EC85−65

データストリーム指向関係データベースシステムの構成

中山雅哉・伏見信也
喜連川優・田中英彦
（東大）

1986年2月25日

社団法人 電子通信学会

EC85-65

データストリーム指向関係データベースシステムの構成

Organization of Data Stream Oriented Relational Database System

中山　雅哉　　伏見　信也　　喜連川　優　　田中　英彦
Masaya Nakayama　Shinya Fushimi　Masaru Kitsuregawa　Hidehiko Tanaka
東京大学　工学部
(Faculty of Engineering , The University of Tokyo)

1. はじめに

高度情報化時代の到来と共に，大量の情報を管理し，多種の要求に対応する為のシステム（データベースシステム）に対する重要性は，日々増大し続けており，種々の商用データベースシステムが開発されてきている。これらの商用データベースを含め，現在研究が進められているシステムの多くは，E.F.Coddにより1970年に提案された関係モデルに基づいている。

関係モデルは，高度なデータ独立性を実現する上では非常に良いモデルであるが，その実装に対しては，困難な点が多く，近年になって，やっと商用システムが登場するに至った。

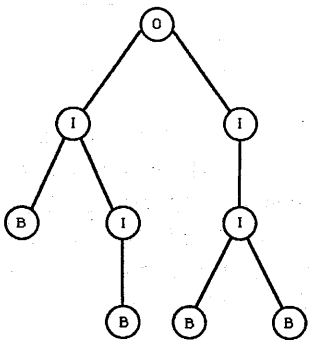
しかしながら，現在の商用システムにおいても，実用の点から見ると不十分な点が多く，特に，大規模なデータベースを効率良く取り扱うことのできるシステムは，皆無に近い。今後のデータベースは，情報の増大化と共に，益々大規模化する傾向にある為，大規模なデータベースを効率良く取り扱うことのできるシステムを研究，開発することは，急務であると考えている。

我々は，大規模なデータベースを効率良く取り扱う為に，従来のデータベースシステムとは異なり，データストリームを処理の単位として取り扱う，データベース処理システムの研究を進めており，現在，汎用計算機上に，その実装を行っている。本論文では，データストリームを単位とする演算処理の概念と，本システムの構成及び実装方式について，報告する。

以下，2章では，データストリームを単位とする演算処理の概念として，データストリームモデルを定義し，その概説を行う。また，3章では，ユーザからの問い合わせを，データストリームモデルに基づいて処理する際の手順を示し，4章では，これらの処理を効率良く実行する為に，複数のモジュールを用いて構成されるデータストリーム指向関係データベースシステムについて述べる。そして5章では，汎用計算機上に実装した試作システムでの，実装方式について説明する。最後に6章で，結論と今後の方針について述べる。

2. データストリームモデル

通常ユーザからの問い合わせは，図1に示すような問い合わせ木に展開されて処理が行われている。ここで，図1に示す各ノードは，選択演算（Selection），射影演算（Projection）等の関係代数演算を含む処理に相当し，アークは，各演算の結果が，次の演算の入力となることを示している。一般に，問い合わせ木の葉（基底タスク）は，ディスク上のリレーションに対する選択演算，射影演算等が対応し，根（出力タスク）は，ユーザに対する結果の出力（ディスプレイやＬＰ等）処理が対応する。また，中間のノード（中間タスク）には，結合演算（Join），集約演算（Aggregation）等の処理が対応する。



O Output Task
I: Internal Task
B: Base Task

図1．問い合わせ木の構成

我々は，このようなすべてのタスクに対して，図2に示すようなモデルを導入し，処理の一般化を行うことを試みている。図中のソース空間は，各タスクにおける入力源であり，この中のデータは，フィルタリング処理に適した構造で保持されていると仮定している。フィルタリング処理では，ソート処理や，通常の関係代数演算処理が各データに対して施される。クラスタリング処理では，次タスクにおけるフィルタリング処理に適した構造でデータを保持する為の処理，具体的には，ハッシュ操作が施され，シンク空間に格納される。図1に示したように，シンク空間は，次タスクにおいては，ソース空間としての役割を果たす。

このように，データは，ソース空間から，シンク空間へ流れる間に，フィルタリング処理，クラスタリング処理が施され，問い合わせ処理が実行されることになる。我々は，図2に示したモデルをデータストリームモデルと呼び，後に述べるシステムは，本モデルを効率良く実行するように，設計，制御されている。

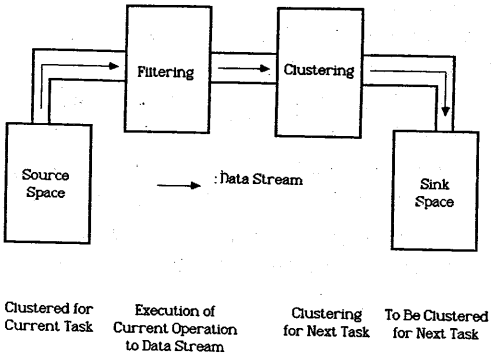


図2．データストリームモデル

本モデルでは，クラスタリング処理の存在により，中間タスク以降に関しては，ソース空間が，タスク内のフィルタリング処理に適した構造でデータを保持することが保証されているが，基底タスクでは必ずしも保証されていない。これに関しては，既に開発しているＧＫＤ木〔1〕構造によってリレーションをディスク上に格納することを考えている。

3．データストリームモデルに基づく関係代数演算処理

3.1．基底タスクにおける関係代数演算処理

前章で述べたように，基底タスクでは，ディスク上のリレーションに対する選択演算，重複除去を伴わない射影演算等の，各タプルに対して独立に施すことが可能な処理負荷の軽い演算が実行される（重複除去を伴う射影演算は，他のタプルとの関係を調べる必要がある為，中間タスクで実行される）が，その処理手順は，以下のようになる。

