

ファイル構造を意識した重複排除による データ格納容量の削減

早坂光雄†

企業内のデータ量は年々増加の一途をたどっており、こうした莫大なデータのバックアップを更に複数世代とるには、膨大なストレージ容量が必要になる。そこで注目されているのが重複排除と圧縮技術を組み合わせた容量最適化技術であり、ストレージ容量を大幅に削減することが可能である。バックアップでは各ファイルを1つのファイルにまとめたアーカイブファイルとしてファイルが保存される。このファイルに通常適用される Content-Defined Chunking を適用しても、重複排除率の向上を抑制してしまう。本論文では、アーカイブファイルの構造を意識したデータ分割による重複排除方式を提案し、提案方式がアーカイブファイルだけでなく圧縮ファイル形式のファイルにも有効に働き、容量を大幅に削減することを示した。

Data Reduction Using De-duplication Technique Considering Backup File Structure

MITSUO HAYASAKA†

Recently, the data volume in a company rapidly increases every year. A huge storage capacity is needed to take two or more generations of backup for such immense data. One of the ways to solve this is a capacity optimization technology, which combine de-duplication with compression techniques. It is possible to dramatically reduce storage capacity using it. In backup systems, a file is saved as an archive file which is a collection of many files. Even if Content-Defined Chunking that is usually used in a de-duplication is applied to it, an improvement in a de-duplication ratio will be limited. In this paper, a de-duplication considering the archive file structure is proposed. It can reduce data capacity of compress data format files as well as the archive files. The results of performance evaluations show that the proposed method is effective to dramatically reduce the storage capacity in backup systems.

1. はじめに

企業内のデータ量は年々増加の一途をたどっている。世界のデータ量が2年ごとに倍増するというデータ¹⁾もあり、こうした莫大なデータのバックアップを更に複数世代とるには、膨大なストレージ容量が必要になる。それに伴いITコストも増加するため、バックアップによるデータ保護を行いつつストレージ容量削減によるITコストの短縮が求められている。

そこで注目されているのが重複排除と圧縮技術を組み合わせた容量最適化ファイルストレージである。重複排除とは、ファイルの全部または一部のデータが既に格納済みデータと一致している場合、その部分は同じであると識別し、インデックスを付けて、同じデータ

を2度格納せずに容量削減を実現する技術である。見方を変えれば、圧縮が1ファイル内のデータ重複を排除する技術であり、重複排除は複数ファイル間のデータ重複を排除する技術であるといえる。

バックアップは、フルバックアップ、差分バックアップ、増分バックアップの3つに分類することができる²⁾。フルバックアップは必要なデータ全てを一度にまとめて一括に複製し、差分バックアップは前回のフルバックアップ時からの変更/追加されたデータのみを複製する。増分バックアップは前回のフルバックアップ時からの変更/追加されたデータのみを複製するが、次回増分バックアップを行う際は直前の増分バックアップの変更/追加分だけが複製される。いずれのバックアップ手法でも、ファイル単位でバックアップされるため、ファイルを小さい単位で分割したチャンク単位で見れば、重複が多い。つまり、バックアップ世代内のファイルの重複排除だけでなく、バックアップ世代

† 日立製作所 横浜研究所
Hitachi Ltd. Yokohama Research Laboratory

間の重複排除の効果が高いことが期待される。

更に、容量最適化によるストレージ保存容量の削減により、省電力を実現できるため、グリーン IT の推進にもつながる。

本論文では、重複排除率 R_{dedupe} を (1)、容量削減率 R_{reduce} を (2) でそれぞれ定義する。

$$R_{dedupe} = \frac{S_{ori} - (S_{meta} + S_{dedupe})}{S_{ori}} \quad (1)$$

$$R_{reduce} = \frac{S_{ori} - (S_{meta} + S_{dedupeCompress})}{S_{ori}} \quad (2)$$

S_{ori} : オリジナルデータサイズ, S_{dedupe} : 重複排除後データサイズ, S_{meta} : 重複排除用メタデータサイズ, $S_{dedupeCompress}$: 重複排除処理後に圧縮を適用したデータサイズを示す。重複排除率は、純粋な重複排除処理により削減された割合である。容量削減率は、圧縮処理適用後に削減された容量削減の割合である。一般的には、重複排除率がそのまま容量削減率を指していることが多いが、本稿ではこの 2 つの評価指標を識別し、詳細な評価を行う。

