Vol.2012-ARC-202 No.4
Vol.2012-HPC-137 No.4
2012/12/13

Content-Defined Chunking を用いた重複除外キャッシュ機構の実装と評価

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，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 Gfarm へ適用した際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．クライアントはファイルの読み込み時にチャンクデータをキャッシュ内に保存し，以降のファイルアクセスで再利用する．書き込み時には書き込み処理の裏でデータをチャンクに分割し，その結果をサーバに返す．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，またファイル書き込み時のオーバーヘッドが 30%程度であることを確認した．

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

Implementation and Evaluation of 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 the overhead of file write is about 30 % of the original performance.

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]．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の 1 つで，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メタデータサーバ，ストレージを供給する I/O サーバおよび 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書きすることができる．

[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

Gfarm におけるファイルアクセスは次のように行われる．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，ファイルデータ全体を保存するという方法である．しかし，この方法には問題点が 2 つある．1 つはキャッシュの保持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，保持しているキャッシュデータを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）単位でキャッシュとして保存し，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] では読み込みに対する提案機構導入時の性能評価を行ったが本稿では書き込みに対する性能評価および並列化によるキャッシュ処理の高速化について示す．

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．

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，ストレージを提供する I/O サーバおよびファイルのメタデータを管理するメタデータサーバの 3 つである．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 I/O サーバ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．

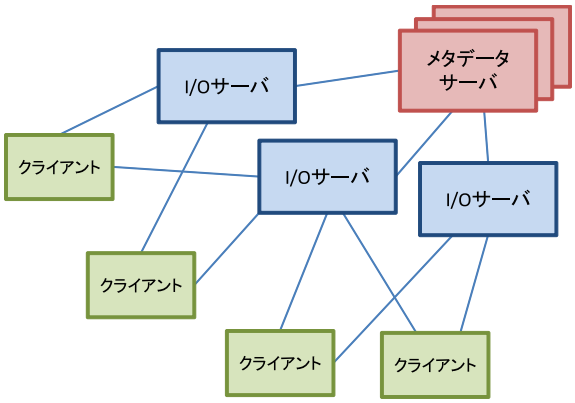

図 1 Gfarm のアーキテクチャ
Fig. 1 The architecture of Gfarm

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．

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

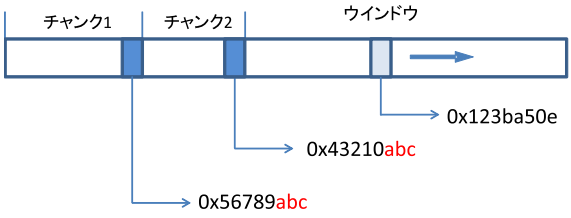

図 2　CDC によるチャンク分割
Fig. 2　Chunking by a CDC method

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き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，重複したデータを持つチャンクの検出に優れていることから，バックアップシステムなどへも応用されている [9]．また，そのアルゴリズムやパラメータの設定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．図中の左側の濃い色の 2 つ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 12 ビットが 0xabc 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 12 ビットを用いた場合には平均チャンクサイズは 4KB になる．

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．すなわち，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 $a_1 ... a_n$ の計算を高速に行うことができる．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．

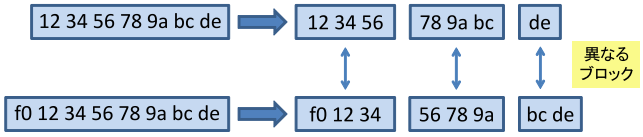

図 3　ファイルの先頭に 1 バイト付加された場合のブロック分割（ブロックサイズ 3 バイト）
Fig. 3　Division into fixed-sized blocks when one byte is added to the beginning of a file (block size is 3 bytes)

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 3 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，影響を受けるのは編集箇所を含むチャンクであり，その他のチャンクは変化しない可能性が高い．というのも，チャンクの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．

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は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ハッシュ値を計算し，そのハッシュ値を用いてチャンクを識別する．以下，このハッシュ値をそのチャンクのチャンク ID と呼ぶ．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，チャンク ID 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ず，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 回目のアクセス時に，クライアントノード上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する．

同一内容のキャッシュ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．キャッシュ内容の同一性の判定は，チャンク ID の一致・不一致の判定により行う．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，次のファイル読み込み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，LBFS などの例に倣い SHA-1 ハッシュ [15] を用いた．

