ブロック化転置ファイルを利用したデータウェアハウス向けデータベース管理システムの評価

上田尚純[†] 郡光則[†] 青野正宏[††]
渡辺尚[†††] 水野忠則[†††]

データウェアハウスの処理ではディスク上の膨大なデータの全件レコードの検索や集計を短時間で行うことが求められる．PC サーバや UNIX サーバなど通常のサーバでこの処理を行う際のネックは，CPU での処理よりも，ディスクから主記憶へのデータ転送の所に生じる場合が多い．このネックを大幅に軽減するためにブロック化転置ファイルと呼ぶファイル形式を工夫し，これ利用したデータウェアハウス向けのデータベース管理システムを作成した．単純な転置ファイルではレコードの転置をファイル全体に一括して行うが，ブロック化転置ファイルでは，複数レコードをまとめたブロック単位で転置する．これにより，問合せ処理において，実際に参照されるレコード・フィールドのデータのみの読み出しを，少ないシーク回数でかつ複数のディスクのつねに均等な負荷のもとでの並列動作により，高速に行うことを可能にした．作成システムを使って評価し，通常のサーバでは IO バスのデータ転送能力が処理のネックであること，ブロック化転置ファイルによりこのネックを大幅に緩和できること，およびその結果実用的な処理性能を持つデータウェアハウスを実現可能なことを確かめた．

Evaluation of Database System for Datawarehouse Using Blocked Transposed File

Takasumi Ueda,[†] Mitsunori Kori,[†] Masahiro Aono,[††]
Takashi Watanabe[†††] and Tadanori Mizuno[†††]

To process queries for datawarehouse, it is required to read all records of huge amount of data stored on the disk in short time. A bottleneck of the process on commonly used servers, such as PC servers or UNIX servers, occurs more often in data transferring from the disk to the main memory rather than CPU processing. In order to solve the bottleneck, we device the Blocked Transposed File and develop the datawarehouse system by using it. The simple transposed file transposes all records in a file at once, but the Blocked Transposed File divides the records of a file into groups then transposes records for each group of records. This idea makes it possible to read only referenced record fields from multple disks under the strictly equal load for any queries. We evaluated the developed system and revealed that the bottleneck of datawarehouse processing on servers happens in IO bus, the Blocked Transposed File can reduce the bottleneck and contributes to realize datawarehouse processing with practical performance on commonly used servers.

1. はじめに

計算機の処理能力やディスク容量の増大により，伝票 1 枚 1 枚といった詳細なデータを蓄積し，あとでそれをさまざまな角度から解析して有効な結果を得ようというデータウェアハウスの利用がさかんになってきている．たとえば，クレジット会社がクレジットカードでの売上詳細の全データをデータウェアハウスに記録し，利用者の購買状況を解析するなどである．

データウェアハウスでは膨大なデータの処理が必要なため，それ専用に設計された計算機の利用も多いが，最近は UNIX サーバや PC サーバといった通常のサーバの利用も増えてきている．データウェアハウスでは全件レコードの検索や集計が必要となる問合せが多く，データベースのデータ量も一般的に多いので，サーバが通常想定している以上の量の入出力データ転送が発

† 三菱電機株式会社
Mitsubishi Electric Corporation
†† 東京工業高等専門学校
Tokyo National College of Technology
††† 静岡大学
Shizuoka University

生する．このため，データウェアハウス処理ではディスクデータを主記憶に転送する過程で処理のネックが発生する場合が多い．これを解決して，通常のサーバで効率良くデータウェアハウスを処理できればユーザの恩恵は大きい．

大量のデータ処理を高速で行う専用ハードウェアの開発が昔から多くなされてきており，最近ではサーバを複数台つないで並列処理させる方法も利用されている．また，ディスクにメモリや処理機能を持たせる研究も最近注目を浴びてきている[2),3)]．さらに，ソフトウェア面では，索引やデータの並べ方工夫して高速化する研究もよく行われてきている．

我々はビジネス分野を対象に，PC サーバでデータウェアハウスを効率良く処理するデータベース管理システムを実現した．このシステムで，サーバ内でのデータ転送ネックを大幅に軽減するためにブロック化転置ファイルと呼ぶファイル形式（Blocked Trasposed File）を工夫して用いた．よく知られた転置ファイル（Trasposed File）ではレコードの転置をファイル全体に対して一括して行うが，ブロック化転置ファイルでは，複数レコードをまとめたグループ単位で転置する．これにより，ほとんどあらゆる問合せパターンに対して，実際に参照されるレコード・フィールドのデータのみの読み出しを，複数のディスクをつねに均等な負荷のもとで並列動作させて高速に行うことを可能にした．作成システムを使って評価し，通常のサーバでは IO バスのデータ転送能力が処理のネックであること，ブロック化転置ファイルによりこのネックを大幅に緩和できることおよびその結果実用的な処理性能を持つデータウェアハウスを実現可能なことを確かめたので報告する．

なお，データウェアハウスではデータ解析のためのロールアップやドリルダウン機能，データマイニング機能などの上位層機能の提供も大事であるが，本論文ではこれら上位層の機能は述べていない．

2. データウェアハウス構築システム

2.1 データウェアハウス処理の特徴

データウェアハウスでのデータ蓄積はリレーショナルデータベースを利用するのが基本であり，複数のフィールド（カラム）からなるレコードを単位として，時系列でデータを蓄積していく．ユーザがデータ解析を行うツールには，SQL をベースにしたものが多く利用される．

データウェアハウス処理の特性は次のとおりである．

- 運用時は，データは読み出し（read）が中心である．データの書き込みや更新は運用時での発生比率は小さく，運用時間外の夜間などに行う場合が多い．
- 問合せは非定型的なものが多く，全件レコード検索が処理の中心となる．索引作成などの高速化手法は，想定の範囲外の問合せに対しては効果が出ない場合も多い．
- データは伝票など詳細レコードの場合が多く，フィールド数が多くレコードは長い．ただし，問合せで実際に参照されるフィールドは一部だけの場合が多い．

2.2 全件レコード検索の高速処理の方策

問合せ処理の中心となる全件レコード検索では，サーバ内でレコードの転送と処理に関わるディスク，ディスク制御装置，IO バス，主記憶，CPU を最大限並行に動作させ，レコード群をよどみのないパイプラインで処理することが高速処理にはまず必要である．ところで，サーバの各装置には性能差があり，速度の遅い装置で性能がおさえられる．IO 装置や IO バスが遅いと主記憶や CPU などにアイドル状態が発生する（**図 1** (a)）．処理のネックとなる装置を高速化し，各装置を同じ速度で動作させることでパイプラインがよどみなく流れ，処理性能は向上する（図 1 (b)）．

サーバの装置では 1 台あたりではディスクが一番遅いが，ディスクは増設が容易であり，複数台接続することでデータ転送速度を図れる．この際，ディスクアームの移動の最小化とともに，各ディスクの負荷の均等化が大事である．ディスクの次に遅いのは IO バスである．IO バスの増設は容易でなく，対策としてデータ転送量を減らす工夫が必要となる．問合せで実際に参照されるのはレコードの一部のフィールドのみの場合が多く，そこだけをディスクから読み出せば転送データ量を減らせる．

次に，全件レコード処理の方法として，レコード単

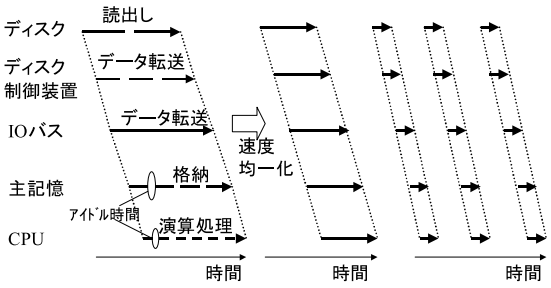


図 1 サーバ内でのデータウェアハウスの並行処理
Fig. 1 Datawarehouse parallel processing in server.

