分散仮想共有メモリ環境に於ける並列関係結合演算アルゴリズムの設計と実装

江　　允・福見　幸一*・天野　浩文**・牧之内　顕文***

倉敷芸術科学大学産業科学技術学部

*日本電気株式会社コンピュータソフトウェア事業部

**九州大学大型計算機センター

***九州大学工学部

(1995年9月30日　受理)

概　　要

我々は分散仮想共有メモリ環境上での「並列・分散ハッシュジョインアルゴリズム」という新しいHash-basedジョインアルゴリズムを提案した。この並列・分散ハッシュジョインアルゴリズムはプログラミングをし易く,共有メモリ型マルチプロセッサ計算機の多CPU単一I/Oチャンネルアーキテクチャに由来する制限が緩和でき,主記憶の大きさがバケットの大きさより小さい時に効率がよくなることが分ってきた。本論文はこのアルゴリズムの設計,定式化分析および分散仮想共有メモリ環境上での実装について述べ,単一サイトの共有メモリ型マルチプロセッサ上でのアルゴリズムと比較する結果を示す。

1　はじめに

現在のデータベース応用では,大量のデータが扱われるようになり,高速なデータベース管理システムが要求されている。広く使用されているデータベース管理システムにとしては関係データベースが挙げられる。関係データベースのジョイン結合操作の効率的な処理はデータベース管理システム全体の高速化につながる。結合演算の高速化を図る方法として,結合演算アルゴリズムの並列化が考えられる([DEW92][HON92][LO93])。しかし,シングルディスクのマルチプロセッサ計算機では,ディスクの入出力がボトルネックとなり,あまり良い結果が得られていない([OKA91][IWA92])と指摘されている。

一方,ここ数年のハードウェア技術の進歩により,マルチプロセッサワークステーションが普及し,複数のワークステーションを高速の通信ネットワークで結んだLAN (Local Area Network)やWAN(Wide Area Network)などのネットワーク分散環境も充実してきた。そこで,我々は,ネットワーク分散環境を利用した並列・分散ハッシュジョインアルゴリズム(Parallel and Distributed Hash Join 以下 PDHJ と略す)を提案した。PDHJアルゴリズムではネットワーク上の安価なマルチプロセッサワークステーションを複数の

使用する。その上に分散仮想共有メモリを用いて，並列処理を行なう場合に問題となるディスクの入出力を各サイトに分散させることにより，結合演算の高速化を図ることができる。

本論文ではこのアルゴリズムを理論的に解析し，マルチプロセッサワークステーション複数台をイーサネットで結んだネットワーク分散環境において実装し，その結果についての考察を行う。

2　Hash-based Join アルゴリズム

Hash Join は関係データベースの結合演算アルゴリズムの中の一種である。例えば，2つの関係(R, S)の同じ属性値を持つタプルを結合させる際に，次のような処理を行なう。

1．関係Rの結合属性にハッシュ関数を摘用する。

2．ハッシュ値よりハッシュテーブルを作成する。

3．関係Sの結合属性に同じハッシュ関数を摘用する。

4．ハッシュ値を用いてハッシュテーブルを検索する。

5．属性の一致したタプルから結果リレーションを作成する。

また，一般にリレーショナルデータベースで取り扱うデータ量は主記憶よりも大きいため，各リレーションをバケットと呼ばれるサブリレーションに分割して結合を行う。それは Hash-Partitioned Join 方法といわれる。例えば，Simple Hash-partitioned Join[GER86], GRACE Hash Join[GOO81][KIT83] や Hybrid Hash Join[DEW84]がある。

マルチプロセッサ計算機に於ける Hash-based Join アルゴリズムは[DEW85]，分割フェーズと結合フェーズとの2つのフェーズに分けてデータを処理するので，リレーションをバケットに分ける方法によって，次のように分類される[LU90]。

- 分割した後，結合を行う方式

  全てのタプルを読み込み，結合属性に分割関数を適用する。その値により，主記憶に入るぐらいの大きさのバケットに分割し，ディスクに書き出す。この方式を用いているアルゴリズムとしては，GRACE ハッシュジョインアルゴリズム [GOO81] Hybrid ハッシュジョインアルゴリズム[DEW84]などがある。

