データベース・システム 65－2
（１９８８．５．１９）

データベース管理システム G-ＢＡＳＥ分散化のためのデータ管理方式

平岡 昭夫, 飯沢 篤志
(株) リコー ソフトウェア研究所

本稿は、拡張リレーショナル・モデルを使用したデータベース管理システムG-ＢＡＳＥの分散化の方式について述べる。特に、部分テーブルを木構造（テーブル分割木）で管理することによって、垂直分割／水平分割／複製が同時に適応できるデータ管理方式を考案し、この方式における問い合わせの処理について考察する。この方法は、エンドユーザに対する分散透過性と、データベース設計の柔軟性を同時に提供できる。

Design of Data Distribution Scheme for Database Management System G-BASE

Akio Hiraoka, Atsushi Iizawa
Software Research Center
Ricoh Co., Ltd.

1-17 Koishikawa-cho 1-Chome
Bunkyo-ku, Tokyo, 112, Japan

This paper investigates design of a distributed system for G-BASE which is a DBMS based on an extended relational model. In particular, we introduce a data distribution scheme that manages partitioned tables with a tree structure (Table Partitioning Tree). This realizes vertical, horizontal partitioning and replication simultaneously. A query processing mechanism according to this method is also investigated. It provides transparency for end-users and flexibility of database design.

はじめに

グラフ・データ・モデル [1] とは、従来の関係データ・モデルに対し、リンクをいう概念を付加することによって、レコード間の関係を表現することを可能にした拡張リレーショナル・データモデルである。

G-BASEはこのグラフ・データ・モデルをデータモデルとして採用した本格的データベース管理システム(DBMS)として(株)リコー, ソフトウェア研究所で開発されたシステムである。現在 G-BASEは UNIX をベースとしたワークステーション (WS) および、ミニ・コンピュータ向けの DBMS として出荷されている。

昨今の WS は、CPU の性能、ネットワーク機能、グラフィックス機能等が強化され、実用に堪えうるものが低価格で入手できるようになってきた。しかし、個々の WS に追加できる二次記憶容量(磁気ディスク装置)に関しては、まだまだ十分とはいえない。そこで、複数の WS 間でファイルを共有したり、1つのデータを複数のサイトに分割し管理する機能が必要となり、多くのシステムが商品化されている。

DBMS の分野でも、データベースを複数の独立したシステム間で共有したり、1つのデータ実体を複数のマシンに分割して保持することを可能にする分散 DBMS に対する期待が高まってきている。

(株)リコーでも G-BASEに対して、分散化の機能拡張を検討している。現在は、将来の本格的分散 DBMS 実現を目標として、要素技術の研究・試作を行なっている。本稿では、G-BASE分散化の方式検討を行ない、本格的 DBMS 実現のための分散データ管理方式について考察する。

1章ではG-BASE分散化の方式について述べる。2章では本格的分散データベース実現のための検討項目について記述する。3章では、分散DBMSに必要なデータ分散/複製の方式について紹介する。4章は3章で紹介した分散データ管理方式に基づく、問い合わせ処理の実現について検討する。

テーブルの分割/複製については、従来より多くの研究・開発が行われてきた。SDD-I [3] はテーブルの垂直/水平分割を最初に導入した分散 DBMS の試作品である。System R* [4] は、ユーザに位置透過性は提供しているが、動的にテーブルの分割/複製を行なうことはできない。テーブル複製の問題は、コンピュータ・ネットワークにおけるファイル・アロケーションの問題 [5] で議論されている。複製データの整合性を保つための更新同期メカニズムついても多くの手法が提案されている [6][7]。現在までの多くの研究でも、テーブル分割と複製を同時に実現し、柔軟なデータベース設計を可能にしたシステムはあまりない [8]。

1. G-BASE分散化の方式

既に商品化されているG-BASEに対して分散機能の拡張を行なうため、いくつかの段階に分けて少しづつ改良を行なう方針である。

分散化の環境として、複数の WS が高速の LAN によって接続されている環境を想定している。各 WS は独立していて、それぞれネットワークへのインタフェースを提供しているものとする。

G-BASEを分散化するため、以下の方法を検討する。

(1) NFS を利用した方法 [9][10]

DBMS で管理されているデータベース (DB) の物理ファイルを、他のサイトから NFS (Network File System) [11] を介してアクセス可能なファイルシステム上に置くことで、遠隔地サイトから DB へアクセスできるようにする方法である。G-BASEの本体をほとんど変更せず、分散環境に適応させるという消極的な方法である。(図1)