位（行方向）に処理する方法とレコードのカラム単位（列方向）で処理する方法とがある．前者は自明なので，後者を説明する．例として `select a from DB1 where b>15 and c=1` という `select` 文をとる．全レコードに対してまず `b>15` の判定のみを行い，次いで `c=1` の判定のみを全レコードに行い，最後に全レコードの `a` を読むのが，列方向の処理方法である．これの利点は，`b>15` の条件を満たしたレコードに対してのみ `c=1` の判定を行い，同様に `c=1` の条件を満たしたレコードの `a` だけを処理するといった最適化処理が利用できることである．

データウェアハウスでは列方向のパイプライン処理は適さないと判断し，我々は行方向のパイプライン処理方式を採用した．これは次の理由による．

① データウェアハウスでは非定型的な問合せが多く，どの条件判定から行うと最適化が最もよく効くか事前の予測が困難な場合が多い．

② データウェアハウスに限らず，最近はブロック単位でレコードやフィールドを大量にまとめて読み込む．1つでも必要なレコード（フィールド）を保持するブロックは読み込む必要があり，結局ほとんどすべてのブロックを読み込むことになる場合が多い．

③ 仮にいくつかのブロックの読み込みを不要化できても，パイプライン処理時はパイプが途切れる結果を招いてむしろ性能が低下する危険性も生じる．

④ CPUでの処理が重い条件判定の場合，CPUネックとなりディスクがアイドル状態になる場合が生じうる．行方向処理では複数の条件判定をまとめて行うため，条件判定処理の重いものと軽いものが混在することになり，CPUネックの発生確率は列方向処理よりも少ない．

① について補足する．先に示した `select` 文の例で示すと，仮に `b` と `c` の各々の条件を満たすレコードが全体の60%および30%あったとすると，`c` → `b` の順に処理した方が `b` → `c` の順に処理するよりも条件判定の総回数は少なくてすみ，一般的には効率が上がる．しかし，`c` のフィールド長が非常に長かったり `c` の条件判定のCPU負荷が大きい場合は，`b` → `c` の順の方が効率が上がる．つまり，事前に各判定条件を満たすレコードの比率が分かっていないと，最適化が不十分なものとなる．② について補足する．実際に読み込む必要のあるフィールドが全体の p%として，1ブロックに n 件のフィールドが詰まっている場合，読まなくてすむブロックの出現確率は $(1-p/100)^n$ となる．$p=1$ でも，$n=1{,}000$ の場合は，読み込みをスキップできるブロックは1万ブロックに1つ以下しかない．メインメモリやディスクの容量が拡大している今日では1ブロックに1,000件程度のレコードやフィールドを詰めるのは普通のケースでしかない．すなわち，条件を満たすレコードやフィールドの比率が少ない場合でも，それらが局所的に偏在しているということがなければ，ほとんどすべてのレコードを読まざるをえない事態になる．

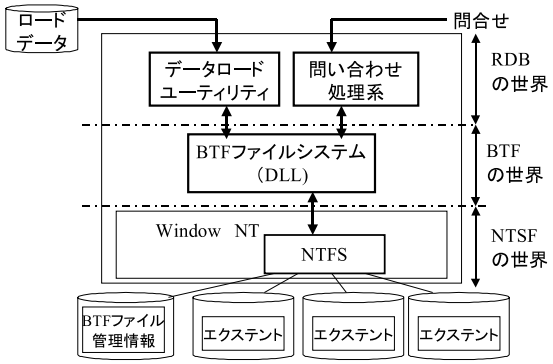


図2 データベース管理システムのソフトウェア構成
Fig. 2 Software structure of database management system.

2.3 作成したデータベース管理システム

PCサーバ上に作成したデータベース管理システムを**図2**に示す．OSはWindows NTを使用し，ブロック化転置ファイル（以下BTFと略す）はWindows NTのNTFSファイルシステム[4),5)]を下位層に利用して実現している．BTFファイルシステムのプログラムはDLL（Dynamic Link Library）として作成し，Win32 APIを介してNTFSファイルシステムの機能を利用している．ソースステップ数はCで約40Kである．

問合せ処理系は既存のSQLベースの処理系を改修して実現している．BTFファイルシステムとのインタフェースを合わせる改修を行っている．

3. ブロック化転置ファイル

3.1 転置ファイルの問題点

実際に参照されるフィールドのみを読むためのファイルとして，よく知られている転置ファイル（以下TFと略す）がある[6)]．実際のデータベースでも利用しているとの報告もある[7)]．TFでは元のファイルのデータを，各レコードの同一論理フィールドのデータをカラム方向に集めたサブファイル（以下SFと呼ぶ）単位に分割してディスクに記録する（**図3**）．1つのファイルは複数のSFに分割される．問合せ処理時に実際

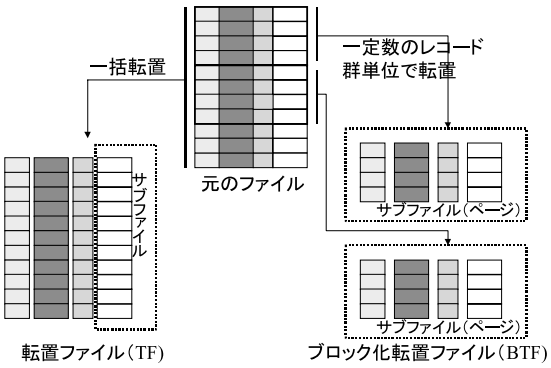


図 3 転置ファイルとブロック化転置ファイル
Fig. 3 Transposed file and blocked transposed file.

に参照されるデータを持つ SF のみを読むことでデータ転送量を削減できる．

しかし，TF の真価は問合せ処理でのレコード全件処理を前述した列方向で行う場合に出せるのであり，行方向処理の場合はその長所を十分には発揮できない．これについては後述する．

3.2 ブロック化転置ファイルの詳細

ブロック化転置ファイル（BTF）は TF の利点を活かしつつ，より効率良く参照フィールドのみをディスクから読めるように工夫したファイルである．TF ではファイルの全レコードに対して一括して転置を行うが，BTF ではファイルを一定数のレコード単位でグループに区切り，このグループ単位で転置を行う．

BTF の詳細を，作成システムに即して述べる．レコードをグループ単位で転置するが，さらにレコードの各フィールド（ユーザが定義したフィールドを論理フィールドと呼ぶ）を固定長の内部フィールドに分割することも行っている．

(1) BTF の作り方

① レコードを内部フィールド（以下 IF と呼ぶ）と呼ぶ固定長サイズ（N バイト）のフィールドに分割する．レコードの論理フィールド（以下 LF と呼ぶ）は通常はバイト単位で任意の境界にあるが，BTF ではまずすべての LF を N バイト境界に合わせる．その後，N バイト単位の IF に分割する．各 LF は 1 個以上の IF に分割される．逆に，1 つの IF には 1 つの LF しか入らない（**図 4** (a)）．

② 1 レコードが k 個の IF（IF_0，IF_1，... とする）に分割されたとする．ブロックと呼ぶ固定長の容器（バッファ）を IF ごとに k 個用意し（BLK_0，BLK_1，... とする），IF_j のデータをレコードの先頭からカラム方向に BLK_j に詰める．ブロックのサイズを B バイトとし，B を N の整数倍にとると，各ブロックには B/N 個の IF が詰まり，B/N 個のレコードを k 個のブロックに詰めて転置処理が終わる．k 個のブロックを BLK_0，BLK_1，... の順に重ねたものをページと呼ぶ（図 4 (b)）．

③ ファイルの全レコードに対して，② の処理を B/N 個のレコードを単位として繰り返す．全レコードを詰め終えると，BTF 化が完了する．

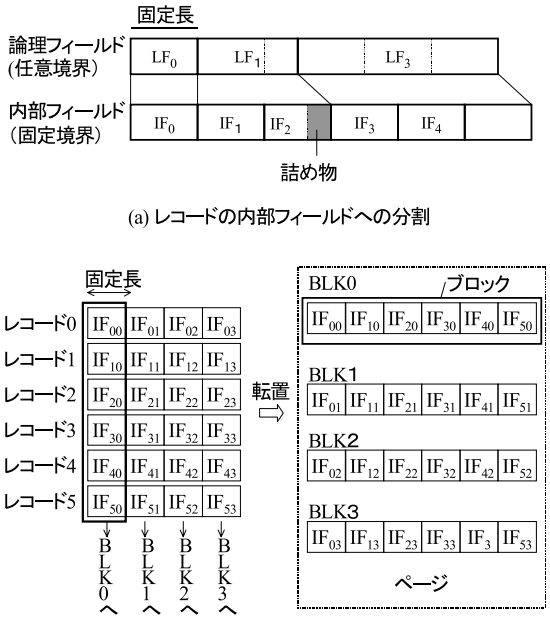


図 4 レコードの転置方法
Fig. 4 Transpose method for record.

上記の BTF 化手順中，フィールド分割を LF 単位でなく固定長の IF 単位で行うのはブロック長およびブロック内の IF 数を同じ値に統一するためである．LF のまま分割すると，レコードの各 LF の長さは一般的に LF ごとに異なり，ブロック長かブロック内の LF 数のいずれかが統一できないことになり，処理ロジックが複雑化して処理速度低下を招く恐れがある．

IF 長の設定について，値が小さいと IF 数が増えすぎて処理オーバヘッドが増大する．反対に，大きすぎると境界合わせの詰め物が増え，内部でのデータ量が増大する．今回は，使用した PC サーバが 32 ビット・マシンであることもあり，4 バイトを採用した．この場合，最悪ケースは 1 バイト長の LF が 4 バイト長の IF となる．手に入るいくつかのデータベースに対して 4 バイト境界に合わせるペナルティ（1 レコード長の増大）を計ってみたが，平均して 10〜20%の範囲であった．この値はデータベースに依存すると思われる

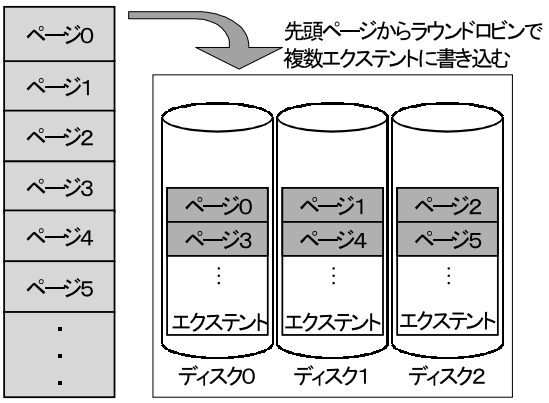


図 5 エクステントとページの配置
Fig. 5 Extent and storing layout of pages.

