Vol.2014-HPC-145 No.33
2014/7/30

InfiniBandを利用したCooperative Cachingの実装と評価

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

概要：高性能科学計算の分野におけるアプリケーションには，ペタバイトスケールのデータに対して計算を行うものが存在する．これらはデータインテンシブアプリケーションと呼ばれ，分散ファイルシステム上に入出力データを保存することが多い．入出力データに対するアクセスがボトルネックとなり，アプリケーションの計算性能が低下するのを避けるため，分散ファイルシステムの性能向上が求められている．ディスクへのアクセスはメモリへのアクセスに比べて非常に低速であるため，可能な限り多くのデータをメモリ上にキャッシュしディスクへのアクセスを低減することが，分散ファイルシステムの性能を向上させる上で重要である．本論文では分散ファイルシステムのクライアントノード間でメモリ上のキャッシュを融通する手法である cooperative caching を，高性能科学計算に利用される分散ファイルシステムである Gfarm ファイルシステムにおいて実現する機構を提案する．我々はクライアントノード間が InfiniBand で接続された環境を想定し，クライアントノード間のデータの転送に InfiniBand の RDMA を使用した．Montage ワークフローを用いて提案機構を評価したところ，実行時間が最大で 15.4%削減された．

Implementation and Evaluation of Cooperative Caching Using InfiniBand

Shin Sasaki[1,a)]　Ryo Matsumiya[1,b)]　Kazushi Takahashi[1,2,c)]　Yoshihiro Oyama[1,2,d)]

Abstract: In the field of high performance computing (HPC), distributed file systems are widely used to store large data processed by data-intensive applications. The amount of data used by these applications is increasing, and there is a demand for improving the performance of distributed file systems to deal with data more efficiently. It is important to reduce the amount of disk access and to cache as much as data in memory, because disk access throughput is much lower than memory access throughput. We propose a cooperative caching mechanism for Gfarm distributed file system. We enabled Gfarm client nodes to read the memory cache of other client nodes, and compose large amount of memory cache as a whole. We assume that Gfarm client nodes are connected by InfiniBand. We achieved high-speed data transportation between client nodes with InfiniBand remote direct memory access (RDMA). In this paper, we show the design and implementation of the mechanism and report the evaluation results. The results show that elapsed time of Montage workflow is reduced by 15.4%.

1. はじめに

高性能科学計算の分野において，大規模なデータを保存するために分散ファイルシステムが広く用いられている．分散ファイルシステムとは，単一または複数のノード上に保存された共有データに対して，複数のノードがネットワークを介してアクセスすることを可能にするファイルシステムである．分散ファイルシステムとして Gfarm [1],

[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
[a)] sasashin@ol.inf.uec.ac.jp
[b)] r.matsumiya@ol.inf.uec.ac.jp
[c)] kazushi@inf.uec.ac.jp
[d)] oyama@inf.uec.ac.jp

© 2014 Information Processing Society of Japan

Lustre [2], Ceph [3], GlusterFS [4] などが知られている．一般に，分散ファイルシステムはメタデータを管理するメタデータサーバ，データを保存するストレージサーバから構成される．また，分散ファイルシステム上のデータにアクセスするノードはクライアントノードと呼ばれる．

天文学，生命情報学，気象学分野などのアプリケーション [5–7] の中には，ペタバイトスケールのデータに対して計算を行うものが存在する．これらのアプリケーションはデータインテンシブアプリケーションと呼ばれ，分散ファイルシステム上に入出力データを保存することが多い．データインテンシブアプリケーションの入出力データは増加し続けており，高性能科学計算の分野における分散ファイルシステムの需要は高まることが予想される．入出力データへのアクセスによりアプリケーションの計算性能が低下するのを避けるため，分散ファイルシステムの性能向上が求められている．

ディスクへのアクセスはメモリへのアクセスに比べて非常に低速であるため，可能な限り多くのデータをメモリ上にキャッシュしディスクへのアクセスを低減することが重要である．クライアントノードが分散ファイルシステム上のデータを読み込む際，自身のメモリ上にデータがキャッシュされていれば，それを読み込むことができるためディスクアクセスは発生しない．自身のメモリ上にデータがキャッシュされていなければ，ネットワークを介したストレージサーバから読み込む．ストレージサーバから読み込む場合でも，ストレージサーバのメモリ上にデータがキャッシュされていれば，ディスクアクセスは発生しない．ストレージサーバのメモリ上にもデータがキャッシュされていない場合に，はじめてディスクアクセスが発生する．

ここでの問題は，クライアントノードがデータを読み込む際，自身のメモリ上またはストレージサーバのメモリ上にデータがキャッシュされていなければ，他のクライアントノードのメモリ上に当該データがキャッシュされていたとしても，ディスクへのアクセスが発生してしまうことである．この問題を解決する手法として *cooperative caching* という手法がある．

Cooperative caching とは，分散ファイルシステムおけるクライアントノードのメモリ上のデータを，クライアントノードが互いに利用することを可能にする手法である．クライアントノードを含めた分散ファイルシステム全体で，大容量のメモリキャッシュを構成することができる．InfiniBand や 10-gigabit Ethernet など高バンド幅，低レイテンシのネットワークの登場により，ネットワークを介したデータ転送のコストはディスクアクセスのコストに比べて遥かに小さくなっているため，cooperative caching の導入が分散ファイルシステムの性能向上に与える影響は大きいと予想できる．Cooperative caching に関する研究 [8–11] は数多く行われているが，高速なネットワーク環境において cooperative caching を導入した際にデータインテンシブアプリケーションに与える影響については明らかでない．

