情報処理学会論文誌　データベース　Vol. 4　No. 2　185–199 (July 2011)

リレーショナルデータベース管理システムにおける並列問合せ処理機構の開発

土田正士[†1]　河村信男[†1]　中野幸生[†1]
原憲宏[†1]　石川博[†2]

並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式を示す．SQL文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化にともなう技術からなる．本論文の目的は，SQLの問合せ処理に，パイプライン並列およびデータ並列の考え方を適用した問合せ処理機構について示すことである．そのために，負荷平準化を目的とするフロータブルサーバ方式の処理時間および効果を評価する．また，フロータブルサーバ方式を好適に活用するために，変数を含むSQL文を解析処理する段階と，実際にそのSQLの処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2段階最適化方式を評価し，有効性を確認する．

Development of Parallel Query Processing Architecture on Relational Database Management System

Masashi Tsuchida,[†1] Nobuo Kawamura,[†1]
Yukio Nakano,[†1] Norihiro Hara[†1]
and Hiroshi Ishikawa[†2]

This paper shows the method of parallel query processing which is basis of the development on parallel database system. These technologies are composed of processing procedure description method for parallel execution, query transformation method, and parallel execution environment according to introduce the parallelism by SQL statement analysis. The purpose of this paper is to show the query processing mechanism that applies the idea of a pipeline parallel and data parallel for the SQL query processing. Therefore, this paper evaluates the processing time and the effect of the floating server method to aim at the load balance. Moreover, this paper also evaluates two stage optimization method to use floating server method suitably, which consisted of the query analysis stage to analyse SQL statement including a variable and the query execution stage to optimize the processing procedure of SQL statement according to the value of a variable.

1. はじめに

実世界においてデータベース化が望まれるデータ量は増加する一方である．大量なデータに対してDBMS（Database Management System）での処理性能はデータ量に比例しない応答時間が望まれている．しかし，大量のデータを逐次的に処理していたのでは限界がある．そこで，データベース処理の並列化が必要とされている．

最近のハードウェア基盤の進展によって，並列システムが構築できる環境が揃ってきていると考えられる．データ処理に並列処理を適用する実践は実応用システムで行われている．なかでもSQLを対象にする並列データベースシステムは様々なアプローチが行われてきている[1)–3)]．SQL問合せ処理ではデータ操作が入力，出力ともに同じく表と呼ばれる形式で閉じた演算で表現され，多種類のデータ操作をパイプライン的につなぎ，各々の操作を並列に実行するパイプライン並列化が可能である．また，表は行と呼ぶデータの集まりであるため，並列システムにおけるハードウェアのShared-Nothing方式を活用するキーレンジ分割，ハッシュ分割などデータ分割方式を適用することで，データを均等の量に分割し，1つの操作を各々分割されたデータに対して並列に実行するデータ並列化が実現される．このように，SQL問合せ処理に対する並列化技術が確立されてきたが，SQLでは表への参照手段を記述しないため，SQLの問合せ処理，なかでも最適化処理で並列処理に向く参照手段を作成することが必要である．

このような背景から，並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式が開発された．SQL文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化にともなう技術からなる．本論文の目的は，SQLの問合せ処理に，パイプライン並列およびデータ並列の考え方を適用した問合せ処理機構について示すことである．

具体的には，負荷平準化を目的とするフロータブルサーバ方式は，DBを格納していない

†1 株式会社日立製作所ソフトウェア事業部
Software Division, Hitachi, Ltd.
†2 静岡大学情報学部
Faculty of Informatics, Shizuoka University

プロセサに処理の一部を分担させ，ジョイン処理にデータ並列およびパイプライン並列を適用することで，処理時間および効果を評価する．また，フロータブルサーバ方式を好適に活用するために，変数を含む SQL 文を解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2 段階の最適化方式を評価した．

以下では，2 章で課題について述べ，3 章で並列問合せ処理機構のアプローチについて解説し，4 章で具体的な並列問合せ処理機構の実装による適用評価を示し，5 章で関連研究を述べ，6 章でまとめる．

2. 課　　題

2.1 処理負荷の平準化

並列データベースシステムにおいて，並列処理の DBMS のアーキテクチャは 3 つの方式に分類できる[4]．これらは，すべてのプロセサが主記憶と HDD（Hard Disk Drive）を共用する SE（Shared-Everything）方式，HDD だけを共用する SD（Shared-Disk）方式，およびプロセサ間で主記憶も HDD も共有しない SN（Shared-Nothing）方式からなる．

本論文の並列データベースシステムは，プロセサ数および HDD 数が多く，かつ大規模な DBMS で更新処理の並列化を実現しやすいので，SN 方式のアーキテクチャを採用した．

ただし，SN 方式の課題は，特定のデータにアクセスするプロセサが固定されてしまうので，複数あるプロセサの処理負荷の平準化を図るのが難しいことである．この課題を解決するために，負荷平準化を目的とする，フロータブルサーバ方式を実現する[5)–9)]．このフロータブルサーバ方式は，プロセサ間での処理負荷の平準化のために，DB を格納していないプロセサに処理の一部を分担させる方法である．したがって，フロータブルサーバ方式の目的は，ジョイン処理やソート処理などの負荷を DB を格納しない専用のサーバ（プロセサも異なる）に分散させることによって DB をアクセスするサーバの負荷を軽減させることである．

2.2 並列問合せ処理

本論文で提案する DBMS は，フロータブルサーバ方式を好適に活用するために 2 段階からなる SQL の最適化方式を採用している[10)–12)]．すなわち，ユーザアプリケーションプログラムが発行した SQL 文を後述する SQL 受付サーバが受け取って解析処理[⋆1]する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に最適化する段階からなる．ユーザアプリケーションプログラムで書かれる多くの SQL 文には変数が含まれており，これらの変数の値は問合せ実行時に決まるので，SQL をどのような処理手順で実行すれば応答時間を最短にできるかは，この値に依存する．したがって，SQL 文の問合せ解析時には必ずしも最適な処理方法を選択できない．特に，大規模な表に対するジョイン処理およびソート処理は，処理手順の選択を間違うと大幅に性能が低下する．しかも，変数の値が決まった問合せの実行処理時点で SQL 文を改めて最適化処理を含めて解析処理するのでは，解析処理の負荷が問題となる．2 段階の最適化方式は，これらの課題を解決するものである．

具体的には，最適な処理手順を選択するのにコストに基づく方式を採用している．まず，問合せ処理を構成する個々の処理単位の処理時間（コスト）を見積もる．SQL 文の解析結果と，アクセス対象となる表のデータの分布状態などの統計情報をもとに，問合せの解析時点で全体のコストが最小になるような処理手順候補の組合せを選ぶ．これを最適な処理手順の候補群とする．最終的に決定される処理手順の最適性は，問合せ解析時に作成する処理手順候補への，問合せ実行時に決まる変数の値の設定方法により決まる．コストモデルに実際の情報システムを反映させることができるかどうかに依存している．なお，この方法は，なるべく負荷をかけずに統計情報を精度良く得ることが必要になる．提案 DBMS は，データの分布情報などをディクショナリ[⋆2]で一元的に管理しているので，迅速にコストを評価できる．本論文では，問合せ最適化によって生成される処理手順の最適性に影響を与えると考えられる，変数を含む問合せの最適化方式を議論する．

3. アプローチ

3.1 フロータブルサーバ方式

3.1.1 分散マルチサーバ構成

並列データベースシステムを実現するために，DBMS を複数のソフトウェア機能，すなわち用途別のサーバ[⋆3]に分割している．プロセサと主記憶，HDD を 1 つの単位としたノード[⋆4]に対して，サーバを複数割り当てる．これを分散マルチサーバ構成と呼ぶ．また，サー

⋆1 SQL 文を実行形式あるいはそれに近い処理手順に変換する最適化処理も含まれる．

⋆2 データベースの論理的な構造や内部状態を管理する仕組み．

⋆3 ここでいうサーバは，DBMS の機能単位．1 つのサーバは同じ機能を持つ複数のプロセスからなる．1 つのプロセスはさらに複数のスレッドを持つ．

⋆4 ここでいうノードは，並列コンピュータを構成する単位．プロセサを基本とする．SN 方式では，プロセサ，主記憶，HDD の 3 要素をまとめてノードとする．SD 方式では，プロセサと主記憶の組合せがノードとなる．SE 方式にはノードという概念はない．

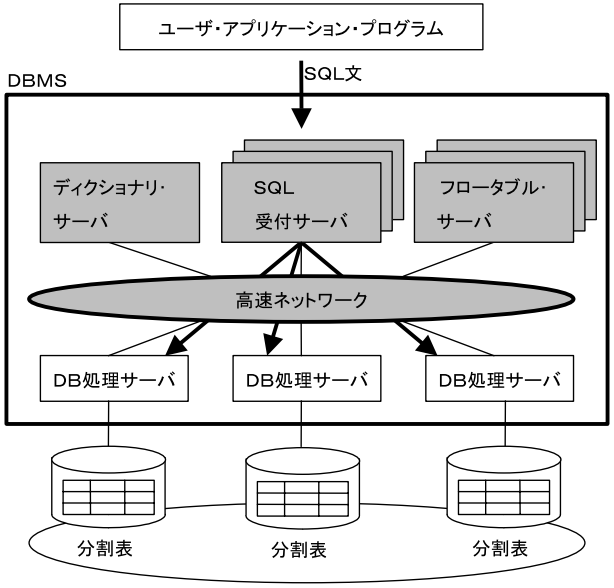


図 1　分散マルチサーバ構成
Fig. 1　Configuration of distributed multi-server.

