情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2011-OS-118 No.17
2011/7/28

Gfarmのためのカーネルドライバへのキャッシュ機構導入の検討

石 黒 駿[†1] 村 上 じゅん[†1] 大 山 恵 弘[†1,†2]

Gfarm は，大容量，高信頼，高性能を実現する広域分散ファイルシステムである．Gfarm ファイルシステムは FUSE を利用して，Linux クライアントにマウントできる．FUSE とは，ユーザレベルで動作するファイルシステムを実現するフレームワークである．しかし，現在の FUSE では，Gfarm ファイルシステムをマウントしたとき、ファイルをオープンする度にページキャッシュを破棄するため，ページキャッシュを有効活用できていない．そこで本研究では，FUSE 内でのキャッシュの利用を最適化することにより，ページキャッシュを有効利用するキャッシュ機構を提案する．本論文では，提案機構の実装について述べ，その性能評価結果を示す．

An Investigation of Implementing Cache Mechanism in Kernel Driver for Gfarm

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

Gfarm is a global distributed file system that achieves large capacity, high reliability, and high performance. Gfarm file systems can be mounted on Linux clients by using FUSE, which is a framework for implementing user-level file systems. However, the current implementation of FUSE cannot make an effective use of page caches in the case a Gfarm file system is mounted because FUSE discards all page caches of a file when the file is opened. In this paper, we propose a cache mechanism that makes an effective use of page caches by optimizing the use of caches in FUSE. We describe the implementation of the mechanism and report the result of performance evaluation.

†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

1. は じ め に

大規模なデータを扱うために，分散ファイルシステムが利用される．分散ファイルシステムは，複数の端末のストレージにデータを配置し，それらをまとめて一つのストレージのように扱う機能を提供するファイルシステムである[1)2)3)4)5)]．多くの分散ファイルシステムでは，複数端末にデータを配置することによって，スループットや耐故障性，スケーラビリティを向上させている．Gfarm[6)] は NFS の代替ととして利用でき，大容量，高信頼，高性能を実現する広域分散ファイルシステムである．Gfarm は単一のメタデータサーバと複数のファイルサーバで構成される．ファイルのメタデータはメタデータサーバで一括して管理され，実際のファイルの内容はファイルサーバに保存される．クライアントはメタデータサーバやファイルサーバにアクセスすることで，Gfarm ファイルシステムを利用できる．Gfarm では，他のいくつかの分散ファイルシステムと同じく，ユーザは基本的には専用の API を通じてファイルシステムを利用する．しかしながら、ユーザレベルのファイルシステムを実装するためのフレームワークである FUSE[7)] を利用することにより，Gfarm が提供するファイルシステムを Linux クライアントにマウントできる．マウントした後には，open や read などのファイル操作のためのシステムコールによってファイルシステムを利用できる．

一般にファイルシステムでは，ファイルアクセスを高速化するために，一度読み込んだファイルの内容をメモリ上にキャッシュしている．Linux ではこのキャッシュはページ単位で作成されるため，ページキャッシュと呼ばれる．一般的に FUSE を利用する場合，`kernel_cache` オプションを利用してファイルシステムをマウントすると，ページキャッシュを使用するためファイルアクセスが高速化する．しかし，Gfarm ファイルシステムをマウントする際にこのオプションを利用するにあたっては二つの問題がある．第一の問題は，POSIX はもとより close-to-open コンシステンシでさえ保証されないことである．close-to-open コンシステンシとは，ファイルをクローズした後に，他のプロセスがそのファイルをオープンしたときに，ファイル内容の更新が観測できることを保証するセマンティクスである．このセマンティクスは，POSIX で定められたセマンティクスを緩めたものであるが，ある程度妥当である．複数のユーザが異なるマウントポイントに Gfarm ファイルシステムをマウントしている場合，キャッシュはマウントポイントごとに存在し，そのキャッシュが最新であるかどうかに関わらずキャッシュアクセスが行われるため，close-to-open コンシステンシを保

© 2011 Information Processing Society of Japan

証できない．第二の問題は，メモリの無駄な消費である．同一ファイルのキャッシュであっても，マウントポイントごとにキャッシュが作成されるため，その分だけ無駄にメモリが消費される．その結果，Gfarm ではデフォルトでは `kernel_cache` オプションをつけないでファイルシステムをマウントするようになっている．この場合，ファイルオープンの度にキャッシュが破棄されるため，ページキャッシュが有効利用されない．