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 (**図 4**)．まずクライアントはファイルオープン時にチャンクリストを I/O サーバに要求する．チャンクリストとは，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・チャンクサイズおよびチャンク ID をリストの要素に持つ．チャンクリストは書き込み時に計算され，I/O サーバ上にファイルとして保存される．チャンクリストの大きさは平均チャンクサイズにより異なる．例えば平均チャンクサイズが

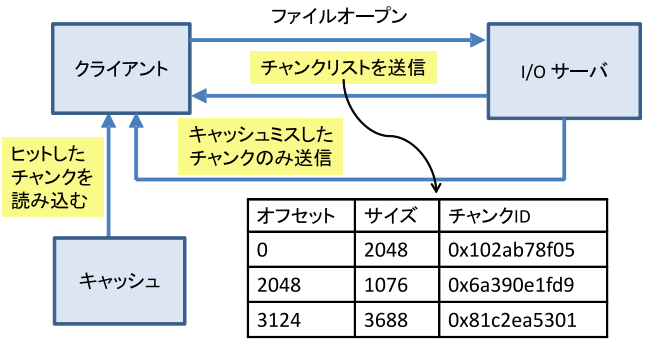

図 4 本機構導入後の read アクセス
Fig. 4 Read access after implementing our system

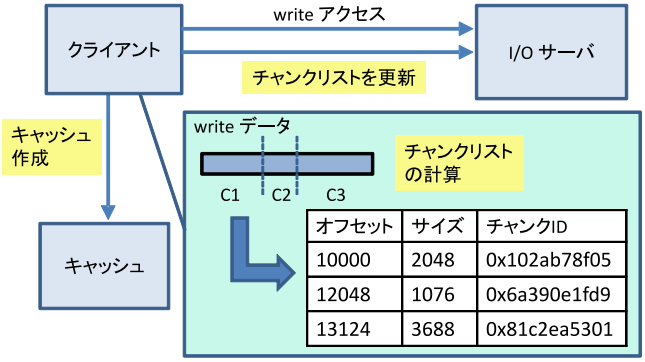

図 5 本機構導入後の write アクセス
Fig. 5 Write access after implementing our system

4KB の場合には，チャンクリストの大きさは分割対象ファイルのサイズの約 1%となる．クライアントはチャンクリストを元に読み込み対象部分を含むチャンクを求め，それらのチャンク ID からチャンクデータがクライアントのキャッシュ内に存在するかどうかを調べる．チャンクデータがキャッシュ内に存在する場合にはキャッシュからチャンクデータを読み，存在しない場合には I/O サーバにそのチャンクデータを要求する．なお，チャンクデータの要求は複数のチャンクをまとめて行う．このようにして I/O サーバから新たに得られたチャンクデータはクライアントのキャッシュに保存される．

次に，ファイルへの書き込み時の処理について述べる (**図 5**)．クライアントは通常の書き込み処理に加え，書き込みデータのチャンク分割・チャンクリスト作成および I/O サーバへのチャンクリストの送信処理を行う．ここで通常の書き込み処理とは，単一のスレッドが I/O サーバに対して書き込みデータを転送する処理を指す．提案機構導入後の Gfarm では，通常の書き込み処理と同時に複数スレッドで書き込みデータのチャンク分割およびチャンク ID の計算を行う．通常の書き込み処理およびチャンク分割・チャンク ID の計算が終了したら，クライアントは I/O サーバに対してチャンクリストの更新要求を発行し，書き込みデータから作成された新たなチャンクリストの断片を

サーバに送信する．

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) を改造することで実装を行った．Gfarm ライブラリについては read, write に対応する API 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．

また，Gfarm クライアントと gfsd の間でチャンクリスト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，チャンクリスト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キャッシュをまとめて要求することができる．

キャッシュの実装方法については，チャンクごとにファイルを作成する方法を採用した．すなわち，チャンクデータのチャンク ID をファイル名とし，チャンクデータを中身に持つファイルとして実装した．

