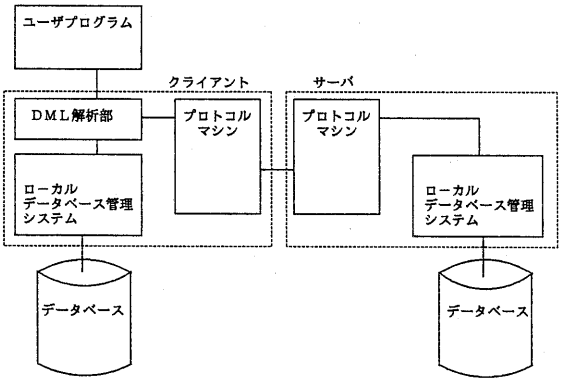

図１　分散データベースシステムの構成

分散ＤＢＭＳは大きく分けて、ＤＭＬ解析部と、プロトロルマシンとからなる。ＤＭＬ解析部はユーザプログラムからＤＭＬ要求を受け、それを後述する分散スキーマ定義等を使って解析して目的ノードを決定する。プロトロルマシンは、ＯＳＩのＲＤＡ[5)]とＣＣＲ[6)7)]をもとに、それを拡張したプロトコルを使用したトランザクション管理、障害回復等を行う。ユーザプログラムの存在するノード（クライアント）においてユーザプログラムからのＤＭＬ要求はまずＤＭＬ解析部が受け、目的ノードを決定する。自ノードのデータベースへのアクセス要求はそのままローカルＤＢＭＳに渡される。これに対して、他ノードのデータベースへのアクセス要求はプロトロルマシンに渡される。プロトロルマシンでは、この要求をデータの存在するノード（サーバ）へ転送し、当該ノードのプロトロルマシンがローカルＤＢＭＳをアクセスする。得られたデータは、逆のルートで元のユーザプログラムへ返される。

上記の構成をとることにより、以下の利点がある。

（１）既存の非分散処理の移行性確保

分散機能を組み込んだシステムにおいても、既存のユーザプログラムはそのまま使用可能であり、その場合ＤＭＬ解析部は通らず、直接ローカルＤＢＭＳへのアクセスが行われる。

（２）ローカルアクセス性能の低下防止

既存のローカルＤＢＭＳにおいてはＤＭＬ処理はユーザプログラムのアドレス空間及びタスクでそのまま実行される。分散ＤＢＭＳにおいてもＤＭＬ解析部はこのユーザプログラムのタスクでそのまま動き、目的ノード決定後それが自ノードなら同一タスクでローカルデータベースアクセスまで行う。一方、プロトコルマシンは独自のアドレス空間を持っているため、他ノードへのアクセスをするときは空間およびタスクの切り換えが必要である。

（３）障害の分離

ＤＭＬ解析部をプロトコルマシンと別アドレス空間のモジュールとすることにより、プロトコルマシン部分に障害が発生しても、自ノードへのデータベースアクセスは可能である。逆に、あるユーザプログラムの空間で障害が発生しても、プロトコルマシンは他のユーザプログラムへのサービスを続けることができる。

