

データウェアハウス向け高性能データ圧縮方式

郡 光 則†

本稿では、データウェアハウス向けの高性能なデータ圧縮方式について報告する。近年、データウェアハウスの分野では、データを圧縮形式で格納することにより記憶装置コストの削減と入出力負荷の削減を図る方式が実用化されつつある。本稿で報告する圧縮方式は、データを列方向に分割したデータストリームを生成し、行方向の履歴を利用した符号化を行うことを特徴とする。個々のデータストリームにはランレングス符号化、インデックス符号化および差分符号化という単純だが高速な符号化方式を適用する。クエリの処理においては処理に必要なデータストリームがストレージより読み出され、各データストリームが独立に伸張される。本圧縮方式を商用データベース管理システム DIAPRISM/AQL 上に付加し、実データを用いた性能評価を行った。その結果、データ圧縮の適用により必要なストレージ容量がおおむね 2~14%に削減され、また、処理時間がおおむね 5~23%に削減されることを確認し、本圧縮方式の有効性を確認した。

A High-performance Data Compression for Data Warehouse

MITSUNORI KORI†

In this paper, new data compression architecture for data warehouses is presented. Recently, some database management systems have a capability of storing data in a compressed format to reduce storage cost and I/O traffic. In our compression architecture, the data is being split into multiple fixed length streams in the column direction. Each data stream is encoded using the history in the line direction by simple but fast encoding scheme such as run length encoding, index encoding and difference encoding. In the processing of a query, only the needed data streams are read from storage, and decompressed individually. We have attached the compress functionality on a commercial database management system called DIAPRISM/AQL. Our evaluation results using the actual application data show that the storage capacity is reduced to 2-14%, and that the elapse time is reduced to 5-23%. The results show the effectiveness of the data compression architecture.

1. はじめに

近年の情報システムにおいて処理されるデータ量は増大を続けている^{1),2)}。これに対応してプロセッサの性能向上、主メモリや外部記憶装置の大容量化といった基盤技術の進歩も顕著である。しかし、CPU に対してディスク装置をはじめとする入出力装置の性能向上率は低く⁴⁾、データベース処理において入出力が性能のボトルネックとなる傾向が高まりつつある。また、近年ではペタバイト級のデータベースの構築事例も見られるようになり、データ量の削減によるストレージコストや管理コストの低減は依然として重要な課題である³⁾。このような問題を解決する方法として、データベースへのデータ圧縮技術の適用が試みられてきた。筆者らはデータウェアハウスに代表される大規模な

履歴蓄積型のデータベースに適用可能な圧縮方式を開発した。本圧縮方式を既存のデータベース管理システムに適用し、その実用性を検証したので報告する。

2 章では、本分野における従来の研究例を示す。3 章では筆者らが提案するデータ圧縮方式について示す。4 章では、本データ圧縮方式を適用した圧縮データベース管理システムの構成について示す。5 章では、圧縮データベース管理システムの評価を行った結果について示す。

2. 従来の研究

2.1 汎用データベース向けデータ圧縮

データベースにデータ圧縮技術を適用し、データを圧縮形式で格納することにより、記憶装置コストを削減するとともに入出力のデータ転送ボトルネックを解消するというアイディアは古くから存在している⁵⁾⁻¹⁶⁾。

商用データベース IMS では、Huffman 符号化の変形によるデータ圧縮の事例が報告されている⁹⁾。フィー

† 三菱電機株式会社情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

ルドごとの型を推定し、データ型ごとに異なる頻度表を用いて圧縮する。圧縮により CPU 負荷は 17.2% 増えたが、データサイズは 42.1% 減ったと報告されている。

DB2 ではレコードごとに Lempel-Ziv 法^{17),18)} の変形のアルゴリズムを適用した事例が報告されている⁶⁾。これによると、テーブルのスキャンに要する時間が 50% に短縮され、CPU 時間は約 20% 増加したとされている。

圧縮によりデータ量を削減し、データベースのメモリ常駐化を図る試みも報告されている¹²⁾。静的な辞書を用いた圧縮によりデータサイズが 1/4 ~ 1/10 になったとしており、メモリ常駐化による効果とあわせて、10 ~ 10 万倍の性能が得られたとしている。

しかし、汎用データベースにデータ圧縮技術を適用するには一般に以下のような課題がある。

- (1) 更新処理において、データ長が不規則に変化し処理の複雑化と負荷の増大を招く。
- (2) 一般に伸張処理よりも圧縮処理の方が処理負荷が高いため、圧縮処理の必要な書き込みの比率が高いと性能低下を招く。
- (3) 圧縮効率を高めるためには大域的な情報の利用が有効であり、ランダムアクセス中心の用途には不向きである。

このため、トランザクション処理など上記の課題が問題になりやすい分野においては、今日に至るまでデータ圧縮技術の適用は主流となるには至っていない。

2.2 データウェアハウス向けデータ圧縮

一方、1990 年代以降利用が拡大してきたデータウェアハウスは以下に示す特徴を備えており、データ圧縮を適用しやすい。

- (1) 実時間的な更新性能に対する要求が低い。
- (2) 読み出しの比率が高く、書き込みの比率が低い。
- (3) 順次アクセスの比率が高い。

このため、近年、商用データベースにおいて、データウェアハウスを主対象とするデータ圧縮の適用が進んでいる。Oracle では、ブロック内で重複した値を取り除くことによりデータ圧縮を行う方式が報告されている⁵⁾。これによると容量がおおむね 67 ~ 75% 程度削減され、CPU 時間は 5% 増加したがテーブルのフルスキャンに要する時間が 50% 削減されたと報告されている。Teradata では、10 ~ 58% の容量が削減されたとの事例が報告されている¹³⁾。

筆者らはデータウェアハウス向けの商用データベース管理システム DIAPRISM/AQL²⁶⁾ (以下、「AQL」とする) にデータ圧縮技術を適用した。本圧縮方式で

は、データを列方向に分割したデータストリームを生成し、行方向の履歴を利用した符号化を行うことにより高いデータ削減効果を実現する。また、個々のデータストリームにランレングス符号化やインデックス符号化などの単純だが高速な符号化方式を適用することにより、高速な復号化を実現する。本稿ではその実現方式と評価の結果について報告する。

3. データウェアハウス向けデータ圧縮方式

3.1 データベース管理システム AQL

本稿で提案するデータ圧縮方式の説明の準備として、AQL の概要について説明する。

3.1.1 AQL の基本機能

AQL は以下のような特徴を備えたデータベース管理システムである。

- (1) RDB に似たデータモデルを持ち、データは行、列の二次元構造を持つテーブルに格納される。各列は固定長である。SQL に類似したクエリによってテーブルの操作を行う。
- (2) 対象とするデータ規模として数十テラバイト程度までを想定している。テーブルのフルスキャンを重要視した設計であり、ランダムアクセスを行わない。このため、基本的にテーブル内データのメモリ上へのキャッシュを行わない。
- (3) データの「追加」は可能だが、レコード(行)を単位とする「更新」「削除」の機能を備えていない。
- (4) データの列を単位としたブロック化を行い、処理に必要な列を選択的に読み出す。
- (5) 並列処理により高速化を図る。

なお、本稿では、以下、圧縮機能を持たない AQL と区別するために、圧縮機能を付加した AQL を、「圧縮 AQL」と呼ぶ。圧縮 AQL の管理する圧縮形式のテーブルを「圧縮テーブル」と呼ぶ。

3.1.2 AQL のデータ格納方式

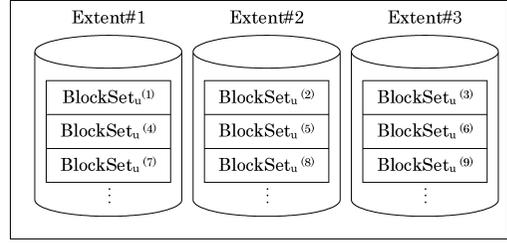
本稿に示すデータ圧縮方式の説明の準備として AQL のデータ格納方式²¹⁾ について示す。AQL では、元データの各列を表 1 に示す形式で格納し、これを「論理フィールド」とする。次いで各論理フィールドを 32 ビットを単位として「内部フィールド」に分割する(図 1)。このとき 32 ビットに満たない部分はパディングを行う。各論理レコードから生成される内部フィールドを連結したものを「内部レコード」と呼ぶ。

i ($i = 1, 2, \dots, M$) 番目の内部フィールドを全内部レコードにわたって連結したデータ列を「データストリーム」と呼び、 $Stream_u^{(i)}$ と表記する。ここで、 i

表 1 AQL の主要なデータ型とデータ格納形式
Table 1 Data format in AQL.