- 分割しながら，結合を行う方式

  まだ処理されていない全てのタプルを読み込み，結合属性にハッシュ関数を適用する。現在のバケットに属するタプルはハッシュテーブルに挿入し，属さないタプルは読み捨てるか，ディスクに書き戻し，次のステップで処理される。この方式に属するアルゴリズムとしては，Simple ハッシュジョインアルゴリズム[DEW85]が挙げられる。

- 分割を行わない方式

  主記憶が一杯になるまでタプルを読み込み，ハッシュテーブルを作成する。Hash-based Nested Loop Join アルゴリズム[DEW85]がこの方式に属している。

この3つの方式を比較してみると，リレーションの大きさが主記憶の大きさに比べて小

さい場合，分割を行わない方式が他の２つの方式と比べて優れている。しかし，近年の情報量の増加に伴い，取り扱うリレーションの大きさも増大してきているため，リレーションの大きさが主記憶よりも小さいという仮定は余り現実的であるとは言えない。そこで，逆に，リレーションの大きさが主記憶の大きさに比べて大きい場合について考えると，分割しながら結合を行う方式や分割を行わない方式では，リレーションの大きさが大きくなると，ディスクの入出力が増加して，非常に効率が悪くなる。それに比べて，分割した後結合を行う方式では，リレーションの大きさが増加しても，バケットに分割することにより効率が極端に悪くなるとは考えられない。

従って，分割した後結合を行う方式が他の２つの方式と比べて優れていると言える。また，分割した後結合を行う方式では，各バケットはそれぞれ独立に処理をすることができるので，並列処理に適していると言える。特に，ディスクの入出力を並列に行えるシステムでは，この方式の並列性を更に引き出すことができるので，大変有効な方式であると考えられている。

3　並列・分散ハッシュジョインアルゴリズム

並列・分散ハッシュジョインアルゴリズムはGRACEハッシュジョインアルゴリズムを基にし，分散・並列環境上で動作するHash-based Joinアルゴリズムである。ネットワークコンピューティング資源を有効に利用することにより，並列処理を行なうとき問題となるディスクの入出力を分散させ，結合演算の高速化を図ることが本提案の目標である。

3.1　分散・並列環境

分散・並列環境とは図１のような計算環境であり，この環境を前提として並列・分散ハッシュジョインアルゴリズムの提案を行なう。

即ち，

- 安価な共有メモリ型マルチプロセッサワークステーションを用いる。
- 各ワークステーションは比較的高速の通信ネットワークで接続されている。
- データは分散して格納されている。

3.2　PDHJ アルゴリズムの設計

PDHJアルゴリズムの設計には２つの目的がある。それは，複数台のマルチプロセッサ計算機の分散共有仮想空間を有効に利用することと，複数のマルチプロセッサ計算機のI/Oチャンネルを利用してシングルディスク制限を改善することである。

3.2.1　分散仮想共有空間の利用

従来のデータベース管理システムは，ユーザの要求に対して，データベースのデータをアクセスするため，ディスクとバッファプールの間のデータを頻繁に転送する必要がある。そのため，並列・分散ハッシュジョインアルゴリズムはWAKASHIサーバ[BAI92]が提供した分散仮想共有空間を利用し，二次記憶に置いているデータを直接アクセスすること

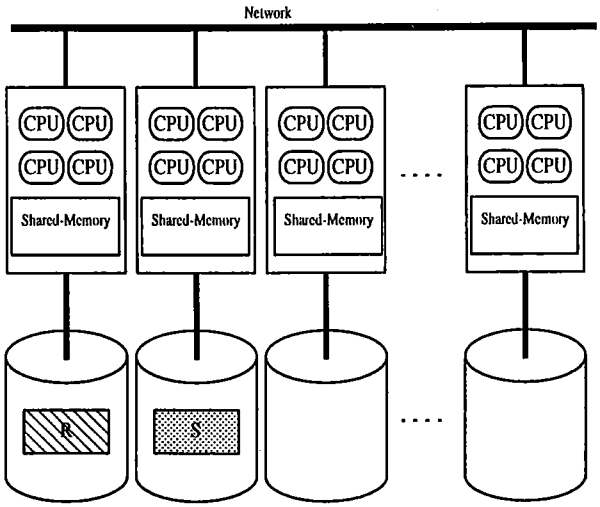

図1　分散・並列環境

ができる。また，複数のサイトに置いているデータを共有して扱うには，仮想共有空間を利用すると，並列プログラムを書くのが便利である。

GRACE ハッシュジョインアルゴリズムでは，分割フェイズにおいて，バケットが実メモリに入りきらないとき，そのバケット内のタプルに対し，別のハッシュ関数を適用してさらに小さなバケットに分割しなければならない。これをバケットのオーバーフローという。また，Hybrid ハッシュジョインアルゴリズムでは，分割フェイズにおいてメモリの空きスペースで最初のバケットのハッシュテーブルを作っているため，メモリの大きさはある程度以上でなければならない。PDHJ は仮想空間を使っているため，上記のようなメモリに関する制約を実質受けない。

