情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2012-ARC-200 No.20
Vol.2012-OS-121 No.20
2012/5/8

Content-Defined Chunking を用いた重複除外キャッシュ機構によるGfarmの性能向上

村上 じゅん[1]　石黒　駿[1]　大山 恵弘[1,2]

概要：データインテンシブサイエンスにおけるアプリケーションにはその実行時に重複するデータを含む多くのファイルを生成するものが存在する．Gfarm は，大規模データの効率的な利用を可能にする分散ファイルシステムである．本稿では，Gfarm のための Content-Defined Chunking (CDC) による重複除外キャッシュ機構の設計と実装および，その機構を評価するための実験結果について報告する．CDC はファイルをその中身に基づき可変長のブロック（チャンク）に分割する方式のことである．クライアントはローカルファイルシステム上にチャンクをキャッシュファイルとして保存し，以降のファイルアクセスで再利用する．重複除外の導入により，データ転送量やストレージ消費容量が減少する．本研究の実験を通じて，提案機構によりファイル読み込み性能が向上し，データ転送量とストレージ消費容量が減少したことを確認した．

キーワード：分散ファイルシステム，ファイルキャッシュ，CDC，重複除外

Improving the Performance of Gfarm by a Cache Deduplication Mechanism with Content-Defined Chunking

Jun Murakami[1]　Shun Ishiguro[1]　Yoshihiro Oyama[1,2]

Abstract: Some application programs in data-intensive science create many large files that contain redundant data. Gfarm is a global distributed file system that enables an efficient use of large-scale data. In this paper, we describe the design and implementation of a cache deduplication mechanism with Content-Defined Chunking (CDC) for Gfarm, and report experimental results for evaluating the mechanism. CDC is a method of dividing a file into variable-size blocks (chunks) based on the contents of the file. The client stores the chunks in the local file system as cache files, and reuses them in the following file accesses. Deduplication of chunks reduces the amount of transmitted data and storage consumption. We confirmed through experiments that the proposed mechanism improved the performance of file read and reduced the amount of transmitted data and storage consumption.

Keywords: distributed file system, file cache, CDC, deduplication

1. はじめに

近年，大規模なデータの解析を必要とするアプリケーションのための分散ファイルシステムの研究が盛んに行われている [1], [2], [3], [4]．そのようなファイルシステムの一つである Gfarm [5] は大規模データ解析を支援する分散ファイルシステムで，スケールアウトするファイルシステムの構築を可能にするものである．Gfarm の主要構成要素はファイルのメタデータを管理する単一のメタデータサーバ，ストレージを供給する複数の I/O サーバ及び Gfarm ファイルシステムにアクセスするクライアントである．クライアントは Gfarm が提供するライブラリ API を呼び出すことにより Gfarm ファイルシステム上のデータを読み

[1] 電気通信大学
The University of Electro-Communications
[2] 独立行政法人科学技術振興機構， CREST
JST, CREST

© 2012 Information Processing Society of Japan

書きすることができる．

Gfarm においてはファイルアクセスは以下のように行われる．クライアントはファイルアクセスを行う際，まずメタデータサーバに問い合わせ，当該ファイルを格納するI/O サーバについての情報を得る．その後はメタデータサーバを介さずI/O サーバに対して直接ファイルデータを要求する．そのデータは通常，クライアントのマシンに複製されて再利用されることはない．また，ページキャッシュに載ったとしても通常は利用されない．これはデータの一貫性を保証するためである．そこで一貫性を保証しながらキャッシュを利用できるような仕組みが導入できれば，クライアントへのデータの読み込み性能が向上すると考えられる．

クライアントにキャッシュを作成する仕組みとしてまず考えられるのは，クライアントのローカルファイルに読み込んだファイルを丸ごと保存しておくという方法である．しかし，この方法には問題点が 2 つある．1 つはファイルの全体をローカルファイルに保持するので，保持のために必要なストレージ容量がファイルサイズに等しく大きいことである．もう 1 つは，ファイルデータが更新される度に，保持しているローカルファイルも更新もしくは破棄しなければならず，ファイルの一部が頻繁に更新されるような場合においてはキャッシュの効果が発揮されないことである．