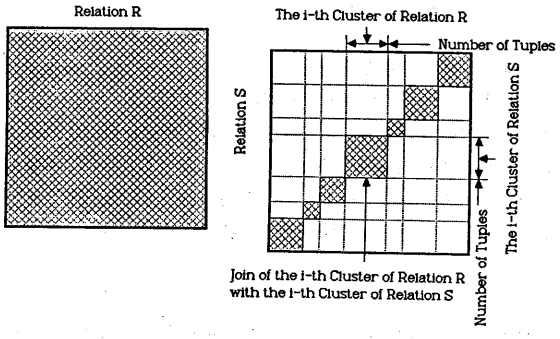
（ｉ）ＧＫＤ木を用いることにより，選択演算において必要となるリレーション内の必要最小限の論理ページを決定し，ソース空間とする。決定したソース空間内の各タプルに関して，選択演算，射影演算を実施する。

（ii）条件を満足した必要なアトリビュート列（結果タプルと呼ぶ）に対して，次タスクの為のクラスタリング処理を施し，シンク空間に格納する。

このように，基底タスクを，「ディスク上のリレーションに関する情報を保持して効率良くソース空間を決定し，各々のタプルに関係代数演算処理を実行するモジュール」と，「結果タプルに対して，次タスクの為のクラスタリング処理を施し，効率良くシンク空間を保持するモジュール」に分離して実現することは，有効な手段であると考えられる。

3.2．中間タスクにおける関係代数演算処理

中間タスクでは，結合演算，集約演算，重複除去を含む射影演算等の処理負荷の重い演算や，ソート処理を実行する。



（a）　　（b）

図3．ハッシュ操作による結合演算の処理負荷の軽減

これらの負荷の重い演算，例えば結合演算を，nested loop アルゴリズムにより実行すると，Ｏ（Ｍ・Ｎ）時間（Ｍ，Ｎ：リレーションのタプル数）だけ必要となり（図3（ａ）），システム全体の性能を大きく制限することになる。これに対して我々は，図3（ｂ）に示すように，各々のリレーションにハッシュ操作を施し，処理の負荷を大幅に軽減させる方法を提案してきた〔2〕。この方法では，『リレーションＡと，リレーションＢの等結合演算を行う場合，各リレーションに同じハッシュ関数を適用させると，違うハッシュ値を有するものは，結合

される可能性が無い。』ことに基づき，処理負荷をO.（Σmi・ni）時間（mi，ni：バケットiのタプル数）に抑えている。（ここでバケットとは，同じハッシュ値を有するタプルの集合を表している。）前節までに述べてきたクラスタリング処理は，ここで述べたハッシュ操作を行うことに相当する。明らかなように，ハッシュ操作は，次タスクで結合演算，集約演算を行うアトリビュートについて施される。このハッシュ操作を用いて結合演算を実行する方式は，最近その改良が進められている〔3〕。

また結合演算は，ソート処理を施してから実行すると，更に処理負荷を軽減することが可能となる。例えば，我々が研究，開発を進めているハードウェアソータ〔4，5〕を用いると，ソート処理がO（n）（n：データ数）で実行できる為，結合演算の処理負荷は，

$$O(\Sigma(mi+ni)) = O(M+N) \qquad (1)$$

として，選択演算の処理負荷と同じにすることができる。

また，通常のソートアルゴリズムを使用した場合でも，結合演算に対する処理負荷は，

$$O(\Sigma(mi+ni)\log(mi+ni)) \qquad (2)$$

に減少することができる。この場合は，バケット数の多いハッシュ関数を選び，各バケットの大きさを小さく抑えるようにすれば，結合処理を高速に実行することが可能となる。

以下に，中間タスクにおける処理手順を示す。

（i.）前タスクで，クラスタリング処理を施したソース空間のデータに対して，各バケット毎に，ソート処理を行う。
（ii）ソート処理を施したバケットに対して，結合演算，集約演算，重複除去を含む射影演算等の処理を行う。
（iii）結果タプルに対して，次タスクの為のクラスタリング処理を施し，シンク空間に格納する。

このように，中間タスクを，「ソート処理を効率良く実行するモジュール」，「ソート処理を施したデータ流に対して，結合演算，集約演算等の関係代数演算を施すモジュール」と，「次タスクの為に，結果タプルに対してクラスタリング処理を行い，シンク空間の管理をするモジュール」に分離して実現することは，有効であると考えられる。

3.3. 出力タスクにおける関係代数演算処理

前章で述べたように，出力タスクにおいては，ユーザに対する結果の出力を行っている。この為，このタスクでは，特に関係代数演算は施されずに，あるアトリビュートに対するソート処理程度が実行される。

以下に，出力タスクにおける処理手順を示す。

（i）必要があれば，前タスクでクラスタリング処理を施したソース空間のデータに対して，バケット順に，ソート処理を行う。
（ii）結果タプルを，順にディスプレイや，ＬＰに出力する。

このように，出力タスクでは，「中間タスクと同様のソート処理を効率良く実行するモジュール」と，「ディスプレイ，ＬＰ等に効率良く出力を行うモジュール」に分割して実現することは，有効であると考えられる。

4. データストリーム指向関係ベースシステム

4.1. システムの全体構成

前章で示したように，データストリームモデルに基づく関係データベースシステムを作成するには，いくつかの機能分割したモジュールで構成すると良く，その全体構成は，図４に示すようになる。

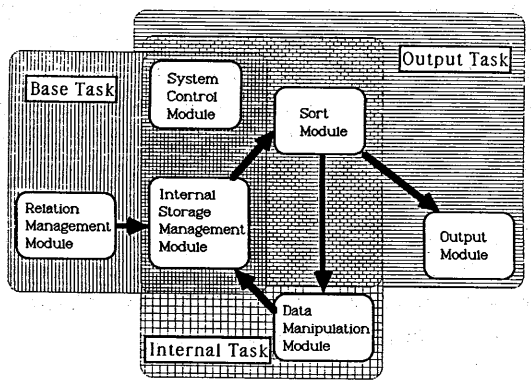


図４．データストリーム指向データベースシステムの構成

次節以降には，各モジュールの構成と，処理の概要について説明する。

4.2. Relation Management Module (RMM)

本モジュールでは，まず，3.1.で述べたように，ディスク上のリレーションに関する情報を保持することで，選択演算，射影演算を実行する際のソース空間を効率良く決定する。