データ型	格納形式	
数値	INT(b)	b ビット整数 ($b = 32$ または 64)
	DEC(p)	$p \leq 9$: 32 ビット整数 $p \geq 10$: 64 ビット整数
文字列	CHAR(l)	l バイトのバイト列
日付	DATE	8 バイト文字列 (YYYYMMDD)

(*) 整数は little endian 形式



エクステント数=3の場合

図 3 AQL のエクステント

Fig. 3 The extent of AQL.

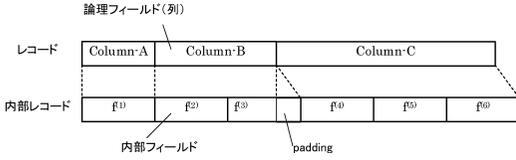


図 1 内部フィールド

Fig. 1 The internal field.

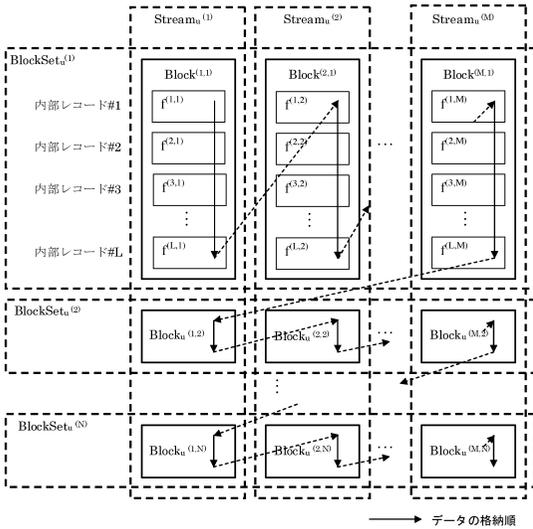


図 2 AQL のブロック構造

Fig. 2 The block structure of AQL.

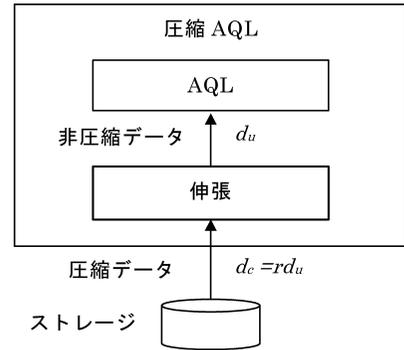


図 4 圧縮テーブル検索の基本データフロー

Fig. 4 Basic data flow of the compressed scan.

含まれるブロックを選択的に読み出す．図 1 の構成において Column-A, Column-C が必要とされる処理の場合, ブロックセット $BlockSet_u^{(j)}$ の中から $Block_u^{(i,j)}$ ($i = 1, 4, 5, 6$) が選択的に読み出される．

3.2 AQL 向けデータ圧縮方式の要件

以上の AQL の特徴から, AQL 向けのデータ圧縮方式は以下の要件を満たす必要があった．

3.2.1 伸張処理における各列の独立性

AQL のデータ処理方式を保ったままデータ圧縮を適用するには, 各列に対応する圧縮データストリームを他の列の内容に依存せず伸張可能とする必要がある．一方, AQL は順次処理によるテーブルのフルスキャンを基本としているため, 圧縮において複数の行にまたがる依存性や履歴を利用可能である．

3.2.2 伸張処理速度

AQL の基本設計を踏襲し, 本圧縮方式ではフルスキャンによる検索処理速度を重要視する．圧縮 AQL によって圧縮されたテーブルを検索する場合の基本的なデータフローを図 4 に示す．圧縮 AQL の処理可能な伸張速度 d_u [Bytes/sec] を

$$d_u = \frac{s_u}{t_u} \tag{1}$$

をデータストリーム番号と呼ぶことにする．各データストリームはストレージ入出力の単位となる「ブロック」と呼ぶ単位に分割される． $Stream_u^{(i)}$ の j 番目のブロックを $Block_u^{(i,j)}$ と表記するとき, $Block_u^{(i,j)}$ の集合を「ブロックセット」と呼び $BlockSet_u^{(j)}$ と表記する (図 2)．

AQL のテーブルは複数のブロックセットから構成され, ブロックセットを単位として複数の「エクステント」と呼ぶストレージ領域に分割されて格納される．AQL はエクステントを単位とした並列処理を行う²¹⁾ (図 3)．

AQL は問合せに応じて必要なデータストリームに

とする．ここで s_u は圧縮前データのサイズ [Bytes] , t_u は伸張処理時間 [sec] とする．

以下，本稿では圧縮率 r を式 (2) とする．

$$r = \frac{s_c}{s_u} \quad (2)$$

ただし s_c は圧縮後のデータサイズ [Bytes] である．このとき，圧縮 AQL の処理可能な圧縮データの速度 d_c は

$$d_c = r \cdot d_u \quad (3)$$

となる．ストレージ，プロセッサの稼働率を最大に保つには，ストレージからのデータ転送能力と d_c がほぼバランスする必要がある．

3.3 AQL 向け圧縮方式

以上の要件から，AQL では以下に示すデータ圧縮方式を採用した．

3.3.1 マルチストリームデータ圧縮

本圧縮方式では，図 5 に示すように，AQL のデータストリームを独立に圧縮して圧縮データストリームを生成する．問合せの処理時には必要な列に関する圧縮データストリームだけを読み出し，各圧縮データストリームを独立に伸張し処理する．このような処理方式をマルチストリームデータ圧縮と呼ぶことにする．

3.3.2 RID 符号化

分解されたデータストリームは RID (Run-length, Index and Difference) 符号化方式と呼ぶ方式によって符号化される．

RID 符号化方式は，基本的に古くから知られている以下の符号化方式¹⁹⁾ の組合せによって実現される．

- (1) ランレングス符号化 (Run-length Encoding)
- (2) インデックス符号化 (Index Encoding)
- (3) 差分符号化 (Difference Encoding)

ランレングス符号化は，同一符号の繰返しを (符号，長さ) の組で置換する符号化方式である．インデックス符号化は，データの値の種類ごとに対応するビット幅の小さい符号を割り当て，その符号で置換する符号化方式である．差分符号化は隣接する符号の差分をと

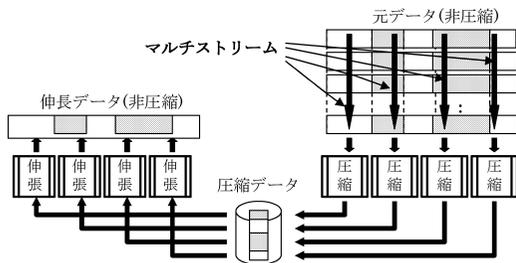


図 5 マルチストリームデータ圧縮

Fig. 5 The multiple stream data compression.

る符号化方式であり，それ自体にはデータ圧縮の効果はないが，差分符号化によりランレングス符号化，インデックス符号化の圧縮効率が高まる場合がある．

3 種類の符号化方式のうち，インデックス符号化は行方向の履歴を利用しない符号化方式だがランレングス符号化および差分符号化はデータの行方向の履歴を利用する．

RID 符号化方式は，比較的単純な符号化方式の組合せにより実現されており，複雑な演算処理を必要としない．また，RID 符号化方式は，非適応的 (non-adaptive) 圧縮¹⁹⁾ であり，圧縮対象のデータを 2 回走査する 2 パスの手順によって符号化を行う．すなわち，まず大域的な情報を取得して圧縮用のパラメータを決定してから (パス 1)，決定したパラメータに基づいて圧縮伸張処理を行う (パス 2)．非適応的圧縮は，伸張時に統計情報を動的に更新する必要がないため伸張処理速度の点で有利である．

3.3.3 圧縮データストリーム

各圧縮データストリームは複数の圧縮ブロックに分割され，各圧縮ブロック内で RID 符号化が行われる．図 6 に圧縮データストリームの構造を示す．圧縮データストリームはデータ圧縮非適用時のデータストリームと同様の構造を持ち，ブロックに相当する構造を圧縮ブロック $Block_c^{(i,j)}$ ブロックセットに相当する構造を圧縮ブロックセット $BlockSet_c^{(j)}$ と呼ぶ．各圧縮ブロック，圧縮ブロックセットのサイズは可変である．

圧縮ブロックセット内のすべての圧縮ブロックを同時に読み出し可能とするため，各圧縮ブロックセット $BlockSet_c^{(j)}$ のサイズ $s_{blockset}^{(j)}$ は事前に設定された主記憶上のバッファサイズ S_{buffer} を超えないようにする．すなわち， $Block_c^{(i,j)}$ のサイズを $s_{block}^{(i,j)}$ と表記するとき，