3．2　スキーマ構成

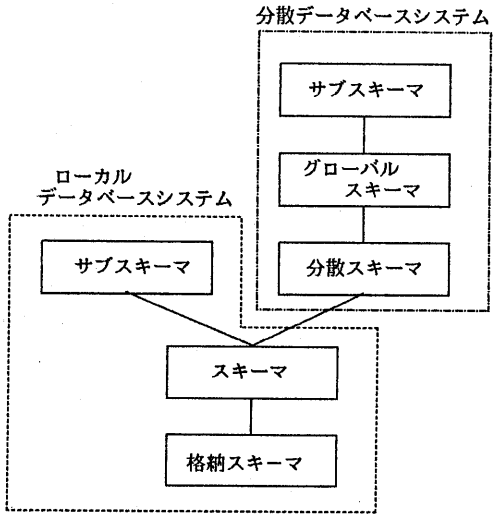

図２　スキーマ構成

本システムで使用するデータベースのスキーマの構成は、図２に示すようにＩＳＯのＮＤＬデータベースシステムを出典とし、ユーザビューであるサブスキーマ、データベースの論理構造を定義するスキーマ、およびそれ以下の物理構造を定義する格納スキーマ、より成っている。分散データベースのアクセスに使用するグローバルスキーマはローカルなスキーマと同一の論理的構造を定義することができるようになっており、グローバルスキーマに対してユーザビューとして定義したサブスキーマにより分散データベースのアクセスを可能とする。グローバルスキーマでは複数のノードにまたがるデータベースを定義することができる。分散スキーマは、このグローバルスキーマとローカルなスキーマとの対応付けを定義する。各ノードへの配置はレコードごとに定めることができるが、ＮＤＬのセットオカレンスは必ず一つのノード内になくてはいけない（オーナとメンバは別のノードには存在できない。）ため、分散配置を指定できるのは、セットの最上位のオーナレコードについてだけである。レコードの配置はコンポーネントの値についての条件式で指定するか、もしくは直接ノードを指定して行う。

分散データベースの定義の例を図３に示す。グローバルスキーマには各ノードのスキーマと同じく、レコードタイプＡ，Ｂ，Ｃ，Ｄと、それらの間のセット関係が定義されている。このグローバルスキーマを参照するサブスキーマは２つある。

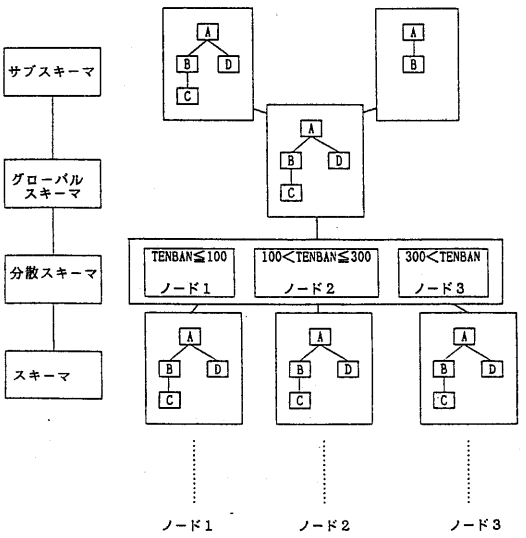

図３　分散スキーマ定義例その１

また、グローバルスキーマで定義されているレコードは、セットの最上位のオーナレコードであるＡのコンポーネントＴＥＮＢＡＮの値によりノード１，２，３に分散して格納されることが分散スキーマにより定義されている。

グローバルスキーマと分散スキーマをローカルなスキーマの上に定義することにより、以下の利点がある。

（１）既存の処理プログラムとデータベースの移行性の保証

グローバルスキーマをもとのローカルスキーマと同じ論理的構造で定義し、ユーザプログラムの使うサブスキーマを、そのグローバルスキーマを参照するように定義しなおす。これにより、ユーザプログラムは全く変えることなく全ノードのデータベースをアクセスできるようになる。

（２）ローカルスキーマの独立性

ローカルなスキーマから下位の定義部分に関しては、従来のスキーマをそのまま使用している。このことから、個々のノードのローカルなスキーマの生成、運用はノード毎に独立して行うことができる。

なお、ここでは各ノードが全く同じスキーマを持つ場合を取り上げたが、ノードごとに別のスキーマを持っており、それらの全てのレコードをグローバルスキーマとして定義することも可能である。この例を図４に示す。

