ifarm: インライン重複除外機構を導入した並列分散ファイルシステム

松宮 遼[1,a)] 佐々木 慎[1,b)] 高橋 一志[1,2,c)] 大山 恵弘[1,2,d)]

概要：多くのファイルシステムやストレージシステムにおいて，重複除外機構が実装されている．重複除外機構とは，ファイル間で重複したデータを 1 度しかデバイス上に書き込まないというものである．高性能計算で利用される並列分散ファイルシステムに対しても，重複除外機構の実装が期待されている．しかしながら，そのようなファイルシステムに対し重複除外機構を実際に実装し，評価を行った研究は，我々が調べた限りでは稀少である．そこで我々は，高性能計算などで利用されている並列分散ファイルシステムである Gfarm に，重複除外機構を導入した ifarm の開発を行っている．本論文は ifarm を実装する上で明らかとなった課題と，その課題に対して我々がとった解決策について議論を行うものである．また，我々は現段階における ifarm の性能評価を行った．その結果についても報告する．

キーワード：重複除外，並列分散ファイルシステム，高性能計算，Gfarm

1. はじめに

高性能計算において，非常に大きなデータの読み書きが必要とされる場合がある．超高解像度海洋大循環モデルの実装である COCO [9] では，格子点数 $40000 \times 40000 \times 40000$ において，10 年分のシミュレーションを行うと 365PB ものデータをストレージに書き込む [30]．そのようなデータを格納する場合のストレージのコストは膨大なものであるため，そのコストを減らすことが課題となっている．

ストレージのコストを減らすには，デバイスに書き込まれるデータのサイズを削減すればよい．Nath ら [17] は，重複除外機構を高性能計算に導入することを提案，実装し評価を行った．重複除外機構とは，ストレージ内で重複したデータをデバイス上には 1 度しか書き込まないというもので，多くのファイルシステムやストレージシステムに実装されている [7,12,16,18,21,24,26,27]．重複除外の方法はいくつかあるが，Meister ら [14] は高性能計算用のデータでは，Nath らが用いた固定長チャンクを用いる方法よりも，可変長チャンクを用いる方法をとることで重複除外機構によるデータ削減の効果を大きく得られることを示した．しかし，高性能計算で利用される並列分散ファイルシステムに対して可変長チャンクを用いた重複除外機構を実際に実装し，ファイルへのアクセスが多く発生するアプリケーション (データインテンシブアプリケーション) を用いて評価を行った研究は，我々が調べた限りでは存在しない．ゆえに

[1] 電気通信大学
[2] 科学技術振興機構 CREST
[a)] r.matsumiya@ol.inf.uec.ac.jp
[b)] sasashin@ol.inf.uec.ac.jp
[c)] kazushi@inf.uec.ac.jp
[d)] oyama@inf.uec.ac.jp

高性能計算で利用される並列分散ファイルシステムに対し，可変長チャンクを用いた重複除外機構の導入を行う際の課題を調査する必要がある．したがって我々は，高性能計算などで利用されている並列分散ファイルシステムである Gfarm [23] に対して，可変長チャンクによる重複除外機構を実装した ifarm の開発を行っている．

本論文は，ifarm を開発する上で明らかとなった課題と，その課題に対して我々がとった解決策について議論を行うものである．また，現時点における ifarm の性能評価を行ったので，それについても報告する．

本論文の貢献は次のようになる．

- 高性能計算用並列分散ファイルシステムに，可変長チャンクによる重複除外機構を導入する上での課題を明らかにし，その解決策を提示した．
- 重複除外機構を導入した並列分散ファイルシステムで，アプリケーションの実行時間を測定した．
- 重複率の高いデータセットでは，重複除外機構導入後の並列分散ファイルシステムが，導入前に比べてアプリケーションの実行時間が短くなる可能性を示した．

本論文の構成を以下に示す．まず，背景となる Gfarm 及び重複除外について説明する．その後，ifarm の設計と実装について述べたのち，ifarm に対して行った実験について説明する．次に関連研究について解説し，最後にまとめと今後の課題について述べる．

2. 背景

2.1 Gfarm

Gfarm とは，ただ 1 つのメタデータサーバと，1 つ以上の I/O サーバから構成されている並列分散ファイルシステムである．メタデータサーバは，Gfarm 上のファイルのパーミッションや I/O サーバの情報等を管理する．I/O サーバは Gfarm 上のファイルの内容を管理する．