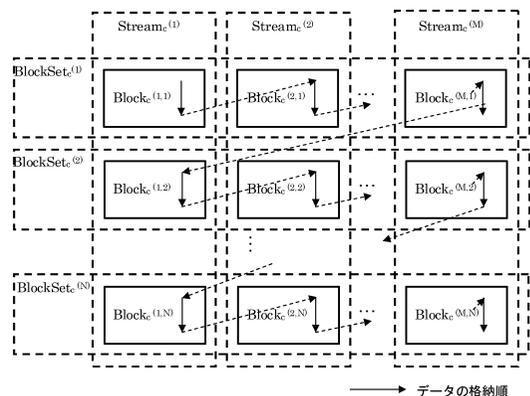


図 6 圧縮データストリームの構造

Fig. 6 The structure of the compressed data stream.

$$s_{blockset}^{(j)} = \sum_{i=1}^M s_{block}^{(i,j)} \leq S_{buffer} \quad (4)$$

という制限のもとに圧縮ブロックのサイズを決定する。

3.3.4 圧縮ブロック

各圧縮データストリームに含まれる圧縮ブロックは「圧縮形式ブロック」「非圧縮形式ブロック」のいずれかの形式をとる。圧縮形式ブロックはRID ヘッダ、RID 符号表、RID パケット列からなる。非圧縮形式ブロックはRID ヘッダ、非圧縮符号列からなる。

RID ヘッダは圧縮形式ブロック/非圧縮形式ブロックの区別、圧縮形式ブロックにおける差分圧縮の有効/無効や圧縮パラメータなどの基本情報を格納する。

RID 符号表はインデックス符号化で使用する構造であり、重複を除去した一意の符号を含む。

図 7 に RID パケットの構造を示す。RID パケットは 1 または複数の RID トークンを含む固定長のデータである。RID トークンのビット幅は同一のブロック内で一定であり、ブロックごとに異なる値をとることができる。RID パケットのビット数が RID トークンのビット数の倍数でない場合はパディングが行われる。RID パケットは固定長であり、そのビット幅 B_{packet} は現在の実装では 32 である。AQL が 32 ビットプロセッサを中心に稼働していることから、処理効率を考慮し 32 ビット幅に設定した。

RID トークンには、RID インデックスと RID ラン長が含まれる。RID インデックス、RID ラン長のビット幅は圧縮ブロックごとに異なる値をとることができる。RID インデックスのビット幅 $b_{index}^{(i,j)}$ は RID 符号表内のオフセットを表現可能な最小のビット幅とする。すなわち、

$$b_{index}^{(i,j)} = \lceil \log_2 w_{index}^{(i,j)} \rceil \quad (5)$$

となる。ただし、 $w_{index}^{(i,j)}$ はブロック $Block^{(i,j)}$ 中に出現する符号 (32 ビットワード) の種類を示す。

一方、同一の RID インデックスを含む RID トークンを繰り返すことにより、RID ラン長で表現可能な回数を超える繰返し回数を表現できる。このため、ブ

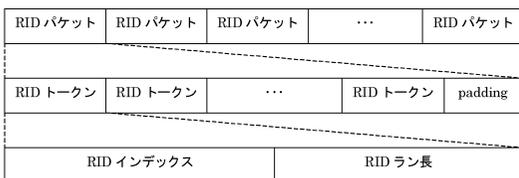


図 7 RID パケット

Fig. 7 The RID packet.

ロック $Block^{(i,j)}$ 内の RID ラン長のビット幅 $b_{length}^{(i,j)}$ は、

$$0 \leq b_{length}^{(i,j)} \leq B_{packet} - b_{index}^{(i,j)} \quad (6)$$

の範囲で選択可能である。 $b_{length}^{(i,j)}$ をゼロに固定すると、ランレングス符号化を無効化することができる。

RID パケットの数、RID インデックスおよび RID ラン長のビット数などのパラメータは RID ヘッダに格納される。

一方、非圧縮形式ブロックの非圧縮符号列には、圧縮前のデータストリームに含まれていた符号列がそのまま格納される。

3.4 符号化/復号化の手順

RID 符号化ではデータストリームに対して以下の手順を適用して圧縮ブロックを生成する。

手順 1: RID 符号化パス 1 (圧縮パラメータの決定)

(1-1) RID インデックスのビット幅 $b_{index}^{(i,j)}$ を式 (5) とする。

(1-2) 式 (6) の範囲のすべての RID ラン長 $b_{length}^{(i,j)}$ につき、差分符号化を適用した場合と適用しない場合の両方について圧縮後のサイズを計算し、最小になる場合を採用する。ただし、いずれの場合の圧縮後のサイズも圧縮前のサイズ $w_u^{(i,j)}$ より小さくならない場合は、圧縮を無効とする。

(1-3) 重複を除去した符号の表、すなわち RID 符号表を生成する。

(1-4) 一定数の符号が追加されるごとに (1-1) ~ (1-3) の手順を実行し、圧縮ブロックのサイズ $s_{block}^{(i,j)}$ が式 (4) の制約を満たすような最大のブロックを生成する。

手順 2: RID 符号化パス 2 (圧縮処理)

(2-1) ブロックの形式が非圧縮形式ブロックの場合は、圧縮前のデータをそのままブロックとし、終了する。圧縮形式ブロックの場合は (2-2) 以下の手順を行う。

(2-2) 差分符号化が有効な場合は符号列に対して差分符号化を行う。

(2-3) 32 ビットワードの符号を単位としたランレングス符号化を行い、 l 回連続する符号 c を (c, l) の組に置換する。

(2-4) (2-3) で生成された (c, l) について、 $l > 2^{b_{length}}$ の場合は l が $2^{b_{length}}$ 以下になるように (c, l) を分割する。

(2-5) (c, l) の符号 c の部分にインデックス符号化を適用する。すなわち、符号 c を RID 符号表内

のインデックス (RID インデックス) によって置換する。

一方、復号化は以下の手順によって行う。

手順 3 : RID 復号化

- (3-1) 非圧縮形式ブロックの場合は符号列を出力し終了する。圧縮形式ブロックの場合は (3-2) 以下の手順を行う。
- (3-2) RID パケット列の RID インデックスに対しインデックス復号化を行う。すなわち、RID 符号表から RID インデックスをオフセットとする符号を取得し、(c, l) の組を生成する。
- (3-3) (3-2) の手順で得られた (c, l) に対しランレングス復号化を行う。符号 c を RID ラン長 l で示される回数だけ繰り返し符号列を生成する。
- (3-4) 差分符号化が行われている場合はさらに差分復号化を行う。

4. 圧縮 AQL の実装

圧縮伸張処理を AQL に付加し、圧縮機能を備えたデータベース管理システム「圧縮 AQL」を構築した。圧縮 AQL のアプリケーションインタフェースや問合せ文など外部インタフェースは基本的に AQL と同一であり、圧縮機能は透過的に実装されている。

4.1 ストレージ上の圧縮テーブルの構造

圧縮テーブルは「一時ページセット」「主ページセット」に分割して格納される。一時ページセットは、圧縮前のデータを一時的に格納するストレージ領域である。主ページセットは圧縮後のデータを格納するストレージ領域である。各「ページセット」は複数のエクステントに分割されて格納され、エクステントを単位とした並列入出力が行われる。

4.2 データのロード

圧縮 AQL のデータロードは以下の 2 段階の手順により行われる。圧縮機能を備えない AQL では基本的に「(1) 初期ロード」だけが行われる。「(2) 圧縮ブロック生成」は圧縮機能追加にともない追加された処理である。

- (1) 初期ロード：ロード対象のデータを非圧縮状態のまま一時ページセットに格納する。一時ページセット内のブロックは非圧縮形式ブロックとなる。
- (2) 圧縮ブロック生成：一時ページセット内に事前に設定した閾値以上の量のデータが蓄積されると RID 符号化の処理を起動し、生成された圧縮ブロックセットを主ページセットに格納する。RID 符号化は 2 パスで実行されるため、圧縮

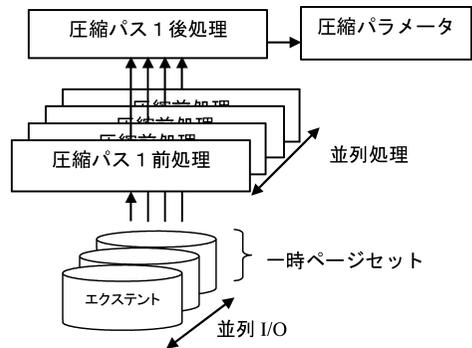


図 8 圧縮処理 (パス 1)
Fig. 8 The compress processing (Pass 1).