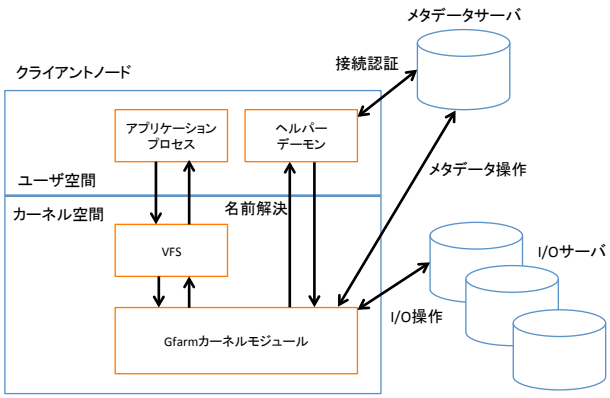

図 1 Gfarm カーネルモジュールと Gfarm の構成
Fig. 1 The architecture of Gfarm with kernel module

本研究では，高性能科学計算に利用される分散ファイルシステムである Gfarm ファイルシステムにおいて cooperative caching を実現する機構について提案する．また，Montage ワークフローを用いて行った評価の結果を報告する．Montage を用いた評価では，実行時間が最大で 15.4%削減された．

本研究において，我々は次のような環境を想定する．分散ファイルシステムを構成するサーバとクライアントノードは，ローカルディスクとして磁気ディスクを搭載している．全てのノードは同一 LAN 内に存在し，InfiniBand により相互に接続されている．また，クライアントノード上ではデータインテンシブアプリケーションを並列に実行し，分散ファイルシステム上のデータを利用した計算を行う．対象とする OS は Linux である．

本論文の構成は以下の通りである．2 章で Gfarm ファイルシステムについて述べる．3 章で提案機構の設計と実装について説明し，4 章で提案機構の評価と考察を述べる．5 章で関連研究について述べ，6 章でまとめを述べる．

2. Gfarm ファイルシステム

2.1 概要

Gfarm ファイルシステム [1] は，ペタバイトスケールの大規模なデータに対してスケーラブルな性能を実現する．Gfarm ファイルシステムは，メタデータサーバと I/O サーバで構成される．メタデータサーバは i ノード番号、世代番号，ファイルのロケーションなどのメタデータを管理する．メタデータサーバは障害に備えて冗長化することが可能であるが，複数のメタデータサーバが並列にメタデータを操作することはできない．I/O サーバは複数存在し，自身のローカルストレージにデータを保存する．Gfarm ファイルシステムでは，I/O サーバがクライアントノードを兼

© 2014 Information Processing Society of Japan

ねることが可能である．この場合，クライアントノードは，同一ノード上の I/O サーバが保持するデータに対してネットワークを介さずにアクセスする．そのため，アプリケーションとデータを同一ノードに配置することで，アプリケーションの計算性能を向上させることができる．

2.2 Gfarm カーネルモジュール

クライアントノードは専用のカーネルモジュールをロードすることで，Gfarm ファイルシステムを任意のマウントポイントにマウントすることができる．これにより，クライアントノードは Gfarm ファイルシステムに対して，カーネル空間から直接アクセスすることが可能となる．カーネルモジュールを使用した Gfarm ファイルシステムの構成を**図 1** に示す．クライアントノード上で専用のスクリプトを実行すると，Gfarm カーネルモジュールがロードされ，カーネルにファイルシステムとして登録される．それと同時に，ユーザ空間で動作するヘルパーデーモンが起動される．ヘルパーデーモンは接続認証やユーザ名，グループ名などの変換を行う．クライアントノード上のアプリケーションプロセスが，Gfarm ファイルシステムのマウントポイントに対してファイル I/O を発行すると，要求は VFS を介して Gfarm カーネルモジュールへと渡される．Gfarm カーネルモジュールは，メタデータサーバや I/O サーバと通信した後，VFS を介してアプリケーションプロセスに結果を返す．Gfarm ファイルシステムをマウントする方法として，カーネルモジュールを利用する方法の他に gfarm2fs を利用する方法がある．gfarm2fs はユーザーレベル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 [12] [13] を利用して実装されている．カーネルモジュールには，gfarm2fs に比べてユーザ空間とカーネル空間とのデータコピーや，コンテキストスイッチのオーバーヘッドが削減できるという利点がある．我々の提案機構は，クライアントノードが Gfarm カーネルモジュールを用いて Gfarm をマウントすることを仮定する．

3. 提案機構

3.1 設計

提案機構は Gfarm ファイルシステムにおいて cooperative caching を実現する．通常の Gfarm ファイルシステムでは，クライアントノードがデータを読み込む際，自身のメモリ上にデータがキャッシュされていなければ I/O サーバに対してデータを要求する．提案機構では，メモリ上にデータをキャッシュしている可能性のある他のクライアントノードが存在すれば，そのクライアントノードに対して当該データを要求する．そして，要求先クライアントノードがデータをキャッシュしていれば，要求元クライアントノードに対して直接転送する．提案機構において，クライアントノードが Gfarm ファイルシステム上のデータを読み込む際の流れを**図 2** に示す．複数存在するクライアントノードの中からデータの要求先クライアントノードを決定する仕組みと，要求先クライアントノードが要求元クライアントノードに対してデータを転送する仕組みについて以下で説明する．Linux においてデータはページ単位でメモリ上にキャッシュされるため，クライアントノード間でのデータ転送の単位はページ単位とした．