本研究では，Gfarm において，クライアントがファイルデータを可変長のブロック（チャンク）単位でローカルファイルとして保存し，それらを後のアクセスで利用するキャッシュ機構を提案する．ファイルの分割方法としてはCDC を採用した．CDC はファイルをその中身に基づき可変長のチャンクに分割する方式である．このキャッシュ機構には，冗長性がある複数のデータを 1 つにまとめる重複除外処理が実現できるという利点がある．すなわち，データの中身が同一であるような複数のファイル断片に対して，同一のチャンクをキャッシュとして利用することが可能になる．提案する機構により，キャッシュによるローカルストレージの消費量を抑えつつデータ転送量を減少させることができる．さらに，ファイルの中身に基づくキャッシュなので，ファイルが更新された場合におけるキャッシュの更新を最小限に抑えることができる．また，過去にアクセスしたチャンクと中身が同一であるチャンクを含むファイルを転送する際にも，キャッシュが利用できる．

文献 [6] では重複除外の予備評価を行ったが本稿では提案機構導入時の性能評価を示す．

以下，2 章で Gfarm の概要，3 章で CDC を用いた重複除外処理，4 章でシステムの設計と実装，5 章で性能評価，6 章で関連研究についてそれぞれ述べ，最後に 7 章でまとめと今後の課題について述べる．

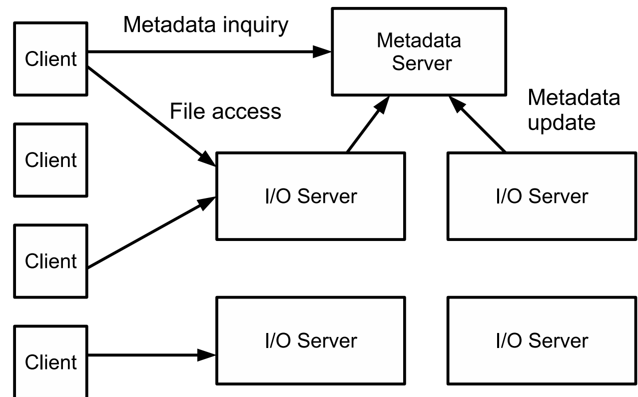

図 1　Gfarm のアーキテクチャ
Fig. 1　The architecture of Gfarm

2. Gfarm

Gfarm は大規模データ解析に利用可能な分散ファイルシステムである．Gfarm のアーキテクチャを**図 1** に示す．Gfarm の主な構成要素はファイルアクセスを行うクライアント，I/O サーバ，メタデータサーバの 3 つである．I/O サーバは Gfarm ファイルシステムにストレージを提供するサーバで，Gfarm ファイルシステム上に複数存在する．メタデータサーバはファイルのメタデータを管理するサーバで，ファイルシステム上に一つだけ存在する．Gfarm ファイルシステムにおけるファイルは，ファイル単位またはファイル断片の形で複数のファイルシステムノード上に分散配置される．クライアントはファイルアクセスを行う際，まずメタデータサーバに問い合わせ，当該ファイルを格納する I/O サーバについての情報を得る．そして得られた I/O サーバの中から 1 つを選択し，ファイルデータを要求する．