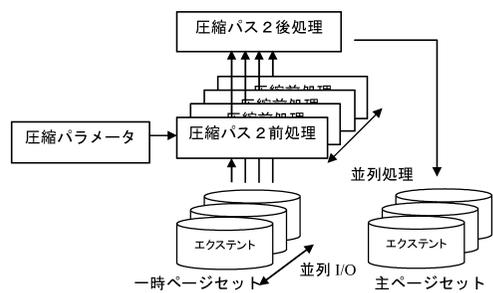


図 9 圧縮処理 (パス 2)
Fig. 9 The compress processing (Pass 2).

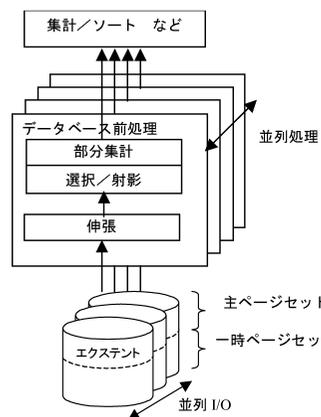


図 10 圧縮 AQL の検索集計処理の一例
Fig. 10 An example of the query processing in the AQL with compression.

ブロック生成においては一時ページセットを 2 回走査する。各パスにおいて、圧縮ブロック生成の処理は圧縮ブロックセットを単位として並列処理される (図 8, 図 9)。主ページセットに移されたデータは一時ページセットから削除される。

表 2 評価において使用するデータ
Table 2 Data for the evaluation.

データ名称	列数	レコード長 [Bytes]	レコード数	概要
データ 1	52	518	200,000,000	予算管理システムにおける購買品目の明細（仕訳明細）
データ 2	75	979	200,000,000	売上分析システムにおける売上の明細（売上明細）
データ 3	22	211	200,000,000	販売管理システムにおける顧客の明細（保有明細）

表 3 表の構成（データ 1）
Table 3 Table characteristics (Data 1).

列名	DS	データ型	データ種類	列名	DS	データ型	データ種類
Column1	1	INT(64)	2000000000	Column27	48	CHAR(7)	1
Column2	3	DATE	1	Column28	50	CHAR(7)	1
Column3	5	CHAR(7)	11	Column29	52	CHAR(2)	3
Column4	7	CHAR(4)	1	Column30	53	CHAR(6)	11
Column5	8	CHAR(3)	1	Column31	55	CHAR(5)	393
Column6	9	CHAR(4)	1	Column32	57	CHAR(2)	10
Column7	10	CHAR(3)	42	Column33	58	CHAR(8)	11518
Column8	11	CHAR(21)	14737	Column34	60	CHAR(4)	333
Column9	17	CHAR(2)	21	Column35	61	CHAR(1)	7
Column10	18	CHAR(7)	1	Column36	62	CHAR(1)	4
Column11	20	CHAR(2)	5	Column37	63	CHAR(3)	121
Column12	21	CHAR(1)	1	Column38	64	CHAR(2)	34
Column13	22	CHAR(12)	1	Column39	65	CHAR(2)	7
Column14	25	CHAR(8)	1	Column40	66	CHAR(1)	1
Column15	27	CHAR(6)	26	Column41	67	CHAR(6)	26
Column16	29	CHAR(1)	5	Column42	69	CHAR(4)	1
Column17	30	CHAR(1)	3	Column43	70	CHAR(1)	5
Column18	31	CHAR(1)	4	Column44	71	CHAR(4)	22
Column19	32	CHAR(7)	1	Column45	72	CHAR(4)	46
Column20	34	CHAR(7)	1	Column46	73	CHAR(4)	20
Column21	36	CHAR(7)	1	Column47	74	CHAR(4)	40
Column22	38	CHAR(7)	1	Column48	75	CHAR(3)	1
Column23	40	CHAR(7)	1	Column49	76	CHAR(140)	35301
Column24	42	CHAR(7)	1	Column50	111	CHAR(130)	26773
Column25	44	CHAR(7)	1	Column51	144	CHAR(4)	49
Column26	46	CHAR(7)	1	Column52	145	INT(64)	22069

DS：先頭のデータストリーム番号

4.3 検索集計処理

図 10 に圧縮 AQL における検索集計処理の処理方法を示す²³⁾。問合せ要求が発行されると、圧縮テーブルを構成する主ページセット、一時ページセットに含まれる圧縮ブロックセットから、処理に必要な圧縮ブロックが順に読み出される。圧縮テーブルの読み出しにあたっては OS のバッファキャッシュはバイパスされ、つねにストレージからの読み出しが行われる。

読み出された圧縮ブロックに対してデータベース前処理が行われる。データベース前処理ではまず、圧縮ブロックを単位として伸張処理を行い、伸張されたデータをバッファに格納し、バッファ上のデータに対して選択/射影や部分集計などの処理を行う。データベース前処理は独立したスレッドによって並列処理可能である。またインテリジェントストレージ^{22),24)}を備えたハードウェア構成では、複数のサーバによるデータ

ベース前処理の並列処理が可能である。これにより、処理負荷の高い伸張処理の処理時間短縮を図る。データベース前処理を経た後、集計やソートなどの後処理が行われる。

5. 評価

本圧縮方式の有効性を検証するため、圧縮率および速度性能の観点から評価を行った。

5.1 評価対象データ

本評価では、3種類の業務処理で使用されているデータを使用した（表 2）。各データを格納するテーブルの各列のデータ型およびデータ中に出現する相異なる値の種類（「データ種類」）を表 3、表 4、表 5 に示す。なお、各列の列名（Column1 等）は今回の評価において設定した名前であるが、データ型やデータ内容は元の業務処理で使用されていたものと同一である。

表 4 表の構成 (データ 2)
Table 4 Table characteristics (Data 2).

列名	DS	データ型	データ種類	列名	DS	データ型	データ種類
Column1	1	INT(64)	2000000000	Column39	127	CHAR(40)	195
Column2	3	DATE	1	Column40	137	CHAR(40)	20
Column3	5	CHAR(10)	10000	Column41	147	CHAR(40)	25
Column4	8	CHAR(20)	10000	Column42	157	CHAR(40)	9
Column5	13	CHAR(30)	195	Column43	167	CHAR(3)	376
Column6	21	CHAR(30)	242	Column44	168	CHAR(4)	371
Column7	29	CHAR(30)	195	Column45	169	CHAR(40)	541
Column8	37	CHAR(30)	189	Column46	179	CHAR(40)	1
Column9	45	CHAR(30)	195	Column47	189	CHAR(40)	1
Column10	53	CHAR(30)	189	Column48	199	CHAR(12)	9998
Column11	61	CHAR(1)	1	Column49	202	CHAR(12)	10000
Column12	62	CHAR(8)	6344	Column50	205	CHAR(20)	1000000
Column13	64	CHAR(2)	12	Column51	210	CHAR(10)	5
Column14	65	CHAR(1)	3	Column52	213	CHAR(3)	3
Column15	66	CHAR(3)	376	Column53	214	CHAR(10)	10
Column16	67	CHAR(4)	371	Column54	217	CHAR(10)	5
Column17	68	CHAR(40)	541	Column55	220	CHAR(10)	4
Column18	78	CHAR(40)	1	Column56	223	CHAR(6)	24
Column19	88	CHAR(40)	1	Column57	225	CHAR(2)	28
Column20	98	CHAR(12)	9999	Column58	226	CHAR(16)	20000
Column21	101	CHAR(12)	10000	Column59	230	CHAR(50)	20000
Column22	104	CHAR(1)	2	Column60	243	CHAR(10)	18
Column23	105	CHAR(1)	2	Column61	246	CHAR(1)	3
Column24	106	CHAR(1)	2	Column62	247	CHAR(1)	1
Column25	107	CHAR(1)	2	Column63	248	CHAR(1)	1
Column26	108	CHAR(1)	2	Column64	249	DEC(5)	36
Column27	109	CHAR(1)	2	Column65	250	DEC(9)	3023
Column28	110	CHAR(3)	7	Column66	251	CHAR(2)	5
Column29	111	CHAR(2)	5	Column67	252	DEC(2)	10
Column30	112	DEC(3)	30	Column68	253	CHAR(8)	672
Column31	113	CHAR(1)	3	Column69	255	CHAR(8)	672
Column32	114	CHAR(1)	2	Column70	257	DEC(9)	7505
Column33	115	DEC(10)	1622	Column71	258	DEC(9)	1
Column34	117	CHAR(2)	3	Column72	259	CHAR(10)	1
Column35	118	CHAR(2)	5	Column73	262	CHAR(8)	1
Column36	119	CHAR(16)	10000	Column74	264	CHAR(8)	1
Column37	123	CHAR(8)	1	Column75	266	CHAR(8)	1
Column38	125	CHAR(8)	1				

DS : 先頭のデータストリーム番号

表 5 表の構成 (データ 3)
Table 5 Table characteristics (Data 3).