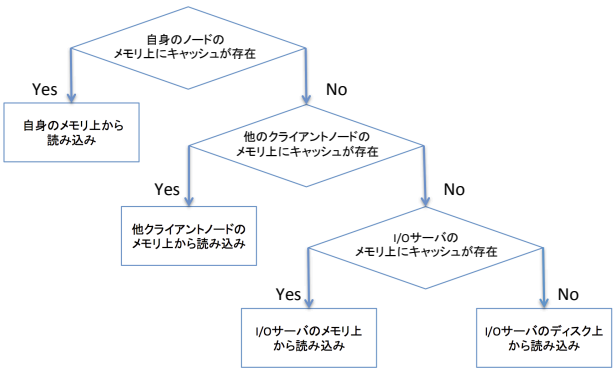

図 2 提案機構におけるクライアントノードのデータ読み込みの流れ
Fig. 2 Flow of data read by client nodes in the proposed mechanism

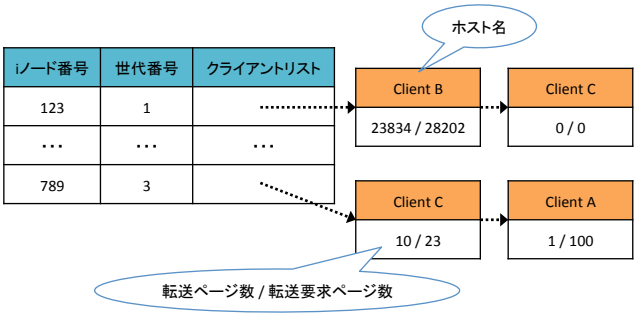

図 3 キャッシュ管理テーブル
Fig. 3 Cache management table

3.1.1 要求先クライアントノードの決定

クライアントノードは，メタデータサーバに問い合わせることで，データの要求先クライアントノードの情報を得る．メタデータサーバはファイルの i ノード番号と，当該ファイルをメモリ上にキャッシュしている可能性があるクライアントノードを対応付けるテーブル（**図 3**）を保持する．このテーブルをキャッシュ管理テーブルと呼ぶことにする．このテーブルのエントリは，i ノード番号，世代番号，クライアントリストへのポインタである．i ノード番号とはファイルシステム上のファイルを一意に特定する識別番号である．i ノード番号をキーとしてこのテーブルのエントリを参照できるものとする．世代番号とはファイルが更新されたときにインクリメントされる番号であり，ファイルの世代を表す．クライアントリストとは，i ノード番号に対応するファイルをメモリ上にキャッシュしている可能性のあるクライアントノードのリストである．クライア

© 2014 Information Processing Society of Japan

ントリストの要素は，ホスト名，転送要求ページ数，転送ページ数である．転送要求ページ数とは当該クライアントノードが転送を要求されたページの累計であり，転送ページ数とは当該クライアントノードが他のクライアントノードに対して転送したページの累計である．クライアントリストは**（転送要求ページ数）-（転送ページ数）**の値が小さい順にソートされている．**（転送要求ページ数）-（転送ページ数）**の値が小さいということは，他のクライアントノードからの転送を要求されたページの多くを，メモリ上にキャッシュしていたということを意味する．

クライアントノードは open したファイルの一部または全てをメモリ上にキャッシュする可能性がある．そのため，メタデータサーバはクライアントノードからファイルの open が要求されたときに，open を要求されたファイルの i ノード番号に対応するエントリをキャッシュ管理テーブルに追加する．また，メタデータサーバはクライアントノードからファイルの削除が要求されたときに，削除が要求されたファイルの i ノード番号に対応するエントリを削除する．

Client A が他のクライアントノードからデータを読み込む流れを**図 4** に示す．Client A はメタデータサーバに対して i ノード番号と世代番号で，それらに対応するファイルをメモリ上にキャッシュしている可能性のあるクライアントノードを問い合わせる（1）．メタデータサーバはキャッシュ管理テーブルを検索し，問い合わせを受けた i ノード番号と世代番号に対応するエントリが存在するかをチェックする（2）．エントリが存在した場合，クライアントリストを参照しリストの先頭に位置する *Client B* の情報を返す（3）．エントリが存在しない場合，存在しないことを通知する．メタデータサーバからの返答を受けた Client A は，Client B に対してデータを要求する（4）．要求を受けた Client B はページキャッシュを検索し，要求されたデータがページキャッシュに存在するかどうかをチェックする．要求されたページが存在する場合，それらのページを要求元の Client A に対して転送する（5）．要求されたページが存在しない場合，存在しないことを通知する．この場合，要求元の Client A は I/O サーバに対してデータを要求し（4’），I/O サーバからデータを読み込む（5’）．最後に要求元の Client A は要求先の Client B に要求したページ数と Client B から転送されたページ数の情報を，メタデータサーバに対して送信する（6）．メタデータサーバは，Client B の情報を更新しクライアントリストをソートし直す．

3.1.2 クライアントノード間データ転送

我々はクライアントノード間が InfiniBand により相互に接続された環境を想定している．そこで，InfiniBand の RDMA を使用してクライアントノード間で高速なデータ転送を行う．RDMA とは異なるノード間のメモリからメモリへと直接データを転送する技術である．データの転送は専用のデバイスにより OS を介さずに行われるため，高速な転送が可能である．