従来のデータベースシステムにおいては，二次記憶系に対する入出力が，ほぼその性能を決定しており，本モジュールでは，いかに，二次記憶系に対する入出力回数を減少させるかが，大きな問題となる。通常は，アクセスの多いアトリビュートに対して，クラスタリング処理を行ってディスクに格納したり，二

次インデクスを設けたりしている。しかし，前者は，クラスタリング処理されていないアトリビュートに対しては，ディスク入出力の回数を抑える保証は無く，後者は，選択率の高い場合には，インデクス処理のオーバーヘッドが無視できなくなる等の欠点を有している。これらの欠点は，データベースが大規模になる程顕著となる為，好ましくない。これに対して我々は，既に開発を行っているＧＫＤ木構造〔1〕で，リレーションをディスク上で保持，管理することにより，平均的な入出力回数の大幅な減少を計る方法を検討している。この方法は，各アトリビュートの実現値の分布と，各々に対する問い合わせの分布に基づいて，データのディスク内でのページ分割方法を決定するものであり，リレーションの大きさには依存しない。よって，大規模なデータベースに対しても，効率良くデータをアクセスできる。

このように，必要最小限のディスクＩ／Ｏにより得られたデータは，個々に選択演算，射影演算等が施され，結果タプル群として，Internal Storage Management Module（ＩＳＭＭ）に送られる。

4.3. Internal Storage Management Module（ＩＳＭＭ）

本モジュールは，ＲＭＭや，Data Manipulation Module（ＤＭＭ）から送られてくる結果タプル群に対して，次タスクの為のクラスタリング処理を行い，効率良く中間リレーションを記憶するシンク空間のフェーズと，Sort Module （ＳＭ）の容量に基づいて，プロセシングクラスタの生成を行い，データ転送を行うソース空間のフェーズに分離することができる。

4.3.1. シンク空間としてのＩＳＭＭの処理

3.1.や，3.2.で示したように，シンク空間としてＩＳＭＭが働く場合には，次タスクで行う結合演算や，集約演算処理の為に，ハッシュ操作を行い，記憶の管理を行う。

ここで，ＲＭＭや，ＤＭＭから送られる結果タプルが，ＩＳＭＭで扱う記憶容量を越える場合は，何らかの方法を用いて仮想化を計る必要がある。これに対して，我々は，以下に示すような，方法を用いることにしている。

（ｉ）バケットの数の大きい適当なハッシュ関数を用いて，結果タプル群を複数のバケット空間に分割していく（同じハッシュ値をとる結果タプルの集合をバケットと呼ぶ）。

（ii）ＩＳＭＭの主記憶空間に空き領域がなくなった時には，その都度，適当なバケットを選択し，作業用のディスクに書き出し，空き領域を作る。この時，作業用ディスク内でのデータは，バケットとして認識できるような構造で保持しておく。

すべての結果タプルに対して，この処理が終了した時には，ある複数のバケット空間のみが主記憶を占め，他のバケットは，作業用ディスクに書き出された状態となる（図5）。

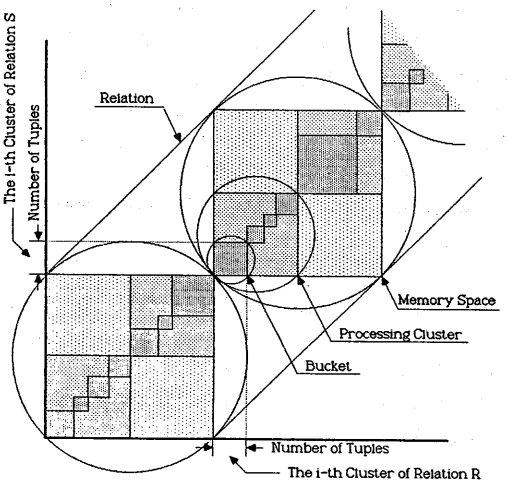


図5．シンク空間における記憶の仮想化

4.3.2. ソース空間としてのＩＳＭＭの処理

ソース空間としてＩＳＭＭが働く際には，3.2.に示したように，バケットを単位としてデータ転送のスケジュールを行い，ＳＭに転送する。このスケジュールは，主記憶上のものを優先してすべてのバケットについて行われる。作業用ディスク内のバケットは，主記憶上のバケットをＳＭに転送している間に，再ステージするようにスケジュールしておく。これにより，ＳＭに対して連続的なデータ流を供給することが可能となる。また，作業用ディスク内に，ＳＭで処理できない程大きなバケット空間が存在する場合には，そのバケットを再ステージする際に，シンク空間としてのＩＳＭＭの処理を再帰的に適用することで，処理可能な大きさのバケット群に分割することができる。

ＳＭに3.2.で述べた，ハードウェアソータを用いる場合は，その容量を有効に利用する為に，いくつかのバケットを統合して一度に処理することが望ましい。このように，複数のバケットを統合し，ソータ容量に近くなるようにすることを，プロセシングクラスタへの統合と呼んでいる。この場合，ＳＭ，ＤＭＭにおける処理の単位は，プロセシングクラスタとなる。

これに対し，ＳＭで通常のソートアルゴリズムを用いる場合には，3.2.の（2）式で示されるように，プロセシングクラスタへの統合が却ってオーバーヘッドを増すことになる為，統合は行わない。