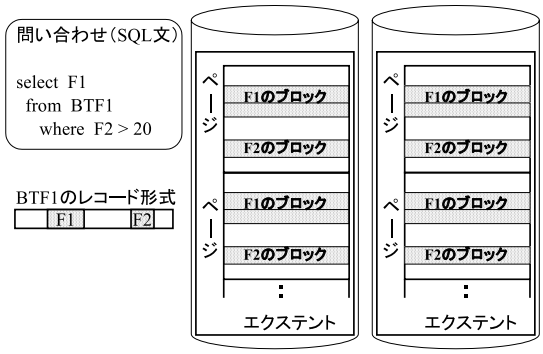


図 6 参照フィールドのブロック読み込み
Fig. 6 Read blocks of referred fields.

が，許容範囲と判断した．なお，今後は PC サーバは 64 ビット化していくので，8 バイトの採用も将来は有力と考えている．

次に，ブロック長は 64 K バイトを採用した．ブロック長が大きいほど読み込み回数が減り，CPU での処理オーバヘッドの減る．反面，メモリ上の入出力バッファも大きくなり，メモリ量の制約から上限が生じる．ここで，バッファに必要なメモリ量を簡単な例で試算する．IF 長が 4 バイト，ブロック長が 64 K バイト時，ブロックには IF が 16 K 個詰まるので，転置は 16 K 個のレコード単位で行われる．あるレコードを BTF 化して 64 個の IF に分割したとする（1 レコードの内部でのレコード長は 256 バイトとなる）．256/4 = 64 個のブロックが作られ，ページ長は 64 K*64 = 4,096 K バイト（4 M バイト）となる．問合せ時にレコードの全フィールドが参照された場合はディスク 1 台あたり 4 M バイト分のバッファが必要となる．ディスクが 16 台の場合は 64 M バイトが必要となる．同様な計算で，内部レコード長が 1 K バイトだと 256 M バイトのバッファが必要となる．実際には全フィールドを参照する問合せは少ないのでバッファはこれより小さくてすむ場合が多いが，参照フィールドが多いとメモリ不足となるので，注意が必要である．

次に，BTF のページのディスク上での配置を説明する．複数のディスクを使用する場合，先頭のページから順にラウンドロビン方式で書き込む（**図 5** にディスク 3 台使用時の例を示す）．各ディスク上に確保した，ページを書き込むための領域をエクステントと呼ぶ．エクステントを物理連続領域にとることで性能を向上できる．1 エクステントは 1 つの NTSF ファイルで実現している．使用するエクステント数は BTF 作成時に必要なだけ指定する．通常は，N 台のディスク使用時は，各ディスクに 1 個ずつの計 N 個のエクステントをとる．

(2) BTF ファイルの読み込み

レコード全件検索時の BTF からのデータ読み出し手順を説明する．

① 問合せ処理システムは問合せ内容（SQL の select 文など）を解析し，読み込みが必要な LF（1 個ないし複数個）を知り，BTF ファイルシステムに読み込みを依頼する．

② BTF ファイルシステムでは，読み込むべき LF に対応する IF（各 LF に対して 1 個ないし複数個ある）を知っており，これら IF を持つブロックのページ内相対位置を計算する．読み出すブロック（1 個ないし複数個）のページ内の相対位置は全ページで同じである（**図 6**）．そして，エクステントが存在する全ディスクにブロック読み込みを指示する．ページ内の複数ブロックを読み出す場合は 1 つの読み出し命令にマージして出す．なお，メモリ上には読み込むデータ量と同じサイズのバッファをディスクごとに用意する．

③ 読み込みを完了したブロックのデータは，ファイル内の順序に関係なく完了したものから問合せ処理システム側に渡す．なお，問合せ処理側は渡されたブロックのレコードを即座に処理する．

3.3 TF と BTF との性能比較

データウェアハウスでは全件レコード検索処理は列方向でなく行方向に処理する方が適していることは前述した．行方向処理を前提に，TF と BTF の性能を比較する．TF では BTF と比較して，ディスク read 回数が多いことおよびディスクアームの移動距離が大きいことのために読み込み速度が遅くなる．

ディスクを複数台使用したとき，BTF ではディスク間の均等な負荷分散がなされる．TF で均等な負荷

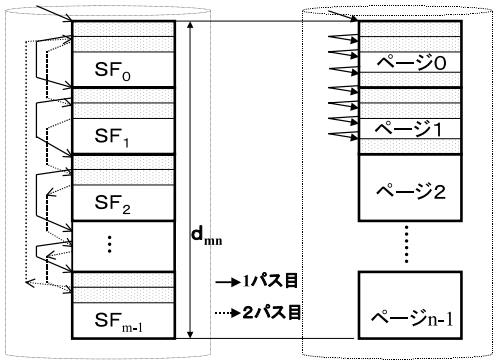


図 7 TF と BTF とのデータ配置の比較
Fig. 7 Comparison of data layout between TF and BTF.

分散を行う方法を考えてみる．TF では各 SF を複数台のディスクをまたいで書き込むストライプ手法を採用することで負荷の均等化が実現できる．ストライプを行わない場合，各 SF は SF 単位でいずれか 1 台のディスクに置かれることになる．ディスク台数と SF 数は一般に異なるので（通常は SF 数は数十以上となり，ディスク台数より多くなる），まず各ディスクに均等に SF を割り振るのが困難である．より問題なのは，問合せで実際に参照される SF は一部の SF だけなので，その SF を保持するディスクだけが作動し，他のディスクは遊ぶことになる．さらには，同一ディスク上の複数 SF を集中的に参照することもありえて，均等な負荷分散にはほど遠いことになる．よって，TF ではストライプ手法によりディスク間の負荷の均等化を図る．すなわち，各 SF において，先頭のブロックをディスク 0 に，次のブロックをディスク 1 に，という順序で置いていく．

TF と BTF での 1 台のディスク上でのデータの配置を**図 7** に示す．いずれもデータ領域を連続物理領域にとり，読み込みが最も速くなるようにしている．また，ブロックを読み込む場合のディスク・アームの動きを示している．ここで，TF では各 SF をストライプ手法により複数ディスクに書き込んでいる．TF も BTF も 1 台のディスクのデータ配置のみを図に示しているが，他のディスクもデータ配置は基本的に同じとなる．データ読み込みも複数台のディスクに同時並行して行われ，その動作も各ディスク間でみな同じなので，1 台のディスクでの TF と BTF の読み込み性能を比較すればよいことになる．このため，説明の都合上，図 7 ではデータベースの全データが 1 台のディスク上にあるようにしている．

条件を合わせるために，1 回で読み込むレコード群のレコード数を TF と BTF とでは同じとする．TF では各論理フィールドの長さが異なるが，各論理フィールドに対して同じレコード件数分を単位にブロッキングすることにする．各 SF のブロック数は同じとなり，ブロック長は SF 間で異なる（ブロック長が一律でないことはデータベースシステムの複雑化を招くので，実際には BTF と同様な内部フィールドの導入が必要と思われるが，ここではこれは考えない）．

図 7 において，レコードの論理フィールド数（SF 数）を m，各 SF のブロック数を n，データがディスク上で占める総トラック数を d_{mn} とする．n は数十以上の数と想定する．ディスク読み込み（read）時の所要時間は，シーク時間 $T_{seek}(d)$，回転待ち時間 t_{rotate} とデータ読み込み時間 $T_{data}(q)$ の合計となる．$T_{seek}(d)$ はディスク・アームの移動トラック数 d の関数で，通常のディスクでは $T_{seek}(d) = t_{const} + T_{move}(d)$ の形式を一般にとる．t_{const} はアームの始動および停止に必要な時間やディスクコントローラのオーバヘッドなどからなり，ディスクに依存した定数値をとる．$T_{move}(d)$ はアームの移動距離 d（トラック数）で定まる値であり，通常は d に線形に比例して $T_{move}(d) = \alpha \cdot d$ となる．ここで，α はディスクで決まる定数である．また，$T_{move}(d1) + T_{move}(d2) = T_{move}(d1 + d2)$ となる．t_{rotate} はディスクが 1 回転する時間の半分の値であり，回転待ち時間の平均値である．$T_{data}(q)$ は読み込むデータ量 q に線形に比例する．