3.2.2　シングルディスク制限の改善

多 CPU 単一 I/O チャンネルマチプロセッサ計算機上で並列ハッシュジョインアルゴリズムを実装した結果（[IWA92]）をみると，ディスクの入出力がボトルネックとなり，あまりよい結果が得られてない。具体的には，各プロセッサ上でプログラムを並列に実行させているためプロセッサによる時間は短縮されるが，実装を行なったシステムはシングルディスクであるためディスクの入出力を並列に行なうことができず，また，プロセッサによる時間はディスクの入出力時間と比べて非常に短いためあまり並列処理効果は上がっていないことが分かる。従って，ディスクの入出力時間を減らすことが高速化を行なう上で重要となる。

ディスクの入出力時間をいかに短くするか，というのは非常に重要な問題であるが，分散環境で高速化を考える場合，ネットワークコストも重要な問題である。例えば，ディスクの入出力を減らすために各サイトにデータをばらまきすぎると，今度は逆に，ネットワ

ークを介したデータの転送時間がボトルネックとなる。つまり，ディスクの入出力と転送時間にはトレードオフの関係があるので，どちらをどの程度重視するかが問題である。そこで，極力ネットワークコストを押え，なおかつ，並列に入出力を行なうことでディスクの入出力時間を減らすような方針で並列・分散ハッシュジョインアルゴリズムを開発した。

3.2.3　PDHJ アルゴリズム

PDHJ アルゴリズムは，分割フェイズと結合フェイズの2つのフェイズからなる。分割フェイズでは，リレーションR，Sがある各々のサイトでそれぞれ R_1……R_n，S_1……S_n に分割する。結合フェイズでは，分割したバケットを全サイトに分配し，各サイトで並列にジョインを行う。ここでは，簡単にするために，2つのサイトに置いてあるR，Sを用いて PDHJ のアルゴリズムを具体的に説明する。

分割フェイズ

〔リレーションRのあるサイト〕

〈フェイズ1〉

リレーションRおよびバケット R_1……R_n のデータファイルを仮想共有空間上にマップする。

〈フェイズ2〉

リレーションRを読み込み，結合属性に分割関数とハッシュ関数を適用する。分割値およびハッシュ値により，タプルをバケットに挿入し，さらに，そのバケット内にハッシュテーブルを作成する（図2参照）。

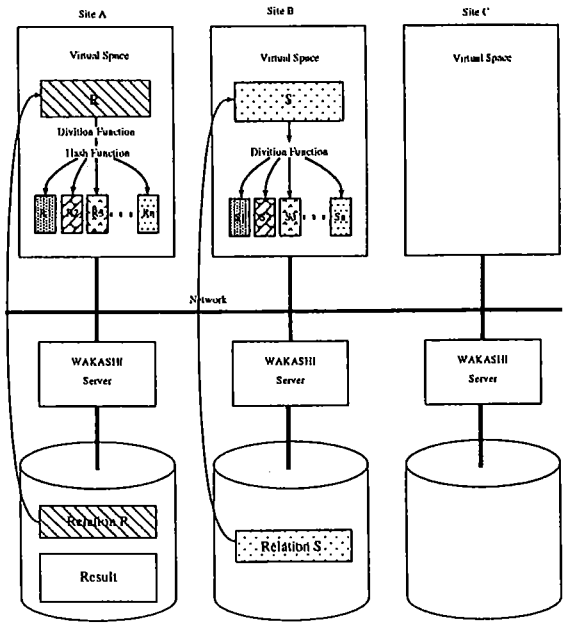

図2　分割フェイズ

〔リレーションSのあるサイト〕

〈フェイズ1〉

リレーションSおよびバケット S_1……S_n のデータファイルを仮想共有空間上にマップする。

〈フェイズ2〉

リレーションSを読み込み，結合属性に分割関数を適用する。効率をあげるために，ハッシュテーブルを作成せず，分割値によって，タプルをバケットに挿入する（図2参照)。

結合フェイズ（全サイトで行なう）

〈フェイズ1〉

自分のサイトに割り当てられたバケット R_i と S_i のデータファイル，結果リレーションファイルを仮想共有空間上にマップする。

〈フェイズ2〉

バケット S_i の結合属性にハッシュ関数を適用し，分割フェイズで作成したバケット R_i 内のハッシュテーブルとマッチングを行う。属性の一致したタプルは結果リレーションファイルに挿入する（図3参照)。

