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

社团法人電子通信学会
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ためのシステム（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、携帯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ができたシステムは、皆無に近い。今後データベースは、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が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、その機能を行う。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、複数のモデ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、アーキは、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L P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。

図1. 問い合わせ木の構成

第 67 回
我々は、このようなタスクに対して、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、各タスクにおける人力源であり、この中のデータは、フィルタリング処理に起因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、ソース処理や、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起因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込む際に、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起因した構造でデータを保持することが確認されているが、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、既に発表しているGKD木[1]構造をもってリレーションを従来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の各タスク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、他のタスク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、以下のようにある。

(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、ソース空間として、決定したソース空間内の各タブレットに関して、選択演算、射影演算を行う。

(ii) 条件を満たした必要なアトリビュート列（結果タブレットと呼ぶ）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、基底タスクでは、ディスク上のリレーション毎の情報を保持して効率良くソース空間を決定し、各々のタブレットに関係代数演算処理を実行するモジュールと、「結果タブレットに対して、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事は、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、集合演算、多重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。

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：リレーションのタブレット数）だけ必要となり、図3（a））と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ている[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、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 \( O(\Sigma m_i \cdot n_i) \) 時間（\( m_i \cdot n_i \) がパケットのタブル数）に抑えてい る。（ここでパケットとは、同じハッシュ値を有するタブルの集合を表している。）前節で述べてきたクラスターング処理は、ここで述べたハッシュ操作を行うことに相当する。明らかのように、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、

\[
O(\Sigma (m_i \cdot n_i)) = O(M+N)
\]

（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、

\[
O(\Sigma (m_i \cdot n_i)) \log(m_i \cdot n_i)
\]

（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、クラスター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パケットに対して、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) 結果タブールに対して、次タスクの為のクラスターング処理を施し、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、結果タブールに対してクラスターング処理を行 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

前節で述べたように、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ーング処理を施した 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) 結果タブールを、順にディスプレイや、HP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、H P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、有効であると思われる。

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

前章で述べたように、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

次節以降では、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、ディスプレイ上 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 削出力が、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、クラスターング処理を行ってディスプレイに格納したり、
4.3. Internal Storage Management Module (ISMМ)

本モジュールは、RMMや、Data Manipulation Module (DMM) から送られてくる結果タブ群に対して、次タスクの為のクラスタリング処理を行い、効率良く周辺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、プロセッ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ISMМの処理

3.1、3.2で示したように、シンク空間としてISMМ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または、DMMから送られる結果タブ群がISMМ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、方法を用いている。

(i) パケット数の数の大きい適当なハッシュ関数を用いて、結果タブ群を複数のパ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ブの集合をパケットと呼ぶ）。
(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、適当なパケットを選択し、作業用のディスクに書き出し、空き領域を作る。この時に、作業用ディスク内でのデータは、パケットとして認識できるような構造で保持しておく。

すべての結果タブ群に対して、この処理が完了した時には、ある複数のパケット空間のみが主記憶を占め、他のパケットは、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. ワーカス空間としてISMМの処理

ワーカス空間としてISMМが働く際には、3.2に示したように、パ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パケットについて行われる。作業用ディスク内のパケットは、主記憶上のパケットをSMに転送している間に、再スケジュールすることをスケジュールしておく。これにより、SMに対して流動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パケット空間が存在する場合には、そのパケットを再スケジュールする際、シンク空間としてISMМの処理を再帰的に適用することで、処理可能な大きさのパケット群に分割することができる。

SMに3.2で述べた。ハードウェアソーダを用いる場合は、その容量を有効に利用する為に、いくつかのパ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、複数のパケットを統合し、ソケット容量に近くなるようにすることを、プロセッ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、プロセッ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、プロセッシングクラスタへの統合が一結で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、ISMМ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（NlogN）時間だけ必要とするが、
プロセッサを \( \log N \) 段接続し、処理を並列に行えば、0 \( (N) \) 時間で実行することができる。この為、ハードウェアソースに関する研究は、盛んに行われており、我々も \( \log N \) 段のプロセッサを用いたデータベースシステムによるハードウェアソースの設計、試作を行っている。試作したソースは、3 MB/sec の処理能力を持っている（4, 5）。

ハードウェアソース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ッ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の数が \( \log N \)（ノードユニットの台数）だけ変更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、データ長に（段階長）も短い場合には、そのオーバーヘッドを無視できない。プロセッシングクラスタへの統合を行うければ、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が変動が大きい場合、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。プロセッシングクラスタへの統合を行うと、入力データの変動のある程度小さく抑えることができる。

では、キーの等しいタブレは、その一つだけを結果タブレ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に効率よ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システムにおいて、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の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、M86COMBO/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行っている（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スを中心とした処理モジュールである（8）。

図6．入力データの変動に対するソート処理の流れ

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、重複除去と伴う射影演算等の処理を行う。

各々の演算と、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 \( \times D \) の異なるキーの等しいタブル同志を順に結合する操作です。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ブルとすればよい。また、重複除去を伴う射影演算

図7．試作システムのハードウェア構成
また、Internal Storage Management Moduleは、主記憶上のバクル管理と、作業用ディスク上のバクル管理を分離して、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、問い合わせ中の中断タスクにおいて、ソース空間とシンク空間の2つの個を同時に行う必要がある事、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形式等で通信することは、オーバーヘッドが大きく、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てしま、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1/Oにおいては、入出力完了の為の同期を、ユーザが個に指定できる特殊な1/O（非同期1/Oと呼ぶ）を用いて装実行を行っている。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、次の節に述べる。