(1) 全フィールド読み込み時の性能比較

レコードの全フィールドを読み込む場合の，TF と BTF の読み込み速度を比較する．

まず TF の読み込み所要時間を考える．m 個ある各 SF の 1 ブロック分を読む（これを 1 パスと呼ぶことにする）には m 回のディスク read が必要である．各 read ごとに，シーク，回転待ち，データ読み込みの時間が発生する．ディスク上の全件レコード処理にはこれを n 回繰り返す．したがって，ディスクには合計 $m \cdot n$ 回の read が発行される．1 パスでのアームの移動距離の合計はほぼ d_{mn} となる（厳密にはこれより小さいが，m と n が数十以上と大きいと d_{mn} と見なしても誤差は小さい）．2 回目以降のパスでは，アーム位置を SF_0 のブロックを読み込むためにほぼ d_{mn} の距離戻す必要があり，アーム移動距離は往路と復路の計で $2d_{mn}$ となる．なお，1 回目のパスにおいても，前回の問合せの処理の完了時にはアームはデータ領域の末尾にきているのでアームをまず d_{mn} 戻す必要があり，1 回目もアームは合計で $2d_{mn}$ 動くと考えても差し支えない．回転待ちについては，各ブ

ロックの read ごとに平均回転待ち時間が必要となる．データ読み込み時間は，1 パスでの読み込みデータ量の合計を q_m とすると，総時間は $nT_{read}(q_m)$ となる．以上より，TF の読み込み所要時間 T_{TF} は式 (1) のようになる．なお，ブロックがトラック境界にまたがる場合はデータ読み込み中にアーム移動が生じ，それにともない回転待ちも生じるが，1 トラック容量がブロックサイズと比較して十分大きい場合はその発生頻度は小さく，またソフト側でブロックがトラックにまたがるような領域割付を避けることも可能なので，このケースは考えないことにする．

$$\begin{aligned} T_{TF} &= nmt_{const} + 2nT_{move}(d_{mn}) \\ &\quad + nmt_{rotate} + nT_{data}(q_m) \qquad (1) \\ &= n(m(t_{const} + t_{rotate}) + 2T_{move}(d_{mn}) \\ &\quad + T_{data}(q_m)) \qquad (1\text{-a}) \end{aligned}$$

次に BTF の読み込み所要時間を考察する．BTF はデータは n ページで格納されており，1 ページを読み込むことが TF の 1 パスに対応する．BTF では論理フィールドを内部フィールドに分割するが，分割された内部フィールドは物理的連続領域に置かれるため，これら内部フィールドの読み込みは 1 つの read にマージされて発行される．このため，read 回数は論理フィールド数で勘定すればよい．さらには，トラックのサイズがブロックサイズより大きい場合は 1 トラックに複数のブロックを置くことができ，複数の論理フィールド分のブロックを 1 回の read で読み込める．1 トラックで平均して k 個の論理フィールドのブロックを格納していると，read 回数は $1/k$ になる．この分，シークおよび回転待ち関連の時間が減少することになる．BTF の n ページ分の読み込みでのアームの移動距離の合計は，図 7 から明らかなようにデータ領域の頭から末尾まで 1 回移動するだけであり，d_{mn} である．前回の問合せ処理で末尾まで移動しているアームを d_{mn} 移動させて先頭に戻す処理が最初に必要なので，これも含めて $2d_{mn}$ 移動する．読み込むデータの総量は，内部フィールドへの分割によるデータ量の増大を無視すると，TF と同じく $n \cdot q_m$ である．よって，BTF の読み込み時間 T_{BTF} は式 (2) のようになる．式 (2) において，アーム移動時間 $2T_{move}(d_{mn})$ は他の項目の値と比較して非常に小さな値となるので無視でき，式 (2-a) となる．なお，論理フィールドを内部フィールドに分割した場合のデータ量増大は考慮していない．これは，TF においても実際に TF に基づくデータベースシステムを作成する場合は内部フィールドの導入が同様に必要になると予想され，BTF に対してのみデータ量増大を含めるのは不公平となるためである．

$$\begin{aligned} T_{BTF} &= n(m/k)t_{const} + 2T_{move}(d_{mn}) \\ &\quad + n(m/k)t_{rotate} + nT_{data}(q_m) \qquad (2) \\ &= n((m/k)(t_{const} + t_{rotate}) \\ &\quad + 2T_{move}(d_{mn})/n + T_{data}(q_m)) \\ &\fallingdotseq n((m/k)(t_{const} + t_{rotate}) + T_{data}(q_m)) \\ &\qquad (2\text{-a}) \end{aligned}$$

TF と BTF の性能比較は式 (1-a) と (2-a) とで行える．データ読み込み時間 $T_{data}(q_m)$ は同じであるが，BTF では TF と比較して read 命令発行回数減少によりシーク・オーバヘッド時間および回転待ち時間が $1/k$ に減少する．この減少分の値は後で例を示すが，高速化への寄与は大となる．また，BTF では TF で必要となるアーム移動時間が無視できるが，全フィールド読み込みの場合は TF においてもこの $T_{move}(d_{mn})$ 値の比重は小さい．

(2) 一部フィールド読み込み時の性能比較

前項ではレコードの全フィールドを読み込む場合で比較したが，実際の問合せ処理では参照されるフィールドは一部のフィールドである場合が多い．この場合の TF と BTF の性能比較を考察する．

実際に参照されるフィールドの数を m'，それに応じて実際に読み込まれる 1 パス（1 ページ）でのデータ量を $q_{m'}$，参照される先頭と末尾のデータ間のトラック数を $d_{m'n}$ とする．また，1 トラック内に存在する参照フィールドの平均数を k' とする．このときの TF および BTF のデータ呼び出しの所要時間を ${T'}_{TF}$，${T'}_{BTF}$ とすると，

$$\begin{aligned} {T'}_{TF} &= nm't_{const} + 2nT_{move}(d_{m'n}) \\ &\quad + nm't_{rotate} + nT_{data}(q_{m'}) \\ &= n(m'(t_{const} + t_{rotate}) + 2T_{move}(d_{m'n}) \\ &\quad + T_{data}(q_{m'})) \qquad (3) \\ {T'}_{BTF} &= n(m'/k')t_{const} + 2T_{move}(d_{mn}) \\ &\quad + n(m'/k')t_{rotate} + nT_{data}(q_{m'}) \\ &= n((m'/k')(t_{const} + t_{rotate}) \\ &\quad + 2T_{move}(d_{mn})/n + T_{data}(q_{m'})) \qquad (4) \\ &\fallingdotseq n((m'/k')(t_{const} + t_{rotate}) \\ &\quad + T_{data}(q_{m'})) \qquad (4\text{-a}) \end{aligned}$$

BTF の式 (4) は，k' 個の参照フィールドがトラック内において隣接している場合の時間である．この場合は 2 番目以降の参照フィールドを回転待ちなしで読み込める．しかし，実際には参照フィールドがトラック内で離散している場合が一般的であり，この場合はトラック内の 2 つ目以降の参照フィールドでは回転待ちが発生する．この回転待ちの平均時間は k' が大きい場合は t_{rotate} 以下の値になると考えられる．なぜ

なら，ディスクでは回転待ち時間が一番少なくてすむブロックから読み込むはずだからである．この場合の回転待ち時間の平均値を t'_{rotate} とすると，これは k'，ブロックサイズ，ディスクのトラックサイズとサーフェス数（物理的な記録面の数）などで決まる値となる．また，この回転待ちを考慮した BTF の読み込み時間 T''_{BTF} は式 (5)，(5-a) のようになる．

$$\begin{aligned} T''_{BTF} &= n(m'/k')t_{const} + 2T_{move}(d_{m'n}) \\ &\quad + n(m/k')t_{rotate} + nT_{data}(q_{m'}) \\ &\quad + n(k'-1)(m'/k')t'_{rotate} \qquad (5) \\ &\fallingdotseq n((m'/k')(t_{const} + t_{rotate}) \\ &\quad + T_{data}(q_{m'}) \\ &\quad + (k'-1)(m'/k')'t'_{rotate}) \qquad (5\text{-a}) \end{aligned}$$