4　並列・分散ハッシュジョインアルゴリズムの解析

PDHJ アルゴリズムの解析には，ウィスコンシンベンチマーク[DEW91]で用いられてい

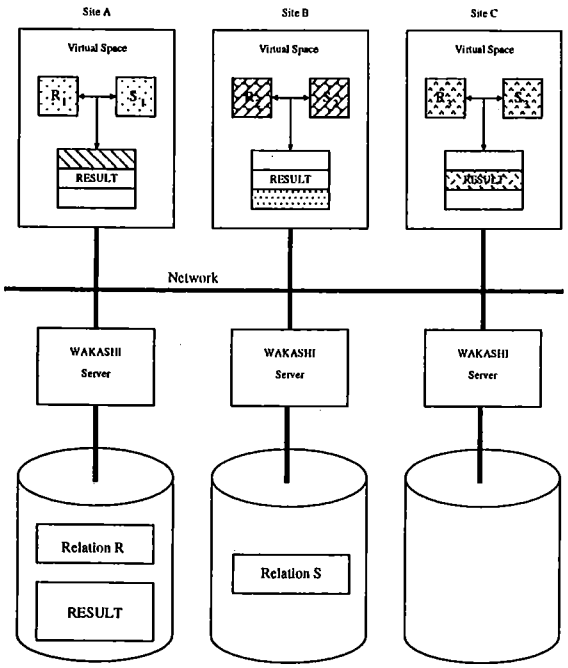

図3　結合フェイズ

るリレーションを使用する。ベンチマークで用いているタプルは，13個の整数（各4バイト）と3個の文字列（各52バイト）で構成されており，1タプルあたり208バイトとなる。

4.1　アルゴリズムの定式化

アルゴリズムを定式化する際，パラメータとして表1を使用する。

●分割フェイズ

分割フェイズではリレーションRとSをそれぞれ別々のサイトで同時に処理するため，2つのサイトでの分割処理が完全に終了するまで，結合フェイズへは移れない。このことは逆に考えると，処理時間の遅いサイトの CPU 時間及びディスクの入出力時間が分割フェイズにおける CPU 時間及びディスクの入出力時間となる。従って，次のようにリレーションRがあるサイトとSがあるサイトに分けて，CPU 時間とディスクの入出力時間を考える。

◎CPU 時間（リレーションRのあるサイト）

マルチプロセッサ・サイトであるため，リレーションRの分割フェイズにおける CPU の

Symbol	Meaning
$\|R\|$	The size of relation R (pages)
$\|S\|$	The size of relation S (pages)
$\|Res\|$	The size of the result relation (pages)
$\|\|R\|\|$	The number of tuples in relation R
$\|\|S\|\|$	The number of tuples in relation S
$\|\|Res\|\|$	The number of tuples of result relation
D_{seq}	I/O time of disk sequential access ($15msec/page$, $8kb/page$)
D_{rand}	I/O time of disk random access ($25msec/page$)
N	The data transmission time in network ($40msec/page$)
t_{split}	The calculation time of division value ($20\mu sec$)
t_{hash}	The calculation time of hash value ($20\mu sec$)
t_{bucket}	The time to insert a tuple of R into a bucket ($50\mu sec$)
t_{match}	The time on matching operation ($70\mu sec$)
t_{build}	The time to build a result tuple ($100\mu sec$)
$\|M\|$	The size of available main memory (page)
b	The number of buckets
p	The number of processors of all
s	The number of sites of all
β	The failure probability