バ間は RPC（Remote Procedure Call）により連携する．他のプロセサのサーバにデータを転送し，リモートで他のプロセサのサーバを起動することでプロセサ間の処理負荷を調整する．

提案 DBMS を構成する代表的なサーバには，以下のものがある．

- DB 処理サーバ
- SQL 受付サーバ
- フロータブルサーバ
- ディクショナリサーバ

各サーバの機能について，**図 1** に示す．

(1)　DB 処理サーバ

DB 処理サーバは，各ノードに複数割り当てる．DB 処理サーバは，そのノードの HDD に分割して格納してある個々の表（分割表*1）に 1 対 1 で対応する．特定の分割表に対するデータベース処理を専任的に行う．入出力処理だけでなく，データの取り出しおよび並べ替えの処理も行う．

(2)　SQL 受付サーバ

SQL 受付サーバは，ユーザアプリケーションプログラムから受け取った SQL 文を解釈して DB 処理サーバに処理手順を割り当てる．処理手順に必要なデータの存在するノードを調べ，そのノードの DB 処理サーバに処理要求を指示する．DB 処理サーバが問合せ処理を終えたら結果を受け取り，ユーザアプリケーションプログラムに渡す．

(3)　フロータブルサーバ

フロータブルサーバは，特定の分割表を割り当てていない DB 処理サーバである．プロセサの処理負荷の平準化を図る場合に利用する．たとえば，大量のデータをジョイン処理する場合，プロセサの処理負荷が重いマージ処理などを受け持つ．DB を持たない別のノードのフロータブルサーバにマージ処理を任せることにより，通常の DB 処理サーバの処理負荷を減らすことで，特定のノードに集中した負荷を分散できる．どの処理をフロータブルサーバに割り当てるかは SQL 受付サーバで決める．フロータブルサーバは，データベース処理の負荷を考慮してノードに割り当てておく．システム性能を平準化する目的では，ハッシュ分割によるデータの分散配置を行う．

(4)　ディクショナリサーバ

ディクショナリサーバは，システムおよび分割表などのデータベースの構成情報を管理する．

3.1.2　フロータブルサーバを用いた負荷平準化

SQL の実行を単に並列化しても，プロセサの処理負荷が平準化されなければ応答時間は短縮されない．この課題に対して，プロセサの処理負荷の平準化が難しい SN 方式は不利とされていたが，フロータブルサーバを導入することで，動的にプロセサの負荷平準化が図れるようにした．すなわち，フロータブルサーバに負荷の重い処理を割り当てることで効果を高めた．

具体的には，通常は DB 処理サーバが受け持つソート処理とデータのマージ処理を，別のノードのフロータブルサーバに移す．その結果，ノード間の通信が増えるが，そうした負

⋆1 DBMS では，1 つの表（テーブル）を複数の HDD に分割して格納する．分割された個々の表を分割表と呼ぶ．個々の表を分割して複数の HDD に分配する方法として，キーレンジ分割とハッシュ分割の両方に対応する．

荷よりも全体の応答時間の短縮を優先させた．また，個々のフロータブルサーバの処理負荷についても平準化できるようにしている．DB 処理サーバの処理するデータがある程度均一になっていたとしても，DB 処理サーバが出力する中間結果のデータ量は偏りが生じる場合がある．SQL で指定された条件式によって，DB 処理サーバごとに処理結果が異なるためである．これに対処するために，ハッシュ手法を使ってフロータブルサーバに中間結果のデータを均等に分配する[13]．こうしてフロータブルサーバの処理負荷を均等化する．

フロータブルサーバを導入することの利点は 2 つである．

- システム性能要件の変更に対するチューニングの容易化
  データの移動をともなうデータベースの再構成をしなくても，フロータブルサーバの追加でチューニングができる．処理すべきデータの増大に対しても，フロータブルサーバを追加することで対応できる．
- トランザクション処理と大量データを一括検索する問合せ処理の共存
  トランザクション処理は，一定の応答時間を保証する必要がある．しかし，ジョイン処理などの高負荷な問合せ処理をトランザクション処理と並行して実行すると，トランザクション処理の応答性能に悪影響を及ぼす可能性がある．この場合，問合せ処理の特性に合わせて，ソート処理に要する時間がデータ入力時間に対して一定の比率になるようにフロータブルサーバの割当て数を調整し，問合せ実行時に割当て数だけフロータブルサーバを割り当てる．ただし，ソート処理に要する時間とデータ入力時間を同じにして，問合せ処理時間を最小化する方法も考えられるが，並行して複数のジョイン処理も実行できることを考慮する．システム内で設定するフロータブルサーバ数から按分することで，フロータブルサーバを専有させない方法も併用する．これにより，処理負荷を別のプロセサに分散できるので，トランザクション処理の性能を一定以上に保証できる．

また，このフロータブルサーバによる負荷の平準化を図るために，割当て方式を調整できるようにしている[14]．

(1)　フロータブルサーバ専用方式
　データ入力処理を行う DB 処理サーバに，ソート処理など負荷がかかる処理が割り当てられず，別のノードにそれらの処理を専用に実行するフロータブルサーバが割り当てられる．

(2)　フロータブルサーバ対象限定方式
　データ入力処理を行う DB 処理サーバと同一のノードに対象を限定して，ソート処理など負荷がかかる処理を実行するフロータブルサーバが割り当てられる．

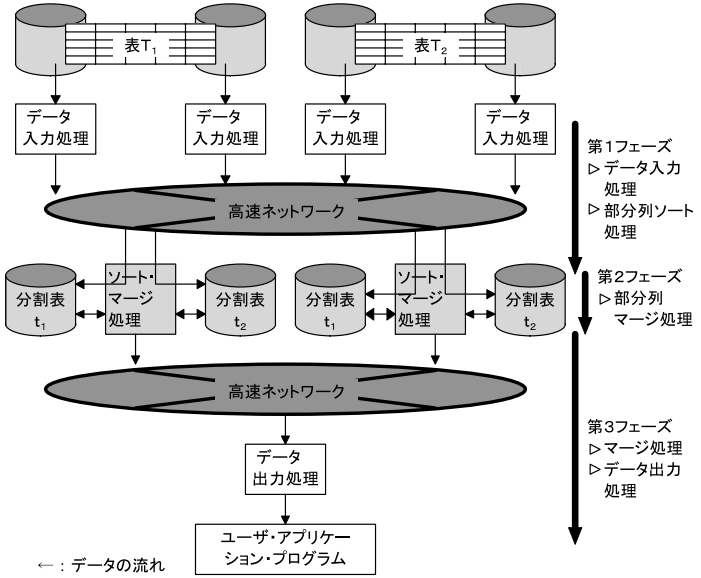


図 2　並列ジョイン処理
Fig. 2　Parallel join processing.

(3)　フロータブルサーバ対象拡大方式
　データ入力処理を行う DB 処理サーバと同一のノードに加えて，別のノードにもソート処理など負荷がかかる処理を実行するフロータブルサーバにも対象が拡大されて割り当てられる．

3.1.3　フロータブルサーバ方式によるジョイン性能の改善

フロータブルサーバ方式によるジョイン性能の改善例を示す．提案 DBMS で大量のソート処理を含むジョイン処理に多大な時間を要するので，できるだけジョイン処理を並列化して応答時間を短縮する必要がある．前述したように，ジョイン処理にデータ並列およびパイプライン並列を適用する．**図 2** にその方法を示す．ジョイン処理は 5 つの処理に分けられる．

- データを HDD から読み出す処理（データ入力処理）
- 部分列ソート処理
  ハッシュ分割された部分表に対してソート処理を行う．

- 部分列マージ処理
  部分列ソート処理によって生成した複数の部分列に対してマージ処理を行い，最終的に 1 つの部分表を作成する．
- データのマージ処理
  2 つの部分表をマージ処理することで，ジョイン処理結果を生成する．
- データをユーザアプリケーションプログラムに送り出す処理（データ出力処理）

この 5 つの処理にパイプライン並列化を適用する．この結果，ジョイン処理は 3 つのフェーズで処理が行われる．第 1 フェーズでは，データ入力処理および部分列ソート処理の間にパイプライン並列化を適用する．データ入力処理が DB 処理サーバで実行され，また部分列ソート処理がフロータブルサーバでパイプライン的に実行される．次の第 2 フェーズでは，第 1 フェーズの結果で得られた部分列に対してマージ処理を行う．ただし，第 2 フェーズの前後でいったんパイプライン処理は途切れる．最後の第 3 フェーズでは，データのマージ処理およびデータ出力処理の間にパイプライン並列化を適用する．データのマージ処理がフロータブルサーバで実行され，またデータ出力処理が SQL 受付サーバでパイプライン的に実行される．