重複排除処理は、a) データ分割, b) 分割データの重複判定処理から構成される。データ分割処理で一般的に用いられるものは、Content-Defined Chunking (CDC)^{3),4)} である。これは、常に同じデータパターンが出現したところを分割点とすることで、重複排除率を向上させることが可能である。重複判定処理では、Hash 比較またはバイナリ比較により、その分割データが既に存在するかどうかを判定し、存在しない場合のみ、格納する。通常、バイナリ比較より Hash 比較の方が重複判定性能を向上させることができる。使用する Hash として SHA1, SHA256, SHA384, SHA512, MD5 などが考えられるが、本稿では SHA1⁵⁾ を用いて評価を行う。

バックアップファイルのファイルサイズに焦点を当てると、ほとんどが 1GB 以上の大きなファイル群である⁶⁾。なぜなら、バックアップファイルは複数のファイルを 1 つのファイルに集めたアーカイブファイルだからである。そこで、このアーカイブファイルの構造を意識したデータ分割方式による重複排除を適用すれば、重複排除率を向上させ、ストレージ容量を大きく削減させることが可能になる。

本論文の構成を以下に示す。2 章で重複排除におけるデータ分割方式を述べる。3 章で提案するバックアップファイル構造を意識したデータ分割方式を詳述する。4 章で特性評価による提案方式の有効性について述べ、5 章で関連研究を示す。最後に、6 章で本論文の結論と今後の課題について述べる。

例) Window サイズが 3Byte の場合
Window は右へずれていく

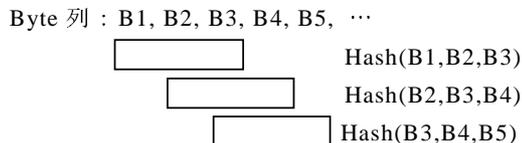


図 1 ローリングハッシュ
Fig. 1 Rolling Hash

2. データ分割方式

データ分割方法は、重複排除処理において最も重要な処理の 1 つであり、重複排除率に大きく影響する部分である。

2.1 データ分割の種類

以下にデータ分割の種類を示す。

- (1) Single Instance
ファイル単位で重複判定を行う。
- (2) 固定長分割
ファイルを固定長のデータに分割し、この分割データ単位で重複判定を行う。分割されたデータをチャンクと呼ぶ。
- (3) 可変長分割
ファイルを可変長のチャンクに分割し、チャンク単位で重複判定を行う。

Single Instance は、同一ファイルのコピーのみ重複排除可能である。固定長分割は、Single Instance と比較して重複排除率を向上させることが可能であるが、ファイルの編集 (データの追加・更新・削除) によるデータずれに対応できない。可変長分割は CDC と呼ばれる方式であり、同じデータパターンが出現した点のみをデータ分割点とするため、ファイルの編集によるずれを吸収し、重複排除率を向上させることができる。図 1 に示すように、データ分割点の検出には、固定サイズの Window 内のデータに対する特徴量 (Hash 値) を逐次求めていく Rolling Hash が用いられる。

Rolling Hash の Sliding Window 内の Hash 値を求め、その Hash 値があらかじめ決めておいた特定の値になったら、そのポイントを分割点とするのが CDC である。Window 内が同一のデータパターンにならない限り、特定の Hash 値にはならないため、データの編集によるバイト列のずれにも対処することができる。

Rabin fingerprint は、Rolling Hash を非常に軽量

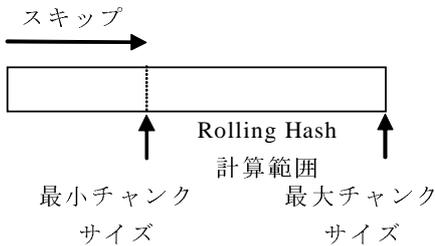


図 2 TTTD によるチャンク分割
Fig.2 Chunk division using TTTD

な計算で求めるアルゴリズムである⁷⁾。図 1 において、B2, B3, B4 の Hash 値を、直前に求めた B1, B2, B3 の Hash 値から求めることにより高速化を行う。具体的には、B2, B3 は既に計算済みの部分であるため、 $\text{Hash}(B2, B3, B4) = \text{Hash}(B1, B2, B3) - (B1 \text{ の要素}) + (B4 \text{ の要素})$ で求める。Window を Sliding させながら同様のことを繰り返し、帰納法的に計算していく。