そこで本研究では，Gfarm において，close-to-open コンシステンシを保証しながらページキャッシュを有効活用できるようにする機構を提案する．提案機構は，複数のマウントポイントから Gfarm ファイルシステム上のファイルにアクセスしたときに，そのファイルのキャッシュを共有する．同一のファイルに対して複数のキャッシュを生成しないことにより，close-to-open コンシステンシが保証されるとともに，メモリ使用量を節約できる．提案機構は，ファイルを更新するのは提案機構を組み込んだ端末に限られるなどの条件が成立するときに利用可能である．例えば，ある一つの端末が Gfarm のクライアントとして InfiniBand のような高速なネットワークで Gfarm ファイルシステムにアクセスできるようになっており，そこに複数のユーザがリモートログインして利用するといった利用法が想定できる．条件の詳細は 3 章で述べる．

本論文の構成は以下のとおりである．2 章では，Gfarm の概要を述べる．3 章では，FUSE と Gfarm のマウントについて述べる．4 章で提案機構の設計と実装について説明し，5 章でその評価を述べる．6 章では関連研究について述べ，7 章で本研究のまとめを述べる．

2. Gfarm と FUSE

2.1 Gfarm

Gfarm は，NFS の代替として利用可能な広域分散ファイルシステムである．Gfarm は，**図 1** で示すように，単一のメタデータサーバ，複数のファイルサーバによって構成される．メタデータサーバは，ファイルサイズやアクセス制御情報などのメタデータの他に，ファイルのオープン状態やファイルの保存場所，Gfarm を利用するユーザ名の管理を行う．またファイルサーバの負荷や利用可能容量などの情報の監視も行う．ファイルサーバは，Gfarm 上のファイルが実際に保存されるサーバである．ファイルは複製が作成され，それぞれ別のファイルサーバに保存される．これにより，耐故障性を向上させたり，同一ファイルへの並列アクセスによるスループットを向上させることができる．

ユーザは Gfarm にアクセスするために Gfarm ライブラリを利用する．このライブラリは，open，close，read，write，seek，stat，rename，unlink といった基本的な API の実

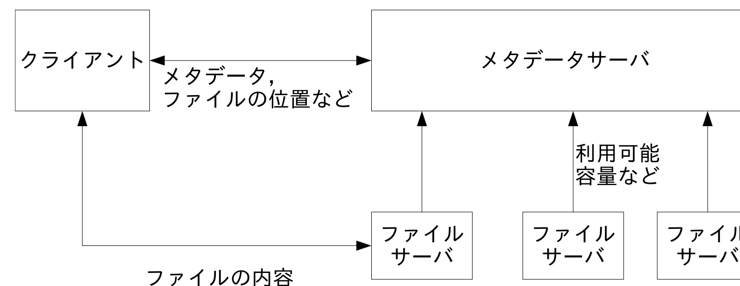

図 1 Gfarm のアーキテクチャ
Fig. 1 The architecture of Gfarm

装である．クライアントは，ファイルをオープンする際にはまずメタデータサーバへ問い合わせる。この問い合わせにより，クライアントはファイルが存在するファイルサーバを知る．次にクライアントは，ファイルサーバに対してオープンを要求し，以降はファイルサーバだけに対して read，write，seek などを要求する．ファイルをクローズする際には，メタデータサーバとファイルサーバ両方に対してクローズを要求する．このように，read，write のようなファイルアクセスについては，直接ファイルサーバにアクセスするため，メタデータサーバやネットワークの負荷を軽減できる．

2.2 FUSE

FUSE は，ユーザレベルのファイルシステムを実装するためのフレームワークである．FUSE は，カーネルモジュールである FUSE モジュールとユーザレベルファイルシステムを提供するデーモンにより，ユーザレベルファイルシステムを実現する．ユーザレベルファイルシステムを作成するには，このデーモンを実装すれば良い．デーモンで実装する処理は，ユーザレベルファイルシステムに対する open，close，read，write といった処理である．ユーザレベルファイルシステムに対して発行される open や read などのシステムコールは，FUSE モジュールが受け取る．そして FUSE モジュールはそれらのシステムコールの実際の処理をデーモンへ要求する．デーモンの処理が完了すると FUSE モジュールはその結果を受け取り，システムコールの結果としてその内容を返す．デーモンは多くの場合，ext3 などのローカルファイルシステムにアクセスする．FUSE を利用したファイルアクセスの例は**図 2** のようになる．