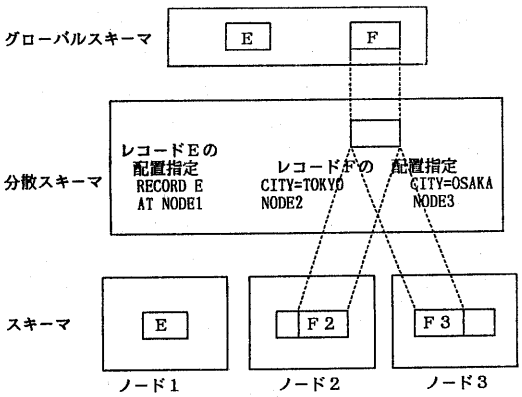

図４　分散スキーマ定義例その２

図４ではグローバルスキーマのレコードＥはノード１に、レコードＦはノード２と３にある。さらに、レコードＦはローカルスキーマのレコードＦ２とＦ

3から共通のコンポーネントを抜き出したものとして定義されている。このように、本システムでは既存のノードやスキーマをそのまま使ったボトムアップ的な分散データベースを構築することもできる。

3．3　DML（データ操作言語）の実行

本システムのローカルDBMSでは、ユーザプログラムのコンパイル時にDML文の構文解析を行い、結果をDMLオブジェクトと呼ばれるテーブルの形にしてユーザプログラムとバインドしておく。このテーブルには、DMLの種別や、探索条件式などが含まれているが、インデクス等の格納スキーマに関する情報は、一切入っておらず、いわゆるデータ独立性を実現している。ユーザプログラムの実行時にはこのテーブルをDBMSに渡して、処理を行う。分散機能を使うユーザプログラムの場合もDMLオブジェクトは、既存のものとまったく同じでありレコードの分散に関する情報は入れないようにした。ユーザプログラムからのDML要求を受けたDML解析部はDML文中の探索条件、目的レコードの分散配置定義、カーソル情報（カレントレコードを指す。3．5参照）等から、目的ノードを決定する。そして必要であれば、DMLオブジェクトを、ローカルスキーマとグローバルスキーマ間のマッピングに従って目的ノードのデータベースアクセス用に変換した後、ローカルDBMSに渡すか、DML実行要求プロトコルデータユニット（r－ExecuteDBL　PDU）に入れて、プロトコルマシンに渡し、送信する。（DBLとはデータベース言語の略。以下、OSI関連の用語、表記は最新のもの[5)6)7)]にならう。）

このとき、DML解析部及びプロトコルマシンはローカルDBMSにとってはユーザプログラムとして動作する。これにより、既存のDBMSプログラムへの影響を最少にでき、また柔軟で拡張性の高いソフトウェアとすることができる。このような比較的に疎なインタフェースを取ることによるオーバヘッドは、3．1に述べたようにDML解析部のモジュールをローカルDBMSのDML処理と共にユーザプログラムのアドレス空間で実行させることで小さくしている。

しかし、どうしても通常のユーザプログラムとは異なるインタフェースが必要になる場合がある。1つは、サブスキーマディレクトリの問題であり、もう一つはカーソル更新情報の問題である。3．4と3．5でこれを説明する。

3．4　サブスキーマ環境の実行時作成

通常、ユーザプログラムのDML要求の処理は、サブスキーマのディレクトリを参照して行われる。このディレクトリはユーザがDBMSに登録したサブスキーマから作成される。分散データベースシステムの場合も、例えばサーバ側のプロトコルマシンではローカルDBMSをアクセスするためにはサブスキーマディレクトリがなくてはならない。このため、オンラインでサブスキーマのディレクトリを作成する必要がある。しかも、一般にはサーバで必要なサブスキーマはクライアントのユーザプログラムが定義したサブスキーマとは同じでない。これは、グローバルスキーマとローカルスキーマとの間には図4に示すようなマッピングが可能であるためである。

このためクライアントノードのDML解析部は、サブスキーマ、グローバルスキーマ、分散スキーマをもとにして、各ノードのローカルDBMSが、DMLを処理するのに必要なサブスキーマをローカルスキーマ上に作成するために必要な情報を作成し、それをダイアログ（クライアントとサーバ間の連絡路）開始時に、資源オープン要求プロトコルデータユニット（r－Open　PDU）に入れて送信する。各ローカルDBMSは、この情報をもとにサブスキーマディレクトリを作成し、DMLを通常のユーザプログラムのDML要求として処理する。