計算ノードは，`gfs_pio_open()` や，`gfs_pio_read()` など POSIX ライクの API を用いることで，Gfarm 上のファイルを操作することができる．本研究における計算ノードは，FUSE [1] 向けクライアント実装である gfarm2fs を用いることで，Gfarm を操作することを前提とする．

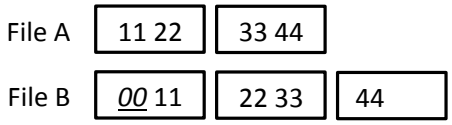

図 1 Byte shift problem の例

Gfarm は close-to-open コンシステンシを保証する．これは，あるファイルが閉じられる前に変更された内容は，再びそのファイルが開かれた後には必ず観測できるというコンシステンシである．close-to-open コンシステンシは，NFS [3] などの分散ファイルシステムでも保証されている．ifarm でもこのコンシステンシを保証するようにしている．

2.2 重複除外の方法

2.2.1 固定長チャンクによる重複除外

ファイルシステムに対する重複除外機構では，一般的に，1 つのファイルを複数のチャンクに分割する．チャンクはデバイス上に記録されるが，デバイス上に既に存在するチャンクと内容が同一のチャンクは記録されない．ZFS [7] や Venti [18] などでは，ファイルを固定長のチャンクに分割し，重複除外を行うという手法を取っている．

しかしながら，固定長のチャンクに分割するという方法では，byte shift problem と呼ばれる問題が発生する．Byte shift problem とは，重複データの位置にズレがあった場合，データの重複を重複除外機構が検知できないことがあるというものである．

Byte shift problem の例を図 1 に表す．チャンクのサイズは 2 バイトとする．ファイル A と，ファイル A の内容の先頭に 1 バイトだけ追加したファイル B がある．この 2 ファイルの内容は明らかに類似しているが，重複データの位置にズレがあるため，重複除外機構はこの重複を検知できない．

2.2.2 可変長チャンクによる重複除外

Byte shift problem に対する解決策として，チャンクを可変長にするという方法がある．低帯域幅用の分散ファイルシステムである LBFS [16] では，ファイルの内容に基づいてチャンクを分割する手法が実装された．この手法は，固定長のウィンドウをファイルの始端から終端まで 1 バイトずつスライドさせる．重複除外機構はウィンドウ内におけるハッシュ値 (一般的には Rabin fingerprint [19] が用いられる) を計算する．あるウィンドウ内におけるハッシュ値の下位ビットが，事前に定義された特定の値であったとする．このとき，重複除外機構は，そのウィンドウの終端をチャンクの境界線としてファイルの分割を行う．

チャンクを可変長にすることで，byte shift problem の多くが解決される．事実，多くのデータにおいて，固定長チャンクに分割する手法よりも，可変長チャンクに分割する手法の方が，より多くの重複データを検出できていることが，複数の研究によって実験的に示されている [14, 15, 29]．そのような理由から，可変長チャンクに分割する手法は，Hydra [8, 24] や Data Domain [27] などいくつかのストレージシステムに実装されている．ifarm でも，可変長チャンクに分割する手法を用いた重複除外機構を導入する．

3. 設計と実装

3.1 インライン重複除外

ファイルシステムに対する重複除外機構は，ファイルシステムがデバイスに記録するタイミングによって 2 種類に分けることができる．

1 つはオフライン重複除外と呼ばれるものである．これは，アプリケーションからファイルシステムに対する書き込みを行っていない間に，ファイルシステムがデバイスに記録を行うというものである．実装は容易であるが，一般的に，いつアプリケーションがファイルシステムに対する書き込みを行うかを推測するのは困難である．ファイルシステムがデバイスに記録している間は，アプリケーションからの書き込みをブロックすれば，アプリケーションがファイルシステムに対する書き込みを行うタイミングを推測する必要はなくなる．しかしながら，高性能計算では，計算ノード間で処理の同期を計算中に行うことが多い．ある計算ノードからの書き込みをブロックした場合，他の計算ノードとの同期に遅延が発生する．この遅延が，全体の計算時間をも大幅に遅延させることがある．即ち，高性能計算においては，オフライン重複除外を導入するべきではない．