3.2　2 段階最適化方式

3.2.1　処理手順の並列化表現

ここでは，処理手順の表現形式について，**図 3** に示す．(a)，(b)，および (c) は，逐次処理のための処理手順の表現形式である．

(a) は単一表へのアクセスの処理手順である．scan オペレータには，インデクススキャンを利用するのか，テーブルスキャンを利用するのか指定されている．

(b)，(c) はいずれもジョイン処理の処理手順である．(b) はネストループジョインの処理手順であり，foreach オペレータより上に位置する scan オペレータが，最初にアクセスされる外側表へのアクセスを示し，下に位置する scan オペレータが繰り返しアクセスされる内側表へのアクセスを示す．これらの scan オペレータには，インデクススキャンを利用するのか，テーブルスキャンを利用するのかが指定されているが，ネストループジョインの処理要件から上に位置する scan オペレータには選択条件に出現する列に付与されるインデクスへのインデクススキャン，および下に位置する scan オペレータにはジョイン条件に出現する列に付与されるインデクスへのインデクススキャンが設定されることが想定される．(c) はソートマージジョインの処理手順であり，merge オペレータより左右に位置する scan オペレータがそれぞれの表へのアクセスを示す．この場合，ソートマージジョインの処理要件

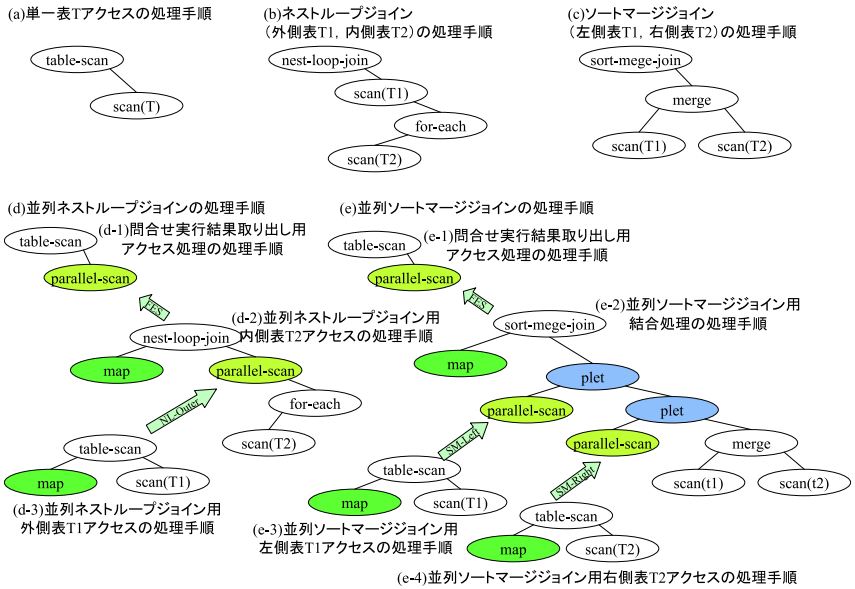


図 3　処理手順の並列化
Fig. 3　Parallel processing procudure.

からそれぞれの表の scan オペレータから出力されるデータはソート済みであることが想定される．

次に，問合せ処理を並列化するうえで，並列手順の表現方法を規定する必要がある．(d)，(e) はいずれも並列ジョイン処理の処理手順である．

(d) は並列ネストループジョイン（以下では，PNL（Parallel nest-loop join）と略記する）の処理手順である．(d) は，下記の各処理手順からなる．

- (d-1) - 問合せ実行結果取り出し用アクセス処理の処理手順
- (d-2) - 並列ネストループジョイン用内側表アクセスの処理手順
- (d-3) - 並列ネストループジョイン用外側表アクセスの処理手順

複数のサーバからのデータを受け取る parallel-scan オペレータを導入する．(d-2) では，外側表からのデータを，PNL 処理の parallel-scan オペレータで受け取ることを表現する．また，データ転送先のサーバに分散して配布する記述のために，map オペレータを導入する．この map オペレータは，外側表からのデータを，PNL 処理の parallel-scan オペレー

タでブロードキャスト転送，あるいはハッシュ分割して受け取る場合（(d-3) から (d-2) へのデータ配布）と，PNL 処理の結果のデータを SQL 受付サーバの parallel-scan オペレータに集約して受け取る場合（(d-2) から (d-1) へのデータ配布）に表現されている．これらの，parallel-scan オペレータおよび map オペレータを利用するデータ転送処理は，パイプライン的に行われる．

(e) は並列ソートマージジョイン（以下では，PSM（Parallel sort-merge join）と略記する）の処理手順である．(e) は，下記の各処理手順からなる．

- (e-1) - 問合せ実行結果取り出し用アクセス処理の処理手順
- (e-2) - 並列ソートマージジョイン用結合処理の処理手順
- (e-3) - 並列ソートマージジョイン用左側表アクセスの処理手順
- (e-4) - 並列ソートマージジョイン用右側表アクセスの処理手順

この場合，PNL と同様に，parallel-scan オペレータおよび map オペレータを利用するデータ転送処理（(e-3) から (e-2)，および (e-4) から (e-2) へのデータ配布）は，パイプライン的に行われるが，部分列に対するマージ処理でいったんパイプライン処理は途切れる．たとえば，集合演算のためのソート処理，およびジョイン処理は複数の表からデータを受け取り，複数の中間結果を作成する必要がある．各表のデータ読み出しは別々のサーバから並列に送られてくるのに対して，plet オペレータによって中間結果の作成処理（図 2 では，部分列に対するマージ処理に該当）を記述する．

3.2.2　2 段階最適化方式での処理手順候補の作成

SQL 受付サーバが実行する最適化処理の手順を，**図 4** に示す．

まず，ユーザアプリケーションプログラムから受け取った SQL 文の問合せ解析処理時に，問合せ処理のコストを評価する．この時点で唯一最適と評価できる処理手順が決まればその処理手順を[10]，それ以外であれば代替案を含めて複数の処理手順を作成する[12]．問合せ解析処理時に作成した処理手順は SQL 受付サーバが保存する．SQL 文の問合せ解析処理は最初の 1 回だけでよい．同じ SQL 文を何回も実行するときは，2 回目以降は問合せ解析処理を省略できる．具体的には問合せ解析処理時には，まずユーザアプリケーションプログラムから渡された SQL 文の構文を解析して並列化する．そのあと，複数の並列化した処理手順候補を含めて出力する．処理手順は，プロセサ間のデータ転送，同期処理，他のプロセサの処理の呼び出しを表現する．図 3 の (d)，(e) に示すように，並列化した処理同士の呼び出し関係は RPC によって記述する．作成した個々の処理手順の相違点は，使用するジョイン方式（PNL，PSM のどちらを使うか），データ取り出し処理にインデクスを利用するか否か，サーバ間でデータをハッシュ手法で配布するか集約して配布するかの点である．この処理手順を保存する．フロータブルサーバを割り当てる数は，SQL 受付サーバが問合せ解析処理時に決める．この際に，ソート処理に要する時間がデータ入力時間に対して一定の比率になるようにフロータブルサーバの割当て数を調整する．

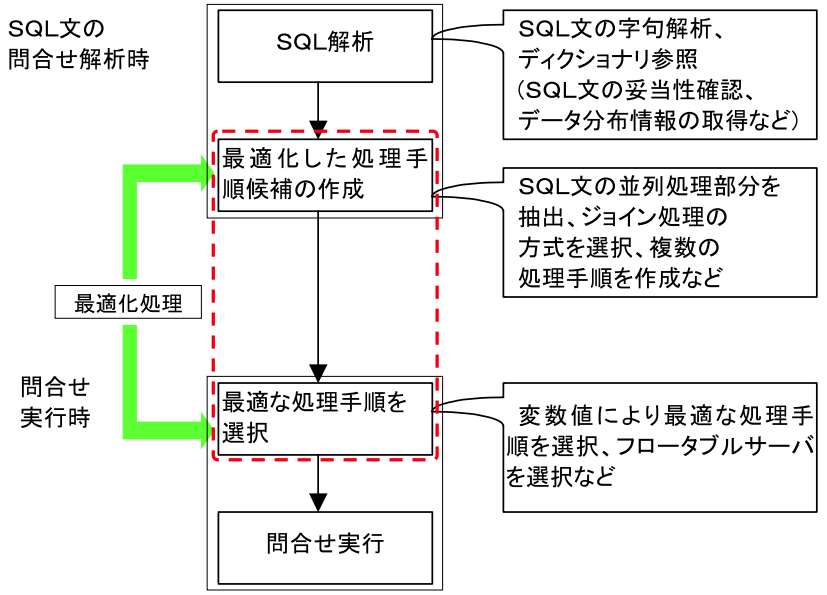


図 4　2 段階最適化方式
Fig. 4　Two-phase optimization method.