CDC は通常、Two Threshold Two Divisor (TTTD)⁸⁾ により、チャンク分割を行う。図 2 に示すように、最小チャンクサイズと最大チャンクサイズを決め、バイト列の先頭から最小チャンクサイズまでスキップし、そこから最大チャンクサイズまでを Rolling Hash の計算範囲とする。TTTD は可変長チャンクのサイズの分散を小さく抑えることができる。

2.2 圧縮ファイル向けデータ分割方式

圧縮ファイルは、オリジナルのデータを圧縮方式に従った別のデータ形式に変更して保存する。そのため、CDC を適用しても重複排除による容量削減効果が抑制されてしまうことが報告されている⁹⁾。そこで、圧縮データを伸長してから重複排除処理を適用する方式が提案されているが、伸長処理に CPU 負荷と処理時間がかかってしまう。また、ZIP 圧縮ファイルについて、伸長処理を行わずに ZIP ヘッダと ZIP ボディ毎に重複排除を適用することにより、重複排除率を向上させている⁹⁾。図 3 に ZIP ファイルの構造を示す。ZIP ファイルは、圧縮ヘッダと圧縮ボディから構成されており、圧縮ボディに格納されたデータの変更は、他の圧縮ボディに影響は無く独立している。そのため、圧縮ヘッダと圧縮ボディ毎にデータ分割し、その単位で重複排除処理を行うことにより、その効果を向上させることが可能である。

更に、Office 2007 以降のファイルはこの ZIP コンテナ方式で保存されている⁹⁾。そのため、本方式を適



CH: Compress Header
CB: Compress Body

図 3 ZIP 圧縮ファイル構造
Fig.3 ZIP file structure

■ 通常ファイル



■ アーカイブファイル



AH: Archive Header
AB: Archive Body

図 4 ファイル構造
Fig.4 File structure

用することにより、バックアップファイルの中でも大きな割合を占める Office Document に対して⁹⁾、効果的な重複排除を適用することができる。

3. バックアップファイル構造を意識したデータ分割方式

図 4 に示すように、ファイルは、通常、ヘッダとボディ、トレイラから構成される。ヘッダは、ファイルに関するメタデータ情報を格納している領域であり、ファイルタイプの識別を行う magic number やボディの長さなどを管理している。ボディはデータの実態が格納されている領域である。トレイラは、ファイル終端を示すための情報である。

また、上記とは異なる構造として、バックアップファイルとして用いられるアーカイブファイルが挙げられる。アーカイブファイルは複数のファイルをまとめて 1 つの大きなファイルにしたものであり、アーカイブヘッダとアーカイブボディ（ここに各ファイルが格納されている）を 1 組とした構造が、ファイル数分だけ繰り返されていく。

3.1 ファイル構造を意識したデータ分割

ヘッダ情報にはファイル名や更新日時等の情報が含まれるものもあるため、図 4 の示したファイルに、CDC をそのまま適用すると、重複排除率が劣化する可能性が高い。そこで ZIP 圧縮ファイルに対するデー