式 (5-a) において，k' が 2 前後以下の小さな値のときは，$t'_{rotate} = t_{rotate}$ と見なせ，これを代入すると，次のようになる．

$$\begin{aligned} &T''_{BTF} \\ &\quad = n((m'/k')t_{const} + m't_{rotate} + T_{data}(q_{m'})) \\ &\qquad\qquad (5\text{-b}) \end{aligned}$$

m' が少なくて，したがって k' が小さいときの TF と BTF の性能比較は，式 (3) と式 (5-b) の比較となる．$k' \fallingdotseq 1$ の場合は，TF と BTF の所要時間の差は式 (3) の項 $2T_{move}(d_{m'n})$ となる．データベースのサイズが小さいと $d_{m'n}$ も小さくなり，$2T_{move}(d_{m'n})$ の値も小さくなって TF と BTF の性能差は 0 に近づく．しかし，データベースが非常に大きい（GB 単位）と $d_{m'n}$ も大きくなり，無視できない値となる．この分，BTF は TF より速くなり，優位性が明確になる．

(3) 性能比較の具体例

上記で導出した計算式を使って，利用環境を想定して TF と BTF の性能比較を行ってみる．

ディスクの例として 4 章の性能評価で使用したディスクをとる．このディスクでは，$t_{const} = 0.8\,\text{ms}$，$\alpha = 0.00065\,\text{ms/トラック}$，$t_{rotate} = 2.99\,\text{ms}$，$T_{data}(33\,\text{MB}) = 1\,\text{s}$，1 トラック $= 609\,\text{KB}$ である．なお，t_{rotate}，t_{data} は各々ディスクでの平均値である．サーフェス（記録面）あたりのトラック数は 14,384 で，$T_{move}(14{,}384) = 9.4\,\text{ms}$ である．この全トラックをアームが移動する時間（フル・ストローク）は 9.4 ms である．また，データベースについて，1 レコード長は 256 B で論理フィールド数 $m = 32$（各論理フィールド長は一律に 8 B とする．こうしても普遍性は失われない），ブロックサイズを 64 KB とする．$m = 32$，$k = 9$ となる．

① 全フィールド読み込み

1 ディスク上のデータ総量が 256 MB の場合をとる．$d_m = 444$ である．これらこの値を式 (1)，(2) に代入すると，

$T_{TF} = 23.36\,\text{s}$（内訳：$nmt_{const} = 3.28\,\text{s}$，$2nT_{move}(d_{mn}) = 0.07\,\text{s}$，$nmt_{rotate} = 12.25\,\text{s}$，$nT_{data}(q_m) = 7.76\,\text{s}$）

$T_{BTF} = 9.48\,\text{s}$（内訳：$n(m/k)t_{const} = 0.36\,\text{s}$，$2T_{move}(d_{mn}) = 0.00057\,\text{s}$，$n(m/k)t_{rotate} = 1.36\,\text{s}$，$nT_{data}(q_m) = 7.76\,\text{s}$）

これより，BTF は TF の約 2.5 倍の速度（時間では 40%の時間）でディスク読み出しが行えることになる．内訳を見ると，TF ではデータ読み出し時間の約 2 倍の時間がシーク動作や回転待ちにかかっていることが分かる．特に，回転待ちの時間が大きい．BTF ではこれらオーバヘッド時間が 1/9 となり，これにより大幅な高速化を達成している．なお，BTF では前述したように $T_{move}(d_{mn})$ の値は他の項と比較して十分小さく，無視できることが分かる．TF においても，全フィールド読み込み時，あるいは読み込むフィールドの割合が高い場合は，$2nT_{move}(d_{mn})$ の値はやはり相対的に小さくて無視できることが分かる．

ディスク上のデータ総量が 256 MB の場合を例で示したが，データ量を変化させた場合，$T_{move}(d_{mn})$ の項の値は相対的に小さいので無視すると，式 (1-a)，(2-a) から分かるように n の値が変わるだけなので，TF と BTF の読み込み速度比も変化しないことになる．

② 一部フィールドの読み込み

参照フィールド数が少ない場合の TF と BTF の性能を比較してみる．ここで TF の読み込み速度には式 (3) を，BTF の読み込み速度には式 (4-b) を使う．$m'/m = a$ とすると，読み込むデータ総量の比の平均値もこれに比例して減るので $Q'/Q = a$ と考えてよい．ただし，$d_{m'n}/d_{mn} = a$ とはならない．極端な場合，レコードの先頭フィールドと末尾フィールドが参照された場合は $d_{m'n} = d_{mn}$ となる．一部フィールド読み込みの場合は，TF では式 (3) の項 $2T_{move}(d_{m'n})$ のアーム移動時間の値が無視できなくなる．データベースのサイズが小さいと $d_{m'n}$ が小さくなり，TF と BTF は性能差はあまりないことになる．逆にデータベースサイズが大きいと，アーム移動時間の占める割合は大きくなり，性能差に影響を及ぼす．なお，$m' = 1$ の場合だけは $d_{m'n}$ は小さな値となることに注意．k' の平均値は a から求まる．1 つのトラックが k 個のブロックを保持でき（ここでの例は $k = 9$），各論理フィールドが参照される確率が a とすると，あるトラックに i 個の参照フィールドのブロックが存在する確率は ${}_k\text{C}_i a^i (1-a)^{k-i}$ となる．1 個も参照フィール

表 1 一部フィールド読み出し時の TF と BTF のデータ読み出し時間の比較
Table 1 Read time comparison between TF and BTF for the case of partial read.

m'		1	2	4	8
k'		1.06	1.28	1.59	2.42
TF	$m'(t_{const}+t_{rotate})$	3.79	7.58	15.16	30.32
	$2T_{move}(d_{m'n})$	-	9.1	9.1	9.1
	$T_{read}(q_{m'})$	1.94	3.98	7.76	15.52
	計(a)	5.73	20.66	32.02	54.94
BTF	$(m'/k')t_{const}+m't_{rotate}$	3.74	7.23	13.97	26.56
	$T_{read}(q_{m'})$	1.94	3.98	7.76	15.52
	計(b)	5.68	11.21	21.73	42.08
BTF/TF (b/a)		0.99	0.54	0.68	0.76

ドのブロックを含まないトラックは何もせずにスキップすればよいので，1 個以上参照フィールド・ブロックを含むトラックにおけるの平均の参照フィールドブロック数を計算すれば k' が求まる．$m'=1,2,4,8$ に対する k' および TF と BTF のデータ読み込み速度を**表 1** に示す．なお，$d_{m'n}=7{,}000$ と大きくして TF でのアーム移動時間が大きくなるようにしている．大きなデータベースに参照率の低い問合せが来た場合，TF ではアームの動作が大きくなって性能が低下するが，BTF ではこの分の性能低下が生じないことが分かる．

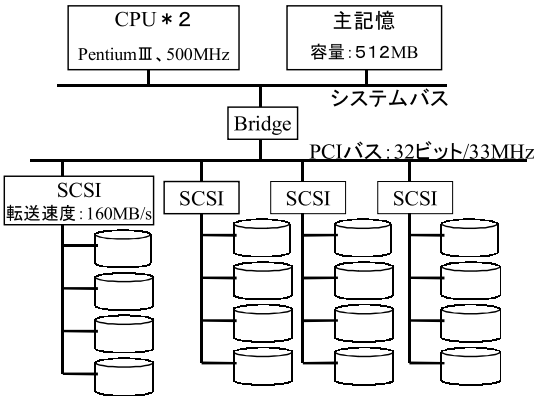
以上のように，データベースを行方向で処理する場合，BTF はつねに TF よりも良い性能を示す．最悪ケースにおいても，BTF は TF よりも性能が低下することはない．

4. 性能評価

データウェアハウス構築システムを PC サーバ上で動作させて評価した．評価の目的は次の 3 つである．

① サーバでのデータウェアハウスの処理ネック個所の確認を行う．
② BTF の性能の測定．ディスクの均等な負荷分散がうまくいっているか，目標としたデータ読み出し速度が実現できているか．
③ 実際のデータベースを利用しての問合せ処理性能の評価．

まず，評価に使用したサーバの構成と諸元を**図 8** に示す（性能はカタログ性能）．最近のディスクはキャッシュ内蔵などで高機能化してきており，性能評価結果がディスクの機種に依存する可能性がある．評価環境を明確にするためディスクの詳しい諸元も示してある．カタログ性能では，1 台の SCSI でディスク 4 台までをほぼフル性能で並行駆動できるので，SCSI へのディスク接続台数は最大 4 台までとした．



(a) システム構成

製造会社、型名： SEAGATE社、 ST39204
容量 : 9.176 GB
回転速度 : 10,033回/分
トラック容量 : 203KB(平均)
トラック数 : 18,145／面＊ 3面
転送速度 : 26-40MB/s(平均 33MB/s)
シーク時間 : Read時 0.8 - 10.2ms(平均 5.4ms)
Write時 1.1 - 11.2ms(平均 6.0ms)

(b)使用したディスクの諸元

図 8 評価に使用したサーバのシステム構成
Fig. 8 Server system configuration used for evaluation.