Gfarm ではファイルシステムへのアクセス手段として Gfarm 並列 I/O API を提供している．Gfarm 並列 I/O API はファイルのオープン，クローズといったアクセス手段を直接提供するインターフェースである．Gfarm 並列 I/O API の一部を**表 1** に示す．クライアントは `gfs_pio_open` を呼び出すことによって Gfarm ファイルシステム上のファイルをオープンすることができる．また，`gfs_pio_read` を呼び出すことによって，オープンしたファイルの内容を読むことができる．クライアントは Gfarm ファイルシステムを利用する際，直接または間接的にこれらの Gfarm 並列 I/O API を利用する．クライアントはこのライブラリを自分のプログラムにリンクして呼び出すことにより，Gfarm ファイルシステムにアクセスすることができる．Gfarm ファイルシステムは，ユーザレベルのファイルシステムを構築するためのフレームワークである FUSE [7] を用いることにより，Linux のファイルシステムにマウントできる．この場合はシステムコールにより Gfarm ファイルシステ

© 2012 Information Processing Society of Japan

表 1　Gfarm 並列 I/O API の一部
Table 1　A part of Gfarm parallel I/O API

gfs_pio_open	Gfarm ファイルのオープン
gfs_pio_read	Gfarm ファイルの読み込み
gfs_pio_write	Gfarm ファイルの書き込み
gfs_pio_close	Gfarm ファイルのクローズ

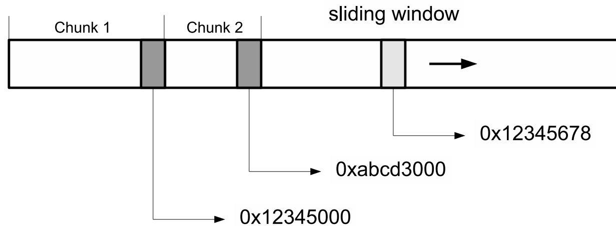

図 2　CDC によるチャンク分割
Fig. 2　Chunking by a CDC method

ムにアクセスすることで間接的に Gfarm 並列 I/O API が呼び出される．よって，Gfarm 並列 I/O API に対し拡張機能を加えることで，あらゆるファイルアクセスに対応することができる．本研究では，Gfarm 並列 I/O API のうちファイルの読み書きに関するものに対し，本機構を実装した．

3. CDC を用いた重複除外処理

3.1 CDC

CDC とは，ファイルの内容に基づいてファイルを可変長のチャンクに分割する方式である．LBFS [8] により用いられたのが最初であるが，その後多くのファイルシステムやストレージシステムに対して広く用いられた．重複したデータを持つチャンクをその位置によらず検出できることから，バックアップシステムの重複除外などで多く用いられるようになった [9]．また，そのアルゴリズムやパラメータの設定等に関して数々の改良・最適化が提案されている [10] [11]．

CDC ではチャンクの境界位置を定めるために固定長（典型的には 48 バイト長）のウインドウが用いられる．この方法では，ウインドウをファイルの先頭から 1 バイトずつスライドさせ，ウインドウに含まれるデータをハッシュ値に変換する．このハッシュ値の下位数ビットが特定の値と一致したときに，該当するウインドウの終点をチャンクの境界とみなすことで，ファイルの分割が行われる．CDC によるチャンク分割の例を**図 2** に示す．図中の左側の 2 つの濃い灰色の四角はチャンクの境界となる 48 バイトの領域である．右側の薄い灰色の四角は現在のウインドウを示す．四角から伸びる矢印の先にある値はその領域のデータから計算されるハッシュ値であり，図ではハッシュ値の下位 13 ビットが 0x1000 であるウインドウの終点をチャンクの境界としている．このときに下位何ビットまでを用いるかによってチャンクのサイズの平均が決まる．これは平均チャンクサイズと呼ばれる CDC のパラメータの一つである．例えば下位 13 ビットを用いた場合には平均チャンクサイズは 8KB になる．

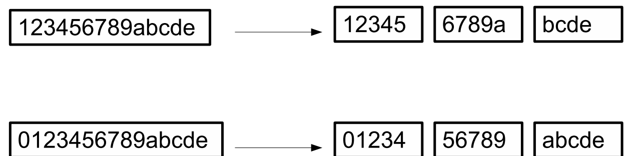

図 3　ファイルの先頭に 1 バイト付加された場合のブロック分割（ブロックサイズ 5 バイト）
Fig. 3　Division into fixed-sized blocks when one byte is added to the beginning of a file (block size is 5 bytes)