列名	DS	データ型	データ種類	列名	DS	データ型	データ種類
Column1	1	INT(64)	2000000000	Column12	20	CHAR(1)	4
Column2	3	DATE	1	Column13	21	CHAR(7)	53077
Column3	5	CHAR(13)	164969	Column14	23	CHAR(4)	32
Column4	9	CHAR(2)	15	Column15	24	CHAR(15)	1127
Column5	10	CHAR(4)	33	Column16	28	CHAR(50)	53567
Column6	11	CHAR(2)	13	Column17	41	CHAR(50)	48926
Column7	12	CHAR(2)	8	Column18	54	CHAR(2)	18
Column8	13	CHAR(2)	9	Column19	55	CHAR(4)	310
Column9	14	CHAR(2)	9	Column20	56	CHAR(4)	1671
Column10	15	CHAR(15)	4822	Column21	57	CHAR(3)	11
Column11	19	CHAR(1)	2	Column22	58	CHAR(12)	39

DS : 先頭のデータストリーム番号

各データについて Column1 と Column2 の形式は共通である。Column1 にはレコード番号として 1 から始まる増分 1 の一意の値が格納される。Column2 にはロード日付として、該当レコードがロードされた日付が格納されるが、本評価においてはすべて同一の日付が格納されている。

本評価では、データ圧縮の必要性が高いと考えられる大規模なデータを使用した場合の評価に重点を置くが、並列化の単位である圧縮ブロックセットの数がエクステント数やデータベース前処理スレッド数と同程度以下では十分な並列化が行われず、大規模データ使用時の定常的な挙動を反映しなくなる。5.3 節に示すように、速度性能の評価では、エクステント数、データベース前処理スレッド数とも 8 とした。これらの条件から、本評価では圧縮ブロックセットの数がおおむね 30 を超えるようなデータを使用することにし、それぞれ 2 億レコードのデータを使用した。なお、実際に生成された圧縮ブロックセットの数を表 6 に示す。

5.2 圧縮率の評価

5.2.1 圧縮率

各データに本方式による圧縮を適用した際のデータサイズを圧縮非適用時と比較し、データ量削減の効果を評価する。本評価では、通常の使用条件でバッファ 1 個あたり利用可能なメモリ量を考慮し、最大バッファサイズ S_{buffer} を 128 [Mbytes] と設定した。

本方式による圧縮を適用した際のデータサイズを圧縮非適用時と比較した結果を表 6 に示す。圧縮の適用により、データ量が 2~14%程度に削減されている。

なお、3.1.2 項に示したとおり、AQL ではデータストリームを生成する際にレコードを内部レコードに変換する。本稿では内部レコードのサイズを基準として圧縮率を評価することにするが、表 6 に示すように、内部レコード長は元の論理的なレコード長よりも 10%程度増加している。

次に、RID 符号化方式を構成するランレングス符号化、インデックス符号化、差分符号化のそれぞれの効果を評価するため、通常の RID 符号化 (1) 以外に以下の (2)~(4) の場合について比較する (図 11)。

- (1) ランレングス符号化、インデックス符号化、差分符号化それぞれをすべて有効にした場合 (以下「RID」と表記)
- (2) RID において差分符号化を無効とした場合 (以下「RI」と表記)
- (3) RID において RID ラン長のビット幅 $b_{length}^{(i,j)}$ をゼロに固定しランレングス符号化を無効化した場合 (以下「ID」と表記)

- (4) RID において差分符号化、ランレングス符号化の両方を無効化した場合 (以下「I」と表記)

さらに、広く利用されている圧縮方式の一例として、各データストリームを gzip²⁷⁾ を用いて圧縮した場合についても図 11 に示す (図中「gzip」と表記)。gzip は LZ77¹⁷⁾ をベースにした圧縮方式を用いており、行方向の履歴を利用した圧縮を行う。

RID と RI の比較から、差分符号化の適用により圧縮後データサイズがおおむね 15%程度縮小されていることが分かる。また、RID と ID、および RI と I の比較から、ランレングス符号化の適用により圧縮後データサイズがおおむね 25~30%程度に削減されていることが分かる。さらに、RID により gzip とおおむね同程度のデータ量削減の効果が得られていることが分かる。

インデックス符号化は行方向の履歴を利用しない符号化方式であるが、ランレングス符号化、差分符号化は行方向の履歴を利用する。本評価結果は、行方向の履歴を利用することにより、ランレングス符号化や差分符号化のような比較的単純な符号化方式であっても、大幅なデータ量削減が可能であることを示している。

一方、行方向の履歴を利用する符号方式は行単位のランダムアクセスに向かないという課題がある。商用データベースについて報告されている事例の多く^{5),6),13)} は行方向の履歴に依存しない圧縮方式を採用している。

5.2.2 各データストリームごとの圧縮率

図 12, 図 13, 図 14 に、データ 1~データ 3 を構成する各データストリームごとの圧縮率を示す。

図 12~図 14 に見られるように、データストリームごとの圧縮率には著しい偏りがある。比較的単純な符号化方式にもかかわらず、大半のデータストリームのサイズは 10%以下に削減される。しかし、一部のデータストリームのサイズはたかだか 50%程度にしか削減されない。このため、データ圧縮の結果、少数の圧縮データストリームが圧縮後のデータサイズの大半を占める傾向がみられる。

表 7 に、各データを構成するデータストリームの中でデータ量削減効果の小さい 5 データストリームを示す。5.1 節に示したとおり、Column1 はレコード連番であるため、RID 圧縮方式の差分符号化の結果、圧縮後のサイズがきわめて小さくなる。また、Column2 はロード日付であるが、本評価時の条件からすべて同一の値が含まれているため、同様に圧縮後サイズがきわめて小さくなる。これら Column1, Column2 を除外した列に関するデータ型ごとの圧縮率の平均値を表 8 に示す。

表 6 評価対象データの圧縮率
Table 6 Compress ratio.

データ	#DS	内部レコード長 [Bytes]	内部レコード長 /レコード長	圧縮前サイズ s_u [MBytes]	圧縮後サイズ s_c [MBytes]	圧縮率 s_c/s_u	圧縮ブロック セット数
データ 1	146	584	1.13	111,389	4,327	0.039	34
データ 2	267	1068	1.09	48,057	4,937	0.024	39
データ 3	60	240	1.14	45,776	6,534	0.145	58

#DS : データストリーム数

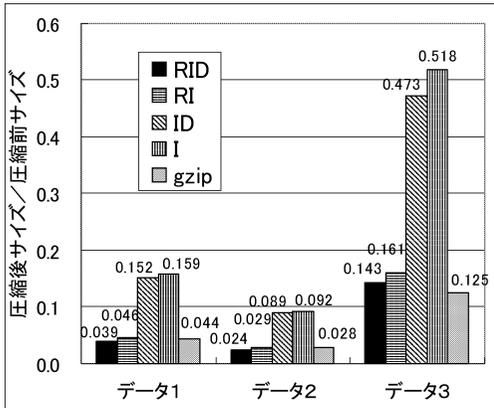


図 11 各符号化方式の効果の比較

Fig. 11 Comparison of the effect of the encoding scheme.

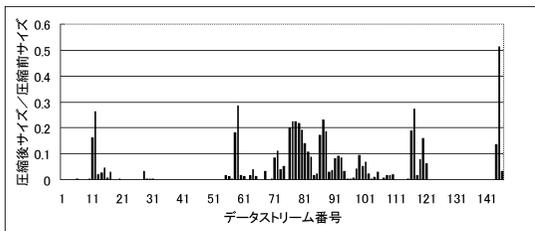


図 12 データストリームごとの圧縮率 (データ 1)

Fig. 12 Compress ratio for each data stream (Data1).

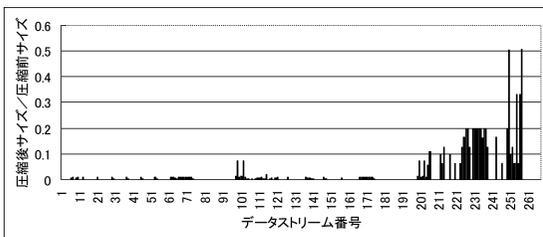


図 13 データストリームごとの圧縮率 (データ 2)

Fig. 13 Compress ratio for each data stream (Data2).

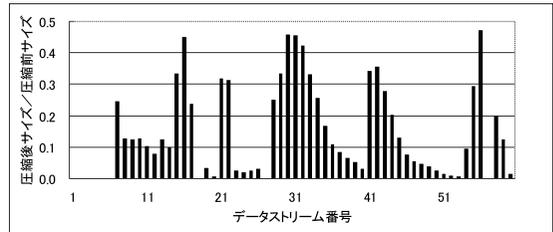


図 14 データストリームごとの圧縮率 (データ 3)

Fig. 14 Compress ratio for each data stream (Data3).