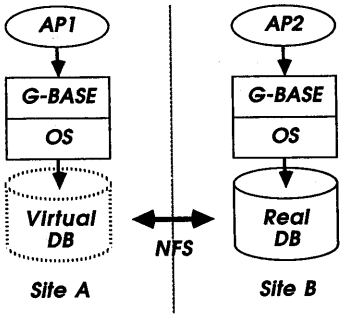


図1 NFS を利用した方法

この方法は、OS が NFS 機能を提供していることを前提する。(米) サンマイクロシステムズが開発し業界標準となりつつある NFS を採用する予定である。

NFS が実現しているローカル・ファイルとリモート・ファイルに対するアクセスの透過性によって、WSのユーザはリモート・サイト上の DB にサイトを意識することなくアクセスできるようになる。

この方法はリモート・サイトへのアクセスの回数が非常に多いので、ネットワークやサーバ・マシンの負荷の増大に伴って、リモートDBへのアクセスの性能が落ちることが問題となる。

(2) リモート・データベース・アクセス

G-BASEでは、ユーザから起動された応用プログラム (AP) からの問い合わせを受けとり、データベースにアクセスするプログラムをデータ・マネジャ (DM) と呼ぶ。AP から DM が起動され (逆の場合もある) 2つのプロセス間には PIPE による通信路が設定される。AP からは問い合わせが、DM からは検索結果が、共に LISP の S 式の形で送られる。この G-BASEの特徴を生かして、リモート・データベース・アクセスを実現する。AP は必要な DB にアクセスするため、DB が存在するサイト上で DM を起動する。次に AP-DM 間に通信路を設定して、DB に対する問い合わせを行なう。DM は常に DB の実体が存在するホスト上で実行されるものとする。(図2)

この方式では1つの AP が同時に複数サイトの DB(複数)にアクセスできるようになる。実現のためには以下の項目について検討しなければならない。

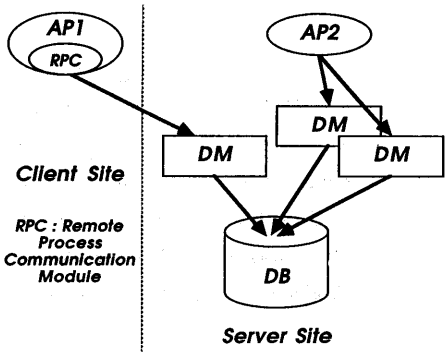


図2 リモート・データベース・アクセス

a．AP-DM 間の通信のオーバーヘッドを軽減する。

b．DM 起動のオーバーヘッドを軽減する。

c．ネーム・サーバの実現

d．アクセス権管理機能の検討

この方式でもデータを保持しているのはサーバ・マシンである。サーバ・マシンの障害によってすべてのWSはデータベースを使用できなくなるため、WSの独立性は期待できない。DM プロセスがリモートサイト側にあるため、サイト間のデータ転送量は格段に減少する。また、複数の DM を並列に実行することも可能になり性能向上のための最適化を考慮できる。

（3）ファイル単位での分散

G-ＢＡＳＥではユーザの問い合わせに対して、最終的にはいくつかのファイル・アクセス・ライブラリの関数を実行する。このライブラリのことをアクセス・メソッド (AM) モジュールと呼ぶ。

AM は、B 木ファイル、プレーン・ファイル等を均一に扱うためのインタフェースを提供している。この AM モジュール内に遠隔地サイト上のファイルに対してアクセス可能なアクセス・メソッドを追加する。これにより、遠隔地のファイルにアクセスできるようになる。ローカルな AM もリモートの AM も透過的に扱えるように、『仮想アクセス・メソッド』と呼ばれるインターフェースを用意する。DM は仮想アクセス・メソッドを介して、遠隔地の『AM サーバ』に対して、ファイル操作を要求する。(図3)

データ実体だけでなく、アクセス・メソッドを含めたファイルシステムを遠隔地ホストに置くことによって、サイト間のデータ転送量を軽減することができる。DB の実体（物理ファイル）を複数のサイトに分散させることが可能になる。

データベース単位（または、テーブル単位）で分散させることによって、一部のサイトの障害を避け、データベース・システムの一部だけで運用することが可能になる。最小限のデータを自サイト上に置くことで、WS ごとの独立性を確保することができる。