次に，問合せ実行時に，確定した変数値とディクショナリで管理されているデータ分布の最新情報を基に，問合せ解析処理時に作成した処理手順候補の中から最適な処理手順を選択する．この時点で初めてフロータブルサーバを確保する．どのフロータブルサーバにデータを割り振って処理を実行するかは，確保したフロータブルサーバの中からランダムに選択する[14]．これは，特定のフロータブルサーバに負荷が集中しないようにするためである．

この 2 段階最適化方式を適用して，問合せ実行時に PNL および PSM を処理手順として選択する場合，問合せ解析処理時に作成される処理手順候補を**図 5** に示す．ここでは，図 3 に前述する処理手順の並列化表現に加えて，select-procedure オペレータが導入される．select-procedure オペレータが指すデータ構造のオペランドに選択条件が設定される．

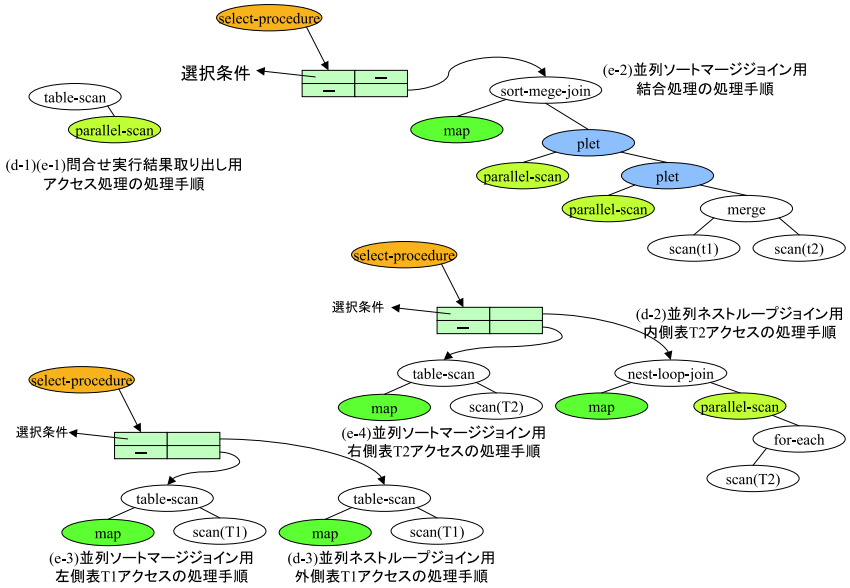


図 5　処理手順の作成例
Fig. 5　Example of processing procedure.

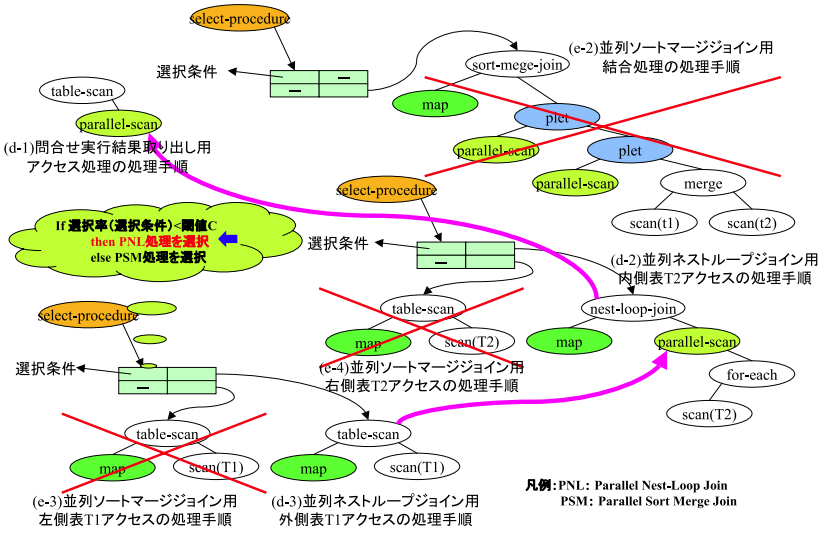


図 6　PNL の処理フロー
Fig. 6　Processing flow of PNL.

この選択条件は，問合せの中に出現する選択条件である．この選択条件に，問合せ実行時に確定した変数値とデータ分布の最新情報を基に選択率を算出し，3.2.3 項に後述する閾値と比較することで，問合せ解析処理時に作成した処理手順候補の中から最適な処理手順を選択する．図 5 では，3 つの select-procedure オペレータが出現するが，いずれも同様に算出が行われる．

問合せ実行時に PNL を処理手順として選択する場合，各オペレータ間での処理の流れを**図 6** に示す．3.2.3 項に後述するが，選択率が閾値以内であれば，PNL の外側表のアクセスでデータ数が絞り込まれるので，(d-3) の PNL 用外側表アクセスの処理手順，および (d-2) の PNL 用内側表アクセスの処理手順が，各々選択されて，結果的に (d-3)，(d-2)，および (d-1) によって PNL 処理が実行される．

一方，問合せ実行時に PSM を処理手順として選択する場合，各オペレータ間での処理の流れを**図 7** に示す．3.2.3 項に並列ジョイン方式の選択方法を後述するが，選択率が閾値以上であれば，PNL の外側表のアクセスでデータ数が絞り込まれないので，(e-3) の PSM 用左側表アクセスの処理手順，(e-4) の PSM 用右側表アクセスの処理手順，および (e-2) の PSM 用結合処理の処理手順が，各々選択されて，結果的に (e-3)，(e-4)，(e-2)，および (e-1) によって PSM 処理が実行される．

著者らはこれまでの研究において，問合せ解析処理時に作成した処理手順候補の中から最適な処理手順を選択するための基準として，並列結合処理で絞り込んだデータを受け取る場合，およびインデクスを利用してデータを取り出す場合を想定して，処理時間の算出式を作成し，選択率，行数，および行長などを変化させることで，各並列ジョイン方式の特性を抽出した[10),12)]．問合せ解析時に行数，および行長などが変化しない前提とすることとし，2 段階最適化方式では，以下に示す研究結果を適用する．

- PNL 処理の処理時間は，最外側表（最も早く取り出される対象の表を指す）が選択条件によって十分に絞ることができ，かつその内側表の結合列にインデクスが存在する場合を想定して見積もり，またその選択列の選択率に比例する．この場合，問合せ解析処理時にジョイン処理でアクセスする表の順序を決めている．一方で，PSM 処理の片側表の選択率を変化させ処理時間を対数で表しても，10 万件規模以上の行数の場合，曲

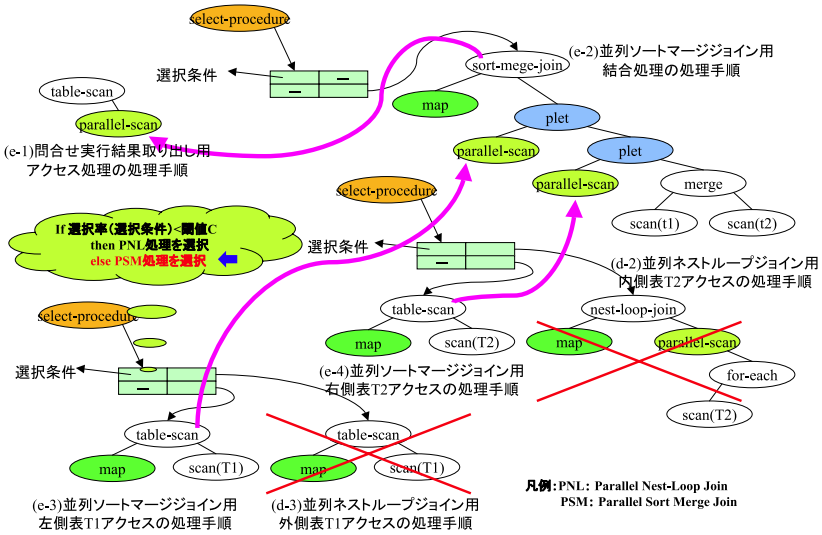


図 7　PSM の処理フロー
Fig. 7　Processing flow of PSM.