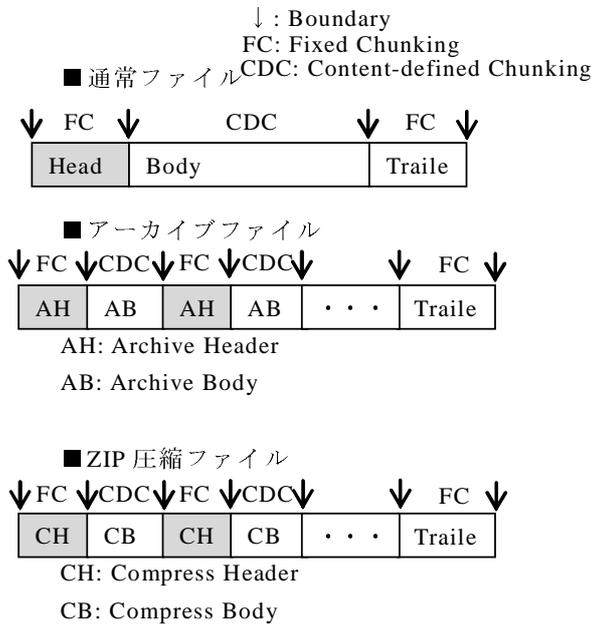


図 5 データ分割
Fig. 5 Data division

データ分割方式を応用し、その他のファイルに対してファイル構造を意識したデータ分割を行う。図 5 に本方式におけるデータ分割方法を示す。

まず、通常ファイルでは、ヘッダとボディ、トレイラの各境界を認識する。ヘッダにおいて magic number からファイルタイプを識別しヘッダサイズを取得する。そして、ヘッダの解析を行うことによりボディの長さを取得し、残った部分をトレイラと判断する。ヘッダとトレイラは固定長分割を適用し、ボディ部は CDC を適用する。ヘッダは各ファイルの固有の情報を持っており、CDC を適用しても重複排除されにくいと考えられる、トレイラはファイル終端を示す決まったパターンのデータであり CDC を適用しなくても重複排除を見込むことができる。そこで、両方とも固定長分割を適用し、CPU 処理を軽減する。

同様に、アーカイブファイルでは、アーカイブファイルヘッダとアーカイブボディ、トレイラの境界を認識し、ヘッダとトレイラには固定長分割を、Body 部には CDC を適用する。

また、最小チャンクサイズ以下の小さいファイルは、Single Instance を適用する。小さいファイルに対して、データ分割を行い、重複排除用のメタデータを保持すると、メタデータの保持量に対して重複があまり

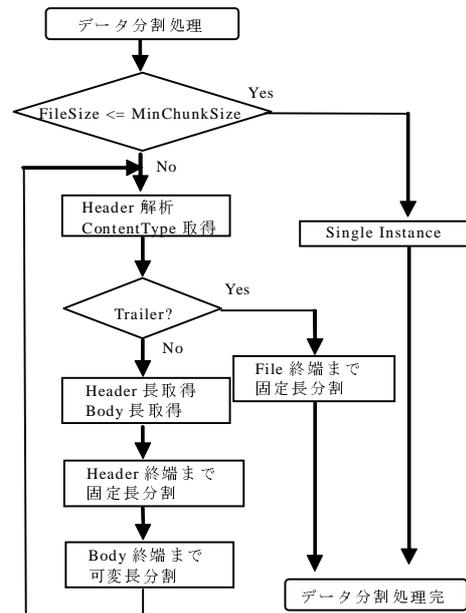


図 6 データ分割処理
Fig. 6 Data division flow

効かず、データ量が削減されない結果になる。そこで、最初にファイルサイズが最小チャンクサイズ以下かどうかを判定し、その後にファイル構造を意識したデータ分割を適用する。

この時の処理手順を図 6 に示す。

3.2 アーカイブファイル向けデータ分割

アーカイブファイルは、複数のファイルを 1 つのファイルにまとめたものであり、バックアップでは本ファイル形式を用いる。アーカイブファイルのヘッダとボディ・トレイラを認識してデータ分割を行ってもボディ部に格納されたファイルに CDC を適用するだけでは重複排除率が劣化する可能性が高い。なぜなら、ボディ部のファイルも同様にヘッダ・ボディ・トレイラに分割できるからであり、2.2 で述べた CDC の課題がそのまま再現される。そこで、アーカイブファイルそのものだけでなく、アーカイブボディ部に格納されたデータについてもファイル構造を意識したデータ分割を適用する方式を提案する。図 7 に示すように、アーカイブファイルヘッダの処理の後、引き続きアーカイブボディ部にもヘッダ解析処理を行い、図 6 と同様の処理を適用するものである。アーカイブボディ部の処理が終わると再度アーカイブヘッダの処理を行い、以後、この処理を繰り返していく。