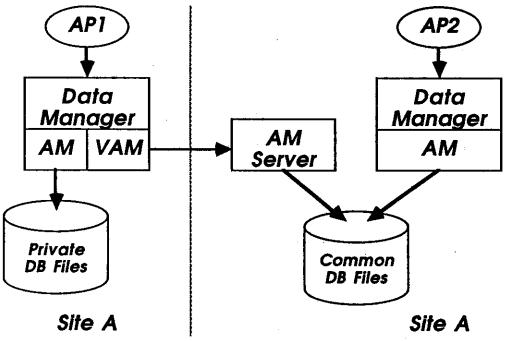


図3 ファイル単位での分散

（4）データ定義情報の分散化

（1）～（3）の方法では、データ定義に関する情報はすべてのサイト間で共通である。(3) では各サイトで同じデータ定義情報を重複して持つ。ここでは、データ定義情報の分散を検討する。

複数のサイトからデータベースの一部を使用する場合、すべてのデータ定義情報をサイトごとに保持しておく必要はない。そこで、必要な情報だけを各サイトで分散管理する方式を検討する。G-ＢＡＳＥでは、データ定義情報はデータ辞書と呼ばれる特殊なデータベースで管理されている。データ定義情報の分散化はデータ辞書を分散させることによって実現される。

データ辞書の分散化も通常のデータベースの分散と同様の手法を使用するが、重複データが多いので更新時の同期が課題となる。また、アクセスが非常に頻繁に起こるので、危険でない部分についてはロックを使用しない更新を検討しなければならない。

データ定義、データ実体の両方をローカルとグローバルに分けて管理できるようになると、共有データを使用しながら、他のサイトの障害等と全く独立に自ホストの DB にアクセスができるようになる。

（1）～（4）の方法は、G-ＢＡＳＥの現在の版の改良の形で開発を進めていく。性能向上と障害回復が重要課題となる。以下の章では、分散化の最終目標である本格的分散 DBMS の実現について検討する。

2．本格的分散データベース

G-ＢＡＳＥを本格的分散データベースとして実現する場合、検討しなければならない項目は多いが、主なものは以下の５項目である。

a．データ分割・管理方式

G-ＢＡＳＥではデータ実体がデータベース／レコード／フィールドの単位に分類される。さらに、レコード値間の関連を示すためのリンクという情報も存在する。すべてのデータ定義はデータ辞書と呼ばれる特殊なデータベースで管理されている。

柔軟な分散データベースの設計を可能にするために、レコード単位での垂直分割、水平分割、サイト間複製を同時に可能とする DBMS の実現を検討する。

１サイトの１つのデータベースで管理されていたデータ辞書も複数のサイトで共有しなければならない。さらにはネットワーク全体で共通の情報と各サイトごとに固有のデータ定義情報に分けて管理する必要が生ずる。

以下、同じ分散 DBMS の管理下にあるサイト、データベース、ネットワークを含めて『分散 DB 環境』と呼ぶ。

b．問い合わせ処理／最適化

データベースの実体が複数のサイトに分散している場合、エンドユーザからの問い合わせは、問い合わせ処理系で処理され、各サイト上のデータに対する問い合わせに変換され、サイトに送られる。サイト別の問い合わせのことを『下位問い合わせ』と呼ぶ。処理系は、各サイトからの結果を統合してユーザに問い合わせの結果として返す。

エンドユーザに対する分散データベースの位置透過性を実現するのが問い合わせ処理系である。問い合わせに対して、完全な結果を得るため必要最小限度の『下位問い合わせ』を生成することを目標とする。

問い合わせ処理系では、性能向上、障害回避のための最適化を行なうことが重要課題になる。性能向上のために下位問い合わせの同時実行を計画し、局所性の判定を行なう。他のサイトの障害時には、必要十分な結果が得られる範囲で複製データを使用し、障害サイトを避ける。

ユーザに対する透過性を実現するためには、問い合わせ処理系が大きな役割を担う。異種 DBMS の結合を考える場合、下位問い合わせとして他の DBMS への問い合わせを生成することも期待される。

c．同時実行制御

現在のG-ＢＡＳＥには、すべてのデータ実体に対する同時実行を制御するためのプロセスとして、ロック・マネジャ (LM) と呼ばれる常駐プロセスが存在する。すべて DM は LM に対してロック要求を出し、許可された DM のみがデータの変更操作を行なうことができる。

分散 DB 環境では、複数のマシン上にデータ実体が存在している上、すべてのサイトが稼働しているという保証はない。したがって、すべてのデータ実体に対して１つのプロセスが同時実行制御を行なうことは現実的ではない。

複数の LM が共同して、分散 DB 環境での各データ実体に対する同時実行制御を行なうモデルを検討しなければならない。たとえば、データ実体の存在するサイトごとに LM を起動しておき、LM は自サイトのデータに関してのみ同時実行制御を行なう。データの変更操作の大半が、自サイトのデータに対して行なわれるような場合、このようなモデルも有効と思われる。さらに、分散 DB 環境全体にわたるようなデッドロック検知のメカニズムを実現しなければならない。