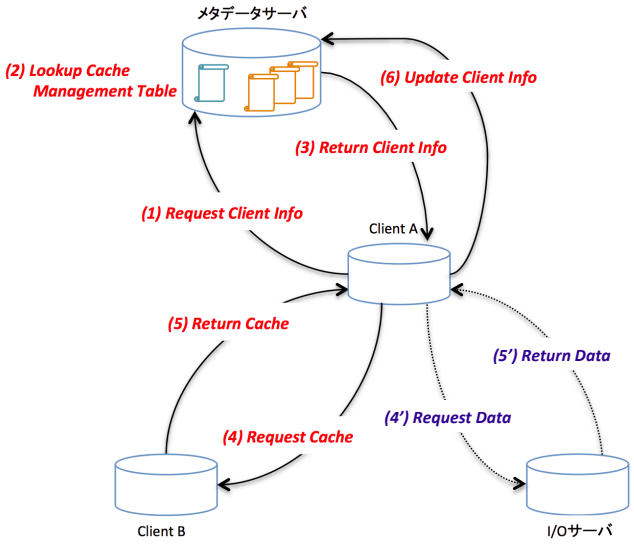

図 4 データを要求するクライアントノードの決定
Fig. 4 Determination of the client node to which a data request is sent

InfiniBand では，同一サブネット内のノードを特定するために *Local Identifier*（*LID*）という 16bit の識別子が用いられる．サブネットを超えてノードを特定するためには，*Globally Unique Identifier*（*GUID*）という 64bit の識別子が用いられるが，本論文では各ノードが同一サブネットに存在することを仮定しているため考慮していない．また，通信の端点として *Queue Pair*（*QP*）というオブジェクトが用いられる．これは，TCP や UDP におけるソケットに対応する．

クライアントノード間でデータの転送を行うためには，各クライアントノードの LID や QP の識別子である *Queue Pair Number*（*QPN*） などの情報が必要である．これらの情報は，クライアントノードが Gfarm ファイルシステムをマウントした際にメタデータサーバに登録される．クライアントノードがメタデータサーバに対して，データをキャッシュしている可能性があるクライアントノードを問い合わせた際（1），メタデータサーバからこれらの情報が返される設計とした．要求元クライアントノードは，メタデータサーバから返された情報に基づき，要求先クライアントノードへデータの転送を要求する．

3.2 実装

提案機構を実現するために Gfarm ファイルシステムのメタデータサーバ，I/O サーバ，カーネルモジュール，ライブラリ API に対して修正を加えた．

3.2.1 要求先クライアントノードの決定

キャッシュ管理テーブルは，i ノード番号をキーとするハッシュテーブルで実装されており，クライアントリスト

© 2014 Information Processing Society of Japan

表 1　測定環境
Table 1　Evaluation environment

CPU	Intel Xeon E5645
メモリ	48 GB
ストレージ	SAS 15,000 rpm 600 GB
OS	CentOS 6.3 64 bit
カーネル	2.6.32-279.14.1.el6.x86_64

は片方向連結リストにより実装されている．キャッシュ管理テーブルを操作するプロトコルを，クライアントノードとメタデータサーバとの通信プロトコルに追加した．

3.2.2　クライアントノード間データ転送

各クライアントノードは，マウント時に要求受付用 QP を作成する．そして，自身の LID と要求受付用 QP の QPN をメタデータサーバに対して登録する．

クライアントノードが他のクライアントノードに対してデータを要求する際，データ受信用 QP を作成する．そして，要求するデータの i ノード番号と世代番号とともにその QPN を要求先クライアントノードに対して送信する．

自身の接続受付用 QP に要求を受信したクライアントノードは，要求のあった i ノード番号と世代番号に対応するデータがページキャッシュに存在するかどうかをチェックする．要求されたページが存在する場合は，要求元クライアントノードのデータ受信用 QP に対してページを送信する．このとき送信するページは，要求元クライアントノードのページキャッシュに直接コピーされる．要求されたページが存在しない場合は，そのことを通知する．

InfiniBand では，複数の転送モデルが提供されており，一般的には *Reliable Connection*（*RC*）と *Unreliable Datagram*（*UD*）のどちらかが使用されている．RC は TCP のように通信ノード間で通信路を確立するコネクション型転送モデルであり，UD は UDP のように複数のノード間で通信路を確立せずに通信可能なコネクションレス転送モデルである．提案機構では全てのクライアントノード間でデータが送受信される可能性があるため，UD を使用した．RC を使用した場合，N ノード全てのクライアントノード間で相互に接続を確立するのに $N \times N$ の接続が必要となる．ノード数に応じて接続が増大すると性能がスケールしない可能性がある．

4.　評価

ベンチマークソフトを用いての I/O 性能の測定と，データインテンシブアプリケーションの実行時間の測定を行い提案機構を評価した．提案機構と通常の Gfarm ファイルシステムに対して測定を行い結果を比較した．測定には**表 1** に示す性能の計算機 8 ノードを使用した．全てのノードは同一 LAN 内に存在し，InfiniBand QDR 4X で相互に接続されている．

4.1　IOR

4.1.1　測定方法

I/O ベンチマークソフト IOR [14] を用いて提案機構を評価した．IOR は分散ファイルシステムの性能評価に広く用いられている．Gfarm ファイルシステムの構成はメタデータサーバ 1 ノード，I/O サーバ 1 ノード，クライアントノード 6 ノードである．IOR の実行時には以下のオプションを使用した．

```
$ mpirun -hostfile hosts IOR -t t_size -b b_size -F -C -e
```

各オプションの意味を説明する．-hostfile はプロセスを実行するノードを指定する．クライアントノード 6 ノードでそれぞれ 1 プロセスが実行されるように指定した．-t は I/O バッファのサイズを指定し，-b は 1 プロセスが読み書きするファイルサイズを指定する．-F は各プロセスがそれぞれ 1 ファイルを作成し読み書きを行うことを指定する．-C は他のプロセスが作成したファイルを読み込むことを指定する．-e は書き込み後ファイルを `close` する前にディスクへフラッシュすることを保証する．上記コマンドの実行により，各クライアントノードでは次のような処理が並列に行われる．