表 7 データ削減効果の小さいデータストリーム

Table 7 Data streams with low space saving.

データ	DS	列名	データ型	圧縮率
データ 1	145	Column52	INT(64)	0.495
	59	Column33	CHAR(8)	0.282
	117	Column50	CHAR(130)	0.269
	12	Column8	CHAR(21)	0.259
	87	Column49	CHAR(140)	0.226
データ 2	257	Column70	DEC(9)	0.501
	250	Column65	DEC(9)	0.500
	254	Column68	CHAR(8)	0.332
	256	Column69	CHAR(8)	0.332
	231	Column59	CHAR(50)	0.200
データ 3	56	Column20	CHAR(4)	0.469
	30	Column16	CHAR(50)	0.456
	31	Column16	CHAR(50)	0.453
	16	Column10	CHAR(15)	0.448
	32	Column16	CHAR(50)	0.421

DS : データストリーム番号

圧縮率 = 圧縮後サイズ/圧縮前サイズ, とする

表 8 データ型ごとの平均圧縮率

Table 8 Average compress ratio for each data type.

データ	数値型			文字型		
	列数	#DS	圧縮率	列数	#DS	圧縮率
データ 1	1	2	0.263	49	140	0.037
データ 2	7	8	0.169	66	255	0.020
データ 3	0	0	-	20	56	0.153

#DS : データストリーム数

圧縮率 = 圧縮後サイズ/圧縮前サイズ, とする

表 7, 表 8 から数値型データのデータストリームに対するデータ量の削減効果が文字型よりも低い傾向が見られる。

なお, データ 3 のデータストリーム番号 28~40, 41~53 には長さ 50 バイトの文字列データ (Col-

umn16, Column17) が存在しているが, 図 14 より, 文字列の末尾に進むに従い圧縮率が高くなっていることが分かる。これは, 固定長文字列の末尾に近づくにつれ空白の比率が高まることが原因と考えられる。

表 9 性能評価環境

Table 9 Performance evaluation platform.

OS	Windows 2003 Server x64 Enterprise Edition
File System	NTFS
CPU	Intel Xeon MP 3.66 GHz × 4 1 MB Secondary Cache
Chipset	Intel E8500
Memory	DDR-2/400 MHz SDRAM 16 GB
HDD	MAXTOR ATLAS 15K2.146SCA × 10
SCSI	Ultra320 × 1

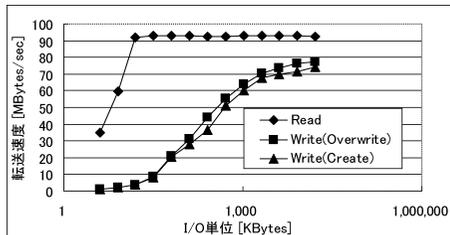


図 15 評価環境におけるディスク装置の単体性能

Fig. 15 The data transfer performance of the disk unit in the evaluation platform.

5.3 速度性能の評価

次に圧縮 AQL と圧縮機能を持たない AQL の速度性能を比較する。

5.3.1 評価環境

本評価では表 9 に示す PC サーバを用いて評価を行った。圧縮テーブルは 8 個のエクステントに分散配置され、それぞれのエクステントが 1 個のディスク装置と対応する。

評価対象ハードウェアは 4 個の CPU を備えているが、評価対象 CPU の SMT (Simultaneous-Multi Threading) 機能 (HyperThreading) を利用し、1 CPU あたり 2 個の論理プロセッサによる 2 スレッドの同時実行が可能のため、圧縮前処理 (パス 1 およびパス 2) およびデータベース前処理では 8 スレッドによる並列処理を行う。

5.3.2 予備評価

本評価環境では、ディスク装置に対する入出力性能が性能ボトルネックとなることが多いため、予備評価としてディスク入出力の基礎性能の実測結果を示す。

図 15 に、本評価環境におけるディスク装置の順次アクセス性能の実測値を示す。ただし、1 [GBytes] のファイルに対して入出力単位を変動させたときの経過時間から転送速度を算出した。なお、このディスク装置はライトスルーに設定されている。図 16 には 1~10 個のディスク装置を同時に動作させたときの総転送速度の実測値を示す。ただし、入出力単位は 1 [MBytes]

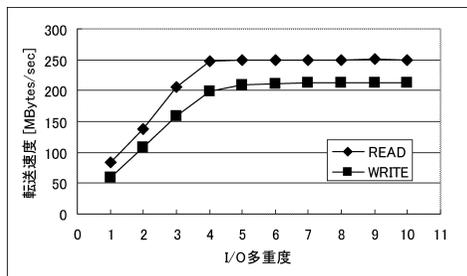


図 16 評価環境における SCSI 転送速度

Fig. 16 The SCSI transfer rate in the evaluation platform.

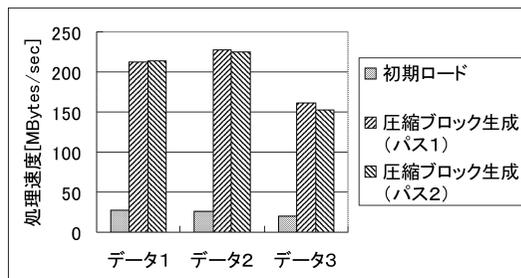


図 17 データロードの処理速度

Fig. 17 The data load performance.

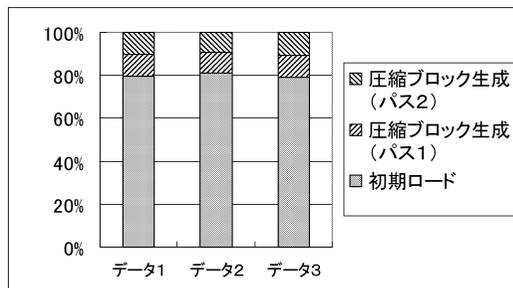


図 18 データロード時間の内訳

Fig. 18 The detail of the data load time.

とした。本評価環境では読み出しの場合、おおむね 250 [MBytes/sec] 程度で Ultra-320 SCSI 転送能力の限界に達している。

5.3.3 データロード性能

図 17 に、表 2 の各データについて CSV 形式の元データを AQL の表データロードツール²⁹⁾を用いて圧縮テーブルにロードする際のロード速度 [MBytes/sec]を示す。また、図 18 に、ロード時間の内訳を示す。ロード時間の約 80%は初期ロード時間が占めており、圧縮機能を付加したことによる処理時間の増加は約 20%程度にとどまっている。

なお、初期ロード時間のほとんどは表データロードツールによって CSV 形式ファイルから列データを抽

表 10 性能評価において使用する問合せ
Table 10 The queries for the performance evaluation.

Query	処理内容	データ	対象列数	#DS	出力データ量
query-1	仕訳明細をコード番号ごとに集計する	データ 1	3	5	192 [Bytes]
query-2	仕訳明細から条件に合致する明細データを取り出す	データ 1	10	14	21.7 [MBytes]
query-3	売上明細を期間、店舗ごとに集計する	データ 2	7	24	50 [Bytes]
query-4	売上明細から条件に合致する明細データを取り出す	データ 2	13	58	19.3 [MBytes]
query-5	保有明細を住所、銘柄ごとに集計する	データ 3	5	7	1958 [Bytes]
query-6	保有明細から条件に合致する明細データを取り出す	データ 3	18	52	0.172 [MBytes]

#DS: 処理対象データストリーム数

出する処理の CPU 時間が占めていることが確認されている。また、圧縮の各段階はストレージからの非圧縮データの読み出しが性能のボトルネックになっていることが確認されている。

5.3.4 検索集計性能

圧縮 AQL と圧縮機能を持たない AQL を用いて同一の検索集計の問合せを行い、その処理時間を比較することにより、速度性能の評価を行う。

本評価では、3 種類の実業務システムにおいて最も典型的と考えられる基本処理を 2 種類ずつ抽出し使用した (表 10)。これらはデータ 1~データ 3 のいずれかを使用する。表 10 において、「対象列数」は実際に処理の対象となる列 (論理フィールド) の数を示す。「処理対象データストリーム数」(#DS) は実際に処理の対象となるデータストリームの数を示す。各処理に対する問合せの詳細を表 11 に示す。