4.3 OpenMP によるスレッド並列化

本システム導入によるオーバーヘッドの低減および性能向上のため，CDC・SHA-1 計算，read 処理，write 処理のそれぞれに対しスレッド並列化を行った．いずれも OpenMP を用いて記述した．まず CDC については，対象データをスレッドと同数のブロックに分割し，各ブロックを各スレッドに割り当てることで並列化を行った．チャンク境界は基本的に 48 バイト長の局所的なデータにより決まるため，ブロックごとの計算が可能である．ただし極端な大きさのチャンクの生成を防ぐため，チャンクサイズの最大値および最小値を設定している．そして各スレッドによるチャンク分割の後に，ブロック境界を含むチャンクのサイズが設定した最小値と最大値の間に収まるよう，適宜修正を行う．SHA-1 の計算では，得られたチャンクの数をスレッド数で割り，チャンク単位で各スレッドに割り当てている．

次に read 処理では，I/O サーバから読み込んだチャンクデータの処理部分に対し並列化を適用した．I/O サーバからチャンクデータを読み込む際には，複数のチャンクをまとめて読み込む．これらのチャンクを均等に各スレッドに割り当て，キャッシュファイルの作成および read バッファへのコピーを並列に行った．

また write 処理では I/O サーバへの write データの送信を 1 スレッドで行い，それと並列に write データの CDC・SHA-1 計算およびキャッシュの作成を含むキャッシュ処理を複数スレッドで行う．データの送信を行うスレッドとキャッシュ処理を行うスレッドは 1 回の write 処理ごとに同期する．よって次の write 処理により write バッファの中身が上書きされるときには必ず前の write 処理が終了している．CDC・SHA-1 計算の並列処理については前述のとおりである．

表 2　実験環境
Table 2　Experimental environment

CPU	Intel Xeon 2.40GHz (6 コア) × 2
Memory	48GB
HDD	15000rpm 600GB
OS	CentOS 6.2 64bit
kernel	ver 2.6.32
Gfarm	ver 2.5.5
OpenMP	ver 3.0
ノード間接続	InfiniBand

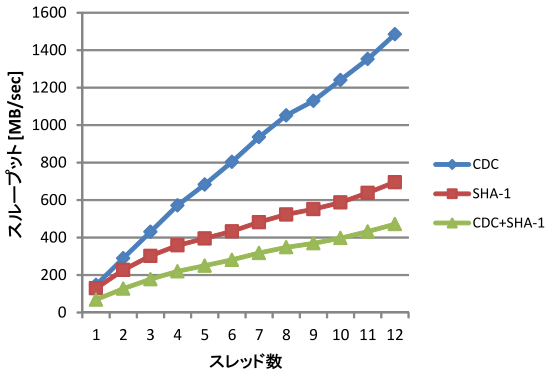

図 6　複数スレッドによる CDC・SHA-1 計算のスループット [MB/sec]
Fig. 6　Throughput of creating a chunk list of a file by multiple threads [MB/sec]

5. 評価

5.1 CDC・SHA-1 計算の並列化

CDC および SHA-1 ハッシュの計算を同一ノード上の複数スレッドで並列に行った場合の実行時間を評価した．10GB サイズのランダムデータからなるファイルを先頭から末尾まで 256KB ずつ読み込み，1 回の読み込みごとにデータのチャンク分割および SHA-1 ハッシュの計算を行った．このときのチャンク分割および SHA-1 ハッシュの計算に要する時間をそれぞれ測定した．実験環境は**表 2** のとおりであり，ノード 1 台を使用した．また読み込み対象のファイルがページキャッシュに載った状態で測定を行った．測定は複数回行い，そのうち最小値を選択した．

図 6 はスレッド数を 1 から 12 まで変化させたときの各計算のスループットを示している．CDC のみの結果，SHA-1 のみの結果および CDC と SHA-1 の両方を実行し

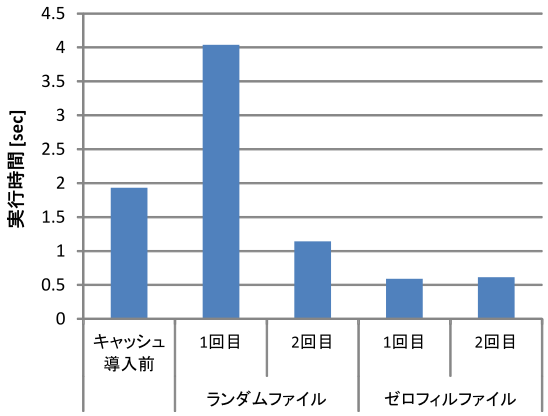

図 7 ファイル読み込みの実行時間 [sec]
Fig. 7 Execution time of file read [sec]