上述のように、ローカルスキーマ上の3種類のスキーマをローカルスキーマ上のサブスキーマにあらかじめ変換しておくことにより、DML実行時の各種のマッピングのオーバヘッドは従来のローカルな処理におけるものと同一レベルに抑えることができる。

上記の、サブスキーマ作成のためのパラメタを作り、それからディレクトリを作るという操作は、比較的複雑で時間のかかる処理である。このため特に頻繁に使われるサブスキーマについては、システムの開始時にその参照するグローバルスキーマ、分散スキーマを解析して、関連ノードにパラメタを送信し、あらかじめサブスキーマディレクトリを作って

おくことにより、ＤＭＬ実行時のオーバヘッドを削減する機能を実現している。

３．５　カーソル更新情報

カーソルとは、ユーザプログラムの実行中に、最も最近にアクセスされたレコードオカレンスを常に指しているものでＮＤＬでは非常に重要な役割を持っている。分散データベースシステムでは、レコードオカレンスは複数のノードに存在するため、従来のローカルＤＢＭＳの管理するカーソルに加えて、現在ユーザプログラムがどのノードのレコードオカレンスをアクセスしているかという情報を管理する必要がある。これをグローバルカーソルと呼び、クライアントノードのＤＭＬ解析部がその管理を行っている。

グローバルカーソルはローカルＤＢＭＳのカーソルと同じ数だけ存在する。すなわち、セションカーソル、サブスキーマに定義されたレコードの数だけのレコードカーソルおよびサブスキーマに定義されたセットの数だけのセットカーソルがある。これらのグローバルカーソルは空値（どのノードも指していない。）であるか、現在ユーザプログラムによって指されているノードの識別子を値として持つ。なお、ＤＭＬ解析部がＤＭＬ要求を解析するとき、そのＤＭＬを実行する目的ノードは、目的レコードタイプに対応するグローバルレコードカーソルの指しているノードに決められることが多い。目的ノードの決定のため、どのグローバルカーソルを参照するかはローカルＤＢＭＳにおいて、目的レコードオカレンスを決定するためのカーソルの参照の仕方と完全に一致させており、またグローバルカーソルの更新の仕方もローカルなカーソルに関するＮＤＬの仕様どおりとしているため、ユーザプログラムにはカーソルのふるまいは、集中型データベースシステムの場合と全く同じに見える。

しかし、ある場合にはＤＭＬ解析部は、ローカルＤＢＭＳからカーソルの更新情報を受け取らなければ上に述べた、正しいグローバルカーソルのメンテナンスができない。ＥＲＡＳＥ文を例に、それを説明する。

図５はノード２における、レコードＡのオカレンスの様子と、ローカルＤＢＭＳのカーソル及びクライアントノードにおけるグローバルカーソルを示す。

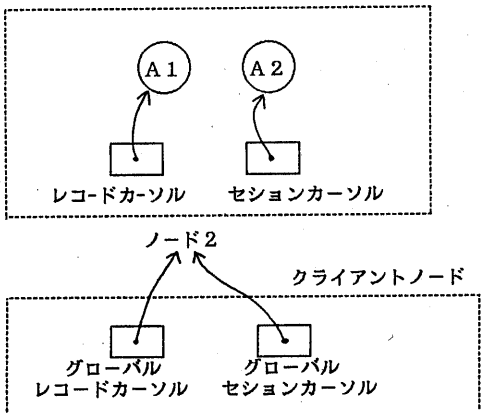

図５　ＥＲＡＳＥ文を実行する前の
レコードカーソルとセションカーソル

ローカルＤＢＭＳのレコードカーソルはＡ１というレコードオカレンスを、セションカーソルはＡ２を指している。また、グローバルカーソルは、レコードカーソル、セションカーソルともにノード２を指している。