(1) 全てのクライアントノードで IOR プロセスが 1 プロセスずつ実行される．
(2) 1 プロセスが 1 つのファイルを生成する．*b_size* のデータをバッファサイズ *t_size* でシーケンシャルに書き込み，ディスクへフラッシュする．
(3) 他のプロセスが生成した *b_size* のファイルを，バッファサイズ *t_size* でシーケンシャルに読み込む．

t_size を 1MiB とし，*b_size* を 1GiB から 64GiB まで 2 倍ずつに増やしながら測定を繰り返した．全プロセスが読み書きする合計ファイルサイズは，6GiB から 384GiB までとなる．なお，測定は全てのファイルサイズで 1 回ずつ行い，測定毎に全てのノードのキャッシュをクリアした．キャッシュのクリアは`/proc/sys/vm/drop_caches` に 3 を書き込むことで行っている．

4.1.2　書き込み性能

書き込み性能の測定結果を**図 5** に示す．書き込みスループットは書き込みファイルの合計サイズ（*b_size* × プロセス数）を，いずれかのプロセスがファイルを `open` してから全てのプロセスが `close` するまでに要した時間で割ることで計算される．提案機構と通常の Gfarm で大きな差は観測されなかった．測定結果にばらつきがあるが，これは測定回数が 1 回と少なかったためだと考えられる．提案機構は書き込み処理に対して変更を加えていないため，書き込み性能には影響を与えない．ファイルの `open` 時と `close`

© 2014 Information Processing Society of Japan

時にメタデータサーバとの通信が発生するが，測定における IOR の処理フローでは，`open` と `close` は書き込みの前後で 1 回ずつしか行われないためオーバーヘッドは無視できる程度だと思われる．

4.1.3 読み込み性能

読み込み性能の測定結果を**図 6** に示す．読み込みスループットは読み込みファイルの合計サイズ（$b\_size \times$ プロセス数）を，いずれかのプロセスがファイルを `open` してから全てのプロセスが `close` するまでに要した時間で割ることで計算される．全てのファイルサイズにおいて，通常の Gfarm ファイルシステムと同等かそれ以上の読み込み性能が得られた．

まず，合計ファイルサイズが 2GiB から 24GiB において，提案機構では通常の Gfarm ファイルシステムに対して約 2 倍のスループットが得られた．測定に使用した計算機のメモリは 48GB であることから，各ノードのメモリ上にキャッシュ可能なデータは最大で 48GB である．ただし，各ノード上では OS やデーモンも起動しており，全てのメモリをデータのキャッシュのために使用することはできない．そのため，メモリ上にキャッシュすることができるデータは 48GB よりも少なくなる．合計ファイルサイズが 24GiB までデータは，1 ノードのメモリサイズに対して十分に小さいため，読み込み開始時点において I/O サーバとクライアントノードのメモリ上に全てキャッシュされていたと考えられる．よって，提案機構では全ての読み込みが他のクライアントノードのメモリから行われ，通常の Gfarm ファイルシステムでは I/O サーバのメモリから行われるため，どちらの場合もディスクへのアクセスは発生しない．I/O サーバとクライアントノード間のデータ転送は IP over InfiniBand（IPoIB）プロトコルで行われ，クライアントノードと他のクライアントノード間のデータ転送は RDMA で行われたことによりスループットに差が生じたものと考えられる．

次に，合計ファイルサイズ 48GiB と 96GiB において，提案機構では通常の Gfarm ファイルシステムに対して約 40〜60 倍のスループットが得られた．合計ファイルサイズが 48GiB 以上のデータは，前述した理由から読み込み開始時点において I/O サーバのメモリ上に全てがキャッシュされていなかったと考えられる．一方で，クライアントノード 1 ノードあたりが書き込んだデータのサイズは，合計ファイルサイズ 48GiB では 8GiB，96GiB では 16GiB であり，各クライアントノードは自身が書き込んだデータ全てをメモリ上にキャッシュしていたと考えられる．通常の Gfarm ファイルシステムでは，他のクライアントノードのメモリ上のデータを読み込むことはできないため，I/O サーバのディスクへのアクセスが発生しスループットが急激に低下している．提案機構では，他のクライアントノードのメモリ上のデータを読み込むことができるため，全ての読み込みがクライアントノードのメモリから行われスループットの低下は観測されなかった．