表1　パラメータ

処理時間は以下のようになる。

$$\frac{s \times \|R\|}{p} \times (t_{split} + t_{hash} + t_{bucket})$$

◎ディスクの入出力時間（リレーションRのあるサイト）

リレーションRのタプルをディスクから順々に呼び出して，ランダムにディスクに書き込むので，分割フェイズのディスクの入出力時間は以下のようになる。

$$|R| \times (D_{seq} + D_{rand})$$

◎CPU 時間（リレーションSのあるサイト）

リレーションSのバケットの中にハッシュテーブルを作成するため，分割フェイズにおける CPU の処理時間は以下のようになる。

$$\frac{s \times \|S\|}{p} \times (t_{split} + t_{bucket})$$

◎ディスクの入出力時間（リレーションSのあるサイト）

リレーションSのタプルをディスクから順々に呼び出して，ランダムにディスクに書き込むので，分割フェイズのディスクの入出力時間は以下のようになる。

$$|S| \times (D_{seq} + D_{rand})$$

◎ネットワーク上でのデータの転送時間

分割フェイズではネットワークを介したデータ転送は行わないので，

$$T_{Network} = 0$$

となる。

◎分割フェイズにおける全処理時間

$$T_{partition} = T_{CPU} + T_{Disk} + T_{Network}$$

$$= \begin{cases} \dfrac{s \times \|R\|}{p} \times (t_{split} + t_{hash} + t_{bucket}) \\ + |R| \times (D_{seq} + D_{rand}) \quad (1) \\ \dfrac{s \times \|S\|}{p} \times (t_{split} + t_{bucket}) \\ + |S| \times (D_{seq} + D_{rand}) \quad (2) \end{cases}$$

式(1)はRのあるサイトの方が遅くなる場合を，式(2)はSのあるサイトの方が遅くなる場合を指す。

●結合フェイズ

並列・分散ハッシュジョインアルゴリズムでは，異なるサイトのマルチプロセッサ上でジョイン操作を並列的に行なうので，ジョインフェイズの操作時間は，CPU 時間，ディスクの入出力時間とネットワーク上でのデータの転送時間からなる。

◎CPU 時間

s 個のサイトで，総計 p 個のプロセッサが動いているので，1サイトあたり p/s 個のプロセッサが動いていることになる。従って，1つのバケットを処理するための CPU の処理時間は

$$\frac{\|S\|}{b}\times\frac{s}{p}\times(t_{hash}+t_{match})+\frac{\|Res\|}{b}\times\frac{s}{p}\times t_{build}$$

となり，1サイトあたり b/s のバケットを処理するので，結合フェイズにおける CPU の処理時間は，

$$T_{CPU}=\frac{\|S\|}{p}\times(t_{hash}+t_{match})+\frac{\|Res\|}{p}\times t_{build}$$

となる。

◎ディスクの入出力時間

サイトはバケット S_i(i =1, 2, ……)を順次に，しかも一度に一つしか読み込まられないので，バケット S_iを読み込む時間は1バケットあたり，

$$\frac{|S|}{b}\times\frac{b}{s}\times D_{seq}=\frac{|S|}{s}\times D_{seq}$$

となる。バケット R_i(i =1, 2, ……) を一度に一つしか読み込まられないが，R_iのサイズがメインメモリのサイズより大きい時と小さい時では処理する時間が違う。バケット R_iの読み込み時間は1バケットあたり，

1．$\frac{|R|}{b}\leq|M|$

$$\frac{|R|}{s}\times D_{seq}$$

2．$\frac{|R|}{b}>|M|$

$$\frac{b}{s}\left(\left(|M|\times\left\lfloor\frac{|R|}{b\times|M|}\right\rfloor\times D_{seq}\right)+\left(\beta\times\left(\frac{\|S\|}{b}-\left\lfloor\frac{|R|}{b\times|M|}\right\rfloor\right)\times D_{seq}\right)\right)$$

$$=\frac{|R|}{s}\times D_{seq}+\left(\beta\times\left(\frac{\|S\|}{s}-\left\lfloor\frac{|R|}{s\times|M|}\right\rfloor\right)\times D_{seq}\right)$$

結果リレーションをディスクに書き込む時間は，

$$\frac{|Res|}{s}\times D_{seq}$$

となる。

よって，結合フェイズにおけるディスクの入出力時間は以下のようである。

$$T_{Disk}=(|S|+|Res|)\times\frac{D_{seq}}{s}$$
$$+\begin{cases}\dfrac{|R|}{s}\times D_{sep} & (\dfrac{|R|}{b}\leq|M|)\\ (\dfrac{|R|}{s}\times D_{seq})+(\beta\times(\dfrac{\|S\|}{s}-\left\lfloor\dfrac{|R|}{s\times|M|}\right\rfloor)\times D_{seq}) & (\dfrac{|R|}{b}>|M|)\end{cases}$$