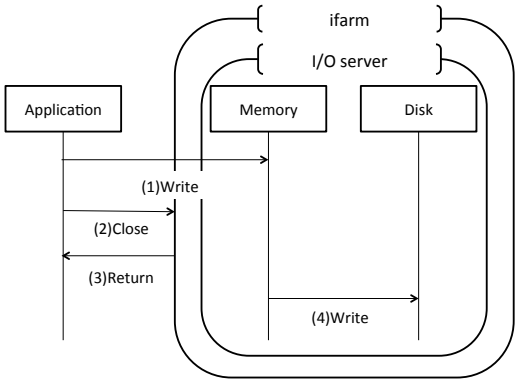

図 2 非同期処理の概要

ファイルシステムに対する重複除外のもう 1 種類は，インライン重複除外と呼ばれている．インライン重複除外は，アプリケーションからファイルシステムに対する書き込みの有無に関係なく，ファイルシステムがデバイスに記録するものである．つまり，インライン重複除外は，オフライン重複除外よりも高性能計算にとっては適切なものであるといえる．以上の理由から，ifarm では，インライン重複除外をとる．

3.2 非同期処理

可変長チャンクに分割する手法では，ウィンドウ内のハッシュ値を求める処理によるよる計算オーバーヘッドが発生する．この処理は，大量のデータを読み書きする上で，ファイルシステムやストレージシステム全体の性能を下げるボトルネックとなりうる．この計算オーバーヘッドを削減するために，ifarm では，計算ノードの処理と非同期に可変長チャンクに分割し，デバイスに記録するという方法をとっている．本節では，ifarm がこの非同期処理をどう実現しているかについて述べる．

非同期処理の概要を図 2 に示す．(1) ファイルの変更箇所は，一旦メモリ上に書き留められる．(2) アプリケーションがファイルを閉じるとき，(3) 計算ノードは即時アプリケーションに処理を戻す．それと同時に，計算ノードは I/O サーバに対してファイルを閉じる命令を送信する．(4)I/O サーバは命令を受信すると，変更状況に基づいてファイルを可変長チャンクに分割し，デバイス上にその結果を保存する．この I/O サーバの処理と，処理がアプリケーションに返った後の計算ノードの処理は独立している．これにより ifarm は計算ノードの処理と，I/O サーバによる可変長チャンクへ分割する処理を非同期にすることを可能とした．

非同期処理を実現している具体的な処理を以下に示す．計算ノードが I/O サーバに対してファイルの書き込みをリクエストしたとする．I/O サーバはリクエストを受信すると，リクエストされたファイルのオフセットに対応するチャンクを，メモリ上にコピーする．コピーされたメモリ上のチャンクは仮チャンクといい，I/O サーバはリクエストされた書き込みを仮チャンクに対して行う．リクエストされた書き込みが複数のチャンクにまたがったものであったとする．この時生成される仮チャンクの内容は，全てのチャンクの内容を統合したものである．

計算ノードはファイルを閉じようとするとき，I/O サーバに命令を送る．そして，I/O サーバが命令を受信したことを検知すると速やかにファイルディスクリプタを閉じて，アプリケーションに処理を返す．I/O サーバは命令を受信すると，閉じようとしているファイルと結びついている全ての仮チャンクを可変長のチャンクに分割する．このとき，事前に設定された最大チャンクサイズを超えるようであれば，最大チャンクサイズで強制的にチャンク分割するようにしている．

計算ノードがファイルの読み込みを行うリクエストを I/O サーバに行ったとする．この時の I/O サーバの処理には，リクエストされた箇所に仮チャンクが割り当てられているか否かによって異なる．もし，リクエストされた箇所に仮チャンクが割り当てられていた場合，I/O サーバは仮チャンクの該当箇所の内容を返す．リクエストされた箇所に仮チャンクが割り当てられていなかった場合は，I/O サーバはチャンクの該当箇所の内容を返す．

3.3 コンシステンシ

2.1 節で述べたように，ifarm では，close-to-open コンシステンシを保証する．しかしながら，3.2 節の方法では close-to-open コンシステンシが保証されない．close-to-open コンシステンシが守られない例を図 3 の左に表す．図形の形状の変化はチャンクへの分割によるテーブルの書き換え，図形の色の変化はファイルの内容の変更によるテーブルの書き換えを表している．アプリケーション A によってあるファイルが閉じられ，チャンクへの分割処理が行われている間に，アプリケーション B がそのファイルを開き，ファイルの内容を更新したとする．このとき，チャンクへの分割処理が終了すると，再びテーブルの内容が書き換わるのでアプリケーション B が更新した内容が無効化されてしまう．