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、各タスクに対して、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、関係格子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1/Oでは、1/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。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1/Oには、入出力完了の同期をユーザ（プロセス）が個に指定できる比較的低位レベルの非同期1/Oを用いて実装を行っている。また、この非同期1/Oは、ディスクに対するRaw 1/Oであり、ユーザの指定した主記憶上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1/Oでは、OS内の特定の領域（入出力バッファ）としか、ディスクとの間でのデータ転送ができる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1/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1/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、共有メモリー領域上に、CB（Communication Buffer）を採り、フィルタリング処理を行った結果タブは、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FCS Queue (Free Communication Buffer Queue)から取り出して用いる。FCS Queueでは、CBのアドレスとサイズが警告されている。FCS Queueが、共有メモリー上にあるのは、Queueからの取り出しと、Queueへの追加で、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ー上のQueueは、DSP10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。

ソース空間の経り込みを有効にし、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散布と，それぞれ
に対する問い合わせの散布を調べる必要がある。アトリビュー-
トの実現値の散布は，比較的簡単に求められるが，問い合わせ
の散布は，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散を得ること
は容易でない。この為，現時点における作業システムでは，こ
のGKD木による方式は，実装されていない。

5.3. メープロセスとWMD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，シングルスペース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
理の機械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. メープロセスの実装方式

5.3.1.1. シングルスペースの実装方式

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アドレスサイズをM1 Portから受取る。各テーブルに対して，ハッシャ操作を施してパケットを決定し，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（＝空間）は，WMDプロセス，M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，ハッシャ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探っている。この為，ステージ
ング処理にかかるトータル1/O回数Tは，ハッシャ関数の値
域の数をN，適用するソース空間（リレーションの一部）の大
きさをS，1/O用のパケットサイズをBとすると，次式のよ
うになる。

\[ T = \left( \frac{S}{B} \right) + 2 \sum_{i=1}^{M} \left( \frac{S \cdot (H-i)}{B \cdot H} \right) \] （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
に書き出すパ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，

\[ T = \left( \frac{S}{B} \right) + 2 \sum_{i=1}^{M} \left( \frac{S \cdot (H-1)}{B \cdot H} \right) \] （4）

とすることができる。

また，上記（3）式は，処理を行うパケットの大きさがすべ
て主記憶上のに取り得る程度に小さい場合で，非常に大きな
パ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，シングルスペースとして
の処理が終了した時点で各パケットの大きさがわかっているので，
4.3.に示したように，再ステージの際に再びハッシャ処理を
施すことができる。たいていの場合には，（4）式で示した回数
の1/Oでスケジュールが行なえる。4.2.で述べたように，デ
ータベースの環境では，二次記憶系の高い出力がシステムの性
能を制限することが多く，1/O回数の少ない処理方式を用い
るときは，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) シングルスペースとしてのMプロセスの構成

(b) メモリスペースとしてのMプロセスの構成

図10. Mプロセスの構成

—73—
(b) に示す。ここでは、まず、各バケット毎のタブル数を基にしてプロセッシングクラスタを生成し、PI Portと共有メモリを用いてデータの転送を行う。そこで、プロセッ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ッシングクラスタPCIに、そのパケットが統合できるか調べる（PCIのタブル数に、パケットのタブル数を加え、ソーラ容量を超えるかどうか調べる）、統合できれば、PCIの構成パケットに当バケットを登録し、PCIのタブル数を書き換える。さもなくば、PCIiを1つについて調べる。

(ii) 統合できるプロセッシングクラスタがない場合には、新しいプロセッシングクラスタとして登録を行う。

これを、すべてのパケットについて行えばよい。

この方法を実装する場合に、図10(b)に示すように、PCM T (Processing Cluster Management Table)と、BLT (Bucket Link Table)を用いている。

プロセッシングクラスタの生成時に、再スケジュールも同時に行うことができる。したがって、スケジュールにより要求される順にバケットI Dを、WI Portに転送しておけば、M I 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と、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、パケット単位でまとめて扱うことができるよう管理される。

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。WI Portから、先に述べたように、要求されるパケットI DがM-プロセスから送られる。その各パケットについて、次節で述べるBMT-M (Bucket Management Table 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、

M I 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的領域を用いている。

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まりとして管理され、パケット空間に対して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.に述べた追い出しパ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パケットで、データが1ブロック以上に渡って主記憶上存在しているば、そのパ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の有するブロック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、タブル数、最終ブロック中のタブル数、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、追い出しパ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トを保持しており、これら2つのテーブルを用いて、主記憶
中のパ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、各パケットの作業用ディス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、作業用ディス内のパケットの管理を行っている。

フェイスは、ディスクにおけるそれと同様にしており、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近であり、まだ完全にホストと結合されておらず、今回は、ソフトウェアをソフトウェアで実装を行っている。

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。図12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ツを含むもので、構成は、図15の様になっている。ホストと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おり、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。
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シス템の構成と、処理手段について示した。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法について述べた。

本論の中でも述べたように、試作シスシュムは、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

(4) 喜連川,他。「バイブ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) 楼,他。「ブロック分割記憶管理法によるバイブラインマージソータ」，情報処理学会第31回全国大会，18-9, (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，18-6, (1985)
(7) 金山,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，18-7, (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，18-8, (1985)

（盛文社）

—76—