このときのハッシュ関数として LBFS では Rabin fingerprint [12] が用いられている．Rabin fingerprint はフィンガープリントの一種で，数学的には 2 元体上の既約多項式の除算に基づいたアルゴリズムである．与えられた長さ n のビット列を $n-1$ 次の多項式とみなし，これをある決められた k 次既約多項式で割る．このときの余りは $k-1$ 次の多項式であり，長さ k のビット列に対応するので，これを Rabin fingerprint とするものである．Rabin fingerprint は効率的な計算が可能であるという特徴を持つ．より具体的には，バイト列 $a_0...a_{n-1}$ から求めたハッシュ値を利用してバイト列 $a_1...a_n$ の計算が高速に行えるという性質を持つ．さらに予め計算結果を格納したテーブルを用いた高速化手法も提案されており [13]，本研究でもこの手法を利用した．

CDC の利点は，ファイルの一部が変更された際に，そのファイルのキャッシュへの変更が少なくて済むことである．ファイルを固定長のブロックで分割した場合，ファイルの先頭や途中に対して，データの追加や削除が行われたときに，以降の全てのブロックがずれてしまう問題がある．**図 3** にファイルの先頭に 1 バイトのデータが追加された場合の例を示す．左上の四角はデータの追加前のファイルを表しており，矢印の先の複数の四角はファイルの先頭から 5 バイト長のブロックに分割した結果を表す．そのファイルの先頭に 1 バイトのデータを追加したものが左下の四角であり，データ追加後のファイルをブロック分割した結果がその右の四角である．図より，データの追加前後で全てのブロックの中身が変化しているのがわかる．このように，ファイルの先頭や途中に対してデータの追加や削除が行われると，ファイルが以前とは別の中身を持つブロックに分割されてしまい，重複除外が働かなくなってしまう．

一方，CDC による可変長チャンクを利用した場合は，影響を受けるのは最初のチャンクのみであり，二番目以降のチャンクは変化しない可能性が高い．また，ファイルの途中にデータを挿入した際も同様に，挿入箇所を含むチャンクにしか殆ど影響を及ぼさない．というのも，チャンクの

© 2012 Information Processing Society of Japan

境界はファイルデータで決まるため，ファイルデータのうち変更のない部分はチャンクの境界も変化しないからである．そのため，類似データをチャンクに分割した結果が一致しやすくなり，重複除外が比較的有効に働くと考えられる．実際，文献 [14] の研究における実験では，CDC は固定長ブロックによる分割と比較してファイルの冗長性の検出性能が高いという結果が得られている．

CDC は主なパラメータとしてウインドウサイズ，平均チャンクサイズを持つが，その中でも性能に直接大きな影響を及ぼすのは平均チャンクサイズである．平均チャンクサイズが小さいほどチャンクの重複率は向上する [14] が，同時にチャンクを管理するためのオーバーヘッドも増加する．適切な平均チャンクサイズはアプリケーションやデータに依存するが，LBFS の例では平均チャンクは 8KB に設定されている．本研究では平均チャンクサイズを 4KB に設定した．

3.2 重複除外における CDC の利用

本研究における CDC を用いた重複除外処理の方法について説明する．本機構ではファイルはファイル単位でなくチャンク単位で保存される．また，チャンクはそのチャンクの中身により識別される．具体的にはチャンクの中身を元にそのハッシュ値を計算し，その値をチャンク ID として用いるといった方法が用いられる．これにより，中身の等しいチャンクは等しいハッシュ値，すなわち等しいチャンク ID を持つことになる．ここで，中身の等しいチャンクはそのストレージ内にただ一つしか作成しないことにすると，重複するチャンクは作成されないことになり，その結果として重複除外処理が行われる．