ロックを基本とした同時実行制御以外に、複製データ等に対しては時刻とバージョン管理による更新同期を検討する。

d．障害回復

分散 DB 環境では、ネットワーク障害による孤立サイトの回復が重要な課題となる。障害後再起動されたサイトのデータ定義情報や複製データは、ネットワーク上の他のサイトのデータと同期が取れていないことがある。各サイトに複製データを他サイトと合わせるためのメカニズムを用意しなければならない。

ダンプ／ログについても、サイト個々で実行するダンプを統合してネットワーク全体のデータベースを回復するためのメカニズムを提供することが必要になる。ネットワーク全体にわたり定期的にダンプをとるようなサービス・プロセスを用意するのも有効である。

e．データベース設計

分散 DB 環境でのデータベース設計は、１サイトでのスキーマ設計に比べて、はるかに複雑になる。また、設計の良否がデータベースへのアクセスの性能に大きく影響する。そこで、データベース設計者のための支援ツールを実現する必要がある。また、各サイトのデータに関する参照・更新の状況を記録し、データベース再設計時に、フィードバックできるようにしなければならない。

ここでは検討項目として挙げていないが、実現したデータベース・システムがユーザに対して実用的な性能でサービスできるかどうかが大きな課題となる。

『同時実行制御』『障害回復』『データベース設計』は、問題提起にとどめる。以下の章では上記の検討項目のうち『データ分割／複製方式』、『問い合わせ処理』について考察する。

3．データ分散／複製方式

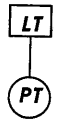
リレーショナル・データモデルにおけるリレーションをテーブルと呼ぶ。１つのテーブルは複数のレコードが集まったファイルに相当する。

a．物理テーブル、論理テーブル

データ実体の格納されているテーブルを物理テーブルと呼ぶ。論理テーブルは物理テーブルへのユーザからのインタフェースとして使用される疑似テーブルである。論理テーブルはユーザの定義したレコード型１つに対応する。分割／複製が起こっていない場合、物理テーブルと論理テーブルは１対１に対応している。エンドユーザのデータ操作は、すべてこの論理テーブルに対する操作として提供される。

LT：論理テーブル

PT：物理テーブル



論理テーブル、物理テーブルそれぞれが１レコード中に含まれるフィールドのリストを保持している。２つのリストの間で、ユーザ側から見えるフィールド（論理フィールド）とデータ実体の中でのフィールド（物理フィールド）のマッピングが行なわれる。

物理テーブルには、レコード値を一意に識別するフィールドが設定されているものとする。G-ＢＡＳＥの場合、プライマリ・キーがこのフィールドに相当する。もしプライマリ・キーを設定していなければ、システムが URI (Unique Record Identifire) を付加する。

データ定義情報には、論理テーブル、論理フィールドを含むユーザ定義のスキーマと、物理テーブル以下のデータ管理情報がある。２つのデータ定義情報を明確に区別し、データ管理情報をエンドユーザから隠すことにより、分散データベースの透過性を実現することができる。エンドユーザはデータ実体の位置や形式にこだわることなく、与えられたスキーマにしたがって操作を行なうことができる。

b．テーブル分割木

物理テーブルの分散・共有の方式として、テーブルに対する垂直分割、水平分割、複製の３つの方式を同時に適用することを検討する。

【垂直分割】
テーブルを列方向、つまりフィールド単位で分割する。１のフィールドが２つ以上のテーブルに重複するような分割は許さない。ただし、レコードを識別するためのフィールドだけは、重複して各物理テーブルに付加される。

【水平分割】
テーブルに含まれるレコード値によって、行方向に分割する方式。テーブルを分割するために、レコードを振り分ける条件式が与えられる。

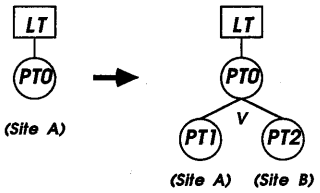
【　複製　】
まったく同じテーブルを２つのサイトに置くことによって、データへのアクセス速度を向上させる方式。

システム内部では、テーブルの分割／複製の情報を木構造で保持する。木構造はオリジナルの物理テーブルをルートノードとし、分割／複製の過程が各ノードで表される。リーフノードの物理テーブルのみが実際のデータを指している。中間のノードはすべて仮想の物理テーブルである。