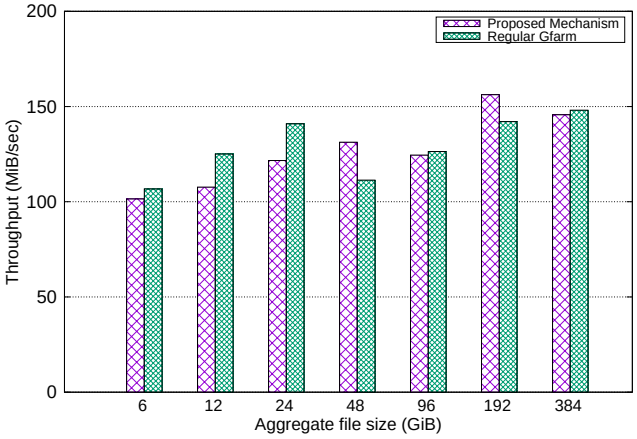

図 5 書き込み性能
Fig. 5 Write throughput

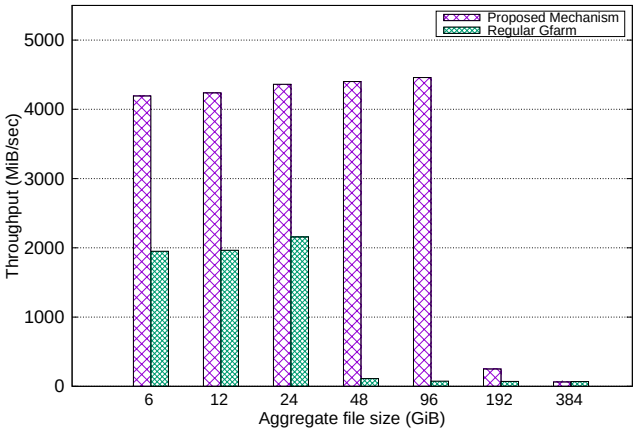

図 6 読み込み性能
Fig. 6 Read throughput

最後に，合計ファイルサイズ 192GiB と 384GiB において，提案機構では急激なスループットの低下が観測された．これはファイルサイズが増加したため，各クライアントノードが自身の書き込んだデータを全てメモリ上にキャッシュしきれなくなったためだと考えられる．

4.2 Montage

4.2.1 測定方法

複数の天体画像を 1 つに合成するアプリケーションである Montage [5] を用いて提案機構を評価した．Montage はデータインテンシブなアプリケーションであることが柴田らの研究 [15] により示されている．Montage を複数クライアントノード上で並列実行するために，ワークフローシステム Pwrake [16] [17] を用いた．次に示す Gfarm 構成でそれぞれ Montage を実行し，実行に要する時間を測定した．それぞれ，測定環境（a），（b）と呼ぶことにする．

© 2014 Information Processing Society of Japan

(a) メタデータサーバ 1 ノード，I/O サーバ 1 ノード，クライアントノード 6 ノード
(b) メタデータサーバ 1 ノード，I/O サーバ兼クライアントノード 7 ノード

Montage の入力データセットとして複数の FITS ファイルを与える．FITS は，天文学の分野において画像データを記述するために標準的に用いられるデータフォーマットである．測定に使用した 2 種類のデータセット X, Y を**表 2** に示す．測定環境（a）では全ての入力データを I/O サーバ 1 ノードに配置し，測定環境（b）では全ての I/O サーバ兼クライアントノードに均等に配置して測定を行った．

4.2.2 実行時間

データセット X を入力としたときの実行時間を**図 7**，データセット Y を入力としたときの実行時間を**図 8** にそれぞれ示す．なお，測定は各データセットで 3 回ずつ行い実行時間はそれらの平均値である．

まず，測定環境（a）において提案機構では通常の Gfarm ファイルシステムよりも実行時間が削減されている．実行時間の削減は，入力データセット X に対して 3.3%，入力データセット Y に対して 15.4%であり，入力データセットのサイズが大きいほうが提案機構の効果が高いことがわかる．提案機構で Montage を実行した場合，通常の Gfarm ファイルシステムであれば I/O サーバから読み込まれるはずのデータの一部が，クライアントノードのメモリ上のキャッシュから読み込まれる．クライアントノード間のデータの転送は RDMA で行われ，I/O ノードからデータを読み込むよりも高速であるため，実行時間が削減されたものと推測できる．また，測定環境（a）において I/O サーバは単一であるため，I/O サーバへのアクセスの集中が緩和されたものと推測できる．

次に，測定環境（b）において提案機構と通常の Gfarm ファイルシステムの実行時間に大きな差は確認できなかった．これは，測定環境（b）において入力データセットは複数の I/O サーバに分散して配置されているため，I/O サーバへのアクセスの集中が起こりにくかったことが要因であると推測できる．

4.2.3 キャッシュヒット率

提案機構においてデータセット Y を入力とし Montage を実行したときの，各クライアントノードにおける転送要求ページ数，転送ページ数，キャッシュヒット率を測定した．測定環境（a）の結果を**表 3**，測定環境（b）の結果を**表 4** に示す．

どちらの測定環境においても，93%から 99%程度の高いキャッシュヒット率を得た．提案機構におけるデータの要求先クライアントノードを決定する仕組みは効果が高いと言える．

表 2 Montage の入力データ
Table 2 Input Data for Montage

	ファイル数 (個)	合計サイズ (GB)
X	3512	6.9
Y	618	1.3

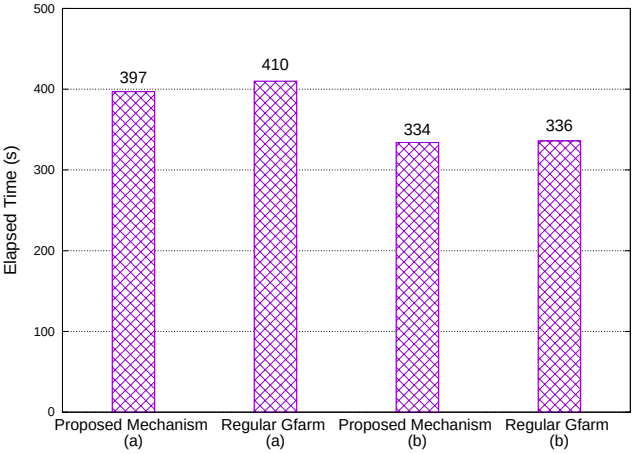

図 7 Montage の実行時間 (入力データセット: X)
Fig. 7 Elapsed time of Montage (Input data set: X)