AQL の問合せ文の記法はおおむね標準的な SQL に準じているが、本評価で使用した処理では、「カテゴリ」と呼ぶ AQL 固有の機能²⁸⁾を使用している。カテゴリとは、データの分類を表現する定義情報であり、階層構造を持つ「主要素」から構成される。カテゴリの一例として図 19 に query-1 で使用されているカテゴリ CAT1 の構成を示す。図において、Level1~Level4 が「主要素」であり、Level4 が最下層の主要素である。AQL の問合せ文の GROUP BY 句において、GROUP BY CATEGORY (column TO cat.level) とは、カテゴリ cat の最下層から、列 column の値と一致する主要素を抽出し、level によって指定された階層から対応する主要素を抽出し、それをキーとして集計を行うことを意味する。たとえば query-1 では、Column51 の値を Level1~Level4 の 4 階層のコード番号ごとに分類して集計する。query-3 のように ROUND DOWN を指定すると、カテゴリ cat の最下層の主要素に対応する値がない場合、column の値より小さく、最も近い主要素と対応付けられる。たとえば、query-3 では COL4, COL5, COL6 の値をその範囲によって分類し、それぞれに含まれる件数を集計する。表 12

に本評価で使用したカテゴリの概要を示す。

表 13 に圧縮非適用時の、各問合せに対する経過時間 (Elapse Time), CPU 時間 (CPU Time), ユーザ CPU 時間 (User Time), システム CPU 時間 (System Time), CPU 使用率 (CPU Usage), ディスク装置からの総読み出し量 (Read Data), 総読み出しスループット (Read Thruput = Read Data / Elapse Time), 非圧縮データ量 (Uncompressed Data), 非圧縮データのスループット (Uncompressed Thruput = Uncompressed Data / Elapse Time) を示す。なお、CPU 時間 (CPU Time), ユーザ CPU 時間 (User Time), システム CPU 時間 (System Time) は 1 論理プロセッサあたりの時間に換算して表示しており、各論理プロセッサの CPU 時間の合計値は表中の値の 8 倍になる。同様に、表 14 には圧縮適用時の値を示す。圧縮非適用時はディスク装置からの読み出し量 (Read Data) と非圧縮データ (Uncompressed Data) は等しくなる。

図 20 に圧縮適用時と圧縮非適用時の経過時間の比較を「圧縮時の経過時間」/「圧縮非適用時の経過時間」によって示す。圧縮の適用により経過時間がおおむね 5~23% に短縮され、圧縮による処理速度の向上が実現されたことが確認された。

図 21 に非圧縮時の CPU 時間を基準とした相対 CPU 時間およびその内訳の比較を示す。図 21 は、圧縮 AQL ではユーザ CPU 時間が増加する反面、システム CPU 時間が減少することを示している。ユーザ CPU 時間の増加は主に伸張処理の負荷によるものであり、一方、システム CPU 時間の減少は入出力量の減少によるものと推定される。

図 22 にストレージからのデータ転送量の比較を示す。圧縮 AQL では問合せに応じて必要なデータストリームに含まれるブロックを選択的に読み出し、また、図 12~図 14 に示したように各データストリームの圧縮率には大きな偏りがある。このため、表 6 に示した静的な圧縮率は必ずしもデータ転送量の削減率を表さない。評価対象とした問合せでは、ストレージからの

表 11 問合せ文の詳細

Table 11 The details of the queries.

Query	問合せ文の内容
query-1	<pre>SELECT KEY1, KEY2, KEY3, KEY4, SUM(Column52) FROM DATA1 WHERE Column33 IN ('K3ZN6053') GROUP BY CATEGORY(Column51 TO CAT1.Level1) AS KEY1, CATEGORY(Column51 TO CAT1.Level2) AS KEY2, CATEGORY(Column51 TO CAT1.Level3) AS KEY3, CATEGORY(Column51 TO CAT1.Level4) AS KEY4</pre>
query-2	<pre>SELECT Column6, Column9, Column33, Column37, Column52, Column41, Column7, Column38, Column31 FROM DATA1 WHERE Column33 IN ('K3ZN6053') AND Column51 = '0001' ORDER BY 1,2,5,6,7,8,9</pre>
query-3	<pre>SELECT KEY1, KEY2, KEY3, COUNT(*) FROM (SELECT Column5, Column6, Column20, CAST(MAX(Column56 Column57) AS INT(32)), COUNT(*), CAST(SUM(Column65) AS INT(32)) FROM DATA2 WHERE (Column56 Column57) BETWEEN '19960211' AND '19980210' AND Column52 = '001' GROUP BY Column5, Column6, Column20) AS T(COL1,COL2,COL3,COL4,COL5,COL6) GROUP BY CATEGORY(COL4 TO CAT21.Level1 ROUND DOWN) AS KEY1, CATEGORY(COL5 TO CAT22.Level1 ROUND DOWN) AS KEY2, CATEGORY(COL6 TO CAT23.Level1 ROUND DOWN) AS KEY3</pre>
query-4	<pre>SELECT Column5, Column6, Column14, Column28, Column33, Column17,Column18, Column19, Column20, Column65 FROM DATA2 WHERE (Column56 Column57) = '19960711' AND Column52 = '001' ORDER BY 1, 2, 10</pre>
query-5	<pre>SELECT Column19, Column22, KEY1, COUNT(*) FROM DATA3 WHERE Column7 = 'LB' AND Column4 = '10' GROUP BY Column19, Column22, CATEGORY(Column8 TO CAT3.Level1) AS KEY1 WITH ROLLUP</pre>
query-6	<pre>SELECT Column4, Column13, Column17, Column19, Column20, Column16, Column15, Column18, Column7, Column8, Column10, Column9, Column11, Column12, Column5, Column6, Column3, Column14 FROM DATA3 WHERE Column4 = '10' AND Column14 = '0807' AND Column19 = '0875' AND Column20 = '1341' AND Column5 = '1996' AND Column6 = '01' ORDER BY 1,2</pre>

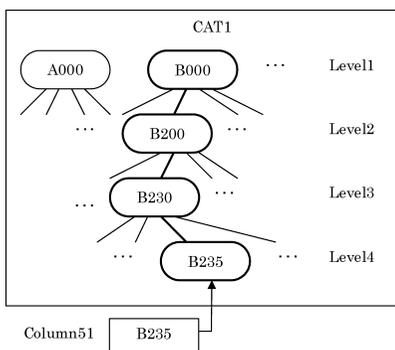


図 19 AQL のカテゴリの例

Fig. 19 An example of the category in AQL.

表 12 評価に使用したカテゴリ

Table 12 The categories used in the evaluation.

カテゴリ名称	階層	主要素	型	要素数	関連するQuery
CAT1	1	Level1	CHAR(4)	4	query-1
	2	Level2	CHAR(4)	11	
	3	Level3	CHAR(4)	22	
	4	Level4	CHAR(4)	77	
CAT21	1	Level1	INT(32)	5	query-3
CAT22	1	Level1	INT(32)	5	
CAT23	1	Level1	INT(32)	5	
CAT3	1	Level1	CHAR(3)	8	query-5

表 13 圧縮非適用時の問合せ性能

Table 13 Query performance for the uncompressed table.

Query	Elapse Time [sec]	CPU Time [sec]	User Time [sec]	System Time [sec]	CPU Usage [%]	Read Data [MBytes]	Read Thruput [MBytes/sec]	Umcompressed Data [MBytes]	Umcompressed Thruput [MBytes/sec]
query-1	18.56	1.24	0.70	0.54	6.67	3875	209	3875	209
query-2	48.50	2.76	1.46	1.30	5.68	10884	224	10884	224
query-3	79.55	9.86	7.76	2.10	12.39	18600	234	18600	234
query-4	186.70	10.08	5.92	4.16	5.40	44971	241	44971	241
query-5	24.47	1.33	0.67	0.66	5.45	5600	229	5600	229
query-6	165.88	6.99	3.66	3.33	4.22	41600	251	41600	251

表 14 圧縮非適用時の問合せ性能

Table 14 Query performance for the uncompressed table.

Query	Elapse Time [sec]	CPU Time [sec]	User Time [sec]	System Time [sec]	CPU Usage [%]	Read Data [MBytes]	Read Thruput [MBytes/sec]	Umcompressed Data [MBytes]	Umcompressed Thruput [MBytes/sec]
query-1	4.30	1.62	1.25	0.38	37.73	859	200	3875	902
query-2	6.08	2.95	2.26	0.69	48.51	972	160	10884	1791
query-3	12.80	11.58	11.05	0.53	90.46	786	61	18600	1454
query-4	8.72	7.41	6.90	0.51	84.95	862	99	44971	5158
query-5	3.66	2.12	1.91	0.21	58.09	742	203	5600	1531
query-6	32.19	11.63	10.21	1.43	36.15	6353	197	41600	1292

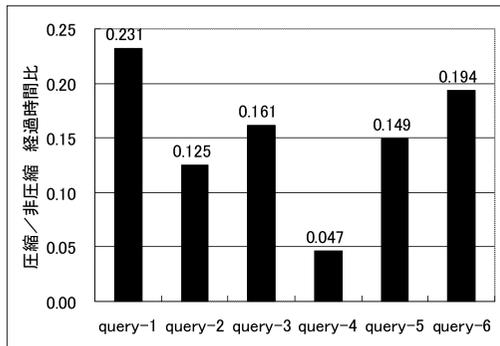


図 20 経過時間の比較

Fig. 20 Comparison of the elapsed time.