この物理テーブルの木構造のことを『テーブル分割木』と呼ぶ。木構造の変更がテーブルの分割／複製／連結を引き起こす。

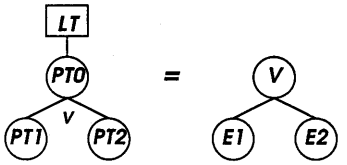
c．テーブル分割木の変更

オリジナルな物理テーブルに対して、垂直分割を行なった時の木構造の変更を下図に示す。サイト A の物理テーブル PT0 がサイト A の PT1 とサイト B の PT2 に分割されたことを示している。V は垂直分割された事を表す。物理テーブル PT0 はもはやデータ実体を持たない仮想のノードとなる。リーフノード（または、初期状態から）の分割／複製では、水平分割の場合 (H) も複製の場合 (R) も同じ構造になる。



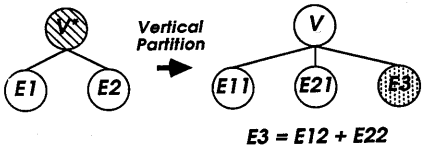
『垂直分割』

以下では論理テーブルの存在を省略する。水平分割された仮想ノードを H 、垂直分割されたノードを V 、複製が起こったノードを R で表す。リーフノードでデータ実体を指しているノードを E で表す。* が付加されているノードを操作の対象ノードとする。



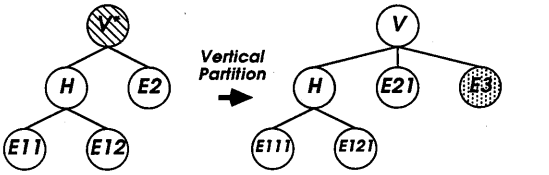
分割の逆操作として、『連結』操作が必要である。連結は対象ノードをルートとする部分木が表している物理テーブルを再構成し、１つのノードを生成する。下図は垂直分割ノード V を連結して、物理テーブル E を再構成したところである。

既に分割／複製が起こっているテーブルにさらに操作を加えることで木構造は成長する。垂直分割を繰り返す場合は、木は横方向（リーフの方向）に成長する。



たとえば、上図では V のノードに新たに加えられる条件で E1 および E2 (一方だけの場合もある) を分割して、E3 を生成している。

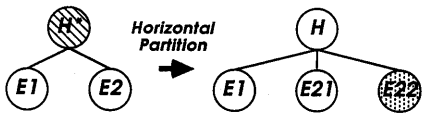
下図のように、下位ノード (E1) がさらに水平分割されている場合は、水平分割ノードへの連結操作を行ない E3 に適当なフィールドを抜き出す。抜き出されたフィールドは元の水平分割ノードからは削除される。



V (または H) のノードに対する分割を行なった場合、基本的には以下のように処理される。

(1) 指定ノードをルートとする部分木を統合した物理テーブルを再構成する。
(2) 再構成したテーブルを分割する。
(3) 分割によって新しく作られた物理テーブルを新しいノードとしてテーブル分割木に登録する。
(4) 元の部分木から、新しいテーブルに移ったフィールド（または、レコード）値が削除される。

水平分割を繰り返す場合は、下位ノードのいずれか一方のみを分割する場合と、両方が分割される場合がある。下図では E2 を水平分割して、E22 を新たに生成している。



『水平分割×２』

E1 と E2 の両方を分割する場合、H ノードの下位に E11 E12, E21, E22 の４ノードができる。

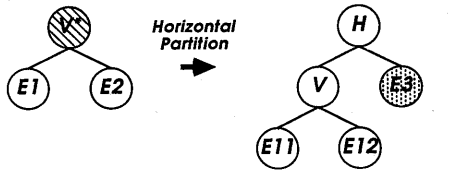
異なるタイプの分割操作が繰り返される場合は、以下のように処理され、新しいノードが付加される。

(1) 指定ノードの部分木の再構成テーブルを生成し、分割する。

(2) 分割によって新しく作られた物理テーブルと、元の物理テーブルの上位に分割を表す仮想ノードを追加する。

(3) 元の部分木から、新しいテーブルに移ったフィールド（または、レコード）値が削除される。

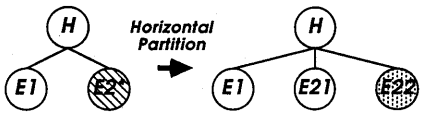
下図に V で示される垂直分割ノードに水平分割を実行した例を示す。



『垂直分割×水平分割』

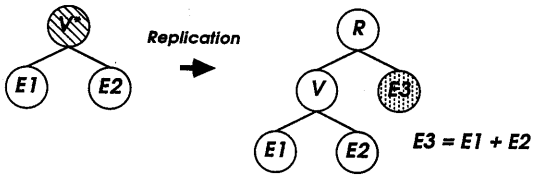
リーフノードを分割する場合で上位のノードと同じタイプの分割を行なう場合は、新しく生成されるノードは１つ上位