4. システムの設計・実装

4.1 システムの設計

本機構は Gfarm 上のファイルの 1 回目の read，write 時に，クライアントのローカルディスクにキャッシュを作成する．キャッシュの作成では，前章で述べた CDC による重複除外処理を適用し，チャンクごとにチャンクファイルを作成する．

同一内容のチャンクファイルはただひとつしか作成されないとする．これにより，チャンク単位の重複除外が実現する．キャッシュヒットの判定は，チャンク ID の一致・不一致により行う．これにより，初めて読み込むファイルであっても以前読み込んだファイルと同じチャンクを持っていればキャッシュの効果が期待できる．また，ファイルが更新されてもキャッシュを捨てる必要はない．なぜなら，ファイルが更新された時点で更新部分を含むチャンクが変化し，それによりチャンク ID も変化するため，次のファイル read 時にキャッシュが読まれることはないためである．なお，チャンク ID を求めるためのハッシュ関数としては，LBFS などの例に倣い SHA-1 ハッシュ [15] を用いた．チャンクファイルは SHA-1 ハッシュ値をファイル名とするファイルであり，その中身がチャンクデータを表す．

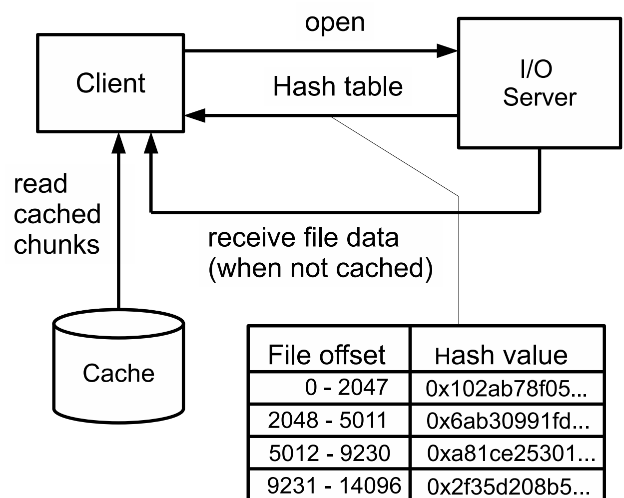

図 4 提案機構導入後のファイルアクセス
Fig. 4 File access using our mechanism

本機構導入後の Gfarm におけるファイル読み込み時の処理について述べる．まずクライアントはファイルオープン時にハッシュ表を I/O サーバに要求する．ハッシュ表とは，ファイルを構成するチャンクのリストであり，各チャンクのファイル内オフセット及び SHA-1 ハッシュ値をエントリに持つ．ハッシュ表は事前に計算され，I/O サーバ上にファイルとして保存される．そのファイル名はハッシュ表が表すファイルの世代番号を元に作成される．クライアントはハッシュ表を元に read 対象部分を含むチャンクを求め，それらの SHA-1 ハッシュ値からチャンクファイルがクライアントのローカルディスク上に存在するかどうかを調べる．チャンクファイルがローカルディスク上に存在する場合にはそのファイルを読み，存在しない場合だけ I/O サーバにそのチャンクデータを要求する．また I/O サーバから新たに得られたデータは，チャンクファイルとしてクライアントのローカルディスクに保存される．

次に，ファイルへの書き込み時の処理について述べる．読み込み時と同様に，クライアントはファイルオープン時にハッシュ表を I/O サーバから取得する．write の際には write により上書きされる範囲のデータについて CDC を適用し，チャンク分割を行う．チャンク境界が見つかる度に新しいチャンクのデータを更新データとして記憶すると共に，クライアント側のハッシュ表に追加する．チャンク分割が終了したら，クライアントは I/O サーバに対して write 要求およびハッシュ表の更新要求を発行する．