率が小さい．そこで，PNL 処理の処理時間を選択率の 1 次式で近似し，また PSM 処理の処理時間を選択率の低次式近似することで，問合せ実行時に決定する選択率の閾値を決める．

- 行数が数倍程度に変化しても，PNL 処理および PSM 処理を切り分けるための評価では PSM 処理の処理時間の曲率は小さく，閾値となる選択率は一定値である．
- インデクススキャンおよびテーブルスキャンを選択するデータ取り出し処理の場合，選択率の閾値は結合方式にかかわらず一定値である．

3.2.3　並列ジョイン方式の選択方法

次に，並列ジョイン方式の選択方法について示す．

(1)　問合せ解析処理時の最適化処理

- 並列ジョイン方式の候補作成
  問合せの条件で指定する列のインデクスの有無から，並列ジョイン方式の候補を作成する．具体的には，結合条件の列にインデクスが存在しなければ PSM 処理だけを，片側の表でも結合条件の列にインデクスが存在すれば PNL 処理および PSM 処理を候補とする．
  次に，問合せ実行時に並列ジョイン処理を決定する判別式を作成する．結合する表を T1 および T2，表 T1 に関する選択条件の選択率を x，表 T2 に関する選択条件の選択率を y，並列ジョイン処理時間の見積りに使用する関数 h，u，v，w とする．
- PNL 処理の処理時間の見積り
  PNL 処理は，最外側表が選択条件によって十分に絞ることができ，かつその内側表の結合列にインデクスが存在する場合に適用できる．また，最外側表のアクセス法としてインデクスを利用するか判断する際に，単一列インデクスであっても，範囲条件（BETWEEN），IN 条件，大小条件式で範囲条件を構成する場合は，複数の変数および選択条件が関与する．さらに，複数列インデクスの際にも，インデクスを構成する列に対して選択条件が指定される場合は，複数の変数および選択条件が関与する．
  これらの場合，PNL 処理時間は最外側表の選択条件によって絞られる行数に比例する．この最外側表（ここでは仮に T1 とする）の選択率 x が，最も小さい．また，PNL 処理は内側表の結合列にインデクスが存在すれば，連続的に適用することができる．その場合，外側表の選択率と，内側表の結合行数比（外側表の 1 行に対して，内側表の何行が対応するかの比率）を掛け合わせることで，結果として生成される中間表の選択率とする．この中間表の選択率によって，中間表の行数が絞られる表を順番に選択し，処理手順を生成する．
  PNL 処理の処理時間の見積り式は，以下とする．関数 h は，選択率 x 以外のパラメータは問合せ解析時に決定されるので，最外側表 T1 の選択率 x を用いて処理時間が導かれる．
  PNL 処理の処理時間
  ＝ h(T1 行数，選択率 x，インデクス段数，結合行数比，I/O 性能，データ分割数)
  ＝ n*x
  （ただし，最外側表 T1 の選択率 x 以外は，問合せ解析処理時に決定される定数 n とする）
- PSM 処理の処理時間の見積り
  次に，PSM 処理は各 2 表の PSM 処理時間が小さい順番で行う．これは，他の

表のデータ取り出し時間の間に，なるべくデータ取り出し時間が短くなると期待できる表どうしでジョイン処理を進めておく期待がある．基本的には，ジョイン処理が可能な 2 つの表のデータ読み出し時間の最大値が最小となる 2 表を見つけて，PSM 処理を行い，その中間表を交えて再びデータ読み出し時間が最小となる 2 表を見つけ出す．これを，中間表が 1 表になるまで繰り返し，処理手順を生成する．

PSM 処理の処理時間の見積り式は，以下とする．関数 u は，表 T1 および表 T2 のデータ取り出し時間，突き合わせ時間，および問合せ結果転送時間からなる．また，関数 v，w は，表 T1 および表 T2 の N ウェイマージ時間からなる．

PSM 処理の処理時間
＝ u(T1 行数，選択率 x，T2 行数，選択率 y，射影行長，CPU 性能，I/O 性能，データ分割数)
＋ v(T1 行数，選択率 x，T2 行数，選択率 y，射影行長，CPU 性能，I/O 性能，データ分割数)
* log(w(T1 行数，選択率 x，T2 行数，選択率 y，射影行長，I/O 性能，ソートウェイ数，データ分割数))

選択率以外のパラメータは問合せ解析処理時に決定されるので，log を低次多項式で近似した式 s(x，y) とおく．

- 並列ジョイン方式の候補決定

以上から，s(x，y) ＜＝ n*x をあらかじめ解いておき判別式とする．

また，片側の表 T1 の選択条件だけに変数が含まれる場合，T2 の選択率も問合せ解析処理時に決定される．問合せの実行時に決定するパラメータを片側表の選択条件の選択率の 1 つだけにする場合，各式を示すグラフの交点は 1 つなので，閾値 C が決まる．

最後に，その選択条件に出現する列のデータ分布情報から，選択率のとりうる最大値および最小値を取得し，最大値が閾値 C より小さければ PNL 処理，最小値が閾値 C より大きければ PSM 処理に処理手順を絞ることができる．

(2)　問合せ実行処理時の最適化処理

問合せ実行処理の最適化処理で，インデクスを利用するか判断する際に複数の変数および選択条件が関与する場合も考慮し，確定した変数値とディクショナリで管理されるデータ分布の最新情報を基に，選択条件の選択率を取得する．次に，その選択率が閾値 C より小さい場合は PNL 処理を，閾値 C 以上である場合は PSM 処理を選択する．これらによって，最適な処理手順を決める．

4. 適 用 評 価

フロータブルサーバ導入の効果を検証するために，TPC-C ベンチマーク[15] で規定される，注文表および注文明細表を利用する．表データは，ハッシュ分割し，各 DB 処理サーバに均等に配置する．問合せは，注文明細表に条件を付与して，絞り込んだ結果の件数を得る処理を評価する．

次に，システム評価環境について示す．ハードウェア環境は，以下の緒元である．

- モデル名：日立 BS320（2 ブレード構成）
- CPU：Intel Xeon X5570 QUAD 2.93 GHz × 2 ⋆1（1 ブレードごと）
- メモリ：48 GB
- ストレージ：日立内蔵 SAS300 GBx2 RAID1
- ネットワーク：ブレード間接続 1 Gbps

また，ソフトウェア環境は以下の緒元であり，このシステム評価環境にソフトウェアを配置した．

- DBMS：HiRDB V08-05（64 bit 版）
- OS：CentOS 5.4（x64）
- DB 処理サーバを 1〜8 つ割り当てる．それらのうち，フロータブルサーバに 1〜4 つの DB 処理サーバを割り当てる．

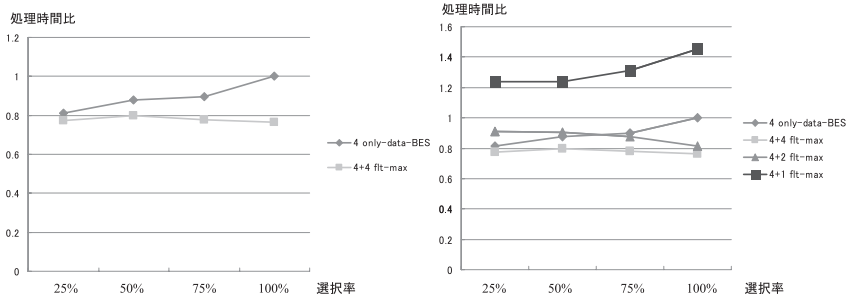
さらに，データベース緒元は以下のとおりである．

- 表のレコード数：注文表（480,000 件），注文明細表（4,819,276 件）
- データベース容量：注文表（24.4 MB），注文明細表（400.6 MB）
- 表の分割方法：ハッシュ分割し，4 つの DB 処理サーバに均等に配置

4.1　フロータブルサーバ方式の効果

フロータブルサーバ導入の効果について，**図 8** に示す．なお，問合せの実行にあたり，問合せ解析処理のための処理時間および表データを読み出す時間の影響を避けるために，2 回目以降の問合せの処理時間を計測する．また，実験では図 8 の問合せを利用するが，注文明細表（order_line）の倉庫番号（ol_w_id）列で指定する選択条件で，where 句のホスト変

⋆1 Intel Xeon は，アメリカ合衆国およびその他の国における Intel Corporation の商標です．



(a)フロータブルサーバ割当て方式による処理性能　　(b)フロータブルサーバ割当て数による処理性能

凡例：4 only-data-BES，選択率100%の処理性能を1とする処理時間比を評価

・問合せ

```
select count(*) from orders join order_line
                              on (o_id,o_d_id,w_id) = (ol_o_id,ol_d_id,ol_w_id)
                 where ol_w_id <= :w_id
```

・4 only-data-BES
4つのデータ入力処理を実行するDB処理サーバに，フロータブルサーバが割り当てられるフロータブルサーバ対象限定方式

・4+n flt-max
4つのデータ入力処理を実行するDB処理サーバ以外で，フロータブルサーバに専用のDB処理サーバがn個割り当てられるフロータブルサーバ専用方式

図 8　フロータブルサーバの評価
Fig. 8　Evaluation of floating server.