のノードとなる。



たとえば、上図では E2 のノードが水平分割された場合には、E2 のノードが分割されて E21 と E22 のノードが H の下位ノードとなる。

複製の場合には、指定の物理テーブルの再構成テーブルと同じテーブルを生成し、2つのノードの上位に複製を表す仮想ノードが追加される。



『垂直分割×複製』

上図は、V のノードに対して複製操作を行なって、E1 と E2 を統合した E3 のノードが生成された状態を示している。V と E3 のノードの上位に R ノードが追加されている。

d．分割木の管理

テーブル分割木に対して各ホストから論理テーブルを定義できる。オリジナルのテーブルの任意の部分（フラグメント）のみを他のサイトに移したり、開放したりすることが分割木に対する操作で可能となる。

テーブル分割木の情報はエンドユーザが知る必要はないが、関連するサイトはこの情報を保持していなければならない。さらにデータ実体は1つなのでテーブル分割の情報もすべてのサイトで共通でなければならない。

4．問い合わせ処理

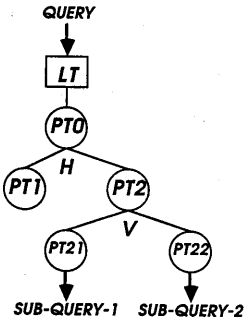
ユーザから論理テーブルに対して与えられた問い合わせ(QUERY) はテーブル分割木をたどり、細分化される。結果として、実存するテーブルに対する下位問い合わせ(SUB-QUERY) に変換される。問い合わせ処理系は QUERY の解析を行ない実テーブルに対する最適な SUB-QUERY を生成する。

テーブル分割木上での最適化は、論理テーブルに与えられた QUERY に対する処理が『局所化』できるかどうかを評価することである。局所化の評価とは、問い合わせがテーブル分割木上の部分木に対する処理だけで完了するかどうかを判断することである。局所化が進むほど、生成される SUB-QUERY の数が減る。

以下では、テーブル分割木に沿って問い合わせ処理が正しく行なわれるかどうかについて考察する。

a．検　索

ユーザから論理テーブルに対して与えられた QUERY に対する変換処理は、論理テーブルが定義されている物理テーブルから開始される。



(1) QUERY は分割木にしたがって分割される。リーフノードに近づくほど検索条件は細分化される。
(2) QUERY の分割がリーフノードに達した段階で実テーブルに対する SUB-QUERY を生成する。
(3) 生成された SUB-QUERY は各サイト上で実行され（可能な限り同時に）その結果が返される。
(4) 結果が揃った段階で分割木を逆向きに（ルート方向に）たどり結果を統合する。
(5) 最後に、得られた結果をユーザに返す。

以下、分割木上の各ノードでの QUERY の分割、局所化判定メカニズム、結果の統合の手続を述べる。

【水平分割ノード】

与えられた QUERY をそのまますべての下位ノードに渡すだけでよい。

QUERY の検索条件とテーブルの分割条件が合致する場合、問い合わせ処理を局所化できる。→ 局所化判定条件1

下位ノードから得られた結果を、単純に結合して上位ノードに渡す。オーダ指定があるような場合には、その場で実行する。

【垂直分割ノード】

下位ノードに対する検索文をそれぞれ生成／実行し、結果をフィールド値（レコード識別フィールド、または URI を含む）の集合として得る。得られたフィールド値から必要なレコードを生成する。

以下に、検索条件にしたがって、どのように検索が実行されるかを記述する。

(1) 検索の条件に関連するフィールドを含む各物理テーブルに対して問い合わせを実行し、条件に適合するフィールド値（複数）を『結果テーブル』に得る。（この操作は並列に処理できる）

例）field1 > 10 → PT1
field2 <= 0 → PT2

得られた結果テーブルのみで結果が得られる場合は終了。→ (4)

(2) 各物理テーブルから得られたフィールド値（複数）について

・レコード識別フィールド（または、URI）をキーとした和集合または、積集合を得る。

例）field1 > 10 && field2 <= 0

・フィールド値とフィールド値の比較演算を行ない、条件に合うフィールド値の集合を得る。（フィールドの相互参照）

例）field1 > field2

結果として、条件に適合したレコードの識別フィールドと、検索時に同時に得られたフィールド値が得られる。この結果テーブルから結果が得られる場合は終了。→ (4)