Close-to-open コンシステンシの保証のために，ifarm はバージョン管理システムである Subversion [5] のような checkout-commit-update 形式を採用した．計算ノードが Gfarm 上のファイルを開く時，I/O サーバは Gfarm 上のファイルとチャンクの対応関係を，メモリ上にコピー (チェックアウト) する．この対応関係は，1 ファイルにつき 1 テーブル割り当てられ，チェックアウトはテーブル単位で行われる．

ファイルの内容の変更により，ファイルとチャンクの対応関係に変更が加わるとき，I/O サーバは，自分がチェックアウトしたテーブルを更新する．更新後，I/O サーバはチェックアウト元となったテーブルにコミットする．計算ノードによりファイルが閉じられ，全ての仮チャンクが可変長チャンクに分割されたとする．その時もファイルとチャンクの対応関係には変更が加わる．この時も I/O サーバはチェックアウト元となったテーブルに対してコミットを行う．即ち，コミットにはファイルの内容の変更に伴うものと，チャンクへの分割処理に伴うものの 2 種類が存在する．

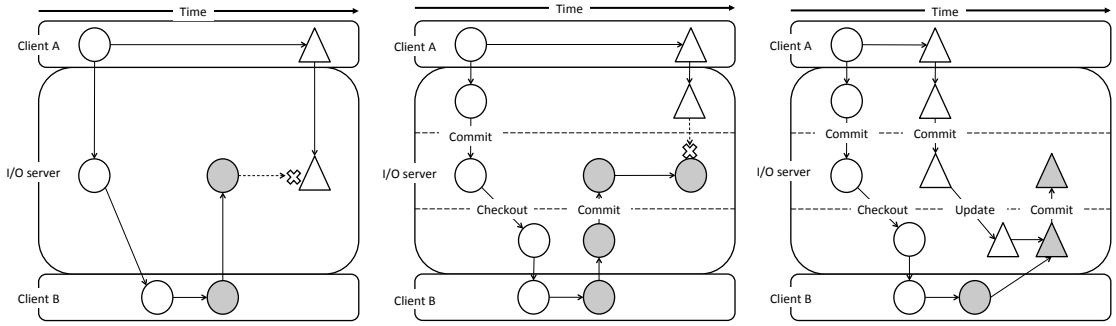

図 3 左は close-to-open コンシステンシが守られない例，中央はチャンクへの分割処理に伴うコミットにおけるコンシステンシの保証方法，右はファイルの内容の変更に伴うコンシステンシの保証方法をそれぞれ表す

ある計算ノードがチェックアウトを行い，コミットを行うまでの間に，別の計算ノードが同じテーブルに対してコミットを行っていたということがある．このときI/Oサーバが行う動作は，I/Oサーバが行うコミットがどちらの種類であるかに依存する．チャンクへの分割処理に伴うコミットであった時について図3の中央を用いて述べる．この時，close-to-open コンシステンシを考えれば，別の計算ノードによって行われたコミットは，ファイルが閉じられた後に開かれ実行された動作であることが自明である．したがって，チャンクへの分割処理に伴うコミットは無効化される．ファイルの内容の変更に伴うコミットであった時 (図3の右) は，チェックアウト元の内容をチェックアウト先に再度コピーする (アップデート)．アップデート後は，ファイルの変更内容を再度テーブルに適用し，コミットを行う．

3.4 GC

ファイルの内容変更や，ファイルの削除によって，生成されたチャンクがどのファイルからも参照されなくなることがある．不要なチャンクが大量に生成されると，デバイスの容量を大量に浪費してしまう可能性がある．そこでifarmでは，不要なチャンクを回収するGCが実装されている．各チャンクはそれぞれ参照カウントを持つ．チャンクに分割される時において，ファイルがあるチャンクを参照することになったとき，参照カウントをインクリメントする．ファイルの内容変更や削除によって，そのチャンクが参照されなくなった時参照カウントをデクリメントする．参照カウントがゼロになったとき，I/Oサーバはそのチャンクをデバイスから削除する．