マウントは次のように行う．

```
$ ./userfs /tmp/user -o kernel_cache
```


© 2011 Information Processing Society of Japan

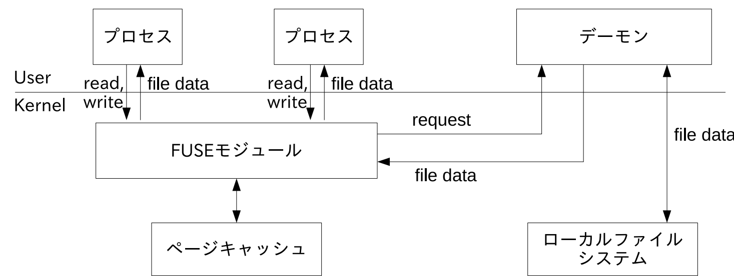

図 2　FUSE のファイルアクセス
Fig. 2　File access using FUSE

userfs がデーモンであり，/tmp/user がマウントポイントである．-o kernel_cache が FUSE に対する kernel_cache オプションの指定である．/tmp/user/以下には，ユーザレベルファイルシステムのディレクトリ階層が出現する．プロセスがユーザレベルファイルシステムのファイルに対して read システムコールを発行すると，FUSE モジュールがデーモンに対して read 要求を行う．デーモンはその要求に対して，バッファにデータを書き込むなどの適切な処理を行う．FUSE モジュールはそのデータを受け取り，プロセスへその結果を返す．FUSE モジュールとデーモンの通信は，/dev/fuse というスペシャルファイルを介して行われる．write，seek などの処理についても同様の通信が行われる．

FUSE は，ページキャッシュを利用してファイルの内容をキャッシュしている．ページキャッシュの読み書きや作成は，FUSE モジュールがカーネルで用意されている関数を呼び出すことでカーネル内部で自動的に行われる。ファイルオープンのときに作成されたキャッシュは，以降のファイルの読み書きに利用される．このキャッシュは，オプション無しでファイルシステムをマウントした場合は，ファイルのオープン毎に破棄される．一方，kernel_cache オプションを指定してマウントすると，オープン時にキャッシュを破棄しない．read 処理の場合，ページキャッシュに要求されたデータが存在するときは，FUSE モジュールはデーモンにデータを要求しない．FUSE モジュールがデーモンにデータを要求する場合，ファイルシステムにシステムコールを発行したプロセスのユーザモードとカーネルモード間のコンテキストスイッチの他に，そのプロセスとデーモン間のコンテキストスイッチ，デーモンでのユーザモードとカーネルモード間のコンテキストスイッチがあるため，最低 6 回のコンテキストスイッチが行われる．ページキャッシュがアクセスされる場合は，ファイルシステムにシステムコールを発行したプロセスのユーザモードとカーネルモード間のコンテキストスイッチのみ行われるので 2 回のコンテキストスイッチで済む．ネットワークファイルシステムのような，外部の要因によりファイルの内容が更新されるファイルシステムの場合，キャッシュの内容が最新ではない可能性があるため，通常は kernel_cache オプションをつけてファイルシステムをマウントしてはならない．

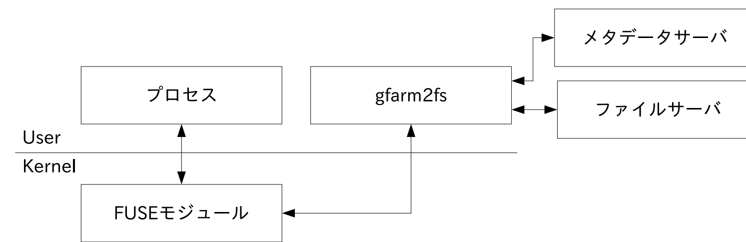

図 3　gfarm2fs による Gfarm ファイルシステムへのアクセス
Fig. 3　Accessing Gfarm file system by gfarm2fs