BTF のエクステント領域がディスク上の連続した物理領域に確保されるかどうか，あるいはディスクの外周と内周のいずれに確保されるかでファイル性能に差が出る．BTF を新たに作成する際は，ディスクを初期化して空の状態にしてからエクステント領域を確保し，ディスク上の状況が同じになるように努めた．

測定にあたってのデータ転送量，CPU 時間，経過時間などの情報は，WindowsNT が提供するログ機能を利用して行った．

4.1 サーバ内のデータ転送速度の測定

まず，サーバにおいてディスク・データを主記憶に転送する際のネック個所を確認するための測定を行った．図 1 に示したように，サーバ内の性能ネック個所がシステム全体の処理性能を制限する．BTF の全フィールドを読み込む重負荷の場合を設定し，ディスク台数を変化させてサーバ内のデータ転送性能を測定した（**図 9**）．ここで，データ転送速度は，読み込んだデータ量を転送に要した経過時間で割って算出している．

1 台あたりではディスクの性能が最も低いため，台数が少ないとディスクがデータ転送上のネックとなり，

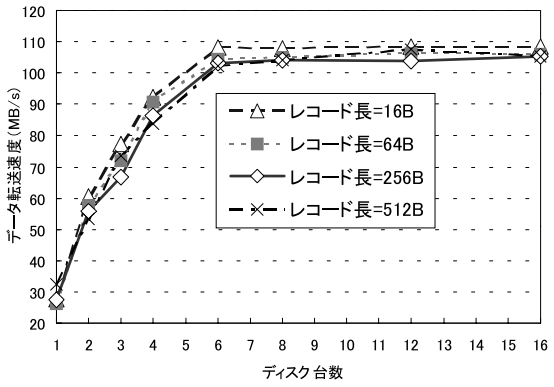


図 9 ディスク台数とデータ転送能力の関係
Fig. 9 Data transfer rate for each number of disks.

台数増加にともない次は PCI バスがネックになると予測できる．図 9 は実態が予測どおりであることを示している．ディスク全体のデータ転送性能は，ディスク 1 台時の 30 MB/s 弱から台数増加にともない向上し，4 台を過ぎたあたりで性能向上の鈍化が目立ち始め，6 台で飽和している．6 台以上ではデータ転送性能は向上していない．これは PCI バスの転送能力が飽和したためと考えるのが妥当である．PCI バスの理論的最大性能は 132 MB/s であるが，実際の使用環境下では複数の IO 装置の制御などのオーバヘッドが発生し，理論値よりも性能が落ちる．測定したサーバでの PCI バスの実効転送能力は最大で 110 MB/s 弱程度となっている．データウェアハウス問合せ処理では，CPU ネックとなる前に IO バスのデータ転送ネックとなる場合が多いが，これを確かめることができた．

ここで，SCSI の使用台数について補足しておく．図 9 の測定では，ディスク台数 1 台，2 台，3 台，4 台，6 台，8 台，12 台，16 台に各々 $s1*d1$，$s1*d2$，$s1*d3$，$s1*d4$，$s2*d3$，$s2*d4$，$s3*d4$，$s4*d4$ の構成を用いた．ここで，SCSI を N 台使用する場合を sN で，SCSI 1 台に接続したディスクを dM で示す．SCSI の台数の差がデータ転送能力に及ぼす影響を調べるため，ディスク台数が同じで SCSI 台数が異なるケース（たとえば $s1*d4$ と $s2*d2$ と $s4*d1$ など）をいくつか測定したが，性能差は 5%程度以内であった．SCSI へのディスク接続台数を 4 台以下にしておけば，今回使用したサーバでは，性能はディスクの台で支配されるといえる．以下の測定では図 9 と同じ SCSI，ディスク構成での測定値を示している．

4.2 ブロック化転置ファイル（BTF）の性能

① BTF の基本性能

最初に参照率を定義する．内部レコード長を r，問合せで実際に参照される IF（1 個ないし複数個）の合計長を f とすると，f/r を参照率と定義する．r が 100 バイトで f が 12 バイトだと，参照率は 0.12（12%）となる．

r が 256 バイト，レコード件数 $2M$ 件（$1M = 1024*1024 = 1{,}048{,}576$）で，総データ量は 512 M バイトのデータを対象に，f を 4 バイト（参照率 1.56%）から 256 バイト（100%）まで変化させた場合を測定した（**図 10**．(a) は参照率が 0%から 100%までを示し，(b) は参照率が 15%までを拡大して示す．(c) は (a) の縦軸をデータ転送速度に変更したものである）．図 10 (a)，(b) の縦軸はファイルの読み込みを開始してから全件レコードの参照 IF を読み終えるまでの経過時間である．なお，参照率が 100%未満の場合，IF のレコード内の存在位置で読み込み性能が異なるが，まずは IF がレコードの先頭から隣接する場合を測定している．読むデータ量が多い場合は，シーク時間は無視でき，読み出し時間はデータ量（すなわち参照率）にほぼ比例している．参照率 100%時はディスク台数が少ない場合に少し性能が良くなっているが，これは読み飛ばしブロックがなくなり CPU 処理やディスク動作が簡単化するためであろう．ディスク台数が増加すると PCI バスネックのためにこの効果は見えなくなっている．参照率が数%以下になるとシーク時間の比重が無視できなくなり比例関係が崩れてくることが，台数の少ないときに見てとれる．

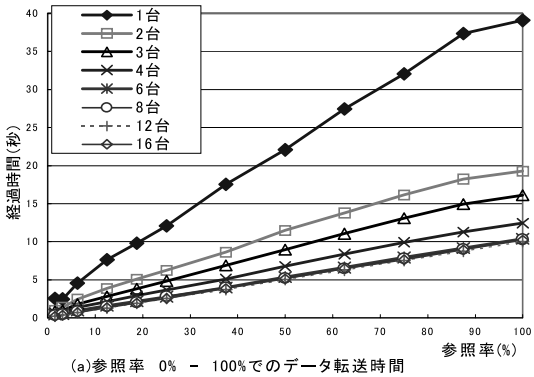
次に，図 10 (a) の縦軸を経過時間からデータ転送速度に変更したグラフを図 10 (c) に示す．データ転送速度は転送データ量を経過時間で割って算出している．図から，参照率が高い場合でもディスク 4 台以下では PCI バスに余裕があり，PCI バスの性能を使い切るにはディスク 6〜8 台以上が必要なことが分かる．参照率が低くなると回転待ち時間やシーク時間の比重が増大してディスクごとの実効的データ転送速度は低下する．12〜16 台のディスクでも台数効果が出ることが分かる．

図 11 はディスク台数効果を示す．各参照率において，ディスク 1 台のときの性能を 1 として，ディスク台数を増やしたときの性能を示している．参照率が低いほどディスクの台数効果が飽和しないことが分かる．また，図に示すいずれのケースにおいても，ディスク間の均等な負荷分散がなされていることを，ディスクごとにとったログ情報で確認している．

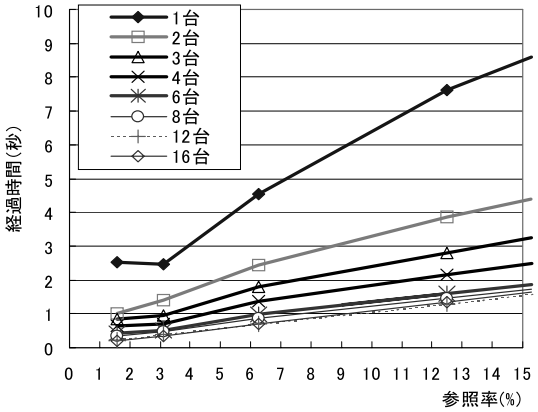
BTF が予想どおりの効果を発揮することを確認できたと考えている．

② 参照フィールドが離散している場合 BTF の性能

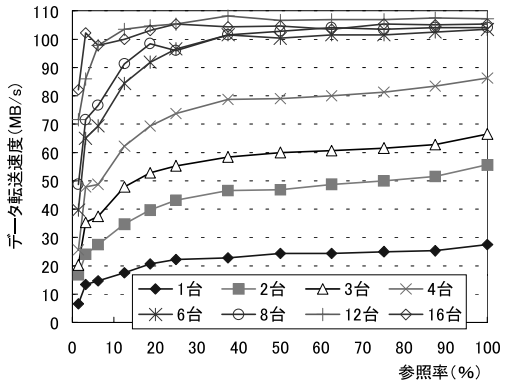
① では参照 IF がレコードの先頭から連続している



(a)参照率 0% － 100%でのデータ転送時間



(b)参照率 0% ー 15%でのデータ転送時間



(c) 参照率とデータ転送速度

図 10 参照率とデータ転送性能
Fig. 10 Refering ratio and data transfer rate.