この時、ユーザプログラムがＥＲＡＳＥ文を発行すると、その要求はグローバルレコードカーソルの指すノード２へ送られ、ローカルＤＢＭＳはレコードカーソルの指すＡ１をデータベースから削除する。この後、Ａ１を指していたレコードカーソルには空値が設定されるが、全く別のレコードを指していたセションカーソルは更新されない。クライアントノードのＤＭＬ解析部は、ノード２のローカルＤＢＭＳのカーソルの更新にならってグローバルレコードカーソルに空値を設定し、グローバルセションカーソルは今までどおりノード２を指した状態とするべきである。しかし今の場合、各グローバルカーソルに空値を設定するかどうかは、ローカルＤＢＭＳのカーソルがどのレコードオカレンスを指していたかに依って決まるため、ＤＭＬ解析部には判断出来ない。

上記のようにグローバルカーソルの更新の有無がローカルＤＢＭＳの管理するカーソルおよびレコードオカレンスの状態に依存するという事態は、ＥＲＡＳＥ文と特殊なＦＩＮＤ文においてのみ発生する

ことがわかっている。その場合にはＤＭＬ解析部は、ローカルＤＢＭＳからカーソルがいかに更新されたかという情報を受け取る必要があるため、ローカルＤＢＭＳはＥＲＡＳＥ文とＦＩＮＤ文のｒ－ＥｘｅｃｕｔｅＤＢＬの応答ＰＤＵに、更新されたカーソルの識別子を設定しクライアントノードのＤＭＬ解析部に渡すようにしている。

３．６　ＯＳＩのＲＤＡ，ＣＣＲとの関係

最後に、本システムにおける分散データベースアクセスの典型的な処理フロー、その特徴およびＯＳＩとの違いについて説明する。

表１に現在ＯＳＩのＲＤＡ[5]で規定されているサービスの一覧を示す。

表１　ＯＳＩ　ＲＤＡ　サービス一覧

ＲＤＡサービス	機能概要
r-Associate	アソシエーションを確立する。
r-Release	アソシエーションを終了する。
A-ABORT	アソシエーションを直ちに中断する。
A-P-ABORT	ACSEプロバイダによるアソシエーション中断。
r-Open	資源を要求する。
r-Close	資源の使用を終了する。
C-BEGIN	トランザクションを開始する。
C-PREPARE	コミット準備を要求する。
C-READY	コミットを受け入れる。
C-REFUSE	コミットを拒否する。
C-COMMIT	コミットを要求する。
C-ROLLBACK	ロールバックを要求する。
C-RESTART	障害の後で同期を取りなおす。
r-ExecuteDBL	DBL(データベース言語)文を実行する。
r-DefineDBL	複数のDBL文からコマンドを定義する。
r-InvokeDBL	コマンドを実行する。
r-DropDBL	コマンドの定義を無効にする。

図６は、本システムにおける、クライアントからサーバへＤＭＬを１つだけ送って終了する処理についての通信メッセージのフローを示した。通常は、ｒ－ＥｘｅｃｕｔｅＤＢＬとその応答は何度か繰り返されるし、１つのダイアログのなかでトランザクションは何度も実行される。これらの場合、ｒ－ＥｘｅｃｕｔｅＤＢＬが単独で、もしくはトランザクションの開始時にはＣ－ＢＥＧＩＮとともに送信される。