2.3　gfarm2fs

Gfarm ファイルシステムは gfarm2fs を用いて，クライアントの端末のファイルシステムにマウントできる．gfarm2fs は，FUSE を利用して Gfarm ファイルシステムをマウントするためのデーモンである．マウントすることにより，Gfarm ライブラリを利用していない一般的なアプリケーションも Gfarm ファイルシステムにアクセス可能である．gfarm2fs を用いて Gfarm ファイルシステムにアクセスする様子を**図 3** に示す．マウントされた Gfarm へのアクセスは，FUSE モジュール及び gfarm2fs を介して行われる．gfarm2fs は，FUSE モジュールからの要求に応じてメタデータサーバやファイルサーバと通信する．その結果を FUSE モジュールが受け取り，最終的に Gfarm にアクセスしたプロセスへ結果が返される．

gfarm2fs は，複数のマウントポイントに Gfarm をマウントできる．例えば，一つの端末にユーザ A,B,C がいるとすると，それぞれのユーザが/tmp/A，/tmp/B，/tmp/C に Gfarm をマウントする，といった利用ができる．マウントポイント以下のディレクトリ構造は Gfarm のファイルシステムのものとなり，各ユーザが同一のファイルを読み書きすることもできる．この場合，同一ファイルへのアクセスであってもそのファイルのキャッシュは FUSE によりマウントポイントごとに別々に作成される．

1 章で述べたとおり，複数のユーザが kernel_cache オプションを指定して gfarm2fs を用いて Gfarm をマウントしている場合，close-to-open コンシステンシが保証されない．これは，Gfarm 上の同一ファイルへのアクセスに対して，マウントポイントごとにファイル

ⓒ 2011 Information Processing Society of Japan

のキャッシュが作成されるため，キャッシュのみへのアクセスが行われると，他のユーザのファイル更新が他のユーザに対して反映されないからである．さらに，同一ファイルのキャッシュが複数存在することから，メモリの無駄な消費につながる．

3. 提案機構

本章では，複数のマウントポイントから参照される Gfarm 上の同一ファイルに対して，そのファイルのキャッシュを共有するキャッシュ機構の設計と実装を述べる．

3.1 仮定

本研究の提案機構にはいくつかの仮定がある．

- 同じ Gfarm ファイルシステムをマウントしている端末のうち，提案機構を組み込んだ端末は一つだけ存在する．ただし，読み込み専用でマウントするならば，提案機構を組み込んだ複数の端末が存在しても良い．
- Gfarm 上のファイルは，提案機構を介さずに更新されることはない．
- 各ユーザは，自分のホームディレクトリなどの異なるディレクトリに Gfarm ファイルシステムをマウントする．

提案機構のキャッシュは，Gfarm のクライアントの端末内のみで共有される．また，`kernel_cache` オプションを指定するとキャッシュアクセスのみ行われる場合があり，そのときはユーザは Gfarm 上のファイル更新を観測できない．ゆえに，ファイルを更新する端末は一つであり，かつ提案機構を介して更新しなければならない．

3.2 設計

提案機構は，FUSE モジュール内で動作する．提案機構の概要を順を追って説明する．まず Gfarm 上のファイルに対してオープン要求があった場合，そのファイルが既に別のマウントポイントからオープンされたことがないかを判定する．ファイルが同一かどうかの判断は，Gfarm ファイルシステム上でのファイルパスが同じかどうかで判定する．もし初めてそのファイルがオープンされた場合は，その情報を保持しておく．そうでない場合は，既にオープンされたファイルのキャッシュを利用するようにする．これにより，read，write などの処理は，共有したキャッシュを利用して行われる．提案機構では，メモリに空きがある限りキャッシュを保持する．キャッシュに対する更新は，提案機構を用いない FUSE の場合と同じタイミングで，Gfarm ファイルシステムに反映される．ファイルアクセスは，必ずキャッシュを経由して行われる．