数「：w_id」で倉庫番号（Warehouse ID）を設定して選択率を変化させる．

4.1.1　フロータブルサーバ割当て方式

(1)　フロータブルサーバ割当て方式による処理性能

図 8 (a) は，フロータブルサーバ割当て方式による性能特性を示す．

4 つの DB 処理サーバに注文表および注文明細表を均等に配置して，各々フロータブルサーバ割当ての 2 方式によるパイプライン並列の効果を PSM 処理の処理時間で評価する．ユーザアプリケーションプログラムで問合せ実行結果を受け取る時間の影響を避けるために，問合せを満たす行数（SQL 文で select 節に count(*) を指定）だけを問合せ実行処理結果とするので，データのマージ処理およびデータ出力処理のオーバラップによる，双方の処理間でパイプライン並列の効果はない．すなわち，データ入力処理および部分列のソート処理がオーバラップによるパイプライン並列の効果が評価できる．

フロータブルサーバ対象限定方式の場合，注文表および注文明細表を格納する 4 つの DB 処理サーバを利用して，ソート処理およびマージ処理を実行する．また，フロータブルサーバ専用方式の場合，注文表および注文明細表の表データを格納する 4 つの DB 処理サーバとは異なるフロータブルサーバを利用してソート処理およびマージ処理を実行する．

その結果，フロータブルサーバ専用方式は，フロータブルサーバ対象限定方式と比較して，選択率が 100%の場合では処理時間が約 24%削減されており，データ入力処理および部分列のソート処理がオーバラップしていることに起因するパイプライン並列の効果が分かる．

(2)　フロータブルサーバ割当て数による処理性能

図 8 (b) は，フロータブルサーバ割当て数による性能特性を示す．

フロータブルサーバ専用方式で，フロータブルサーバの割当て数によって処理時間が短縮されることが期待できる．これは，データ入力処理および部分列のソート処理がオーバラップしている処理間でパイプライン並列の効果が認められることに起因する．

その結果，フロータブルサーバ専用方式の場合，フロータブルサーバの割当て数が 1 から 4 に増えるにつれて処理時間が短縮されることが分かる．これは，データ入力処理および部分列のソート処理がオーバラップしている処理間でのパイプライン並列，およびフロータブルサーバの割当て数の増加によるデータ並列の両効果である．

一方，フロータブルサーバ対象限定方式の場合，データ入力処理および部分列のソート処理の間，データのマージ処理およびデータ出力処理の間で，各々オーバラップが行われない．フロータブルサーバ対象限定方式と，フロータブルサーバ専用方式でフロータブルサーバを割り当てる場合を比較すると，選択率によって処理時間の優劣が決まる．これから，フロータブルサーバを導入する場合，問合せ解析時に決定するフロータブルサーバに割り当てる数が重要であることが分かる．

4.1.2　フロータブルサーバ方式による負荷平準化

図 9 は，フロータブルサーバ方式による負荷平準化を示す．x 軸は，各々4 only-data-BES，4+4 flt-max の処理時間を 1 とした際の各処理の進行度合い (%)，y 軸は各ノードの CPU 利用率 (%) を示す．CPU 利用率は，4 つのデータ入力処理を実行する DB 処理サーバが配置される DB 処理サーバ用ノード，および 4 つのフロータブルサーバが配置されるフロータブルサーバ用ノードで計測する．ここでノードは，適用評価で 1 つのブレードに相当する．

図 9 (a) は，フロータブルサーバ対象限定方式での DB 処理サーバ用ノードの CPU 利用率を示すが，進行度合いの約 90%までがデータ入力処理および部分列ソート処理の実行負荷であり，平均 CPU 利用率は約 33%である．また，進行度合いの約 90%以降が部分列マージ

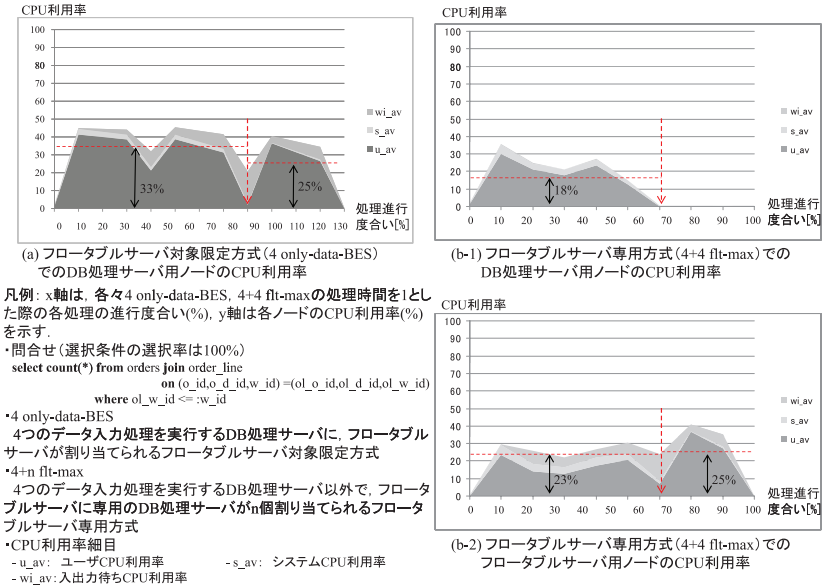


図 9　フロータブルサーバ方式による負荷平準化
Fig. 9　Load balancing by means of floating server method.

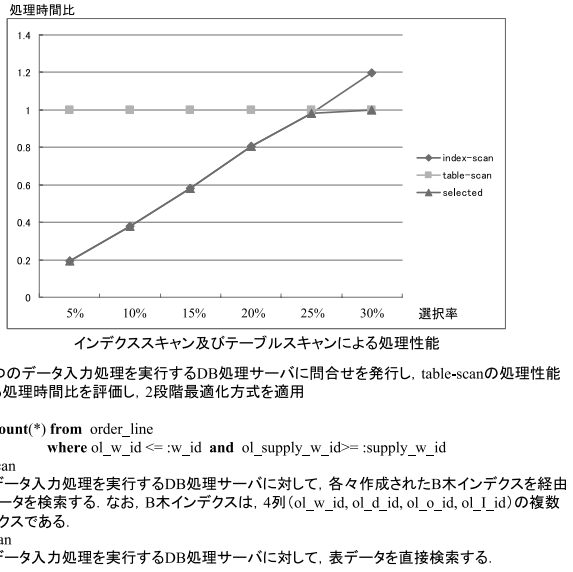


図 10　スキャン法の選択
Fig. 10　Selection of scan method.

処理，マージ処理，およびデータ出力処理の実行負荷であり，平均 CPU 利用率は約 25%である．

一方，図 9 (b-1) は，フロータブルサーバ専用方式での DB 処理サーバ用ノードの CPU 利用率を示すが，進行度合いの約 70%までがデータ入力処理だけの実行負荷であり，平均 CPU 利用率は約 18%である．また，図 9 (b-2) は，進行度合いの約 70%までが部分列ソート処理だけの実行負荷であり，平均 CPU 利用率は約 23%である．さらに，進行度合いの約 70%以降が部分列マージ処理，マージ処理，およびデータ出力処理の実行負荷であり，平均 CPU 利用率は約 25%である．

この結果から，進行度合いの約 70%までで，図 9 (b-1) のデータ入力処理（平均 CPU 利用率約 18%）と図 9 (b-2) の部分列ソート処理（平均 CPU 利用率約 23%）で負荷が平準化されていることが分かる．また，4 only-data-BES，4+4 flt-max の処理時間は異なるが，図 9 (a) のデータ入力処理および部分列ソート処理の実行負荷の平均 CPU 利用率は約 33%であるので，負荷の平準化のために図 9 (b-2) に示すフロータブルサーバ用ノードを導入することで，DB 処理サーバ用ノードの実行負荷である平均 CPU 利用率を約 18%に軽減することが期待できる．

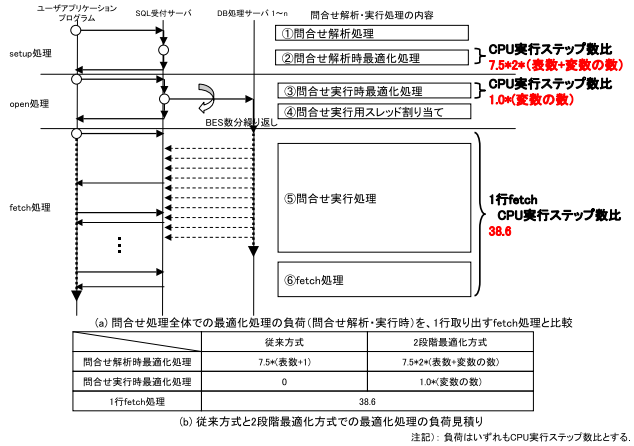
4.2　2 段階最適化方式の評価

4.2.1　スキャン法の選択

図 10 は，2 段階最適化によるスキャン法の選択を示す．

4 つの DB 処理サーバに注文明細表を均等に配置して，選択率を変化させて適切なスキャン法を選択しているか評価する．ユーザアプリケーションプログラムで問合せ実行結果を受け取る時間の影響を避けるために，問合せを満たす行数（SQL 文で select 節に count(*) を指定）だけを問合せ実行処理結果とするので，テーブルスキャン法を選択する場合，表データをすべてアクセスする処理時間はほぼ一定である．

また，インデクススキャン法を選択する場合，各々DB 処理サーバで作成された B 木インデクスを経由して表データを検索するが，アクセスする表データは選択率に比例して増加するので，処理時間も増加する．



(a) 問合せ処理全体での最適化処理の負荷(問合せ解析・実行時)を、1行取り出すfetch処理と比較

	従来方式	2段階最適化方式
問合せ解析時最適化処理	7.5*(表数+1)	7.5*2*(表数+変数の数)
問合せ実行時最適化処理	0	1.0*(変数の数)
1行fetch処理	38.6	

(b) 従来方式と2段階最適化方式での最適化処理の負荷見積り

注記) : 負荷はいずれもCPU実行ステップ数比とする.