また、アーカイブボディ部に格納されたファイルが、

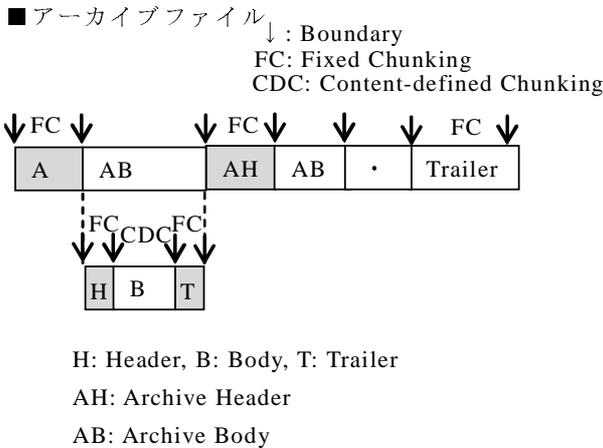


図 7 アーカイブファイル向けデータ分割処理
Fig.7 Data division for archive file

アーカイブファイルだった場合が考えられる。ここで説明を簡単にするため、バックアップファイルとして作成されたアーカイブファイルを 1 段目、アーカイブされたファイルがアーカイブファイルだった場合を 2 段目とする。この場合にも、本処理を適用し、2 段目の解析によって 3 段目があった場合にも再帰的に適用していく。しかし、実際には、再帰的な処理が深くなると、そのファイルに編集が加えられている可能性は低く CDC で十分である。そこで、どこまで再帰的にアーカイブ向けデータ分割処理を行うかを指定するパラメータにより制限し、処理の軽減を図る。

アーカイブファイル向けデータ分割処理の手順を図 8 に示す。図 6 と比較して、異なるのは★をつけた部分の処理である。アーカイブファイル判定後のボディ部の処理で、アーカイブボディ長を指定して、再度図 8 の処理を再帰処理制限まで繰り返していく。

4. 特性評価

提案したアーカイブファイル向けデータ分割方式の特性評価を行う。提案方式と従来方式について、重複排除率および容量削減率を評価する。ここで、比較する従来方式はアーカイブファイルのファイル構造のみを認識してデータ分割する方式とし、アーカイブされた各ファイルには可変長分割が適用されるものとした。

4.1 評価パラメータ

重複排除の評価に用いるべきファイルは、バックアップ世代毎に重複部分と変更部分が存在し、それが複数世代に渡って新規ファイルの追加を伴いながら変わっていくものである。Microsoft Office や Open Office

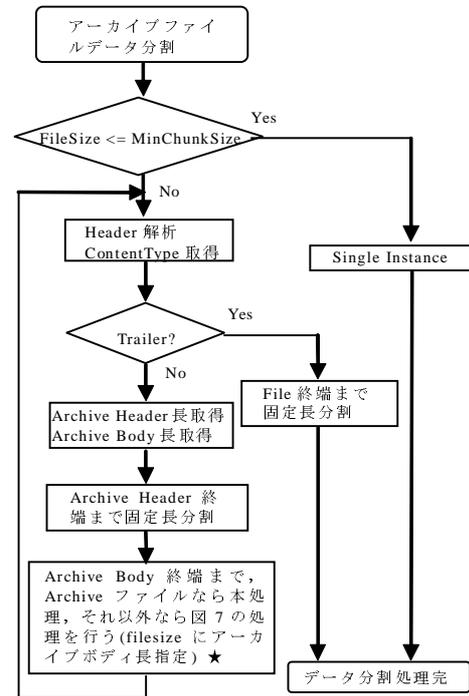


図 8 アーカイブファイル向けデータ分割処理手順
Fig.8 Data division flow for archive file

のファイルに、意図的に修正を加えた別のファイルで、それぞれの重複排除割合を示しているものがある⁹⁾が、故意に都合の良いデータを作成可能であるため、客観性と再現性の実現が困難である。そこで、一般的にも広く使用されている GNU Compiler Collection (GCC)¹⁰⁾ のソースを利用して評価を行う。これらのソースは、複数のバージョンが存在し、更にこれらのファイルを ZIP 形式に変換したものについても評価することで、Microsoft Office 形式で複数世代に渡りバックアップされたデータの特性を模擬することが可能になる。