4.4. Sort Module （ＳＭ）

本モジュールでは，ＩＳＭＰから送られてくるデータ流にソート処理を施し，ＤＭＭ又は，Output Module （ＯＭ）に転送する。

ソート処理は，通常Ｏ（Ｎ logＮ）時間だけ必要とするが，

プロセッサを logN段接続し，処理を並列に行えば，O（N）時間で実行することができる。この為，ハードウェアソータに関する研究は，盛んに行われており，我々も logN段のプロセサを用いたパイプラインマージ方式によるハードウェアソータの設計，試作を行っている。試作したソータは，3MB／sec の処理能力を持っている〔4，5〕。

ハードウェアソータを用いてソート処理を行う場合，次の2つの理由から，前節で述べたプロセシングクラスタへの統合を行うことが望ましい。

（a）入力されたデータ流は，その数によらず必ず logn（n：ソートユニットの台数）だけの遅延を生じて処理される。よって，入力データの数が非常に少なく，データ長（タプル長）も短い場合には，そのオーバーヘッドを無視できなくなる。プロセシングクラスタへの統合を行えば，オーバーヘッドを少なくすることができる。

（b）入力されるデータの数に変動が大きい場合，図6に示すように，連続してソート処理を施すことができなくなる。プロセシングクラスタへの統合を行うと，入力データの変動をある程度小さく抑えることができる。

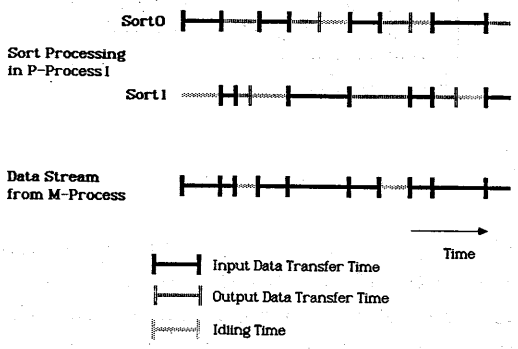


図6．入力データ数の変動に対するソート処理の乱れ

4.5. Data Manipulation Module (DMM)

本モジュールでは，SMによりソート処理を施したデータ流に対して，結合演算，集約演算，重複除去を伴う射影演算等の処理を行う。

各々の演算とも，データ流がソートされている為に，施す処理は簡単なものになる。例えば，結合演算においては，リレーションIDの異なる，キーの等しいタプル同志を順に結合する操作ですみ，必要とするアトリビュートに関して射影演算を施して結果タプルとすればよい。また，重複除去を伴う射影演算では，キーの等しいタプルは，その一つだけを結果タプルとするように処理を施せばよい。集約演算に関しても同様に簡単なものとなる。

4.6. Output Module (OM)

本モジュールでは，SMから受け取ったデータ流をディスプレイや，LPに効率良く出力する処理を行う。

4.7. System Control Module (SCM)

本モジュールでは，ユーザから得られる問い合わせ木に基づいて，上記の各モジュールに処理を割り当て，発火を行う。データストリームモデルに基づく本システムおいて，各モジュールは，データ流が流れている間は制御を受けないで処理を進めることができる。この為，制御のオーバーヘッドは，各モジュールの割りつけと，発火だけに抑えることができる。

5. 試作システムの実装方式

5.1. 試作システムの全体構成

現在我々は，MELCOM80／500 (OS：DPS10) 上で，データストリーム指向関係データベースシステムの試作を進めている〔6，7〕。そのハードウェア構成は，図7に示す通りであり，**図中のデータベースプロセッサは，4.4.で述べたハードウェアソータを中心とした処理モジュールである〔8〕。**

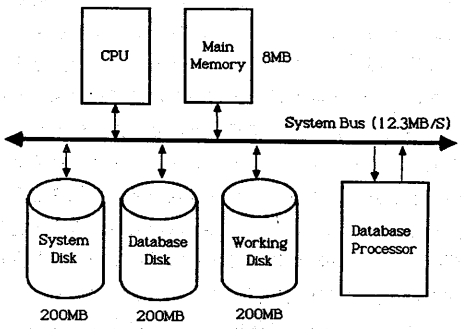


図7．試作システムのハードウェア構成

また，試作システムは，図8に示すような複数のプロセスを用いて実装を行っており，前章の各モジュールとの対応は，次に示す通りである。

D－プロセス　：Relation Management Module
P－プロセス1：Sort Module
P－プロセス2：Data Manipulation Module
O－プロセス　：Output Module
C－プロセス　：System Control Module

また，Internal Storage Management Moduleは，主記憶上のバケット管理と，作業用ディスク上のバケット管理を分離して，それぞれM－プロセスとWD－プロセスとして実装している。これらは，問い合わせ木の中間タスクにおいて，ソース空間とシンク空間の２つの働きを同時に行う必要がある為，それぞれ２つずつ用意している。

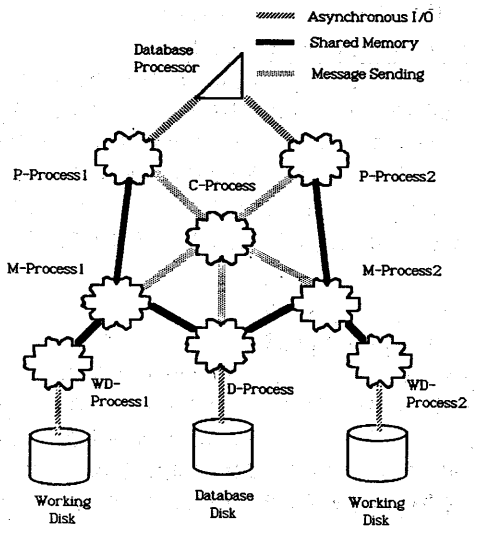


図８．試作システムの全体構成

本試作システムでは，データ流が複数のプロセス間で通信され，処理が施される。我々のシステムにおいてデータ流は，非常に量の多いものとなる。この為，実際のデータ流をメッセージ交換方式等で通信することは，オーバーヘッドが大きくなり，好ましくない。そこで我々は，主記憶上に各プロセスで共通に利用できる領域（共有メモリ領域）を採り，実際のデータはこの領域に置いたまま，データのアドレスとサイズだけをプロセス間で通信する方法により，プロセス間のデータ転送を実現している。また，ディスクとのＩ／Ｏにおいては，入出力完了の為の同期を，ユーザが陽に指定できる特殊なＩ／Ｏ（非同期Ｉ／Ｏと呼ぶ）を用いて実装を行っている。これは，Ｄ－プロセス，ＷＤ－プロセス等が，連続的にデータ流を生成する為に必要となる重要な機能であるが，その詳細に関しては，次節で述べる。