Gfarm 上の同一ファイルへのアクセスに対して，マウントポイントごとにキャッシュが作られる原因は，そのファイルの inode がマウントポイントごとに異なるからである．一度オープンされたファイルの inode はメモリ上にキャッシュされる．二回目以降のファイルオープンでは，inode のキャッシュから inode を取得する．この inode のキャッシュはスーパーブロックごとに作成される．FUSE では，デーモンごとに異なるスーパーブロックを作成する．ゆえに，複数のマウントポイントから Gfarm ファイルシステムにアクセスする場合，それぞれのマウントポイントであるファイルが初めてオープンされる際に，inode のキャッシュにそのファイルの inode が存在しないため，新たに inode が作成される．

3.3 実装

Linux では inode は struct inode 型の構造体で表現されている．`struct inode` 型の構造体は，その inode が表すファイルのページキャッシュを管理するために，`address_space` 型の構造体とそのポインタを持っており，それぞれ `struct address_space i_data`，`struct address_space *i_mapping` として定義されている．`i_mapping` は `i_data` へのポインタである．提案機構では，この `i_mapping` の値を，最初にファイルをオープンしたユーザが利用する inode の `i_data` へのポインタに書き換えることで，キャッシュを共有している．この様子を図 **4** に示す．

同一ファイルのオープンかどうかの判定については，まず初めてオープンされたファイルのパスとその inode の `i_mapping` の値を覚えておく．次のオープンのときの要求ファイルパスと保存したファイルパスを比較することで，既に別のマウントポイントから同一ファイルがオープンされたかどうかを判定する．別のマウントポイントから既にオープンされていた場合は，その inode の `i_mapping` の値を書き換える．ファイルパスの比較は，高速化のためにファイルパスのハッシュ値を併用している．ハッシュ値が一致したときのみファイルパスを比較する．inode と `i_mapping` を管理するためのデータ構造を図 **5** に示す．`mapping_list` により，ファイルパスとそのハッシュ値，そのファイルが利用する `i_mapping` の値を管理する．さらにこのリストの各要素は，キャッシュを共有している inode のリストも持つ．

ファイルシステムをアンマウントすると，そこで利用していた inode が破棄される．もし，初めてファイルをオープンしたときの inode が破棄されると，その inode とキャッシュを共有している inode の `i_mapping` 変数の参照先がなくなってしまう．それを次のようにして回避する．まず同一ファイルを表す inode をリストとして持っておく．参照先の inode が破棄されるときは，キャッシュを共有している `inode` を一つ選び，その inode の `i_data` を参照するように，キャッシュを共有しているすべての inode の `i_mapping` を書き換える．これによりキャッシュの状態は，何もキャッシュされていない状態に戻る．しかしアンマウ

© 2011 Information Processing Society of Japan

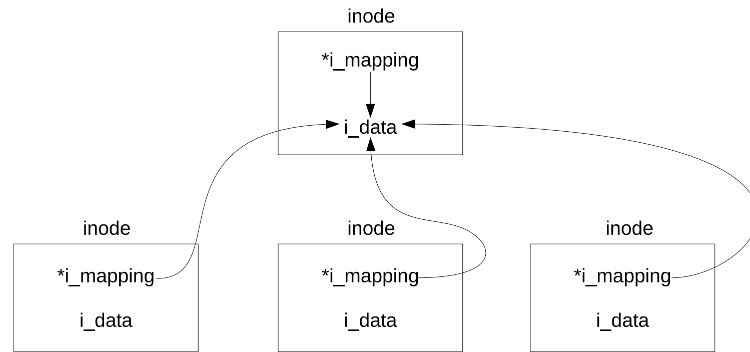

図 4　i_mapping の値の書き換え
Fig. 4　Changing i_mapping values

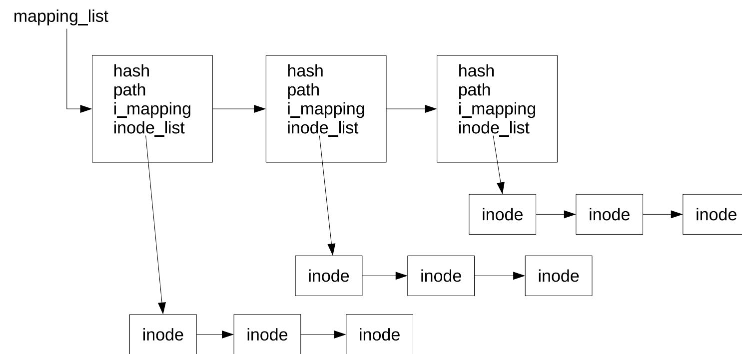

図 5　キャッシュ共有のためのデータを保持するデータ構造
Fig. 5　Data structures keeping data for sharing caches