4.2 システムの実装

本システムは Gfarm ライブラリおよび Gfarm ファイルサーバデーモン (gfsd) を改造することで行った．

© 2012 Information Processing Society of Japan

表 2 実験環境
Table 2 Experimental environment

CPU	Intel Xeon 2.40GHz × 2
Memory	48GB
HDD	15000rpm 600GB
OS	CentOS 5.5 64bit
kernel	ver 2.6.18
Gfarm	ver 2.4.2
ノード間接続	InfiniBand

ライブラリについては read, write 処理を担っている `gfs_client_pread()`, `gfs_client_pwrite()` などの関数に変更を加えた他，チャンク分割を行うための関数などを新たに作成した．メタデータサーバとの通信は元の実装のままで変更を加えていない．追加したコード行数はおよそ 950 行程度である．

また，Gfarm クライアントと gfsd の間でハッシュ表およびチャンクデータの送受信を行うため，gfsd との通信用の RPC プロトコルに `GFS_PROTO_GETHASHLIST` および `GFS_PROTO_GET_CHUNK` を新たに追加した．`GFS_PROTO_GETHASHLIST` はファイルディスクリプタを送信し，ハッシュ表を受信するプロトコルである．`GFS_PROTO_GET_CHUNK` はファイルディスクリプタおよびチャンクのリストを送信し，チャンクデータを受信するプロトコルであり，同一ファイル内の複数のチャンクファイルをまとめて要求することができる．

5. 評価

提案機構導入によるファイル読み込み性能の評価実験を行った．Gfarm ファイルシステム上のファイルを先頭から末尾まで 1MB ずつシーケンシャルに読み込むのにかかる時間を計測した．ファイルの読み込みには Gfarm 並列 I/O API を用いた．計測は内容・サイズの異なる複数のファイルに対してそれぞれ複数回行った．ファイル内容はランダムファイル及びゼロで埋められたファイルの 2 通り，またファイルサイズは 1GB および 10GB の 2 通りを用いて実験を行った．読み込まれるファイルがメモリに載った状態での性能を調べるため，最初の数回を除いた平均値を求めた．提案機構導入後の計測は，ファイルを構成するチャンクが全てクライアントのローカルにチャンクファイルとして存在し，かつページキャッシュに載った状態で行い，全てのチャンクがローカルから読み出される場合の性能を調べた．また，チャンク分割における平均チャンクサイズは 4KB とした．実験環境は**表 2** の通りである．メタサーバ，I/O サーバ，クライアントノードを各 1 台ずつ使用した．

ランダムファイル及びゼロで埋められたファイルの読み込み実験結果をそれぞれ**表 3**, **表 4** に示す．提案機構の導入後は導入前と比べて最大で 19%実行時間が短縮された．また，ゼロで埋められたファイルの読み込み性能がランダムファイルに比べて早い理由は，ゼロで埋められたファイルの場合，単一のチャンクファイルがページキャッシュに載った状態で繰り返し読み出されるためであると考えられる．今回の実験では単一のクライアントプロセスによる読み込み性能を測定したが，複数クライアントによる読み込みを行った場合，導入前のシステムでは I/O サーバとの通信がボトルネックとなり性能がスケールアウトしない．一方で提案機構導入後はチャンクファイルが存在する限り全てローカル読み出しとなるため，性能がクライアント数に対してスケールアウトすると考えられる．

次に，データインテンシブアプリケーションにおける大規模データの重複率を調べるための実験を行った．**表 5** はゲノムデータに対して CDC によるチャンク分割を適用した際のチャンクファイルの重複率を表している．実験に用いたデータは GenBank [16] のゲノムデータベースを BLAST により抽出することで得られる総サイズ 40GB のテキストファイル群である．これらのデータに対して CDC を適用し，得られたチャンクデータの重複率を求めた．重複率の計算は，重複するチャンクファイルの総サイズを全ファイルサイズで割ることにより行った．表 5 より，平均チャンクサイズ 128B の場合の重複率が約 9.3%となることが分かった．ゲノム解析においては同一のデータを複数回読み込むケースが十分に考えられるため，重複排除による転送量の削減や読み書き性能の向上といった効果が期待できる．