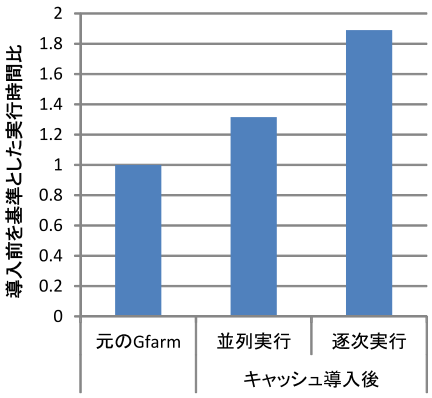

図 8 ファイル書き込みの実行時間比
Fig. 8 Execution time ratio of file write

た場合の結果をそれぞれまとめた．図より，いずれの場合もスレッド数の増加に伴いスループットが直線的に増加していることが分かる．CDC のみ, SHA-1 のみ，CDC と SHA-1 両方の 3 つの場合それぞれについて 12 スレッドでの実行時のスループットを 1 スレッドの場合と比較すると，それぞれ 10.1 倍，5.4 倍，6.9 倍となった．なお，逐次処理の場合と並列化後の 1 スレッドの場合の結果に違いは見られなかった．

以上の結果から，CDC および SHA-1 ハッシュの計算におけるスレッド並列化の有効性を確かめることができた．以降の評価実験における CDC・SHA-1 計算は 12 スレッドで並列に行っている．

5.2 読み込み性能の評価

続いて提案機構導入によるファイル読み込み性能の評価実験について述べる．Gfarm ファイルシステム上のファイルを先頭から末尾まで 100MB ずつシーケンシャルに読み込んだときの実行時間を計測した．読み込みは Gfarm 並列 I/O API を直接用いて行った．ファイルサイズは 1GB, 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 種類を用いた．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すべてのチャンクがクライアントのローカルにキャッシュとして存在し，かつページキャッシュに載った状態で行った．また，チャンク分割における平均チャンクサイズは 16KB とした．実験環境は表 2 のとおりである．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．

結果を**図 7** に示す．まず，中身がランダムなファイルの 1 回目の読み込みについては，キャッシュ導入前と比べて実行時間が 2 倍に増加している．キャッシュ導入によるオーバーヘッドはキャッシュヒットの判定および新しいキャッシュの作成処理のコストである．現在の実装では読み込み処理とキャッシュに関する処理を逐次的に行っているため，キャッシュ導入によるコストが読み込みコストにそのまま上乗せされた結果になっている．今後，これらの処理を並列に行うことでコストを削減できると思われる．

ランダムファイルの 2 回目およびゼロフィルファイルの 1 回目・2 回目の結果はいずれもキャッシュからの読み込み性能を表している．ゼロフィルファイルの読み出しがランダムファイルに比べて高速なのは，読みだされるチャンクの種類が極めて少なく，それらが CPU のキャッシュに載っているためであると考えられる．

5.3 書き込み性能の評価

次に，書き込み性能の評価結果を**図 8** に示す．実験は，ローカルファイルシステム上のファイルから読み込んだデータを Gfarm ファイルシステム上のファイルに書き込む際の書き込みに要する実行時間を計測することで行った．読み込み対象ファイルの中身は 1GB のランダムデータである．書き込みデータをチャンク分割する際の平均チャンクサイズは 16KB とした．縦軸は提案機構導入前の実行時間を 1 としたときの実行時間比を表す．図の棒グラフは左から順にそれぞれ提案機構導入前の Gfarm における書き込み性能，キャッシュの作成処理を通常の write 処理の裏で並列に行った場合の性能および通常の write 処理の完了後に逐次的に行った場合の性能を表している．並列実行の場合および逐次実行の場合の実行時間はそれぞれ導入前の約 1.3 倍および約 1.9 倍となっており，この結果から並列化により実行時間が大きく減少することが確認できた．

5.4 WRF 出力ファイルにおける重複率

最後に実際のアプリケーションにより出力されるファイルの重複除外実験について述べる．Weather Research & Forecasting (WRF) モデルは実用的な天気予報および大気研究のために開発された気象解析予報システムである．WRF の出力ファイルは NetCDF（Network Common