βは R_iがメインメモリの中に存在していない失敗率であり，0から1までの値をとる。一番悪い場合は $\beta=1$ となる。その時には，$\frac{\|S\|}{s}-\left\lfloor\frac{|R|}{b\times|M|}\right\rfloor$ページをディスクに読み込む必要がある。

◎ネットワーク上でのデータの転送時間

バケット R_iと S_iを他サイトに転送しなければならないので，その時間は，

$$T_{Network}=(|R|+|S|)\times(1-\frac{1}{s})\times N$$

となる。

◎結合フェイズにおける全処理時間

$$T_{join}=T_{CPU}+T_{Network}+T_{Disk}$$
$$=\frac{\|S\|}{p}\times(t_{hash}+t_{match})+\frac{\|Res\|}{p}\times t_{build}$$
$$+(|R|+|S|)\times(1-\frac{1}{s})\times N+(|S|+|Res|)\times\frac{D_{seq}}{s}$$
$$+\begin{cases}\dfrac{|R|}{s}\times D_{sep} & (\dfrac{|R|}{b}\leq|M|)\\ (\dfrac{|R|}{s}\times D_{seq})+(\beta\times(\dfrac{\|S\|}{s}-\left\lfloor\dfrac{|R|}{s\times|M|}\right\rfloor)\times D_{seq}) & (\dfrac{|R|}{b}>|M|)\end{cases}$$

5　理論式による評価

PDHJ アルゴリズムの効果を考察するために，定式化した理論式を用いて2つのサイト

のマルチプロセッサを用いる全実行時間を計算し，単一サイトのマルチプロセッサを用いるアルゴリズムの理論式[IWA92]による全実行時間との比較検討を行った。この比較は，プロセッサ数p＝4，サイト数s＝2とs＝1の場合，バケット数b＝10，β=0.4とする。また，バケットR_iを読み込めるメモリの大きさ$|M|$は20ページとする。タプルの個数を5000，7500，10000，12500，と変化させた場合の2つのアルゴリズムの理論的実行時間を図4に示す。

図4から明らかなように，バケットのサイズはメモリの大きさ$|M|$より小さい場合，2つのサイトを用いる並列処理の効果がPDHJと単一サイトとほぼ同じである。それは，各フェイズにおいては，CPUによる処理時間がディスクの入出力時間やネットワークを介したデータの転送時間に比べ非常に短いため，単一サイトと同様にCPU処理の並列化による実行時間の短縮はあまり期待できない。バケットのサイズがメモリの大きさ$|M|$より大きい時に，PDHJの並列効果が単一サイト上の効果より随分良い。それはデータが多くなると，データの転送量も増加するので，単一サイトのディスクの入出力がボトルネックとなり，あまり良い成果が得られていないことである。

サイト間のデータの転送量では，単一サイトでは，リレーションSを全部リレーションRのあるサイトに送って結合演算を行なうことに対し，複数のサイトを同時に結合演算を行なうため，サイト間のデータの転送量を減らした。例えば，2つのサイトの場合を考えると，Rの半分をSのあるサイトへ，Sの半分をRのあるサイトへそれぞれ送って結合演算を行なっている。ディスクの入出力では，PDHJは分割フェイズにおいてリレーション

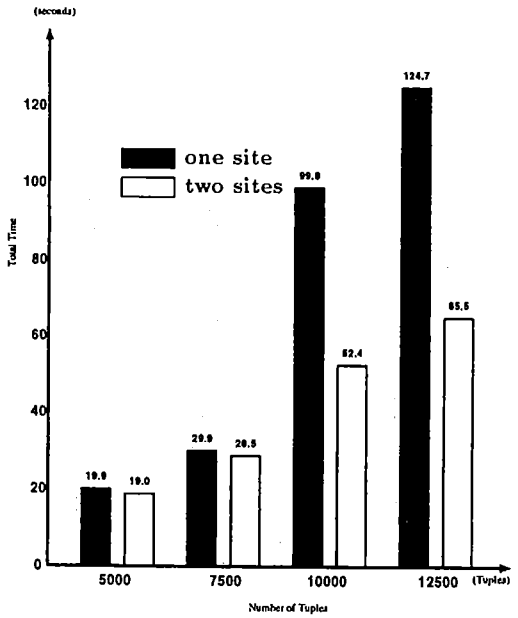

図4　理論式による全実行時間（4threads）

を並列に分割しているため，ディスクの入出力の時間が単一サイトと比べて半分ですんでいる。結合フェイズにおいても同様にバケット R_i を読み込むためのディスクの入出力時間が半分になっている。