また，提案機構の導入前後におけるデータ転送量の比較実験を行った．Gfarm 上に 10MB のランダムファイルを

表 3 ランダムファイル読み込みの実行時間 [s]
Table 3 Execution time of random file read [s]

サイズ	導入前	導入後 キャッシュ無	導入後 キャッシュ有	前後比
1GB	1.6	20.391	1.3	0.81
10GB	17.2	13m10	14.6	0.85

表 4 ゼロで埋められたファイルの読み込みの実行時間 [s]
Table 4 Execution time of zero fill file read [s]

サイズ	導入前	導入後 キャッシュ無	導入後 キャッシュ有	前後比
1GB	1.6	0.621	0.55	0.34
10GB	17.1	5.795	5.25	0.31

表 5 ゲノムデータの重複率
Table 5 Deduplication rate of genome data

平均チャンクサイズ	重複率
1KB	0.37%
512B	3.8%
256B	6.3%
128B	9.3%

© 2012 Information Processing Society of Japan

表 6 データ転送量 [MB]
Table 6 The amount of the transmitted data [MB]

	send	recv
導入前	0.3662	20000
導入後	9.717	10136.7

表 7 ファイル A およびファイル B を構成するチャンクファイルのストレージ消費量 [KB]
Table 7 The amount of the storage consumption of the chunk files of file A and file B [KB]

	固定長ブロック方式	チャンク方式
A のみ	1048576	1048576
B のみ	1048580	1048576
A と B	2097156	1048578.5

1001 個作成し，リモートのクライアントからキャッシュファイルを持たない状態でこれらのファイルを読み出した．作成した 1001 個のファイルのうち 1000 個を 1 回ずつ読み出し，残り 1 個のファイルのみ 1000 回読み出したときのデータ転送量を計測した．**表 6** にデータ転送量の比較結果を示す．結果より，提案機構の導入後におけるデータ転送量は導入前と比較して約半分になっていることが分かる．読み出すファイルそのもののデータに加え，ハッシュ表の転送が行われるため，半分よりもやや多くなっている．

次に固定長ブロック方式とチャンク方式におけるストレージ消費量の比較実験を行った．1GB のランダムデータファイル A およびファイル A の先頭に 1 バイト追加したファイル B を読み出し，作成されたチャンクファイルの合計サイズを固定長ブロック方式とチャンク方式の両方でそれぞれ求めた．固定長ブロック方式におけるブロックサイズおよびチャンク方式における平均チャンクサイズは共に 4KB とした．

結果は**表 7** の通りである．固定長ブロック方式ではファイル A，ファイル B を構成するブロックが一つも一致しないため，両ファイルを読み出した後のキャッシュファイルの総サイズはそれぞれのファイルを単体で読み出した際に作成されるキャッシュファイルの総サイズに等しくなる．一方，チャンク方式では両ファイルを構成するチャンクが先頭チャンク以外全て一致するため，両ファイルを読み出した後のキャッシュファイルの総サイズはファイル A，B を単体で読み出した場合のものと殆ど変わらない．結果としてチャンク方式ではキャッシュファイルの総サイズが固定長ブロック方式の場合の半分程度に収まった．

6. 関連研究

LBFS [8] は低バンド幅ネットワークのためのファイルシステムであり，CDC を初めて提案したシステムである．クライアントおよびサーバは共にチャンクインデックスというチャンクのメタデータを保存するデータベースを持つ．データの送受信をチャンク単位で行うことによりデータ転送量の削減を図っている．キャッシュファイル自体はファイル単位で保存され，重複除外は行われていない．

CDC を用いた重複除外はバックアップシステムやストレージシステムに多く見られ，SSE でハッシュ値を求める手法 [17] や GPGPU でハッシュ値を求める手法 [18] などが提案されている．本機構はこれらの研究と組み合わせることでさらなる性能向上が期待できる．