(3) 検索条件に関係のないフィールド値が結果に必要な場合（結果テーブルにないフィールド値が必要な時）得られていないフィールド値を含む物理テーブルに対して、(2) で求めた結果テーブルのレコード識別フィールドで検索を行なう。得られたフィールド値を結果テーブルに付け加える。

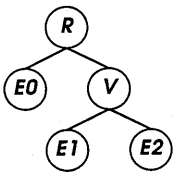
(4) 最後に結果テーブルを上位ノードに渡す。

検索条件から、どの物理テーブルが検索に必要かを検討する。不必要な検索を避け、局所化された検索方針を決定する。→ 局所化判定条件２

物理テーブルの検索順序を決定する。並行して実行できるもの、関連があるものを判定する。→ 最適化判定条件１

【複製ノード】

QUERY が複製のノードに達した場合、どちらの部分木を検索に使用するかを評価する条件判定が必要。→ 最適化判定条件２



上図で E0 がリモートサイト上にあり、V 以下のノードがローカルサイト上にあった場合でも、検索条件によっては V のノードでフィールドの相互参照の処理を行なうより、E0 で結果を得るほうが早いケースもある。

複製ノードでは障害回避も検索対象決定の要因となる。

b．更　新

更新に関しては『検索』と同じ手法で検索対象を限定し、更新が許される場合にはそのレコードに関して更新処理を行なう。対象となる物理テーブル・ノードで局所性を判断し、最適な検索方針を決定する。

局所的更新が許されないケースとして、特に以下の点を考慮しなければならない。

【水平分割】

分割条件に抵触するような変更を行なった場合。つまり、レコードに対する変更によって現在の物理テーブルの分割条件に適合しないレコードが発生してしまう場合は、変更後のレコードを適切なテーブルに移さなければならない (Record Migration) 。このような変更操作は局所化することはできない。→ 局所化判定条件３

変更操作終了後に、部分木で更新されたすべてのレコードを水平分割ノードに集める。水平分割ノードでは変更レコードが分割条件に合致しているかどうか判断し、条件にあわなくなったレコードがある場合には、そのレコードの移動（挿入＋削除）を行なう。

【垂直分割】

フィールドにインテグリティ指定がない場合、あるいは対象となるテーブル上のフィールド値だけで、インテグリティの判定が可能な場合には、変更を局所化できる。→ 局所化判定条件４

フィールド間にまたがるインテグリティが設定されている場合、更新すべきレコードのインテグリティをチェックするため他の物理テーブルに対して検索を行なわなければならない。→ 更新前整合性判定メカニズム

【　複製　】

重複データに対する同期更新のメカニズムが必要になる。→ 同期更新メカニズム２

たとえば、複製テーブルに対する更新はあまり多くないことを前提に『２フェーズ・コミット』方式を採用する。更新中の障害回復と、ダウンしているノードの更新が重要な検討課題として残っている。

c．挿　入

垂直分割のノードでは、挿入レコードを分割して複数の挿入文を生成し、各フラグメントに対して実行する。
水平分割のノードでは、挿入レコードに分割条件を適応して条件にあてはまるフラグメントに対してのみ挿入文を実行する。→ 局所化判定条件１

複製のノードでは、受け取った同じ挿入文をすべての下位ノードに適用する。

挿入レコードに値が設定されていないフィールドが存在する場合は、そのフィールドに『空値』を設定する。したがって、垂直分割できるテーブルは各フィールドが空値を許すことを条件にしなければならない。→ 垂直分割条件

d．削　除

基本的には、更新と同じメカニズムを使用する。

以上、検索／更新／挿入／削除それぞれについて、テーブル分割木を利用した問い合わせ処理を述べてきた。これらの問い合わせに対する処理系としてトランザクション・マネージャ (TM) を実現する。

TM はユーザからの QUERY を受け取り、テーブル分割木にしたがって必要な SUB-QUERY を生成する。SUB-QUERY を受け取りデータ実体へのオペレーションを行なうプロセスをデータ・マネージャ (DM) と呼ぶ。(G-BASE の従来の DM プロセスが、TM と DM に機能分割されたものになる)

TM はユーザ定義のスキーマとテーブル分割木 を保持していて、仮想の物理テーブルに関する同時実行制御を行なう。DM はデータ実体に関する同時実行制御を行なう。

TM は必要に応じて、複数の DM を起動し必要な結果を得るための SUB-QUERY を送る。DM はデータ実体の存在するサイトで起動される。ローカル・サイトの場合とリモート・サイトの場合が考えられる。