6　分散・並列環境上での実装と評価

以上で述べた並列・分散ハッシュジョインアルゴリズムをCプログラムを記述して，実際の分散・並列環境上で実装した。本節では，まず，分散・並列環境上での実装について説明して，次に，単一サイトと2サイト上のPDHJアルゴリズムの測定結果を比較する。

本研究では，表2に示すようなOMRON社製の共有メモリ型マルチプロセッサワークステーションLUNA88kおよびLUNA88k2を使用した。LUNA88kはRISCチップ(MC88100,

表2　実際の並列・分散環境

マシン	CPU	メモリ	ハードディスク
LUNA88k	MC88100 (4CPU)	32MB	1台
LUNA88k2	MC88100 (3CPU)	16MB	1台

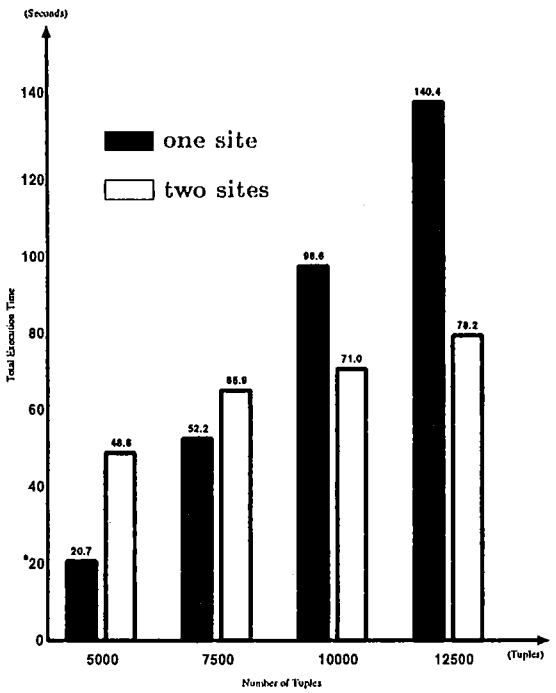

図5　全実行時間（4threads）

25MHz)を4個，32MBの共有メモリ，二次記憶装置としてハードディスクを1台搭載している。LUNA88k2はRISCチップ(MC88100，33MHz)を3個，16MBの共有メモリ，二次記憶装置としてのハードディスクを1台搭載している。OSにはUNIX4.3BSD上位互換で分散OSであるMach2.5[WAL88]を用いた。表2のような2台のマルチプロセッサワークステーションをイーサネットで接続することで分散環境を実現した。また，この実現は分散環境において共有仮想空間をサポートさせるために，分散共有メモリシステムサーバWAKASHIを利用した。

6.1　実測値による評価

Cで記述してPDHJアルゴリズムをそれぞれ単一サイトと2サイトの並列環境で実装し，理論式と同じパラメータ(プロセッサ数p＝4，サイト数s＝2とs＝1，バケット数b＝10)を用いて実際の実行時間を測定した。リレーションR，Sを5000，7500，10000，12500と変化させた時の全処理時間を図5に示す。

グラフから明らかなように，実装した結果と理論式による分析した結果が一致している。5000，7500タプルでは単一サイトの場合と比べて少し遅くなっているのが，5000，7500タプルではバケットR_iが全部メモリにのるため，結合フェイズにおけるディスクの入出力時間差があまり出ず，逆に，実際のサイト間の同期をとるための通信などによるオーバーヘッドが大きくなっていることが原因である。しかし，10000タプルを越えると理論式の場合と同様に多サイトの方が速くなっており，結合フェイズにおける入出力の時間差が出ることがわかる。

7　おわりに

本論文は提案している並列・分散ハッシュジョインアルゴリズムの設計，アルゴリズムの定式化分析を述べた。理論式による結果と分散仮想共有メモリ環境上での実装した結果との比較検討を行なった。その結果，バケットの大きさが実メモリの大きさより大きく，リレーションが大きくなる時に，ネットワーク上での分散仮想共有メモリ環境に於けるPDHJアルゴリズムが高速であることがわかった。

参考文献

[BAI92]　G. Bai and A. Makinouchi：“Implementation and Evaluation of New Approach to Storge Management for Persisted Data — Towards Virtual-Memory Database —”, *Proc. of the 2nd Far-East Workshop on Future Database Systems,* pp. 211—220, Apr. 1992.