図 11　2 段階最適化方式の評価
Fig. 11　Evaluation of two-stage optimization.

図 10 の評価設定で 2 段階最適化方式を実装する提案 DBMS を評価すると，選択率が 25%以下ではインデクススキャン法を，また選択率が 30%以上ではテーブルスキャン法を選択することが確認できる．2 段階最適化方式を採用しない従来方式の場合，問合せ解析時に決定されたスキャン法が実行されることになり，必ずしも最適ではないスキャン法が選択される．この場合，インデクススキャン法が選択されるので，選択率が 30%以上では最適ではないスキャン法を実行する．

4.2.2　2 段階最適化処理の負荷評価

2 段階最適化方式を実装する DBMS の CPU 実行ステップ数を取得し，SQL 文の問合せ解析処理および問合せ実行処理の負荷を評価する．SQL 文の問合せ解析処理は，1 回目の open 処理だけで実行されるが，2 回目の open 処理では実行されない．また，SQL 文の問合せ実行処理は，1 回目以降の open 処理で実行される．**図 11** は，2 段階最適化方式の評価結果である．

まず，図 11 (a) で，問合せ解析処理および問合せ実行処理の内容を示す．問合せ実行処理の最適化処理で，確定した変数値とディクショナリで管理されるデータ分布の最新情報を基に，最適な処理手順を決めるための CPU 実行ステップ数比を 1.0 とする．また，この処理は，たかだか変数の数回だけ繰り返される．

一方で，問合せ解析処理の全体の CPU 実行ステップ数比は 81.2 であり，特に問合せ解析処理の最適化処理で，ジョイン処理の選択および処理手順候補を作成するための CPU 実行ステップ数比は 7.5 である．また，この処理は，SQL 受付サーバの処理手順で 1 回，PNL 処理のために問合せに出現する表数回，PSM 処理のために問合せに出現する（表数 −1）回，単一表の処理手順候補ごとに，たかだか問合せ変数の 2 倍の回数だけ繰り返される．すなわち，繰り返される回数は，2*（表数の数＋変数の数）となる．

図 8 に示す問合せで試算すると，表数の数が 2，変数の数が 1 である．各々の CPU 実行ステップ数比は，問合せ実行処理の最適化処理では 1.0*(変数の数=1)=1.0 となり，問合せ解析処理の最適化処理では 7.5*2*(表数の数=2 ＋変数の数=1)=45.0 となる．また，比較のために，単一表からたかだか 1 行を取り出すための fetch 処理に要する CPU 実行ステップ数比は 38.6 であり，問合せ実行処理の最適化処理は約 3%の負荷に相当する．適用評価の問合せは，ジョイン処理であるので，全体の負荷における割合がさらに小となる．

次に，問合せ解析処理の全体の CPU 実行ステップ数比は 81.2 に対して，問合せ解析処理の最適化処理は約 55%の負荷に相当する．2 段階最適化方式で問合せ解析処理時に，従来方式と同等に唯一最適と評価できる処理手順を決める場合，この負荷が削減できる効果がある．

4.2.3　従来方式との比較

図 11 (b) は，コストモデルに基づき問合せ解析時最適化処理で唯一の処理手順を選択する従来方式[16),17)] と 2 段階最適化方式での最適化処理の負荷見積りを示す．

まず，問合せ解析時最適化処理の負荷は，従来方式と比較して 7.5*(表数-1 ＋ 2*(変数の数)) ほど増加する．図 8 に示す問合せで試算すると，表数の数が 2，変数の数が 1 であるので，22.5 となる．単一表からたかだか 1 行を取り出すための fetch 処理に要する CPU 実行ステップ数比は 38.6 と比較すると，約 58%の負荷増加に相当する．動的 SQL のような，問合せ解析処理と問合せ実行処理が連続動作する場合，単一表からたかだか 1 行を取り出すだけで済む fetch 処理では，負荷増加が問題となることが考えられる．

また，問合せ実行時最適化処理の負荷は，従来方式と比較して 1.0*(変数の数) ほど増加する．図 8 に示す問合せで試算すると，変数の数が 1 であるので，1.0 となる．単一表からたかだか 1 行を取り出すための fetch 処理に要する CPU 実行ステップ数比は 38.6 と比較すると，約 3%の負荷増加に相当する．取り出す行数が増加すれば，負荷増加の割合がさらに小となる．したがって，問合せ実行時においては，負荷増加がたかだか約 3%となり，実システムへの適用性が確認できる．

5. 関連動向

5.1 並列問合せ機構について

パイプライン並列に関する問合せ機構については，Double pipelined hash join によって自明ではないブロッキングオペレータを除去する手法[18] が提案されている．

MapReduce が大規模データ解析に活用されつつある[19),20)]．利用者が key/value ペアを処理する Map と Reduce と呼ぶ 2 つの関数によって，大規模なデータ処理を記述する．具体的には，分散ファイルからの入力を，key にハッシュ関数を適用して分割し，各ノードに格納する Map と，それらを対象にデータ処理を行う Reduce からなる．しかし，MapReduce で提案されているデータ処理フレームワークは，すでに並列 DBMS に内在するものである[1)–3)]．提案方式では，SQL 文で指定されたジョイン処理では，2 つの表の結合処理を実行する前に，各表に指定される選択条件を適用して絞り込めるかどうかを問合せ実行時に判断する．もし絞り込める場合，選択条件を適用してからもう一方の表をアクセスする PNL 処理が実行される．絞り込めない場合，2 つの表をアクセスして，同一のハッシュ関数を適用して，各ノードに分散格納，すなわち Map を実行する．また，分散格納された 2 つの表どうしで結合処理，すなわち Reduce を実行する．SQL をサポートする並列データベースシステムに，フロータブルサーバ方式を併用し 2 段階最適化方式を適用するものは，他では議論されていない．

並列データベースシステムは，問合せの応答時間を短縮するために，複数の処理要素で実行を行いつつ，単一のシステムイメージで振る舞う．並列性の利点は，データ処理の対象が互いに独立に分割されていて，しかもパイプラン操作によって実行することによって得られる．XPRS では，2 段階で最適化処理を行う[21)]．最初の段階で，ジョイン処理の順序およびソート処理など処理手順を決定し，また第 2 段階で，通信処理を加味し，スケジューリング方法を提示するが，最初の段階で処理手順を決定する点で提案方式とは異なる．

5.2 問合せ最適化について

問合せ最適化の課題として，統計情報に基づいて処理手順を作成するのに必要とされる精度，正確さに限界があることである．問合せ最適化の役割は，問合せで参照する表に関連する各種統計情報に基づいて最適な処理手順を探すことである[16),17)]．これらの各種統計情報には，表，インデクス，あるいはそれらから導出される対象の，行数，列値の分布，格納状況などが存在する．また，問合せ最適化は実際の情報システムのストレージシステムでのシステム統計情報に依存する処理のコストモデルに基づいて，アクセスパス，結合方法，結合順序，問合せ変換を決定する．このコストモデルは処理手順の候補から最も効率が良いとされる処理手順を導くために用いられる．

しかし，コストモデルに基づく最適化に課題がある．その中で，正確とはいい難い統計情報に基づいて処理手順を予測することについて，その正確さには課題がある．たとえば，列に存在する値が，極端な場合には数個の特定の値が非常に高い頻度で出現し，一方で他の値は低い頻度で分布するような場合が考えられる．この場合，処理性能が大幅に異なる処理手順が値に依存して生成されることが分かっている．これに対処するために，ヒストグラムを用いて列の値に関する分布を正確に把握し，推定を精緻に行うことも考えられる[22)–24)]．

また，たとえ精緻な分布を統計情報として管理しても，比較した値が定数ではなく問合せで出現する変数である状況で処理手順が作成されることもあるので，必ずしも最適な処理手順が作成されない．この変数は，実行される前に，どの値が束縛されるのか事前に知ることはできない．このために，特定の値のために処理手順を作成することはできない．実用上では，列の値がとりうる分布が精緻に分かっていても，役には立たず，実際には定数と比較する場合よりも，変数と比較する場合の方が多いことが分かっている．もし，変数が束縛されるまで処理手順の作成を遅延させると，変数が束縛されるごとに動的 SQL を利用することになり，毎回 SQL の解析を行うことを意味する．この課題を解決する策として，問合せが初回に実行される場合，初回に設定される値に対して統計情報を利用して最適化処理によって処理手順を生成することも考えられるが，その値がその問合せを代表するという仮定には問題が残され，処理手順の最適性の課題は残る．このほかに，変数を含む問合せの最適化方式が提案されている[25)–27)]．

問合せ実行時まで情報が適用可能にならない限り，完全な処理手順の作成を遅らせることもなされている[25)–28)]．特に，choose-plan 操作を導入して，問合せ実行時に，複数の処理手順の選択肢から選択できる[25)]．本論文では，問合せ解析時に列の統計情報から変数に代入される値にかかわらず，最適な処理手順を選択できる点で異なる．また，並列データベースに適用している点でも異なる．