ントは頻繁に行われないため，性能に問題はない．

`address_space` 型の構造体はその構造体を内部に持つ inode へのポインタを持っており，`struct inode *host` として定義されている．ファイルに書き込みを行うときには `host` が参照されるため，この値を正しい inode のものにしなければならない．そのために，write 要求の度に，`host` を現在利用している inode へのポインタに書き換えている．

4. 評　　価

4.1 実　　験

提案機構の性能を評価するために，二つの実験を行った．一つはキャッシュの利用によるファイルアクセス速度の測定であり、もう一つはキャッシュの共有によるページキャッシュ使用量の測定である．両方の実験を，以下の二つの環境で行った．

local　メタデータサーバ，ファイルサーバ，クライアントが同一の端末に存在する環境

remote　メタデータサーバとファイルサーバが同一の端末にありクライアントがリモートの端末に存在する環境

ファイルアクセス速度の測定実験では，以下の三つのケースについて測定した．

original　提案機構を用いずかつオプションを指定せずに Gfarm ファイルシステムをマウント

kernel_cache　提案機構を用いずかつ `kernel_cache` オプションを指定して Gfarm ファイルシステムをマウント

proposed　提案機構を用いて Gfarm ファイルシステムをマウント

それぞれのケースについて，Gfarm ファイルシステム上に 1GB のファイルを用意し，それを 4KB ずつ `read` システムコールで最後まで読み込むプログラムを実行して，そのプログラムの実行時間を計測する実験を行った．実行時間の計測は 1,000 回行い，その平均時間を求めた．ただし，ファイルのデータをページキャッシュに配置するために，計測を開始する前に一度プログラムを実行した．さらに，それぞれのケースについて 2 人のユーザが別々のマウントポイントに Gfarm ファイルシステムをマウントし，同一ファイルを読み書きしたときの動作を調べた．まず，一人のユーザがファイルを読み込んでその内容をキャッシュしておき，その後もう一人のユーザが同じファイルに対して書き込みを行ってクローズする．次に，最初にファイルの内容をキャッシュしたユーザがファイルをオープンする．このとき，このユーザがファイル内容の更新を観測できるかを調べた．ファイルの更新を観測できる場合は close-to-open コンシステンシが守られ，観測できない場合，close-to-open コンシステンシが守られていないことになる．

ページキャッシュ使用量の測定では，ユーザを 3 人用意してそれぞれ別のディレクトリに Gfarm をマウントし，1 人ずつ 1GB の同一ファイルを読み込んで，そのときのページキャッシュ使用量を測定した．ファイルの読み込みには、ファイルアクセス測定に用いたプログラムを利用した．ユーザ全員が Gfarm をマウントした直後の値，1 人目のユーザがプログラ

© 2011 Information Processing Society of Japan

ムを実行した直後の値，2 人目のユーザがプログラムを実行した直後の値，3 人目のユーザがプログラムを実行した直後の値を，free コマンドで測定した．提案機構を用いないときの測定は，kernel_cache ケースで行った．

実験環境は，local 環境では，CPU が Intel Core i7 2.93GHz，メモリが 8GB，OS が CentOS 5.6 kernel-2.6.18，Gfarm はバージョン 2.4.2 である．remote 環境では，メタデータサーバとファイルサーバの端末が，CPU が Intel Core i7 2.93GHz，メモリが 16GB でありクライアントの端末が，CPU が Core i7 2.93GHz，メモリが 8GB である．両方の端末とも，OS が CentOS 5.6 kernel-2.6.18，Gfarm のバージョンが 2.4.2 であり，両者の間は 100BASE-TX のネットワークにより接続されている．ファイルアクセス速度測定実験の結果を**表 1**，ページキャッシュ使用量測定実験の結果を**図 6** と**図 7** に示す．