表 1 に評価ファイルの形式を示す。それぞれ gcc-4.5.0~gcc-4.6.2 までのソースをバージョン毎に tar 形式、zip 形式、gz 形式に変換したものがバックアップされ、tar+tar or tar+zip or tar+gz になるものとした。これにより、提案方式のバックアップファイル向け分割方式の有効性を評価する。

また、本評価では gcc-4.5.0 のファイルから順番に 1 バージョンずつ重複排除処理を適用し、gcc-4.6.2 まで評価した時の重複排除率および容量削減率を用いて比較を行う。

表 2 に、可変長分割で用いる TTTD のパラメータ

表 1 評価ファイル

Table 1 Files for Evaluations

ファイル形式	評価ファイル
TAR+TAR	gcc-4.5.0.tar.tar ~ gcc-4.6.2.tar.tar
TAR+ZIP	gcc-4.5.0.zip.tar ~ gcc-4.6.2.zip.tar
TAR+gz	gcc-4.5.0.gz.tar ~ gcc-4.6.2.gz.tar

表 2 TTTD パラメータ

Table 2 TTTD parameters

最小チャンクサイズ	最大チャンクサイズ	計算範囲
4KB	12KB	8KB
16KB	48KB	32KB

表 3 実験環境

Table 3 experimental environment

項目	内容
CPU	Intel Core i5 2.67GHz
OS	Linux Fedora15
Kernel	2.6.42
Memory	8GB

を示す。ここでは、チャンクサイズの変化に対する特性を評価するため、チャンクサイズの異なる 2 つのタイプを用いた。最小チャンクサイズ 4KB までスキップし、そこから 8KB の計算範囲において CDC を適用するものと、16KB までスキップし、そこから 32KB の計算範囲で CDC を適用するものである。提案方式では、ボディ部にこの TTTD が適用され、可変長チャンク分割が行われる。

データ分割されたチャンクは圧縮される。圧縮の方法として、gzip¹¹⁾, lzo¹²⁾, snappy¹³⁾ などが考えられる。本評価では、軽量に圧縮を実現できる snappy を用いて評価を行い、圧縮前・圧縮後のサイズで小さい方を格納するものとした。

実験環境を表 3 に示す。

4.2 TAR+TAR フォーマットの評価結果

TAR フォーマットにおける重複排除率と容量削減率を図 9 に示す。全体の傾向として、バージョンを 1 つずつあげて重複排除処理を適用することにより、重複排除率および容量削減率が向上し、実格納データを大幅に減少させている。gcc-4.6.0.tar.tar を格納した時に上記割合が劣化しているのは、gcc-4.5 系と gcc-4.6 系でソースファイルの変更が多く、重複部分が小さくなったためである。

図 9 において、容量削減率は重複排除率よりも大きい割合を示し、容量を大きく削減している。これは圧縮によりデータ量が削減されたためである。また、従来方式と比較して提案方式が、排除率および容量削減率を向上させている。データ構造を意識したデータ分

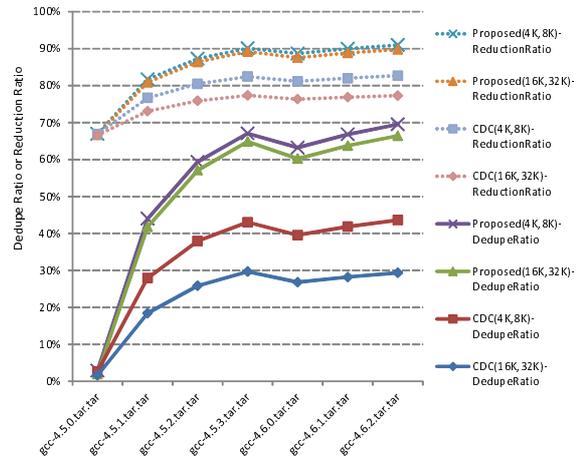


図 9 TAR+TAR における重複排除率・容量削減率

Fig. 9 Dedupe ratio and reduction ratio for TAR format file archived with TAR

割方式が有効に働き、重複部分の検出頻度が多くなったためである。

チャンクサイズに着目すると、チャンクサイズを小さくしたものが排除率を向上させている。チャンクサイズの差に対する各方式の差分は、重複排除率では大きく、容量削減率では小さくなっている。これは、圧縮は小さいデータサイズよりも大きいサイズで処理すると効果が高いため、チャンクサイズが大きい方が効果的に圧縮が働いたことを示している。