4. 実験

ifarmの性能を測定するために，いくつかの実験を行った．実験はオリジナルのGfarmと比較して行うものとした．

4.1 実験環境

実験環境は1台のメタデータサーバ，1台のI/Oサーバ，1台の計算ノードより構成されている．各ノードは同じスペックのマシンであり，相互の通信はInfiniBand QDR 4Xにより行われる．ノードの環境を以下に示す．CPUはXeon E5-2609，メモリは64GB，HDDは15000rpmのSASによる2台構成のRAID 0である．OSはLinux 2.6.32で，ノードのローカルファイルシステムはExt4，Gfarmはバージョン2.5.8.7で，gfarm2fsはバージョン1.2.9.6を使用した．

4.2 マイクロベンチマーク

4.2.1 ゼロデータファイルに対する読み込み速度

ゼロデータファイルをシーケンシャルリードした結果を図4に示す．Gfarmと比較すると，ifarmは最大1.38倍のスループットを出すことができ

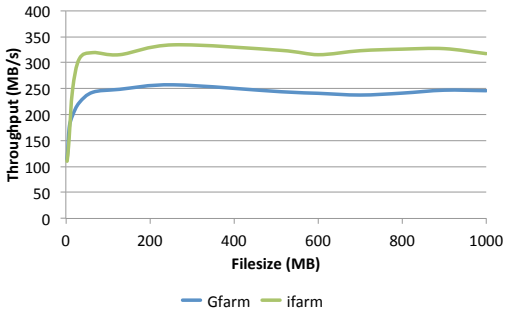

図 4 ゼロデータファイルに対する読み込み速度

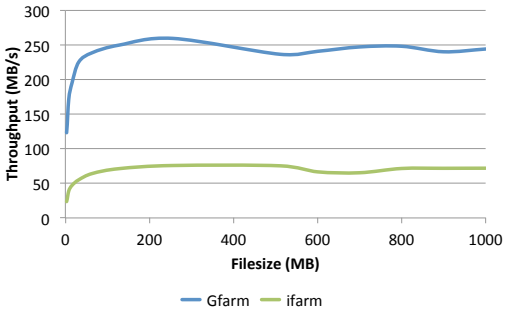

図 5 ランダムデータファイルに対する読み込み速度

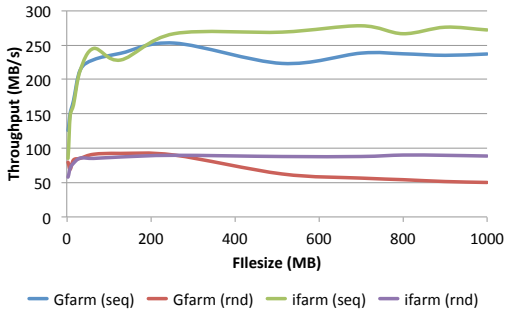

図 6 IOR によるファイル読み込み速度

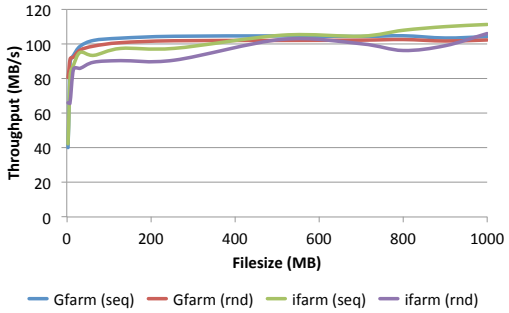

図 7 IOR によるファイル書き込み速度

た．重複除外機構により，重複したチャンクは全て HDD 上の同じ箇所に書き込まれる．ゼロデータファイルは全てのチャンクが重複している．つまり，このファイルをシーケンシャルリードするということは，HDD 上の同じ箇所を何度も読み込むということである．この時，実際に HDD にアクセスするのは先頭のチャンクを読み込むときのみであり，あとは I/O サーバのメモリ上にあるキャッシュを読み込む．メモリへのアクセスは HDD へのアクセスよりも十分に高速であるため，Gfarm よりも ifarm の方が高速となった．この結果は，重複除外率が高いファイルをシーケンシャルリードするようなデータインテンシブアプリケーションは，ファイルシステムに重複除外機構を導入することにより，実行時間を短縮できるという可能性を示唆している．

4.2.2 ランダムデータファイルに対する読み込み速度

ランダムデータファイルをシーケンシャルリードした結果を図 5 に示す．Gfarm と比較すると，ifarm は最大でも 0.32 倍，最低 0.19 倍のスループットとなってしまった．ゼロデータファイルと違い，ランダムデータファイルは全てのチャンクが互いに重複しない．チャンクの内容が HDD 上のどの箇所に記録されているかを探索する処理がボトルネックとなってしまったからだと考えられる．重複除外機構を導入した既存のファイルシステムやストレージにおいても，このボトルネックについて言及されている [17, 21]．