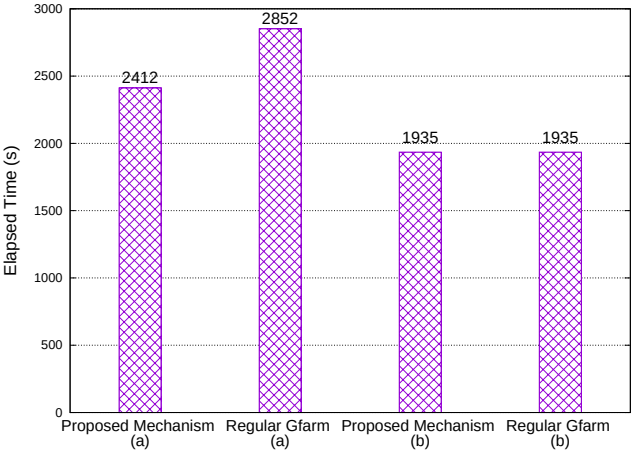

図 8 Montage の実行時間 (入力データセット: Y)
Fig. 8 Elapsed time of Montage (Input data set: Y)

表 3 測定環境 (a) おける各クライアントノードの転送ページ数・転送要求ページ数・キャッシュヒット率
Table 3 # of transferred and requested pages, and cache hit rate of each client node in evaluation environment (a)

クライアント	転送ページ数	転送要求ページ数	ヒット率（%）
1	8858996	9386542	94.38
2	9065744	9106166	99.56
3	9177107	9219760	99.54
4	8658233	8697004	99.55
5	9077877	9123589	99.50
6	8637025	8684031	99.46

© 2014 Information Processing Society of Japan

表 4 測定環境 (b) における各クラインとノードの転送ページ数・転送要求ページ数・キャッシュヒット率

Table 4 # of transferred and requested pages, and cache hit rate of each client node in evaluation environment (b)

クライアント	転送ページ数	転送要求ページ数	ヒット率（%）
1	7012587	7522149	93.23
2	7283617	7325078	99.43
3	7233916	7281514	99.35
4	7471332	7512118	99.46
5	6797149	6839780	99.38
6	7631569	7673038	99.46
7	7039715	7080464	99.42

5. 関連研究

Cooperative caching の既存手法について述べ議論する．

5.1 Direct Client Cooperatioin

Direct Client Cooperation [8] は，アクティブ状態のクライアントノードが，自身のメモリから溢れたキャッシュをアイドル状態のクライアントノードのメモリに転送する手法である．アイドル状態のクライアントノードがアクティブ状態になった場合，キャッシュは破棄される．実装においては，クライアントノードがアクティブ状態かアイドル状態かを判断する基準が必要となる．本手法は，クライアントノードの中でアイドル状態のノードの割合が高い場合において有効であると考えられる．しかしながら，この手法にはアクティブ状態のクライアントノードのキャッシュを，他のクライアントノードが利用できないという問題が存在する．我々が想定する環境において，クライアントノードは並列で計算や I/O 処理を行っており，アイドル状態のクライアントノードは存在しないため有効な手法ではない．

5.2 Greedy Forwarding

Greedy Forwarding [8] は，クライアントノードのメモリを 1 つの共有メモリのように扱う手法である．クライアントノードは自身のメモリ上にキャッシュが存在しなかった場合，ストレージサーバに対してキャッシュを要求する．要求を受けたストレージサーバは自身のメモリに要求されたキャッシュが存在する場合，当該キャッシュをクライアントノードに転送する．要求されたキャッシュが存在しない場合，要求元のクライアントノードとは別のクライアントノードに対して要求をリダイレクトする．この時，ストレージサーバは自身が保持するリストを参照し，要求をリダイレクトするクライアントノードを決定する．リストは予め決められたブロック単位で，データをメモリ上にキャッシュしているクライアントノードを管理する．単一のサーバがキャッシュのロケーションを管理するという点は提案機構と同じであるが，本手法は単一のストレージサーバを想定していると思われる．本手法を複数のストレージサーバを持つ分散ファイルシステムに適用する場合，ストレージサーバ間でリストの同期が必要となる．また，本手法と提案機構とではクライアントノードの管理単位が異なる．ブロック単位での管理はキャッシュのロケーションを正確にトレースすることができる一方で，管理のためのコストが大きい．提案機構ではブロック単位ではなくファイル単位で管理しているが，高いキャッシュヒット率を得ている．

5.3 Centrally Coordinated Caching

Centrally Coordinated Caching [8] は，クライアントノードのメモリを静的にローカルキャッシュ空間とグローバルキャッシュ空間へ分割する手法である．ローカルキャッシュ空間はクライアントノード自身のキャッシュで独占され，グローバルキャッシュ空間はクライアントノード間で 1 つの共有メモリのように扱われる．グローバルキャッシュ空間において，キャッシュは LRU で管理される．クライアントノードのローカルキャッシュ空間とグローバルキャッシュ空間に分割されているため，単一クライアントノードに同じデータのキャッシュが 2 つ作られる可能性がある．提案機構では，クライアントノードのキャッシュ空間を分割していないため，単一クライアントノードに同じデータのキャッシュが 2 つ作られることはない．

5.4 N-Chance Forwarding

N-Chance Forwarding [8] は，Greedy Caching に対して，重要度の高いキャッシュブロックを優先的にキャッシュするというポリシーを加えた手法である．単一のクライアントノードのみがメモリ上にキャッシュしているブロックを重要度が高いと見なし，*singlet* と呼んでいる．クライアントノードが singlet を破棄しようとした場合，当該 singlet はランダムに選ばれた他のクライアントノードに転送される．転送先のクライアントノードで当該 singlet にアクセスがなかった場合，N 回転送された後破棄される．ただし，転送先のクライアントノードでアクセスされた場合，転送回数はリセットされる．singlet の扱い以外は前述の Greedy Forwarding と同様の振る舞いをする．複数のクライアントノードがデータインテンシブアプリケーションを並列に実行するような環境において，singlet を追跡し優先的にキャッシュすることは必ずしも有効であるとは言えないと考えている．このような環境では，分散ファイルシステム上の同一ファイルに対して複数のクライアントノードがアクセスし，同一ファイルをメモリ上にキャッシュする場合がある．我々は，singlet よりもむしろこのような複数のクライアントノードに必要されるデータに対して負荷分散がおこるようにすることが重要であると考える．このような場合，単一のクライアントノードに対してデータの要