4.3 TAR+ZIP フォーマットの評価結果

ZIP フォーマットにおける重複排除率と容量削減率を図 10 に示す。重複排除率 < 容量削減率となる関係は TAR フォーマットの結果と同じであるが、TAR と比較すると全ての割合が劣化している。これは、既に圧縮しランダムに近いデータになったものに、データ分割処理を適用しているためである。特に顕著なのが、従来方式の重複排除率と容量削減率の差分であり、圧縮されたデータに対して圧縮を適用しようとしているため、圧縮の効果が非常に小さくなっている。対して、提案方式は重複排除率を大きく向上させている。アーカイブされた ZIP ファイルにも、ZIP 圧縮形式向けのデータ分割方式が有効に働いたためである。また、提案方式においてチャンクサイズの違いはなく同じ排除率となっている。これは、既に圧縮されて小さいサイズになったファイルに、提案分割方式を適用しているため、両方式で分割点の違いが出なかったためである。

4.4 TAR+gz フォーマットの評価結果

gz ファイルを TAR でまとめたフォーマットに対す

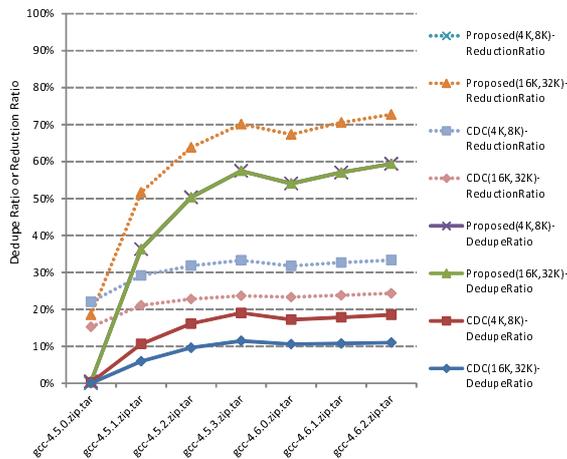


図 10 TAR+ZIP における重複排除率・容量削減率
Fig. 10 Dedupe ratio and reduction ratio for ZIP format file archived with TAR

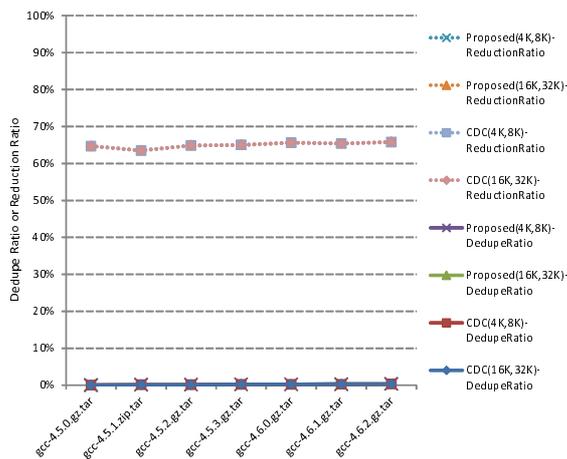


図 11 TAR+gz における重複排除率・容量削減率
Fig. 11 Dedupe ratio and reduction ratio for gz format file archived with TAR

る評価結果を図 11 に示す。全ての方式で、重複排除の効果がほぼない。そのため、容量削減効果はなく、圧縮処理のみの効果になっている。これは gz で圧縮されたファイルは、部分的な修正が gz ファイル全体のバイト列の変更に影響していることを示している。すなわち、gz 圧縮ファイルは、一度伸長処理をしてから重複排除処理を適用すると重複排除率を向上させることができる。しかし、圧縮ファイルの伸長には CPU 負荷と処理時間がかかるため、システム管理者がそのまま重複排除を適用するか、伸長してから重複排除を行うかの選択ができるようにしてあるとよい。

4.5 考 察

以上の評価から、提案方式が従来方式と比較してデータ格納容量を大きく削減している。gcc の各バージョンをバックアップ世代と考えると、7 世代で TAR フォーマットについては従来方式よりも最大で 35% 強の容量削減率の向上に成功し、ZIP フォーマットについては、最大で約 50% の容量削減率の向上を達成している。