4.2.3 IOR ベンチマーク

ランダムアクセスの性能や，書き込み速度の性能を測定するために，我々はベンチマークアプリケーションである IOR [2] を用いたベンチマークも実施した．

IOR によるファイル読み込みベンチマークの結果を図 6 に示す．seq はシーケンシャルアクセス，rnd はランダムアクセスを表す．シーケンシャルリードでは最大 1.20 倍，ランダムリードでは最大 1.77 倍の性能向上が確認された．先の実験の結果と，このシーケンシャルリードの結果より，IOR

が読み込むデータは重複除外率が高いと言える．ゆえにこの結果は，ランダムリードにおいても重複除外率が高いデータを読み書きするアプリケーションは，ファイルシステムに重複除外機構を適用することで実行時間が短くなる可能性を表している．

IOR によるファイル書き込みベンチマークの結果を図 7 に示す．今回は Gfarm と ifarm で大きな性能差を確認することはできなかった．これはファイルの書き込みが多いアプリケーションを実行するとき，ifarm と Gfarm では実行速度に大きな差は出ないことを表している．

4.3 アプリケーションベンチマーク

実際のデータインテンシブアプリケーションに対する重複除外機構による性能的な影響を調査するため，我々は Montage [4] の実行時間を測定した．Montage とは，天体望遠鏡が観測した画像を統合するソフトウェアであり，Shibata らの研究 [20] によりデータインテンシブアプリケーションであることが示されている．本実験では入力データセットと，生成される中間ファイル及び出力される画像ファイルを ifarm (Gfarm) 上に，Montage 本体のファイルを計算ノードのローカルファイルシステム上に保管されるようにして行った．

実行結果を図 8 に示す．Gfarm 上で実行した時に比べて，ifarm 上で実行した場合，実行時間が 1.42 倍となってしまった．Montage の実行においては，入力ファイルや生成されるファイルの重複除外によって得られる性能向上よりも，重複除外による計算オーバヘッドのほうが大きいと考えられる．

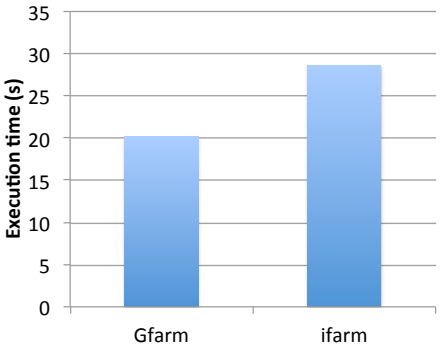

図 8 Montage の実行時間

5. 関連研究

我々が以前に発表した [31] では，実験が非常に不足しており，GC の実装も行っていなかった．本論文は，GC の実装を行った上で，実験の種類を増やしたものである．

Nath ら [17] は，重複除外機構を導入した並列分散ファイルシステムである CAPFS [25] 上にデータインテンシブアプリケーションを実行する上での課題などについて検討を行った．CAPFS では固定長チャンクによる重複除外を行っている．ifarm は可変長チャンクによる重複除外を適用し，Nath らの研究では行っていなかった，アプリケーション実行時間の測定も行った．

Meister ら [14] は，高性能計算で実際に利用されているストレージに対して重複除外を行った場合のシミュレートを行った．結果，高性能計算で利用されるデータでは，固定長チャンクによる重複除外よりも，可変長チャンクによる重複除外の方がより多くの重複データを検出できることがわかった．この研究ではシミュレートのみを行っており，実際に重複除外機構を導入したわけではない．本研究は，この研究を踏まえた上で，高性能計算で利用される並列分散ファイルシステムに対して重複除外機構を導入する上での課題などを議論するものである．

可変長チャンクによるインライン重複除外機構を導入している既存研究として，Hydra [8,24] や Data Domain [27]，SAR [12]，iDedup [21] が挙げられる．Hydra や Data Domain はバックアップ用途であり，SAR はクラウド環境を想定した VM イメージ向けのストレージである．高性能計算での利用を想定していないという点で本研究と異なる．iDedup は高性能計算にも転用可能であるが，データインテンシブアプリケーションによるベンチマークを行っていない．