場合を測定したが，次に参照 IF が離散している場合を示す．参照する IF がレコードの先頭，真中，末尾に位置する場合，ディスク上で対応するブロックに間隔があり，回転待ちや場合によってはディスクアーム移動が生じ，ディスクの実効的な読み込み性能は低下する．フィールドの離散パターンの全ケースの網羅はできないので，**図 12** に示す参照率，離散配置のケース

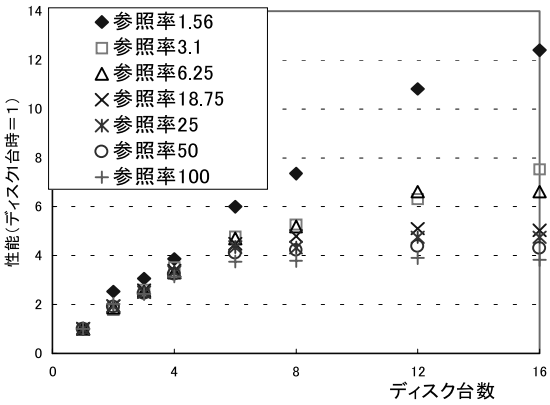


図 11 ディスク台数効果
Fig. 11 Effect of multiple disks.

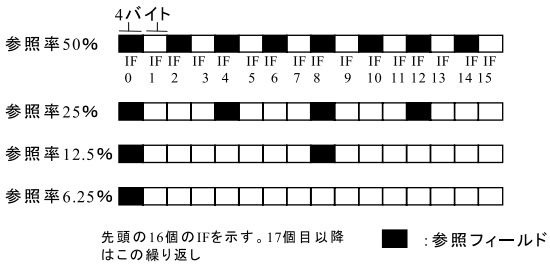
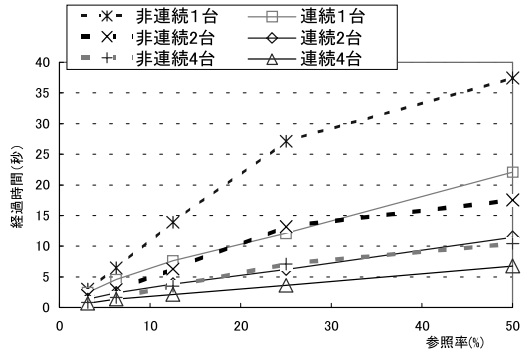


図 12 測定したフィールド離散のケース
Fig. 12 Evaluation cases of non-continuous fields.

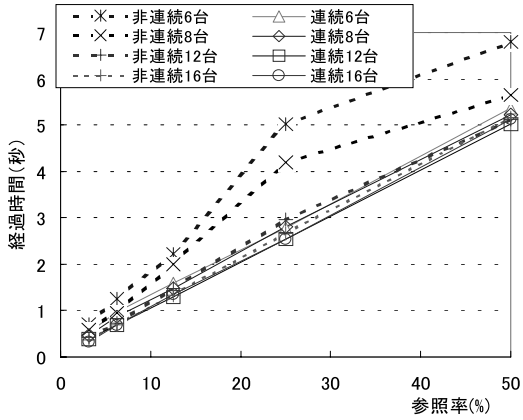
で測定した．結果を**図 13** (a)，(b)，(c) に示す．(a)，(b) は縦軸は経過時間，(c) はデータ転送速度である．参照フィールドが離散しているとディスク 1 台あたりではデータ転送がほぼ半減することが分かる．実際の問合せでは参照フィールドは離散するのが一般的と考えられ，実行性能としてこちらの方を考えておく必要があろう．PCI バス性能を使い切るにはディスク台数は 12 台以上が望ましいことが分かる．なお，(c) を見ると，参照率が 25%前後のところでデータ転送性能が少し落ちている．これは使用したディスクの内部動作に起因するものと推測している．ディスクのカタログに内部に 4 M バイトのキャッシュメモリを持つと記述されている．今回は規則的なパターンで参照 IF を選んだので，ディスクのキャッシュ仕様と相性が悪かった可能性がある．真相解明は今後の課題である．

③ ブロック長の変化

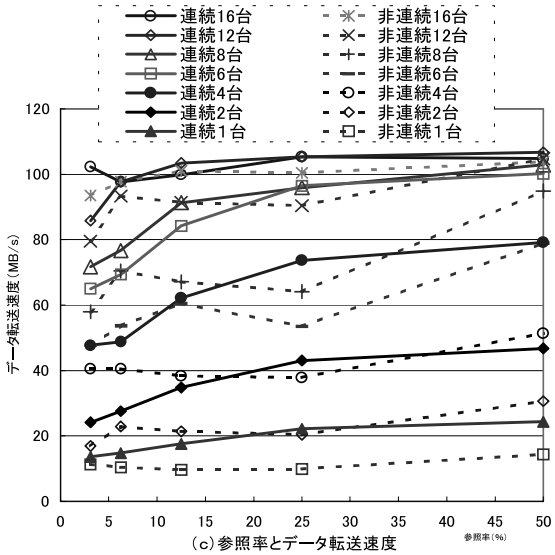
ブロック長の BTF 性能への影響を知るために，ブロック長を 16 KB から 256 KB まで変化させて性能を測定した．台数が 8 台以上では差がほとんどなかったので，ディスク 8 台までを示している（**図 14**）．ディスクが 1 台ではブロック長が大きいほどおおむね性能



(a)参照率とデータ転送時間(ディスク 1/2/4台)

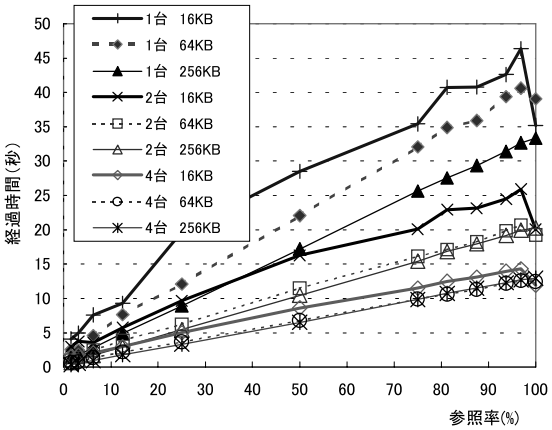


(b)参照率とデータ転送時間(ディスク 6/8/12/16台)

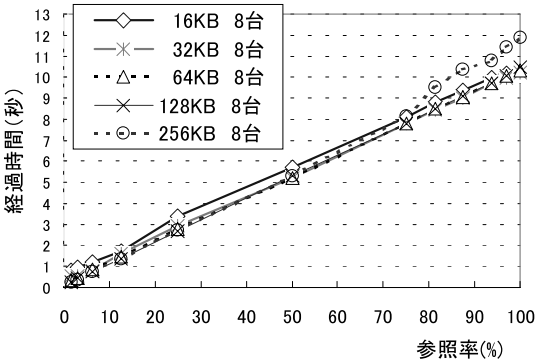


(c)参照率とデータ転送速度

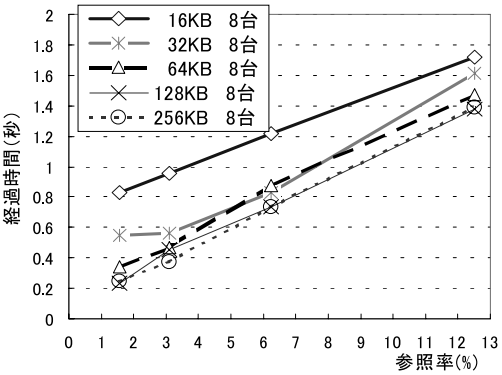
図 13 ブロック連続，非連続の性能比較
Fig. 13 Performance comparison between continuous blocks and non-continuous blocks.



(a)ブロック長とデータ転送時間(ディスク 1/2/4台)



(b)ブロック長とデータ転送時間
(ディスク8台、参照率 0%-100%)



(c)ブロック長とデータ転送時間
(ディスク8台、参照率 0%-13%)

図 14 ブロック長と性能の関係
Fig. 14 Block size and performance.