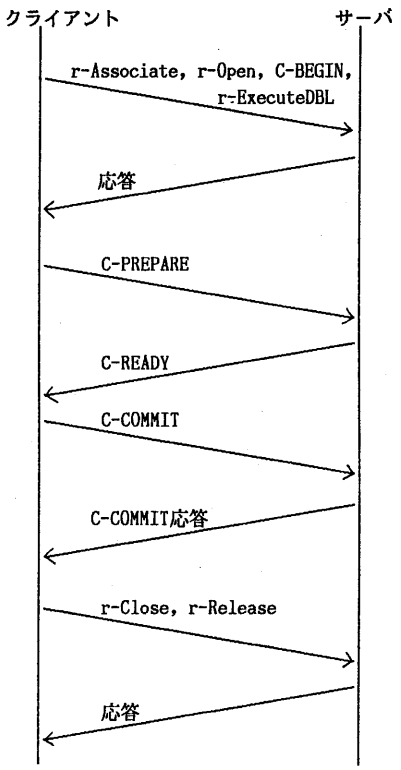

図６　処理フロー

図６に示すように、ｒ－Ａｓｓｏｃｉａｔｅ，ｒ－Ｏｐｅｎ，Ｃ－ＢＥＧＩＮ　およびｒ－ＥｘｃｕｔｅＤＢＬ　のＰＤＵは通信メッセージの回数を削減する為に、一度にまとめて送られる。サーバ側ではこれら要求を順に処理し、最後に各サービスの応答をまた１つにまとめてクライアントへ送信する。ただし、途中で何かエラーが発生した場合は（リソースのオープンができなかったなど）、そこで処理を中断し、エラーの詳細をクライアントに返す。このような、複数のサービスをまとめる機能はＯＳＩには無い。しかし上記の例の場合、最初に他ノードへＤＭＬを発行するときに、応答も含めて合計８つ必要であった通信メッセージがＰＤＵを連結することで２つで済むというように、性能上の効果が大き

いため、この方式を採用した。

ｒ－Ｏｐｅｎには、３．４で述べた、サブスキーマディレクトリをサーバ側で作成するための情報（使用するレコード、セット等の宣言）の他に、プログラムが必要とするデータベースバッファの量や、インテントロッキング（将来必要となるリソースをあらかじめ共用モードでロックしておく。）のための情報などが含まれる。こうしてオープンされたリソースの状態はトランザクションが終わっても引き継がれる。

本システムにおいてダイアログは、上記ｒ－Ｏｐｅｎで確保される特定のプログラム環境と結び付いたものとして実現している。そして、同じプログラム実行環境を使うトランザクションは、同一のダイアログの上で何度も繰返し実行することができる。定型的な業務の場合、関連ノードへダイアログを予め設定しておけば、他ノードのデータベースへのアクセス要求が発生したときに必要な処理オーバヘッドは、多くても新しいトランザクションの生成だけであり、オーバヘッドの少ない分散処理が実現できる。

４．まとめ

オンライントランザクション処理への適用を主に考えた大型水平分散型データベースシステムの基本方式を検討し、高性能、高信頼かつ、既存ユーザプログラム、データベースとの共存を保証する方式を提案した。本システムの特徴は、下記の通りである。

（１）既存データベースシステムへの分散機能の外付けによる既存ローカル処理ユーザプログラム、データベースの移行性、性能の保証と、障害の分離、局所化。

（２）既存スキーマと同一構造のグローバルスキーマと、この両者の間をマッピングする分散スキーマの導入によるローカルスキーマの独立性確保と、既存ユーザプログラムの分散データベースアクセスへの移行性保証。

今後の課題として、分散データベースのオンライン中の構成変更およびそのためのスキーマ定義などの配布の方式の検討、そしてリレーショナル型データベースシステムへの本方式の拡張などがある。

参考文献

１）市場形成に向けて製品化始まる分散データベース、日経コンピュータ別冊ソフトウェア（１９８８）ｐｐ．１４５－１５９．

２）石川博道　他、データマネジメントシステムＸＤＭ（６）、情報処理学会第３６回全国大会４Ｅ－６．

３）根岸和義　他、データマネジメントシステムＸＤＭ（７）、情報処理学会第３６回全国大会４Ｅ－７．

４）ISO/IS 8909 : Information processing systems - Database language NDL (1987).

５）ISO/JTC1/SC21 DP9579: Information Processing Systems - Remote Database Access(R D A) (July 1988).

６）ISO/IEC/DIS 9804.2 : Information Processing Sysyems - Open Systems Interconnection - Service Definition for the Commitment, Concurrency and Recovery Service Element (1988).

７）ISO/IEC/DIS 9805.2 : Information Processing Systems - Open Systems Interconnection - Protocol Specification for the Commitment, Concurrency and Recovery Service Element (1988).