6. おわりに

並列データベースシステムにおいて，問合せ処理の並列化方式を提案した．具体的には，SQL の解析処理によって並列性を導き，具体的に並列に実行するためのパイプライン並列およびデータ並列の処理手順の表現方法，問合せ変換方法，並列実行環境など，実適用をともなう問合せ処理機構の実現方式およびその評価を示した．

負荷平準化を目的とするフロータブルサーバ方式は，プロセサ間での処理負荷の平準化のために，DB を格納していないプロセサに処理の一部を分担させる方法である．これによって，ジョイン処理にデータ並列およびパイプライン並列を適用することで，応答時間が改善される．また，サーバ数を増やすことにより各サーバの処理手順を起動するための処理時間およびスケーラビリティとの関係を評価した．さらに，フロータブルサーバ方式を好適に活用するために 2 段階からなる SQL の最適化方式を提案した．SQL 文を SQL 受付サーバが受け取って解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に最適化する段階からなる．大規模な表に対するジョイン処理およびソート処理は，処理手順の選択を間違えると大幅に性能が低下する懸念があるが，2 段階の最適化方式は，これらの課題を解決するものである．

著者らは，提案 DBMS をこれまで最大百数十台規模の情報システムに適用してきたが，今後クラウド環境でのデータベースサービスを提供するニーズも顕在化しつつあり，対応を考えたい．最後に，今後の課題について述べる．

- 並列ジョイン処理として，PNL 処理と PSM 処理を代替の処理手順として評価しているが，並列ハッシュジョインについても同様の評価を実施する．
- 千台を超えるサーバ環境でのシステム評価によって，提案方式の妥当性を検証する．

参 考 文 献

1) DeWitt, D.J. and Gerber, R.H.: Multiprocessor Hash-based Join Algorithms, *Proc. VLDB*, pp.151–164 (1985).
2) DeWitt, D.J., Gerber, R.H., Graefe, G., et al.: GAMMA – A High Performance Dataflow Database Machine, *Proc. VLDB*, pp.228–237 (1986).
3) Fushimi, S., Kitsuregawa, M. and Tanaka, H.: An Overview of The System Software of A Parallel Relational Database Machine, *Proc. VLDB*, pp.209–219 (1986).
4) DeWitt, D.J. and Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems, *Comm. ACM*, Vol.35, No.6, pp.85–98 (1992).
5) 根岸和義，土田正士，正井一夫：並列リレーショナルデータベースシステム HiRDB の概要と基本技術，電子情報通信学会データ工学研究専門委員会，Vol.95, pp.407–410, DE95-79 (1995).
6) Shimokawa, T., Takayama, H. and Masai, K.: Highly Scalable Parallel Relational DBMS, *Hitachi Review*, Vol.45, No.2, pp.67–70 (1996).
7) 正井一夫，根岸和義，土田正士ほか：更新処理を並列実行する UNIX 向け DBMS を開発—更新処理を並列実行する UNIX 向け DBMS を開発，日経エレクトロニクス，No.630 pp.101–114 (1995).
8) Tsuchida, M., Nakano, Y., Kawamura, N., et al.: Database management system and method for query process for the same, U.S. Patent 5,806,059 (1998-09-08).
9) Tsuchida, M., Masai, K. and Torii, S.: Method and system of database divisional management for parallel database system, U.S. Patent 6,192,359 (2001-02-20).
10) 土田正士，武藤英男：RDB における埋込み型問合せの最適化方式の提案，情報処理学会第 36 回全国大会 (1988).
11) Tsuchida, M. and Oomachi, K.: System for optimizing query processing in a relational database, U.S. Patent 5,091,852 (1992-02-25).
12) 岩田守弘，土田正士，根岸和義ほか：並列 DB サーバシステムにおける問合せ処理方式の提案，情報処理学会第 48 回全国大会 (1994).
13) DeWitt, D.J., Naughton, J.F., Schneider, D.A., et al.: Practical Skew Handling in Parallel Joins, Proc. VLDB, pp.27–40 (1992).
14) HiRDB Version 8 UAP 開発ガイド（第 5 版）．3020-6-356-40 (2009 年 1 月).
15) TPC-C Benchmark, available from (http://www.tpc.org/tpcc/spec/).
16) Satoh, K., Tsuchida, M., Nakamura, F., et al.: Local and Global Query Optimization Mechanisms for Relational Databases, *Proc. VLDB*, pp.405–417 (1985).
17) Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., et al.: Access Path Selection in a Relational Database Management System, *Proc. ACM SIGMOD*, pp.405–417 (1979).
18) Liu, B. and Rundensteiner, E.A.: Revisiting Pipelined Parallelism in Multi-Join Query Processing, *Proc. VLDB*, pp.829–840 (2005).
19) Pavlo, A., Paulson, E., Rasin, A., et al.: A Comarison of Approaches to Large-Scale Data Analysis, *Proc. ACM SIGMOD*, pp.165–178 (2009).
20) Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Proc. OSDI*, pp.137–150 (2004).
21) Hong, W. and Stonebraker, M.: Optimization of Parallel Query Execution Plans in XPRS, *Proc. Conference on Parallel and Distributed Information Systems*, pp.218–225 (1991).
22) Piatetsky-Shapiro, G. and Connell, C.: Accurate Estimation of the Number of Tuples Satisfying a Condition, *Proc. ACM SIGMOD*, pp.256–276 (1984).
23) Muralikrishna, M. and Dewitt, D.J.: Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, *Proc. ACM SIGMOD*, pp.28–36 (1988).
24) Haas, P.J., Naughton, J.F., Seshadri, S., et al.: Sampling-Based Estimation of the Number of Distinct Values of an Attribute, *Proc. VLDB*, pp.311–322 (1995).
25) Graefe, G. and Ward, K.: Dynamic Query Evaluation Plans, *Proc. ACM SIGMOD*, pp.358–366 (1989).
26) Cole, R.L. and Graefe, G.: Optimization of Dynamic Query Evaluation Plans,

Proc. ACM SIGMOD, pp.150–160 (1994).
27) Lee, A.W. and Zait, M.: Closing The Query Processing Loop in Oracle 11g, *Proc. VLDB*, pp.1368–1378 (2008).
28) Hellerstein, J.M., Franklin, M.J., Chandrasekaran, S., et al.: Adaptive Query *Processing: Technology in Evolution*, IEEE Data Engineering Bulletin, Vol.23, No.2, pp.7–18 (2000).

(平成 22 年 12 月 20 日受付)
(平成 23 年 3 月 30 日採録)

(担当編集委員　市川 哲彦)

土田　正士（正会員）

株式会社日立製作所ソフトウェア事業部先端情報システム研究開発部担当部長．1983 年筑波大学大学院理工学研究科修了後，同年株式会社日立製作所システム開発研究所を経て，2003 年より現職．博士（情報学）．著書に『SQL スーパーテキスト』（技術評論社），『SQL2003 ハンドブック』（SRC 社）．訳書に『SQL:1999 リレーショナル言語詳解』（J. メルトン著，共訳，ピアソン・エデュケーション），『標準講座 XQuery』（J. メルトン，S. バクストン著，共訳，翔泳社）．日本データベース学会理事，ISO/IEC JTC 1/SC 32 WG3 日本代表．情報処理学会情報規格調査会 SC32 専門委員会幹事および SC32/WG3(SQL) 小委員会委員，2005 年同標準化貢献賞受賞．日本データベース学会，ACM 各会員．

河村　信男（正会員）

株式会社日立製作所ソフトウェア事業部先端開発プロジェクト室主管技師．1981 年愛媛県立松山工業高等学校情報技術科卒業後，同年株式会社日立製作所システム開発研究所を経て，2002 年より現職．

中野　幸生（正会員）

株式会社日立製作所ソフトウェア事業部先端情報システム研究開発部主任技師．1982 年香川県立多度津工業高等学校電子科卒業後，同年株式会社日立製作所システム開発研究所を経て，2003 年より現職．

原　　憲宏

株式会社日立製作所ソフトウェア事業部 DB 設計部主管技師．1992 年東京工業大学理学部情報科学科卒業後，同年株式会社日立製作所システム開発研究所を経て，2004 年より現職．

石川　　博（フェロー）

静岡大学情報学部情報科学科教授．東京大学理学部情報科学科卒業．東京都立大学を経て 2006 年より現職．東京大学博士（理学）．著書に『次世代データベースとデータマイニング』（CQ 出版社），『JavaScript によるアルゴリズムデザイン』（培風館），『データベース』（森北出版）等．国際的論文誌 ACM TODS，IEEE TKDE，国際学会 VLDB，IEEE ICDE 等を含め学術論文多数．1994 年情報処理学会坂井記念特別賞，1997 年科学技術庁長官賞（研究功績者）受賞．情報処理学会データベースシステム研究会主査，情報処理学会論文誌（データベース）共同編集委員長，International Journal Very Large Data Bases Editorial Board，日本データベース学会理事歴任．電子情報通信学会フェロー．ACM，IEEE 各会員．