が向上している．ブロック読み込みごとのオーバヘッドの減少がそのまま表れている．ディスク台数の増加，および参照率の増加にともない，ブロック長での性能差は小さくなっている．これは，参照率が高まるとデータ転送時間の比重が増大してブロックごとの処理オーバヘッドは目立たなくなること，最終的には PCI バスネックの影に隠れるためであろう．なお，ブロック長が 256 KB では，参照率が高くなると性能が低下しているが，これは主記憶のメモリ不足でスラッシングが発生したためである．実測結果から，ブロック長はメモリ不足を引き起こさない範囲で大きいほうが性能が向上すること，ディスク 4 台以上ではブロックサイズが 32 KB 以上あると性能差はわずかであることが分かる．標準として採用している 64 KB のブロック長は妥当な値であったと判断している．

4.3 問合せ処理の性能見積り

ユーザが問合せを出してから結果が返ってくるまでのレスポンス時間の短縮はデータウェアハウスの使い勝手向上のために大事である．実業務での本システムの性能評価はこれからであるが，レスポンス時間が BTF の読み込み性能でほぼ決まることを，測定データをベースにして説明する．

システム側での問合わせ処理時間（d）は，① 問合せ文の解析（前処理）時間，② ディスクからのデータ読み込み時間，③ レコードの演算と結果表示（処理）時間の 3 つの合計となる（**図 15**）．① は通常は数 ms 以下のきわめて短い時間で処理されるため，データ量が多いと ② と ③ でレスポンス時間が決まる．ところで，本システムでは ② と ③ はオーバラップして処理される．問合せ内容が複雑で CPU 処理が非常に重い場合もあるが，このケースは比率として小さく，データ転送ネックとなる問合せが多い．この場合は ③ の時間は ② の時間の影に隠れる（図 15 の (a) のケース）．この結果，データ転送ネックの問合せのレスポンス時間は ② で決まり，図 12，13 に示した BTF の読み込み時間がほぼレスポンス時間に対応する．このことはテストデータを使用したいくつかの簡単な select 文で実際に確認した．

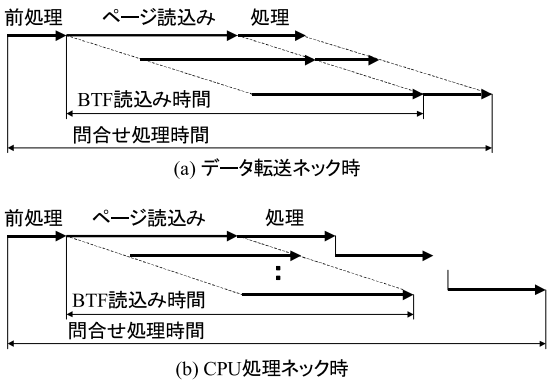


図 15 問合せ処理のレスポンス時間
Fig. 15 Response time for query.

CPU 処理速度について述べる．① とともに，③ は CPU 処理であり処理時間はプログラムの作りに大きく依存する．今回のシステムで，代表的な select 文の処理時間をいくつか測定した．整数データの条件判定，整数データの取り出しなど比較的簡単な実行は平均して 1 データ項目あたり 0.04 μs 前後で，1 データを 4 バイトとすると 100 MB/s の処理速度である．また，BTF ではディスクから読んだデータは転置されており，転置を元に戻して問合せ処理システムに渡す必要がある．この転置復旧の速度は 90 MB/s から 100 MB/s の範囲であり，平均は 95 MB/s であった．これらの値は，使用したサーバのメモリ上でのコピー性能（1 CPU を使用時）とほぼ等しかった．

PCI バス性能が 110 MB/s なので，この速度でデータがディスクから送られてくると，転置の復旧処理で 1 CPU がフルとなり，`select` 文処理でもう 1 CPU がフルとなるが，実際には最適化処理を施しており，CPU 負荷を軽減して簡単にはフル状態にならないようにしている．2.2 節で列方向のパイプライン処理時の最適化手法を述べたが，これを本システムでも採用している．本システムは行方向パイプラインであり，`where` 句で参照されているフィールドデータは全部読み込むが，ブロック単位で `where` 句処理の最適化を行っている．すなわち，`where` 句での各条件判定をブロック内の全レコードに列方向で行い，条件をパスしたレコードに対してのみ転置復旧を含めた以降の処理を行う．この際，条件判定で使用するフィールドデータについては転置復旧と取り出しを同時に行い，転置復旧分の時間を実質 0 にしている．

ソートや like 処理，全件レコードの集計，複雑な条件判定などでは CPU 負荷が大きくなり，問合せ処理が CPU ネックとなりうる（図 15 (b) のケース）．この場合のレスポンス時間は BTF の読み込み時間は CPU 処理時間の影に隠れ，CPU 処理時間がレスポンス時間となる．CPU 処理時間はプログラムの作り，サーバの性能，データベースの内容などに依存するので，一概にはいえない．

以上のように，CPU 処理ネックとならない問合せに対しては，レスポンス時間は BTF 読み込み性能でほぼ決まり，BTF を利用することで通常のサーバでも大きなデータ量を持つデータウェアハウスの利用が

可能となる．BTF は構造と読み込み方式が規則的なため，データ量と参照率，および使用ディスク台数が決まれば，読み込み時間は精度高く計算できるので，レスポンス時間をある値以下におさえるシステムの設計は容易である．今後実システムに適用し，確認していきたい．

5. むすび

ブロック化転置ファイルを利用したデータウェアハウス構築システムを PC サーバ上で作成し，評価を通じて実用的な処理性能を持つデータウェアハウスの提供が可能なことを示した．ここで提案したブロック化転置ファイルはデータベースに対する射影演算をファイルレベルで実現する手法と位置付けられる．さらなるデータウェアハウス処理の高速化には，ディスクの近傍でレコードの選択処理や集計演算を行うことで，主記憶に転送するデータ量のいっそうの絞り込みを図ったり，CPU の負荷を分散したりすることが有効である．また，データ圧縮を行ってディスクに書き込むデータ自体を削減することも効果がある．これらさらなるデータウェアハウス処理の高速化を実現するためのハードウェアや方式の研究および評価も PC サーバを対象に行っており，順次発表していきたい．

参考文献

1) Patterson, D.A. and Hennessy, J.L.: *Computer Architecture A Quantitative Approach*, Second Edition, Morgan Kaufmann Publishers, Inc. (1996).
2) Riedel, E. and Gibson, G.: Active Storage For Large-Scale Data Mining and Multimedia, *Proc.24th VLDB Conference*, New York (1998).
3) Keeton, K. and Patterson, D.A.: A Case for Intelligent Disks (IDISKs), *SIGMOD*, Vol.27, No.3 (1998).
4) Microsoft: *Inside the Windows NT File System*, Microsoft Press (1994).
5) Riedel, E.: A Performance Study of Sequential I/O on Windows NT 4, *Proc. 2nd USENIX Windows NT Symposium* (1998).
6) Batory, D.S.: On Searching Transposed Files, *ACM Trans. Database Syst.*, Vol.4, No.4 (1979).
7) 鵜木昌行：独自データ構造による，データウェアハウスへのアプローチ，信学技報，DE97-75 (Dec. 1997).
8) 鹿島理華，郡　光則ほか：データベースプロセッサ DIAPRISM (1)　I/O プロセッサ制御方式，第 57 回情報処理学会全国大会 4K-07（1998 後期）．
9) 道下　学，郡　光則ほか：データベースプロセッサ DIAPRISM (2)　データ管理方式，第 57 回情報処理学会全国大会 4K-08（1998 後期）．
10) 平岡精一，郡　光則ほか：三菱 DIAPRISM の高速化技術，三菱電機技報，Vol.72, No.11 (1998).

(平成 13 年 4 月 7 日受付)
(平成 13 年 6 月 13 日採録)

（担当編集委員　片岡 良司）

上田　尚純（正会員）

昭和 45 年東京大学工学部電気工学科卒業，同年に三菱電機入社．計算機アーキテクチャ，OS，AI ワークステーション，業務処理用 CASE 等の研究開発に従事し，今日に至る．平成 10 年静岡大学大学院理工学研究科博士後期課程に社会人学生として入学，在学中．ACM 会員．

郡　　光則（正会員）

昭和 59 年東京大学工学部電子工学科卒業，昭和 61 年同大学院電子工学専門課程修了，同年三菱電機入社．計算機アーキテクチャ，並列処理，I/O プロセッサ等の研究開発に従事．平成 4 年本会第 44 回全国大会奨励賞受賞．

青野　正宏（正会員）

昭和 44 年名古屋工業大学工学部経営工学科卒業．同年三菱電機入社．航空管制システム・通信システム等のシステム開発に従事．平成 12 年静岡大学大学院理工学研究科博士後期課程修了．平成 13 年東京工業高等専門学校教授．博士（工学）．技術士（情報工学部門）．電子情報通信学会会員．

渡辺　尚（正会員）

昭和 57 年大阪大学工学部通信学科卒業，昭和 59 年同大学大学院博士前期課程修了．昭和 62 年同大学院博士後期課程修了．工学博士．同年徳島大学工学部情報工学科助手．平成 2 年静岡大学工学部情報知識学科助教授．現在同大学情報学部情報科学科教授．平成 7 年文部省在外研究員（カルフォルニア大学アーバイン校）．計算機ネットワーク，分散システム，マルチエージェントシステムに関する研究等に従事．訳書「計算機設計技法」等．IEEE 会員．

水野　忠則（正会員）

昭和 43 年名古屋工業大学工学部経営工学科卒業．同年三菱電機入社．平成 5 年静岡大学工学部情報知識工学科教授．現在情報学部情報科学科教授．工学博士．情報ネットワーク，プロトコル工学，モーバイルコンピューティングに関する研究に従事．著書「プロトコル言語」，「分散システムコンセプトとデザイン」，「MAP/TOP と生産システム」，「分散システム入門」等．IEEE，ACM 会員．