起動される DM と TM には、以下のような実現方法が考えられる。

a. TM に対して、データ実体（テーブル分割木のリーフノード）ごとに DM を起動する。
複数の物理テーブルに対する処理を並列に実行することができるが、DM 起動時間のオーバヘッドが問題となる。また、複数の DM に対する通信路の維持と、障害回復の複雑さが問題となる。

b. TM に対して、関連サイトごとに１つ DM を起動する。サイト間での並行処理が可能になるが、まだ TM ごとの DM 起動にはかわりなく起動時のオーバヘッドが問題となる。対策として、DM の制御プロセスを用意し、起動済みの複数個の DM（DM プール）を制御プロセスが割り当てる方式を検討する。

c. 各サイトに DM プロセス（常駐）を１個だけ起動しておき、すべての TM からのリクエストにこの DM が答えるようにする。
この方法では、複数の TM からのリクエストに対して同時に実行することができないので、十分な性能が得られない。そこで、軽量プロセスによるクライアント（この場合は TM ）の管理を行ない複数の TM からのリクエストを同時実行できるように制御する。

当面は実現性を考えてｂの方法を検討するが、将来は TM-DM 間のプロトコルを規定し、TM のマシン、DM のマシンの構成によって柔軟にプロセス構成を変更できるようなメカニズムを実現する。

５．まとめ

データベース管理システム G-ＢＡＳＥへの分散 DBMS 機能拡張の実現について考察した。G-ＢＡＳＥ分散化はこれからの研究課題であり、現在検討中の課題を述べるにとどまった。

分散 DBMS 機能実現の要素技術の１つとして、テーブル・データの分割／管理方式を示した。

エンドユーザ・インタフェースとして論理テーブルを導入し、テーブル分割木と呼ばれる木構造によって物理テーブルを管理することによって、以下に述べる操作性の向上と実用性が期待できる。

a. テーブルの分割／複製の混在を許すことで、データベース設計者に対し、より柔軟なデータベース設計を可能にする。

b. エンドユーザ・インターフェースとして論理テーブルの概念を導入することで、ユーザに対してデータベースのデータ実体を完全に透過的にすることができる。

c. 問い合わせ処理の過程で、部分木に対する操作に局所性が認められた問い合わせについては、単一サイトでの処理 (Local Operation) が可能となり、分割されているにも関わらず、かなりの性能が期待できる。実際には、局所性を高めるデータベース設計を行なうことが必要となる。

d. QUERY の解析結果を統計情報として残しておくことにより、データベースの再設計時にフィードバックできる。

e. 地理的に分散したデータベースの管理だけでなく、不均一なメディア上のデータベース（マルチメディア・データベース）に対してもこのテーブル分割法を利用できる。

分散 DBMS への機能拡張を行なうためには、まだまだ解決していかなければならない問題点が多い。

今後、本稿で述べた方式に基づいて、G-ＢＡＳＥの分散 DBMS 機能を拡張していく計画である。さらに、グラフ理論を基盤とするグラフ・データ・モデルに適する分散方式についても検討していきたい。

参考文献

[1] H.S.Kunii: Graph Data Language: A High Level Access-Path Oriented Language, Ph.D.Dissertation of The University of Texas at Austin, May 1983

[2] （株）リコー編：G-ＢＡＳＥシステムマニュアル

[3] Rothnie, J., et al.: Introduction to System for Distributed Databases (SDD-I), ACM Trans. on Database Systems, Vol.5, No.1, pp.1-17, 1980

[4] Williams, R., et al.: R* : An overview of Architecture, IBM Res. Rep. RJ3325, Dec. 1981.

[5] Chu, W.W.: Optimal File Allocation in a Multiple-Computer Information System, IEEE Trans. on Comput., Vol.C-18, No.10, pp885-889, 1969

[6] Minoura. T and Wiederhold, G.: Resilient Extended True-Cory Token Scheme for a Distributed DataBase System, IEEE, Trans. on Software Engineering, Vol. SE-8, No.3, 1982

[7] G. Gardarin and W. W. Chu: An Distributed control algorithms for replicated data in Distributed databases, IEEE, Trasn. Comput., Vol.C-29, pp1060- 1068, Dec.1980

[8] Y.Masunaga: Update and Query Processing Schema in Partitioned Distributed Relational Database System, COMPSAC '87 Proc. pp.521-530

[9] [10] 飯沢篤志,平岡昭夫：データベース管理システム G-ＢＡＳＥのＮＦＳ対応（１）、（２）情報処理学会第３６回全国大会, Mach 1988

[11] R.Sandberg, et al.: Design and Implementation of the Sun Network Filesystem, USENIX Summer 1985