次節以降では，各プロセスの実装方式について説明する。

5.2. Ｄ－プロセスの実装方式

本プロセスでは，ディスク上のリレーションを読み込み，各タプルに対して，選択演算，射影演算の処理を行う。ここで，ディスクからのデータの読み込みと，関係代数演算処理を並行して実行することで，Ｍ－プロセスへの連続したデータ流を供給することが可能となる。しかし通常の，ＯＳがサポートするＩ／Ｏでは，Ｉ／Ｏを行ったプロセスが入出力完了待ちとなり，ＣＰＵが他のプロセスの実行に開放されてしまう為，このような環境を実現することはできない。さらに，本システムの基底タスクでは，他のプロセスの実行が，Ｄ－プロセスの供給するデータ流により駆動される為，システムの処理能力が十分に活かせないことになる。そこで我々は，ディスクとのＩ／Ｏには，入出力完了の同期をユーザ（プロセス）が陽に指定できる比較的低位レベルの非同期Ｉ／Ｏを用いて実装を行っている。また，この非同期Ｉ／Ｏは，ディスクに対するRaw Ｉ／Ｏであり，ユーザの指定した主記憶の空間とディスクとのデータ転送を，直接行うことができる。それに対し，通常のＩ／Ｏでは，ＯＳ内の特定の領域（入出力バッファ）とだけしか，ディスクとの間でのデータ転送ができないようになっている為，ユーザ空間と入出力バッファとのメモリ間の転送が必要となる。この点でも，非同期Ｉ／Ｏでの，入出力のオーバーヘッドは非常に小さくなる。

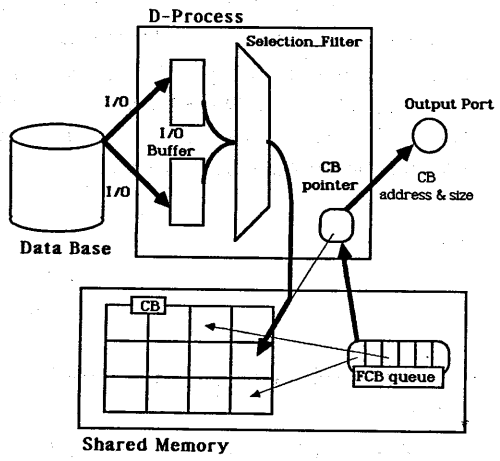


図９．Ｄ－プロセスの構成

非同期Ｉ／Ｏと，ダブルバッファリングを用いて実装したＤ－プロセスの構成を図９に示す。前節で述べたように，プロセス間通信の為に，共有メモリ領域上に，ＣＢ（Communication Buffer）を採り，フィルタリング処理を行った結果タプルは，ここに書き出される。ＣＢが，一杯になったら，Output Port（M1 Port又は，M2 Portのどちらか）にＣＢのアドレスとサイズを送る。また，次のＣＢは，図中のＦＣＢ Queue（Free Communication Buffer Queue）から取り出して用いる。ＦＣＢ Queue中には，ＣＢのアドレスとサイズが書かれている。ＦＣＢ Queueが，共有メモリ上にあるのは，Queueからの取り出しと，Queueへの追加が，別のプロセス（この場合は，Ｄ－プロセスと Ｍ－プロセス）から行われる為である。本システムでの共有メモリ上の Queueは，DPS10によりサポートされている Named Queue機能を使用しており，Queueの管理は，ＯＳが行っている。

ソース空間の絞り込みを有効にし，ディスク入出力の回数を

少なくする手法として，ＧＫＤ木構造〔1〕での物理編成法を挙げてきたが，ＧＫＤ木構造をディスク上に保持する為には，リレーションの各アトリビュートの実現値の分布と，それぞれに対する問い合わせの分布を調べる必要がある。アトリビュートの実現値の分布は，比較的簡単に求められるが，問い合わせの分布は，実際に使用されるリレーションの性質や，利用環境等に大きく左右され，その一般的性質を有する分布を得ることは容易でない。この為，現時点における試作システムでは，このＧＫＤ木による方式は，実装されていない。

5.3. Ｍ－プロセスとＷＤ－プロセスの実装方式

これらのプロセスは，タスクのソース空間として用いられる場合と，シンク空間として用いられる場合で処理が異なることは，既に述べてきた。また，これらは，5.1.で述べたように処理の機能上分離したにすぎない。ここでは，まず個々のプロセスの構成を示し，後に，記憶の仮想化方式の実装という点から，２つのプロセスをまとめて説明する。

5.3.1. Ｍ－プロセスの実装方式

5.3.1.1. シンク空間としての処理の実装

本プロセスをシンク空間として用いる時の実装方式を図10（ａ）に示す。ここでは，前節で述べたように，共有メモリ上のＣＢアドレスとサイズを Ｍ1 Portから受け取り，各タプルに対して，ハッシュ操作を施してバケットを決定し，決められた領域に格納する。ここで，Ｍ－プロセスの主記憶空間（Ｍ空間）は，ＷＤ－プロセス，Ｐ－プロセス１との間で，データ転送が行われる為，共有メモリ上にとられる。