7. おわりに

CDC を用いた重複除外を行うキャッシュ機構の実装・評価を行った．ランダムファイルの読み込みでは提案機構により最大 19%性能向上した．またデータ転送量およびストレージ消費容量が減少することを確認した．今後は write 時のハッシュ表の更新処理の実装を行う他，本機構と圧縮処理を組み合わせることやマルチコアを流用する機構についても検討する．また実用的なアプリケーションを用いた評価も積極的に行う．

謝辞 本研究を行うにあたって，有益な助言を頂いた筑波大学建部研究室の方々に深く感謝する．また本研究は，科学技術振興機構戦略的創造研究推進事業（JST CREST）の研究課題「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」の支援を受けている．

参考文献

[1] Babu, A.: GlusterFS. http://www.gluster.org/.
[2] Braam, P. J., *Lustre*, http://www.lustre.org/.
[3] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn : Ceph: A Scalable, High-Performance Distributed File System, In *Proceedings of the 7th Conference on Operating Systems Design and Implementation*, pp. 320–327 (2006).
[4] S. Ghemawat, H. Gobioff, and S. Leung : The Google file system, In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, pp. 29–43 (2003).
[5] O. Tatebe, K. Hiraga, and N. Soda : Gfarm Grid File System, *New Generation Computing*, Ohmsha, Ltd. and Springer, Vol. 28, No. 3, pp. 257–275 (2010).
[6] 村上じゅん，石黒駿，大山恵弘：重複除外による Gfarm の性能向上に関する検討，並列/分散/協調処理に関するサマー・ワークショップ (SWoPP), (2011).
[7] Szeredi, M. : FUSE: Filesystem in Userspace. http://fuse.sourceforge.net/.
[8] A. Muthitacharoen, B. Chen, and D. Mazieres : A Low-bandwidth Network File System, In *Proceedings of the 8th ACM symposium on Operating systems principles*, pp. 174–187 (2001).
[9] L. P. Cox, C. D. Murray, and B. D. Noble : Pastiche: Making Backup Cheap and Easy, In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pp. 285–298 (2002).
[10] E. Kruus, C. Ungureanu, and C. Dubnicki : Bimodal Content Defined Chunking for Backup Streams, In *Pro-*

© 2012 Information Processing Society of Japan

ceedings of the 8th USENIX Conference on File and Storage Technologies, pp. 239–252 (2010).
[11] J. Min, D. Yoon, and Y. Won : Efficient Deduplication Techniques for Modern Backup Operation, *IEEE Transactions on Computers*, Volume 60, Issue 6, pp. 824–840 (2010).
[12] Michael O. Rabin : Fingerprinting by random polynomials, Center for Research in Computing Technology, Harvard University, Technical Report TR-15–81 (1981).
[13] A. Z. Broder : Some applications of Rabin's fingerprinting method, *Sequences II: Methods in Communications, Security, and Computer Science*, pp. 143–152 (1993).
[14] Calicrates Policroniades and Ian Pratt : Alternatives for Detecting Redundancy in Storage Systems Data, In *Proceedings of the USENIX 2004 Annual Technical Conference*, pp. 73–86 (2004).
[15] D. E. Eastlake and P. E. Jones, "US Secure Hash Algorithm 1 (SHA-1)", RFC 3174, IETF, Sept. (2001).
[16] GenBank. http://www.ncbi.nlm.nih.gov/genbank/.
[17] 坪内佑樹， 置田真生， 伊野文彦，山川聡， 柏木岳彦，萩原兼一，"重複排除ストレージにおける SHA-1 計算の SSE によるスループット向上手法"，情報処理学会研究報告. 2012-HPC-133(31), (2012).
[18] C. Kim, K. Park, and K. Park : GHOST: GPGPU-offloaded high performance storage I/O deduplication for primary storage system, In *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, pp. 17–26 (2012).