© 2014 Information Processing Society of Japan

求が集中するのを避けるため，複数クライアントノードに要求を分散すべきである．しかしながら，提案機構は現在このような機能を有していない．

6. まとめと今後の課題

本論文では Gfarm ファイルシステムに対して，クライアントノード間でキャッシュを共有する手法である cooperative caching の機構を実装し性能を評価した．データインテンシブアプリケーションである Montage を用いた評価では，単一の I/O サーバ上にデータが集中して配置された環境において実行時間が 13.4%削減された．しかしながら，I/O サーバがクライアントノードを兼ね，複数の I/O サーバ上にデータが分散して配置された環境においては提案機構の効果は確認できなかった．

ネットワークの性能は向上し続けており，InfiniBand の仕様を策定する標準化機構である InfiniBand Trade Association が公表する InfiniBand Roadmap [18] によると，InfiniBand のバンド幅は 1000Gbps に到達すると見込まれる．また，CPU の計算性能とストレージシステムの I/O の性能の差は広がり続けることが指摘されている [19]．これらの背景から，高性能科学技術計算の分野におけるデータインテンシブなアプリケーションの計算性能を向上させる手段として，cooperative caching はますます重要となることが予想される．本論文では Gfarm ファイルシステムを対象としたが，提案機構を単一のメタデータサーバと複数のストレージサーバから構成される分散ファイルシステムに対して適用することは可能であると考えられる．

今後の課題として次のようなものが挙げられる．提案機構の効果はアプリケーションの特性に依存する可能性があるので，他のデータインテンシブアプリケーションでも評価を行うことは課題である．また，提案機構ではクライアントノードがメタデータサーバに対して問い合わせを行い，データを要求するクライアントノードを決定している．そのため，クライアントノードの数が増加するとメタデータサーバに対するアクセスが集中し，性能がスケールしないことが予想される．この問題に対処することも課題として挙げられる．

謝　辞

本研究を行うにあたり，有益な助言を頂いた筑波大学建部修見准教授と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] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, pp. 257–275 (2010).
[2] Donovan, S., Huizenga, G., Hutton, A., Ross, C., Petersen, M. and Schwan, P.: Lustre: Building a File System for 1,000-node Clusters, *Proceedings of the Linux Symposium* (2003).
[3] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. and Maltzahn, C.: Ceph: A Scalable, High-performance Distributed File System, *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, USENIX Association, pp. 307–320 (2006).
[4] Babu, A.: GlusterFS, `http://www.gluster.org/`.
[5] Montage: `http://montage.ipac.caltech.edu/`.
[6] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. and Lipman, D. J.: Basic local alignment search tool, *Journal of molecular biology*, Vol. 215, No. 3, pp. 403–410 (1990).
[7] The Weather Research & Forecasting Model Website: `http://www.wrf-model.org/index.php`.
[8] Dahlin, M. D., Wang, R. Y., Anderson, T. E. and Patterson, D. A.: Cooperative Caching: Using Remote Client Memory to Improve File System Performance, *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, USENIX Association, pp. 267–280 (1994).
[9] Sarkar, P. and Hartman, J.: Efficient Cooperative Caching Using Hints, *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'96)*, pp. 35–46 (1996).
[10] Jiang, S., Petrini, F., Ding, X. and Zhang, X.: A Locality-aware Cooperative Cache Management Protocol to Improve Network File System Performance, *Proceedings of the 26th International Conference on Distributed Computing Systems*, IEEE, pp. 42–42 (2006).
[11] Huang, G., Chen, K., Wu, Y., Zheng, W. and Yue, Q.: Improving the System Capacity by Client Cooperation in Distributed File Service, *ChinaGrid Annual Conference (ChinaGrid)*, IEEE, pp. 156–163 (2012).
[12] Szeredi, E.: FUSE: Filesystem in Userspace, `http://fuse.sourceforge.net/`.
[13] Rajgarhia, A. and Gehani, A.: Performance and Extension of User Space File Systems, *Proceedings of the 2010 ACM Symposium on Applied Computing*, ACM, pp. 206–213 (2010).
[14] IOR HPC Benchmark: `http://sourceforge.net/projects/ior-sio/`.
[15] Shibata, T., Choi, S. and Taura, K.: File-access Patterns of Data-intensive Workflow Applications and Their Implications to Distributed Filesystems, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, ACM, pp. 746–755 (2010).
[16] Tanaka, M. and Tatebe, O.: Pwrake: A Parallel and Distributed Flexible Workflow Management Tool for Wide-area Data Intensive Computing, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ACM, pp. 356–359 (2010).
[17] Pwrake: `http://github.com/masa16/pwrake`.
[18] InfiniBand Roadmap: `http://www.infinibandta.org/content/pages.php?pg=technology_overview`.
[19] Zhao, D., Qiao, K. and Raicu, I.: HyCache+: Towards Scalable High-Performance Caching Middleware for Parallel File Systems, *IEEE/ACM CCGrid* (2014).

© 2014 Information Processing Society of Japan