大規模なリレーションを，ハッシュ操作を用いて適当な大きさの空間に分割し，処理を行う方式は，他のシステムにも見ることができるが〔3〕，その方式では，静的に処理空間を決定しておき，それ以外のものは作業用ディスクに書き出すという操作を反復して適用する方法を採っている。この為，ステージング処理にかかるトータルＩ／Ｏ回数Ｔは，ハッシュ関数の値域の数をＨ，適用するソース空間（リレーションの一部）の大きさをＳ，Ｉ／Ｏ用のバケットサイズをＢとすると，次式のようになる。

$$T = \lceil S/B \rceil + 2\sum_{i=1}^{H} \lceil S \cdot (H-i)/B/H \rceil \quad (3)$$

これに対して我々は，4.3.に示したように，作業用ディスクに書き出すバケットを動的に決定し，再ステージ時の順序をスケジュールする方法を用いている為，

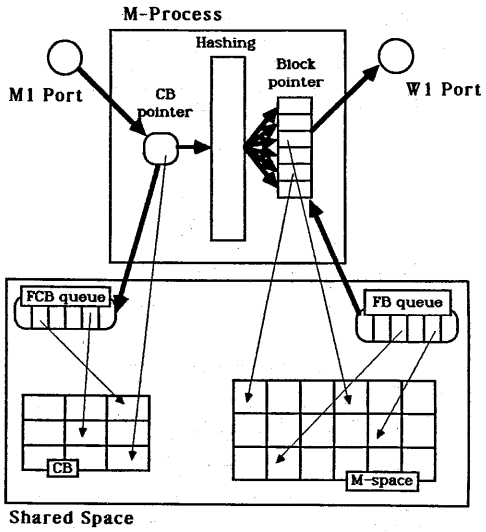
$$T = \lceil S/B \rceil + 2\lceil S \cdot (H-1)/B/H \rceil \quad (4)$$

とすることができる。

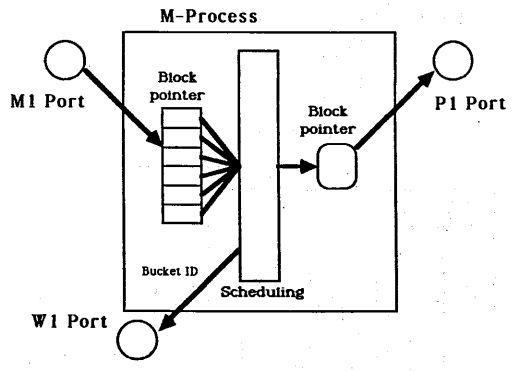
また，上記（３）式は，処理を行うバケットの大きさがすべて主記憶上にのりきる程度に小さい場合であり，非常に大きなバケットがある場合には，さらに多くのディスクアクセスが必要となる。それに対し，我々の方法では，シンク空間としての処理が終了した時点で各バケットの大きさがわかっているので，**4.3.2.で示したように，再ステージの際に再びハッシュ処理を施すことができ，たいていの場合には，（４）式で示した回数のＩ／Ｏでステージングが行なえる。4.2.で述べたように，データベースの環境では，二次記憶系との入出力がシステムの性能を制限することが多く，Ｉ／Ｏ回数の少ない処理方式を用いることは，性能向上の為に，大きな意味を持つ。**

5.3.1.2. ソース空間としての処理の実装

Ｍ－プロセスをソース空間として用いた時の実装方式を図10



（ａ）シンク空間としてのＭ－プロセスの構成



（ｂ）ソース空間としてのＭ－プロセスの構成

図10．Ｍ－プロセスの構成

（b）に示す。ここでは，まず，各バケット毎のタプル数を基にしてプロセシングクラスタを生成し，P1 Portと共有メモリを用いてデータの転送を行う。ここで，プロセシングクラスタの生成方法は，以下に示す通りである。

（i）既に存在するプロセシングクラスタPCi に，そのバケットが統合できるか調べ（PCi のタプル総数に，バケットのタプル数を加え，ソータ容量を越えるかどうか調べる），統合できれば，PCi の構成バケットに当バケットを登録し，PCi のタプル総数を書き換える。さもなければ，PCi+1 について調べる。
（ii）統合できるプロセシングクラスタがない場合には，新しいプロセシングクラスタとして登録を行う。
これを，すべてのバケットについて行えばよい。

この方法を実装する為に，図10（b）に示すように，PCMT（Processing Cluster Management Table）と，BLT（Bucket Link Table）を用いている。

プロセシングクラスタの生成時に，再ステージのスケジュールも同時に行うことができる。この為，スケジュールにより要求される順にバケットIDを，WD1 Portに転送しておけば，M1 Portから要求順にデータの転送を受けることができる。

5.3.2. WD－プロセスの実装方式

5.3.2.1. シンク空間としての処理の実装

本プロセスをシンク空間として用いる時の実装方式を図11（a）に示す。ここでは，M－プロセスから送られてくるデータを，非同期I／Oとダブルバッファを用いて，作業用ディスクに書き出す。各データは，次節で述べるように，バケット単位でまとめて扱うことができるように管理される。

5.3.2.2. ソース空間としての処理の実装

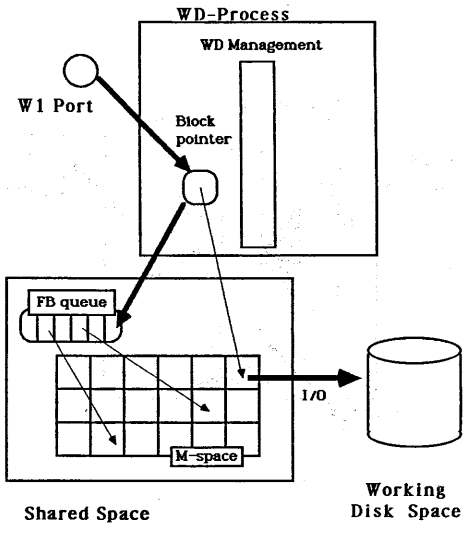
WD－プロセスをソース空間として用いる時の実装方式を図11（b）に示す。WD1 Portからは，先に述べたように，要求されるバケットIDがM－プロセスから送られる。その各バケットについて，次節で述べるBMT－WD（Bucket Mangement Tabke for Working Disk），WD－LT（Working Disk Link Table）を用いることにより，作業用ディスクから読み込み，M1 Portにデータの転送を行う。この時，読み込み先の主記憶領域は，FB Queue（Free Block Queue）から得られる共有メモリ上の領域を用いている。