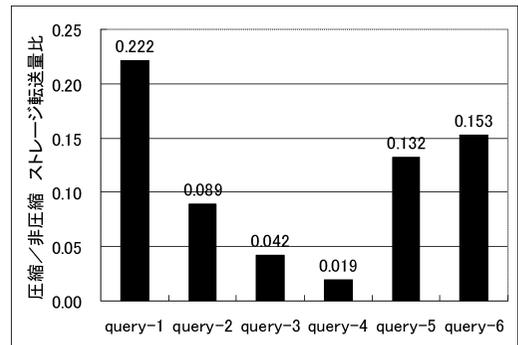


図 22 データ転送量の比較

Fig. 22 Comparison of the amount of transferred data.

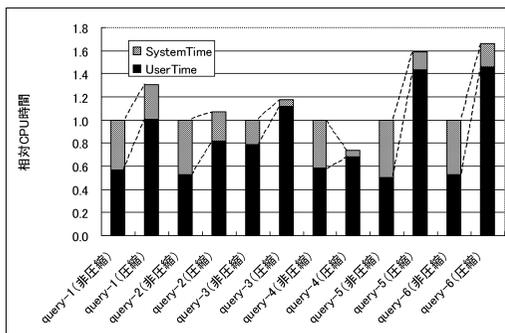


図 21 CPU 時間の比較

Fig. 21 Comparison of the CPU time.

データ転送量が 2~20%程度に削減されていることが示されている。

表 13 より、圧縮非適用時の CPU 負荷はおおむね 4~12%程度にとどまっており、また、総読み出しスループット (Read Thruput) は、おおむね 200~250 [Mbytes/sec] に達しており、いずれの場合もストレージからのデータ転送がボトルネックになっている。

表 14 より、圧縮適用時においてもデータ転送量削減の効果が低い query-1, 2, 5, 6 については依然としてストレージ読み出しが性能のボトルネックとなっており、ほぼストレージからの転送量の比率と同程度の比率で経過時間が短縮されている。一方、データ転送量削減の効果の高い query-3, query-4 では、それ

までボトルネックとなっていたストレージからのデータ転送量の削減の結果、性能のボトルネックが CPU 処理に移り、CPU 負荷が 90～85%程度に上がっている。このため、データ転送量の削減比率ほどに経過時間が短縮されていない。

このように、対象とするデータや処理内容によってボトルネックは異なるが、いずれの場合もストレージからのデータ転送量の削減による経過時間の短縮が実現されている。

6. む す び

本稿ではデータウェアハウス等の大規模データ処理に適したデータ圧縮伸張方式を提案した。また、これを既存のデータベース管理システムに適用して圧縮データベース管理システムを構築し、評価によってその有効性を確認した。

本稿の評価結果は、行方向の履歴の利用により、比較的単純な符号化の組合せにより大半のデータストリーム(列)に対して高い圧縮効果が得られることを示している。しかし一部のデータストリームに対しては必ずしも十分な圧縮効果が得られておらず、少数の圧縮データストリームが圧縮後のデータサイズの大半を占める傾向が見られる。これらのデータストリームに対する対策は今後の課題である。

本稿の評価結果も示すように、データ圧縮の適用効果は対象とするデータに大きく依存する。近年セキュリティ対策へのニーズの高まりにとともに、ネットワーク機器、セキュリティ対策ソフトウェア、サーバ等から生成される大量のログの保存が行われるようになり、ストレージコスト削減と速度性能の向上が課題となりつつある。今後はこのような分野に対するデータ圧縮の適用についても取り組みを進めていきたい。

謝辞 本研究の一部は、情報処理振興事業協会「先端的情報化推進基盤整備事業」による。

参 考 文 献

- 1) School of Information Management and Systems at the University of California at Berkeley: How Much Information? 2003.
<http://www.sims.berkeley.edu/how-much-info-2003>
- 2) Greg Papadopoulos: Future of Computing, *NOW Workshop*, Lake Tahoe CA, USA 27 (1997).
- 3) Becla, J., Daniel, L. and Wang, D.L.: Lessons Learned from Managing a Petabyte, *CIDR 2005*, pp.70-83 (2005).

- 4) Gray, J. and Shenoy, P.: Rules of Thumb in Data Engineering, Microsoft Technical Report, MS-TR-99-100 (2000).
- 5) Poess, M. and Potapov, D.: Data Compression in Oracle, *Proc. 29th VLDB conf.*, pp.937-947 (2003).
- 6) Iyer, B.R. and Wilhite, D.: Data Compression Support in Databases, *Proc. 20th VLDB conf.*, pp.695-704 (1994).
- 7) Alsberg, P.A.: Space and Time Savings Through Large Data Base Compression and Dynamic Restructuring, *Proc. IEEE*, Vol.63, No.8, pp.1114-1122 (1975).
- 8) Bassiouni, M.A.: Data Compression in Scientific and Statistical Databases, *IEEE Trans. Softw. Eng.*, Vol.SE-11, No.10 (1985).
- 9) Cormack, G.V.: Data Compression on a Database System, *CACM*, Vol.28, No.12, pp.1336-1342 (1985).
- 10) Roth, M.A. and Ban Horn, S.J.: Database Compression, *SIGMOD RECORD*, Vol.22, No.3, pp.31-39 (1993).
- 11) Ng, W.K. and Ravishankar, C.V.: Block-Oriented Compression Techniques for Large Statistical Databases, *IEEE Trans. Knowledge and Data Engineering*, Vol.9, No.2, pp.314-328 (1997).
- 12) Cockshott, W.P., McGregor, D. and Wilson, J.: High-Performance Operations Using a Compressed Database Architecture, *The Comput. J.*, Vol.41, No.5, pp.283-296 (1998).
- 13) Morri, M.: Teradata Multi-Value Compression V2R5.0, Teradata Whitepaper (2002).
- 14) Goldstein, J., Ramakrishnan, R. and Shaft, U.: Compressing Relations and Indexes, *Proc. ICDE*, Orlando, FL, USA (1998).
- 15) Graefe, G. and Shapiro, L.D.: Data compression and Database Performance, *Proc. ACM/IEEE-CS Symp. on Applied Computing*, Kansas City, MO (1991).
- 16) Westmann, T., Kossmann, D., Helmer, S. and Moerkotte, G.: The Implementation and Performance of Compressed Databases, *SIGMOD Rec.*, Vol.29, No.3, pp.55-67 (2000).
- 17) Ziv, J. and Lempel, A.: A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Inf. Theory*, Vol.23, No.3, pp.337-343 (1977).
- 18) Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-rate Coding, *IEEE Trans. Inf. Theory*, Vol.24, No.5, pp.530-536 (1978).
- 19) Lelewer, D.A. and Hirschberg, D.S.: Data Compression, *Comput. Surv.*, Vol.19, No.3, pp.261-296 (1987).

- 20) 鶴木昌行：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 (1997).
- 21) 上田尚純，郡 光則，青野正宏，渡辺 尚，水野忠則：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，情報処理学会論文誌：データベース，Vol.42, No.SIG 10 (TOD 11), pp.64-78 (2001).
- 22) 郡 光則，山岸義徳，清水英弘，金子洋介：検索機能を備えたストレージシステムによる大規模並列全文検索，信学技報，CPSY-2002-47 (2002).
- 23) 佐藤重雄，早川孝之：集計処理並列化に関する評価，情報処理学会全国大会第 68 回，5D-2 (2006).
- 24) 清水英弘，郡 光則：スケーラブルインテリジェントストレージアーキテクチャによる並列データウェアハウス，情報処理学会全国大会第 68 回，5D-1 (2006).
- 25) 郡 光則，佐藤重雄，鹿島理華，高山茂伸：大規模データ処理向け超並列可逆データ圧縮伸張処理技術の開発，2001 年度 IPA 先端的情報化推進基盤整備事業報告 (2002).
- 26) <http://www2.mdit.co.jp/service/diaprism/>
- 27) <http://www.gzip.org/>
- 28) 三菱電機インフォメーションテクノロジー：DIAPRISM ビジネスインテリジェントスイート構築運用ガイド．
- 29) 三菱電機インフォメーションテクノロジー：DIAPRISM ビジネスインテリジェントスイートユーティリティガイド．

(平成 18 年 3 月 20 日受付)

(平成 18 年 7 月 19 日採録)

(担当編集委員 大森 匡)



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，情報検索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．