5. 関連研究

データ分割によるチャンクベースの重複排除では、様々な方式が提案されている。固定長分割の重複排除によって容量を削減するシステム^{14),15)}では、固定長分割のためファイルの編集によるバイトずれに対応できず、重複排除率を劣化させている。ネットワーク帯域を節約するためのネットワークファイルシステム³⁾で CDC が適用され、また、ファイル容量を削減する目的でも CDC が適用されている^{4),16),17)}。特にバックアップシステムでは、複数のファイルが 1 つのファイルにまとめられたアーカイブファイルで保存される。関連研究では以下の点について検討されたものはない。

- (1) アーカイブファイルに特化したデータ分割方式
- (2) 分割データの特性に応じてシングルインスタンス、固定長分割、可変長分割を適用し CPU 処理を軽減しつつ重複排除率を向上

6. 結 論

企業内のデータ量は年々増加の一途をたどっており、こうした莫大なデータのバックアップを更に複数世代とるには、膨大なストレージ容量が必要になる。そこで注目されているのが重複排除と圧縮技術を組み合わせた容量最適化技術であり、ストレージ容量を大幅に削減することが可能である。

本論文では、バックアップで用いられるアーカイブファイルに着目し、そのデータ構造を意識したデータ分割による重複排除方式を提案した。特性評価より、提案方式はアーカイブファイルに有効に働き、大きく容量を削減している。

本方式は、バックアップが行われる全ての IT サービス・システムに適用可能であり、ストレージ格納容量の削減に有効である。

参 考 文 献

1) World's Data More Than Doubling Every Two Years, <http://www.emc.com/about/news/press/2011/20110628-01.htm>, EMC Press Re-

- lease, June 2011.
- 2) Backup, <http://en.wikipedia.org/wiki/Backup>
 - 3) A. Muthitacharoen, B. Chen, and D. Mazieres: A low-bandwidth network file system. In Symposium on Operating Systems Principles, pages 174-187 (2001).
 - 4) B. Zhu, K. Li, and H. Patterson: Avoiding the disk bottleneck in the Data Domain deduplications file system. In 6th Usenix Conference on File and Storage Technologies, pages 269-283, Feb 2008.
 - 5) Federal Information Processing Standard (FIPS) 180-3: Secure Hash Standard (SHS). Tech. Rep. 180-3, National Institute of Standards and Technology (NIST), Gaithersburg, MD, Oct 2008.
 - 6) G. Wallace, F. Douglass, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu: Characteristics of Backup Workloads in Production Systems. In Proc. Of 10th USENIX Conference on File and Storage Technologies. (2012).
 - 7) M. O. Rabin: Fingerprinting by random polynomials. Technical Report, Center for Research in Computing Technology, Harvard University, (1981).
 - 8) Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezise, and Peter Camble: Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality, in Proc. of the 7th conference on File and storage technologies (FAST09), Pages 111-123, (2009).
 - 9) Dirk Meister, Andre Brinkmann, Multi-Level Comparison of Data Deduplication in a Backup Scenario, SYSTOR'09, 987-1-60558-623-6/09/05 (2009).
 - 10) GCC, the GNU Compiler Collection, <http://gcc.gnu.org/>
 - 11) gzip, <http://www.gzip.org/>
 - 12) lzo, <http://www.oberhumer.com/opensource/lzo/>
 - 13) snappy, <http://code.google.com/p/snappy/>
 - 14) N. Tolia, M. Kozuch, M. Satyanarayanan, et al: Opportunistic Use of Content Addressable Storage for Distributed File Systems, In Proc. of Usenix 2003 Annual Conference, (2003).
 - 15) S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In Proc. of the 1st USENIX conference on File and Storage Technologies, (2002).
 - 16) N. Jain, M. Dahlin, and R. Tewari. TAPER: tiered approach for eliminating redundancy in replica synchronization. In Proc. of the 4th USENIX conference on File and Storage Technologies, 2005.
 - 17) L. P. Cox, C.D. Murray and B. D. Noble. Pastiche: Making Backup Cheap and Easy, In Proc. of the 5th Symposium on Operating Systems Design and Implementation, pp.285-298, 2002.