5.3.3. 記憶の仮想化の実装法

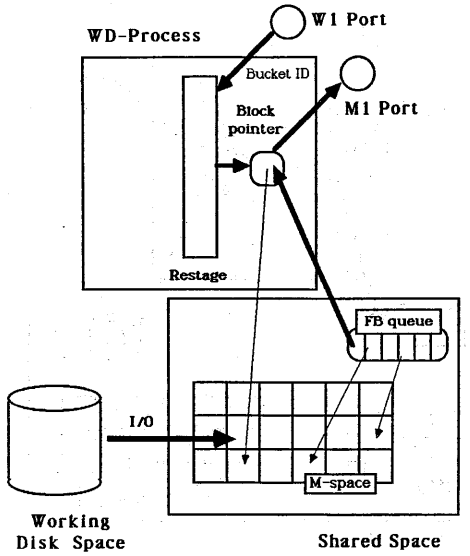
試作システムにおけるM空間は，図12に示すように，複数のブロックの集りとして管理され，バケット空間に対する動的な領域の割り当ては，このブロックを単位として行われる。また，**試作システムでは，4.3.1.で述べた追い出しバケットの選択アルゴリズムとして，以下に示す方法で実装している。**

（i）既に追い出されているブロックを有するバケットで，データが1ブロック以上に渡って主記憶上に存在していれば，そのバケットの各ブロックを追い出す。
（ii）そうでなければ，現時点で，最も多くのブロックを主記憶上に有するバケットの各ブロックを追い出す。

M－プロセス内のBMT－M（Bucket Management Table for Memory）は，各バケットの先頭ブロック，最終ブロック，トータルのブロック数，最終ブロック中のタプル数，スワップアウトフラグを保持するものであり，これを用いて，追い出しバケットが決定される。また，M－LT（Memory Link Table）は，M空間中の各ブロックのアドレスと，次ブロックへのポインタを保持しており，これら2つのテーブルを用いて，主記憶



（a）シンク空間としてのWD－プロセスの構成



（b）ソース空間としてのWD－プロセスの構成

図11. WD－プロセスの構成

中のバケットの管理を行っている。

WD－プロセス内のBMT－WD（Bucket Management Table for Working Disk）は，各バケットの作業用ディスク内の先頭ブロックと最終ブロックを保持している。WD－LT（Working Disk Link Table）は，作業用ディスク内での各ブロックのアドレスと，次ブロックへのポインタを保持しており，これら2つのテーブルを用いて，作業用ディスク内のバケットの管理を行っている。

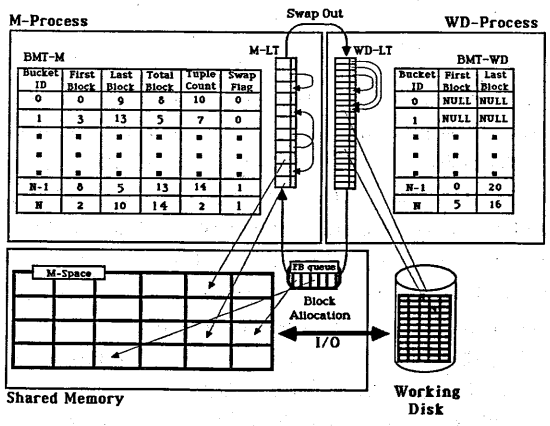


図12．記憶の仮想化の実装方式

5.4．Ｐ－プロセス１とＰ－プロセス２の実装方式

本プロセスの実装方式を図13，図14に示す。両図中のデータベースプロセッサは，4.4.で述べたハードウェアソータを含むもので，構成は，図15の様になっている。ホストとのインターフェイスは，ディスクにおけるそれと同様にしてあり，ソフトウェアの立場からは，先に述べた非同期Ｉ／Ｏを用いて通常にディスクとＩ／Ｏを行うようにしてプログラミングしている。現在，このデータベースプロセッサは，実装を行っている最中であり，まだ完全にホストと結合されておらず，今回は，ソート処理をソフトウェアで実装を行っている。