[DEW84]　D. J. Dewitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker and D. Wood：“Implementation techniques for main memory database systems,” *Proc. of SIGMOD'84,* pp. 1－8，1984.

[DEW85]　D. J. Dewitt and R. Geber：“Multiprocessor Hash-based Join Algorithms,”*Proc. of VLDB'85,* pp. 151—164, 1985.

[DEW91]　D. J. Dewitt：“The Wisconsin Bahmarking : Past, Present and Future”, The Benchmark

Handbook, 1991.

[DEW92] D. J. Dewitt and J. Gray : "Parallel Database Systems : The Future of High Performance Database Systems,"*Commu. of ACM,* Vol. 35, Mo. 6, pp. 85—98, Jun. 1992.

[GER86] R. J. Gerber : "Dataflow query processing using multiprocessor hash-partitioned algorithms," Computer Sciences Tech. Rep. No. 672, Computer Sciences Dept., Univ. of Wisconsin, 1986.

[GOO81] J. R. Goodman : "An Investigation of Multiprocessor Structures and Algorithms for Database Management,"Univ. of California at Berkeley, Technical Report, UCB/ERL, M81/33, May, 1981.

[HON92] W. Hong : "Exploiting Inter-Operator Parallelism in XPRS,"*Proc. of ACM SIGMOD'92,* pp. 19—28, Jun. 1992.

[IWA92] K. Iwase and A. Makinouch : "An Effcient Hash-Based Join Algorithm for Virtual Space Databases," *Technical Report CSCE-92-C03,* Mar. 1992.

[KIT83] M. Kitsuregawa, H. tanaka and T. Moto-oka : "Application of Hash to Data Base Machine and its Architecture", New Generation Computing, Vol. 1, No. 1, 1983.

[LO93] Ming-Ling Lo, Ming-Syan Chen, C. V. Ravishankar and Philip S. Yu : "On Optimal Processor Allocation to Support Pipelined Hash Joins", *ACM Proc. of the SIGMOD'93,* pp. 69—78, Jun. 1993.

[LU90] Hongjun Lu, Kian-Lee Tan and Ming-Chien Shan : "Hash-Based Join Algorithms for Multiprocessor Computers with Shared Memory", *Proc. of the 16th VLDB Conference* pp. 198—209, Aug. 1990.

[OKA91] K.Okazaki : "A Review for Parallel Hash-Join on Shared Memory Multiprocessor Computers", *Technical Report CSCE-91-C02,* Feb. 1991.

[WAL88] L. R. Walmer and M. R. Thompson : "A Programmer's Guide to the Mach System Calls", *Department of Computer Science Carnegie Mellon University version of 19 Feb.,* 1988.

Design and Implementation of a Parallel Join Algorithm on a Distributed Virtual Shared Memory Environment

Yun JIANG, Koichi FUKUMI* Hirofumi AMANO** and Akifumi MAKINOUCHI***

Dept. of Computer Science Science and Mathematics,
Kurashiki University of Science and the Arts,
2640 Nishinoura, Tsurajima-cho, Kurashiki-shi, Okayama, 712, Japan
**Computer Software Operations Unit, NEC Corporation,*
1-10, Nisshincho, Fuchu, Tokyo 183, Japan
***Computer Center, Kyushu University,*
****Dept. of Computer Science and Communication Engineering, Kyushu University,*
6-10-1, Hakozaki, Higashi-ku, Fukuoka-shi, 812 Japan

(Received September 30, 1995)

We propose an algorithm called Parallel and Distributed Hash Join algorithm on a distributed virtual shared memory environment. The hash join algorithm can be easy to program and relax the restriction of the multiple-cup-one-disk architecture and get better performance when the size of available main memory is less than the bucket size. In this paper, we describe design of the PDHJ algorithm and compare it with the algorithm of single site of multiprocessor using analytical formulas as well as benchmark tests.