村上らの研究 [28] においても，Gfarm を拡張し，重複除外機構を導入している．しかしこの研究は

Gfarm のプロトコルにも手を加えており，Gfarm プロトコルとの互換性を持っていない．更に，チャンク分割の処理は計算ノード側が行っているため，計算ノードの計算資源を消費してしまう．ifarm は Gfarm プロトコルとの互換性を維持させ，かつ計算ノードの計算資源を消費することなく重複除外機構を導入することを可能とした．また，データインテンシブアプリケーションによる実験も行っていないという点においても本研究と異なる．

可変長チャンクによる重複除外の計算オーバーヘッドに着目し，並列化によって計算オーバーヘッドの削減を試みた研究は多数存在する [6, 10, 11, 13, 22, 26, 32]．しかしながら，これらの手法を導入した場合でも巨大なデータに対しては十分な計算オーバーヘッドが発生する．その上，並列化によって計算資源を大きく消費してしまうという問題点がある．ifarm では可変長チャンクに分割する処理を非同期で行う．これにより，計算資源を大きく消費することなく，巨大なデータでも計算オーバーヘッドを十分に削減することができる．

6. まとめと今後の課題

我々は，Gfarm に可変長チャンクによる重複除外機構を導入した ifarm の開発を行っている．ifarm では，アプリケーションによるファイルシステムに対する書き込みの有無に関係なく，ファイルシステムがデバイスに記録するインライン重複除外を採用している．また，ファイルを可変長チャンクに分割する際に発生する計算オーバーヘッドを削減するために，ifarm では I/O サーバが行うチャンクへの分割を計算ノードの処理と非同期に行わせている．この非同期処理によって Gfarm で保証されている close-to-open コンシステンシが ifarm では保証されなくなってしまうという問題が発生したが，Subversion [5] のような checkout-commit-update 形式を採用することで解決した．前回の我々の研究 [31] では未実装であった参照カウントによって不要となったチャンクをデバイス上から削除する GC を実装した．ifarm の性能を評価した結果，重複除外率が高いデータを使うアプリケーションでは，重複除外を行うことでアプリケーションの実行速度を短縮される可能性があることが判明した．

今後の課題について述べる．ifarm ではメモリ上のデータを読み書きしているが，この方法は I/O サーバに障害が発生した時に復旧が困難となってしまうという問題点がある．したがって耐障害性の高い実装が求められる．また，仮チャンクのサイズが大きくなることによりメモリスワップが発生してしまい，速度が低下するおそれがある．仮チャンクのサイズが大きくなっても十分な速度が出せる実装が必要となる．これら 2 つはメモリ上で読み書きしているデータを，揮発性メモリではなく，SSD などの不揮発性メモリを利用するという方法で解決できるので検討していきたい．最後に，データインテンシブアプリケーションとして，本研究では Montage によるベンチマークしかとっていない．他のデータインテンシブアプリケーションでも実験されるべきであると我々は考えている．

謝辞 本研究を行うにあたって，筑波大学の建部修見准教授及び HPCS 研究室建部グループの方々より有益な助言を頂いた．また本研究は科学技術振興機構 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] : FUSE: Filesystem in Userspace, `http://fuse.sourceforge.net/`.

[2] : IOR HPC Benchmark — Free System Administration software downloads at SourceForge.net, `http://sourceforge.net/projects/ior-sio/`.

[3] : Linux NFS faq, `http://nfs.sourceforge.net/`.

[4] : Montage - Image Mosaic Software for Astronomers, `http://montage.ipac.caltech.edu/`.

[5] Apache Software Foundation: Apache Subversion, `https://subversion.apache.org/`.

[6] Bhatotia, P., Rodrigues, R. and Verma, A.: Shredder: GPU-accelerated Incremental Storage and Computation, *the Proc. of USENIX FAST '12* (2012).

[7] Bonwick, J.: ZFS Deduplication (Jeff Bonwick's Blog), `https://blogs.oracle.com/bonwick/entry/zfs_dedup` (2009).

[8] Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C. and Welnicki, M.: HYDRAstor: a Scalable Secondary Storage, *the Proc. of USENIX FAST '09* (2009).

[9] Hasumi, H.: *CCSR Ocean Component Model (COCO) Vesrion 4.0*, Center for Climate System Research, The University of Tokyo (2007).