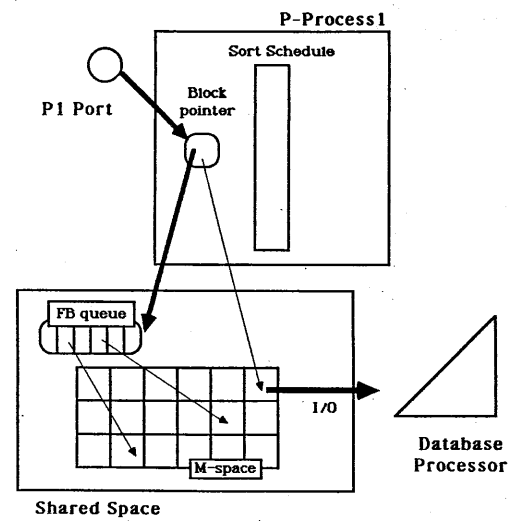


図13．Ｐ－プロセス１の構成

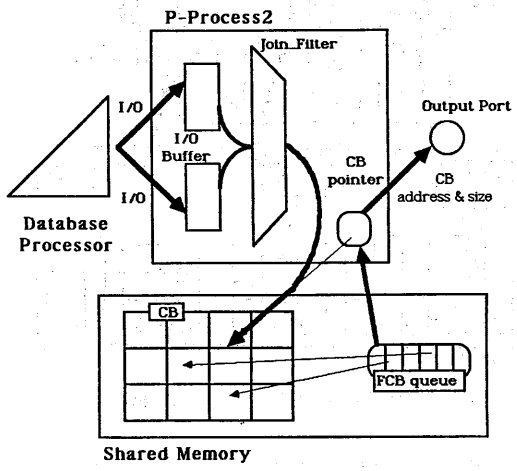


図14．Ｐ－プロセス２の構成

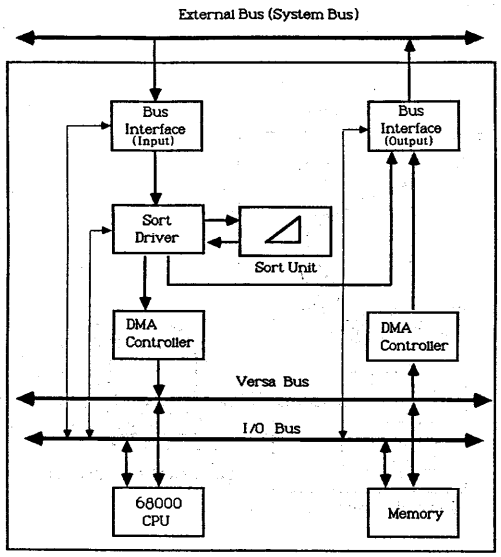


図15．データベースプロセッサの構成

今回実装を行った方式では，図16に示すように，両プロセス間で共有メモリを用いてデータ転送するようにし，他のプロセスとのインターフェイスは同じになるようにしてある。

Ｐ－プロセス２では，4.5.で述べた方法により結合演算等の処理を実行し，結果タプルは，5.2.で述べたように，共有メモ

リ上のＣＢに置いて，Output Portを用いたデータ転送を行っている。

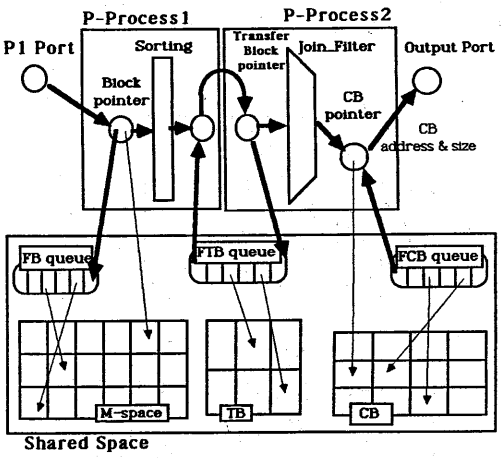


図16．今回実装したＰ－プロセスの構成

5.5．Ｏ－プロセスの実装方式

本プロセスは，図17に示すように実装される。現在，このプロセスでは，ＯＳのサポートする通常のＩ／Ｏを用いて結果の出力を行っており，特に特別な操作を施していない。

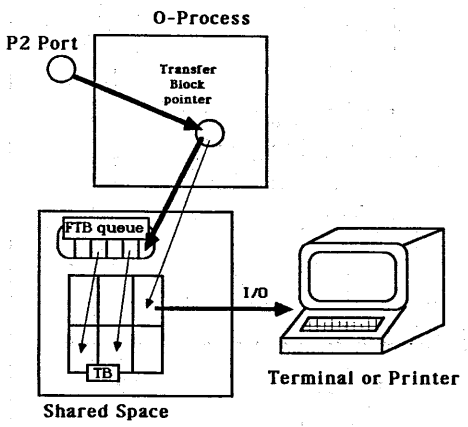


図17．Ｏ－プロセスの構成

5.6．Ｃ－プロセスの実装方式

本プロセスは，問い合わせ木に従ってプロセスを割り当て，各々に対して，処理の方法に関する情報と，Output Port先（M1 Port又は，M2 Port）や，ソータ容量等の制御情報を送ることにより，プロセスを発火する。各プロセスから終了通知を受けると，次のタスクの処理を行う。

6．おわりに

今回は，我々の提案するデータストリームモデルについて概説し，このモデルを有効に実行する為のシステムの構成と，処理手順について示した。また，現在，汎用計算機上に実装している試作システムの実装方式について述べた。

本論の中でも述べたように，試作システムは，まだ，完全な実装を終えておらず，今後は，データベースプロセッサを接続して，実際にＯ（ｎ）時間で処理が行われることを確認することを目指すと共に，ＧＫＤ木構造でのディスク管理方式を実装して，従来のデータベースシステムとの性能の比較を行っていく方針である。

参考文献

〔1〕Fushimi,S. et al,「Algorithm and Performance Evaluation of Adaptive Multidimensional Clustering Technique」, ACM SIGMOD Int. Conf. of Management of Data, (1985)

〔2〕Kitsuregawa,M. et al,「Architecture and Performance of Relational Algebra Machine GRACE」, Proc. of 1984 Int. Conf. on Parallel Processing, (1984)

〔3〕DeWitt,D.J. et al,「Multiprocessor Hash-Based Join Algorithms」, Proc. of VLDB 85, (1985)

〔4〕喜連川，他，「パイプラインマージソータの構成」，電子通信学会論文誌，J66-D，No.3，(1983)

〔5〕楊，他，「ブロック分割記憶管理法によるパイプラインマージソータ」，情報処理学会第31回全国大会，1B-9，(1985)

〔6〕伏見，他，「データベースマシンＧＲＡＣＥのプロトタイプシステム」，情報処理学会第31回全国大会，1B-6，(1985)

〔7〕中山，他，「ＧＲＡＣＥプロトタイプシステムにおけるソフトウェア構成」，情報処理学会第31回全国大会，1B-7，(1985)

〔8〕鈴木，他，「ＧＲＡＣＥプロトタイプシステムにおけるプロセッシングモジュールの設計」，情報処理学会第31回全国大会，1B-8，(1985)

（盛文社）