表 3　WRF 出力ファイルの重複率
Table 3　Deduplication rate of WRF output files

平均チャンクサイズ	重複率 [%]
1KB	37.58
4KB	35.78
8KB	34.55
16KB	32.56
gzip	54.20

Data Form）形式で保存される．これは気温・海面温度・湿度・風速・風向きなどの多次元データを配列形式で格納するものである．今回の実験では WRF により出力された総サイズ 2GB の NetCDF ファイルに対して CDC による重複除外を適用し，除外されるチャンクデータの全体に占める割合（重複率）を求めた．

表 3 は平均チャンクサイズを 1KB, 4KB, 8KB および 16KB に設定した際の重複率および gzip を用いた圧縮による重複率をまとめたものである．平均チャンクサイズが小さいほどデータの重複率が高くなる傾向が見られるが，いずれの場合も 30%以上の高い重複率となっている．gzip の重複率と比較しても約 60%の重複データが CDC により検出されることがわかる．以上の結果から，科学技術計算のアプリケーションが生成するファイルの中には重複率が高いものが存在することが確認された．

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．

HydraFS [16] は CDC により分割された可変長チャンク単位で読み書きを行うバックアップシステム HYDRAStor のためのファイルシステムである．本研究では分散ファイルシステムのクライアントキャッシュに対して CDC を適用した．

7. おわりに

CDC を用いた重複除外を行うキャッシュ機構を分散ファイルシステム Gfarm に対して実装し，書き込み時のチャンクリストの更新処理の実装および通信とハッシュ計算の並列化による高速化を実現し，評価を行った．

今後は実用的なアプリケーションを本機構導入後の分散ファイルシステム上で実行したときの性能評価を行う他，複数クライアント間でのキャッシュの共有についても検討する．

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] Weil, S., Brandt, S., Miller, E., Long, D. and Maltzahn, C.: Ceph: A scalable, high-performance distributed file system, *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 307–320 (2006).
[4] Ghemawat, S., Gobioff, H. and Leung, S.: The Google file system, *ACM SIGOPS Operating Systems Review*, Vol. 37, No. 5, ACM, pp. 29–43 (2003).
[5] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New General Computing*, Vol. 28, No. 3, pp. 257–275 (2010).
[6] 村上じゅん，石黒駿，大山恵弘：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, 情報処理学会研究報告 2012-OS-121-20, pp. 1-7 (2012).
[7] Szeredi, M.: FUSE: Filesystem in Userspace. `http://fuse.sourceforge.net/`.
[8] Muthitacharoen, A., Chen, B. and Mazieres, D.: A low-bandwidth network file system, *ACM SIGOPS Operating Systems Review*, Vol. 35, No. 5, ACM, pp. 174–187 (2001).
[9] Cox, L., Murray, C. and Noble, B.: Pastiche: Making backup cheap and easy, *ACM SIGOPS Operating Systems Review*, Vol. 36, No. SI, pp. 285–298 (2002).
[10] Kruus, E., Ungureanu, C. and Dubnicki, C.: Bimodal content defined chunking for backup streams, *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, USENIX Association, pp. 18–18 (2010).
[11] Min, J., Yoon, D. and Won, Y.: Efficient deduplication techniques for modern backup operation, *Computers, IEEE Transactions on*, Vol. 60, No. 6, pp. 824–840 (2011).
[12] Rabin, M.: *Fingerprinting by random polynomials*, Center for Research in Computing Techn., Aiken Computation Laboratory, Univ. (1981).
[13] Broder, A.: Some applications of Rabin's fingerprinting method, *Sequences II: Methods in Communications, Security, and Computer Science*, Vol. 993, pp. 143–152 (1993).
[14] Policroniades, C. and Pratt, I.: Alternatives for detecting redundancy in storage systems data, *Proceedings of the 2004 USENIX Annual Technical Conference*, pp. 73–86 (2004).
[15] Eastlake, D. and Jones, P.: US secure hash algorithm 1 (SHA1) (2001).
[16] Ungureanu, C., Atkin, B., Aranya, A., Gokhale, S., Rago, S., Calkowski, G., Dubnicki, C. and Bohra, A.: HydraFS: a high-throughput file system for the HYDRAstor content-addressable storage system, *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, USENIX Association (2010).