[10] Kim, C., Park, K.-W. and Park, K. H.: GHOST: GPGPU-offloaded High Performance Storage I/O Deduplication for Primary Storage System, *the Proc. of ACM PMAM '12* (2012).

[11] Ma, J., Zhao, B., Wang, G. and Liu, X.: Adaptive Pipeline for Deduplication, *the Proc. of IEEE MSST '12* (2012).

[12] Mao, B., Jiang, H., Wu, S., Fo, Y. and Tian, L.: Read-Performance Optimization for Deduplication-Based Storage Systems in the Cloud, *ACM TOS*, Vol. 10, No. 6 (2014).

[13] Matsumiya, R., Takahashi, K., Oyama, Y. and Tatebe, O.: Content-Defined Chunking for CPU-GPU Heterogeneous Environments, *Poster at USENIX FAST '14* (2014).

[14] Meister, D., Kaiser, J., Brinkmann, A., Cortes, T., Kuhn, M. and Kunkel, J.: A Study on Data Deduplication in HPC Storage Systems, *the Proc. of IEEE/ACM SC '12* (2012).

[15] Meyer, S. T. and Bolosky, W. J.: A Study of Practical Deduplication, *the Proc. of USENIX FAST '11* (2011).

[16] Muthitacharoen, A., Chen, B. and Mazières, D.: A Low-bandwidth Network File System, *the Proc. of ACM SOSP '01* (2001).

[17] Nath, P., Urgaonkar, B. and Sivasubramaniam, A.: Evaluating the Usefulness of Content Addressable Storage for High-performance Data Intensive Applications, *the Proc. of ACM HPDC '08* (2008).

[18] Quinlan, S. and Dorward, S.: Venti: A New Approach to Archival Storage, *the Proc. of USENIX FAST '02* (2002).

[19] Rabin, M. O.: *Fingerprinting By Random Polynomials*, Center for Research in Computing Techn., Harvard Univ. (1981).

[20] Shibata, T., Choi, S. J. and Taura, K.: File-access Patterns of Data-intensive Workflow Applications and their Implications to Distributed Filesystems, *the Proc. of ACM HPDC '10* (2010).

[21] Srinivasan, K., Bisson, T., Goodson, G. and Voruganti, K.: iDedup: Latency-aware, Inline Data Deduplication for Primary Storage, *the Proc. of USENIX FAST '12* (2012).

[22] Tang, Z. and Won, Y.: Multithread Content Based File Chunking System in CPU-GPGPU Heterogeneous Architecture, *the Proc. of IEEE CCP '11* (2011).

[23] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, Ohmsha, Ltd. and Springer (2010).

[24] 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, *the Proc. of USENIX FAST '10* (2010).

[25] Vilayannur, M., Nath, P. and Sivasubramaniam, A.: Providing Tunable Consistency for a Parallel File Store, *the Proc. of USENIX FAST '05* (2005).

[26] Xia, W., Jiang, H., Feng, D., Tian, L., Fu, M. and Wang, Z.: P-Dedupe: Exploiting Parallelism in Data Deduplication System, *the Proc. of IEEE NAS '12* (2012).

[27] Zhu, B., Li, K. and Patterson, H.: Avoiding the Disk Bottleneck in the Data Domain Deduplication File System, *the Proc. of USENIX FAST '08* (2008).

[28] 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の実装と評価，情報処理学会 HOKKE-20 (2012).

[29] 大山恵弘，松宮　遼：マルウェアデータに対する重複除外の適用，コンピュータセキュリティシンポジウム 2014 論文集 (2014).

[30] 石川　裕，堀　敦史，Balazs, G.，高木将通，島田明男，清水正明，佐伯裕治，白沢智輝，中村　豪，住元真司，小田和友仁：次世代高性能並列計算機のためのシステムソフトウェアスタック，情報処理学会オペレーティングシステム・計算機アーキテクチャ合同研究会 (2013).

[31] 松宮　遼，佐々木慎，高橋一志，大山恵弘：並列分散ファイルシステムに対するオンライン重複除外機構の導入に向けて，2014 年並列／分散／協調処理に関する『新潟』サマー・ワークショップ (2014).

[32] 松宮　遼，平井成海，佐々木慎，高橋一志，大山恵弘：CPU-GPU アーキテクチャを用いた Content-Defined Chunking の実装と評価，情報処理学会 コンピュータシステム・シンポジウム論文集 (2013).