4.2 考　　察

local 環境 remote 環境ともに kernel_cache ケースと proposed ケースでは，FUSE モジュールが gfarm2fs にデータを要求せずにキャッシュにアクセスするため，original ケースよりもファイルアクセスが高速だった．これは実験用のプログラムと gfarm2fs 間のコンテキストスイッチ回数の削減と FUSE モジュールと gfarm2fs 間のデータコピー削減によるものと考えられる．さらに提案機構は，kernel_cache ケースの場合と比べてほぼ同等のファイルアクセス速度を保っている．また，original ケースと proposed ケースではファイル内容の更新を観測した．すなわち，close-to-open コンシステンシが守られていた．しかし，kernel_cache ケースではファイルの更新を確認できず，close-to-open コンシステンシが守られていなかった．これは，kernel_cache ケースではキャッシュの内容が最新であることが保証されないことに起因する．特に，remote 環境では original のファイルアクセス速度が local 環境と比べて遅いが，これはネットワークの通信速度が遅いためである．さらに，ファイルの読み込み処理では，ファイルサーバがディスクからファイルを読み出す．そのときに，ファイルの内容はファイルサーバがアクセスするローカルファイルシステムのページキャッシュに配置される．したがって，local 環境ではファイルサーバがキャッシュからファイル内容を読み込むために，original の場合でも 1.7 sec 程度と比較的高速となったと考えられる．

local 環境では 1 人目のユーザがファイルを読み込むと，提案機構あり、提案機構なしの場合ともにページキャッシュの使用量が 2GB 増加した．2GB 増加した理由は，ファイルサーバがファイルを読み出すときにローカルファイルシステムのページキャッシュに 1GB のファイルをキャッシュし，さらに FUSE モジュールがそのデータを受け取ったときに再

表 1　1GB のファイルを 4KB ずつ read するプログラムの実行時間 (s)
Table 1　Execution time of the test program which reads a 1GB file in steps of 4KB

	original	kernel_cache	proposed
local 環境	1.7	0.26	0.27
remote 環境	93	0.27	0.27

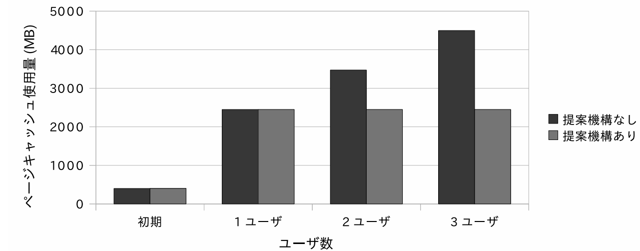

図 6　複数ユーザが 1GB の同一ファイルを読み込んだときのページキャッシュ使用量 (local 環境)
Fig. 6　Amount of page cache when multiple users read the same 1GB file (local)

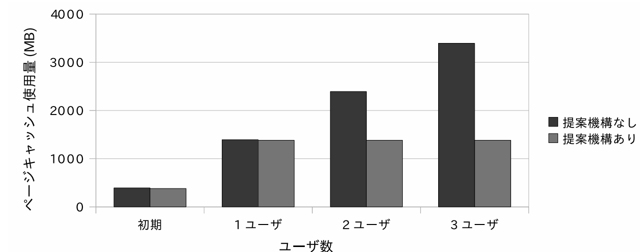

図 7　複数ユーザが 1GB の同一ファイルを読み込んだときのページキャッシュ使用量 (remote 環境)
Fig. 7　Amount of page cache when multiple users read the same 1GB file (remote)

びキャッシュするためであると考えられる．一方，remote 環境では 1 人目のユーザがファイルを読み込んだとき，提案機構あり、提案機構なしの場合ともにページキャッシュの使用量の増加は 1GB であった．これはローカルファイルシステムが利用されず，FUSE によるキャッシュのみ行われたためと考えられる．local 環境と remote 環境の両方の環境ともに，2 人目以降のユーザがファイルを読み込む場合，提案機構なしの場合はページキャッシュ使用量が 1GB ずつ増加したが，提案機構ありの場合は変化しなかった．提案機構なしの場合，

© 2011 Information Processing Society of Japan

ユーザがファイルを読み込む度にファイルのキャッシュが FUSE で作成されるため，1GB ずつ増加したと考えられる．一方提案機構ありの場合は，ファイルのキャッシュを共有するため，ページキャッシュの使用量は変化しなかったとみられる．

5. 関連研究

NFS[1) は，Sun Microsystems 社により開発された分散ファイルシステムであり，UNIX 系 OS で標準的に利用される．NFS を利用すると，リモートにある別の端末のディレクトリを自分の端末にマウントして利用できる．NFS では提案機構と同じくファイルの読み書きにページキャッシュを利用するが，NFS は Gfarm と比較して耐故障性が低い．

GFS[2) は，Google が開発した分散ファイルシステムであり，Google が自社のサービス提供や研究開発のために利用している．GFS は単一のマスタと複数のチャンクサーバで構成される．ファイルは 64MB のチャンクに分割され，チャンクサーバに保存される．それぞれのチャンクはデフォルトで 3 つの複製が作られる．マスタはファイルのメタデータやチャンクサーバの状態把握，クライアントへチャンクの位置情報の提供を行う．クライアントは，マスタへチャンクの情報を問い合わせ，マスタにより返されるチャンクサーバにアクセスし，チャンクをやりとりする．GFS は，ファイルの書き換えよりも追記が多い用途に対して高いパフォーマンスを発揮するように設計されている．GFS のオープンソースのクローンとして HDFS[8) が存在する．GFS へのアクセスは専用 API を用いるため，ページキャッシュを利用しないという点で提案機構と異なっている．

Ceph[3) は，カリフォルニア大学のプロジェクトにより開発された分散ファイルシステムである．Ceph は，メタデータサーバ，オブジェクトストレージクラスタ，クラスタモニタにより構成される．メタデータサーバはがメタデータを管理し，オブジェクトストレージクラスタにファイルの内容が保存される．クラスタモニタが，それらすべてのクラスタを監視する．特徴として，複数のメタデータサーバがメタデータを管理する点が挙げられる．現在はカーネルに組み込まれているため，FUSE を使わずマウントできる．Ceph は，計算端末とストレージを分けた運用を想定して設計されているが，Gfarm では計算端末とストレージが同じ場合に，ローカルディスク I/O を活用するように設計されている．

Lustre[4) は，Cluster File Systems 社により提供される分散ファイルシステムである．Lustre は単一のメタデータサーバと複数のオブジェクトストレージターゲットにより構成される．メタデータサーバはメタデータを管理し，オブジェクトストレージターゲットにファイルの内容が保存される．ファイルは分割されてオブジェクトストレージターゲットに保存される．FUSE によりマウントできる．通常の FUSE ではファイルシステム上の同一ファイルのキャッシュを共有しないという点で本研究と異なる．

GlusterFS[5) は，Gluster 社により提供される分散ファイルシステムである．GlusterFS は，クライアントとサーバで構成される．サーバではデーモンが動作しており，ローカルファイルシステムをそのまま「ボリューム」としてエクスポートする．クライアントは，サーバにアクセスしてそれらのボリュームを一つにまとめる．GlusterFS ではメタデータサーバのような中央サーバが存在しない．GlusterFS では Gfarm と同じく FUSE を用いてファイルシステムをマウントして利用する．通常の FUSE ではファイルシステム上の同一ファイルのキャッシュを共有しないという点で本研究と異なる．

GMount[9) は，東京大学で開発された分散ファイルシステムである．GMount では，GXP[10) を利用して複数端末上で SSHFS[11) を多重化した SSHFS-MUX を実行することにより，分散ファイルシステムを構築できる．GMount はユーザレベルで動作し，ユーザは簡単に分散ファイルシステムを作成できる．SSH-MUX，FUSE，ローカルファイルシステムのそれぞれにキャッシュが作成されるが，GMount はいかなるキャッシュ一貫性モデルも採用していないという点で本研究と異なる．

6. おわりに

本研究では，複数のマウントポイントから Gfarm 上の同一ファイルを参照したときに，そのファイルのキャッシュを共有するキャッシュ機構を提案した．この機構を用いることにより，Gfarm ファイルシステムを `kernel_cache` オプションを指定して複数のマウントポイントにマウントしたときに，close-to-open コンシステンシが保証されない問題と Gfarm ファイルシステム上の同一ファイルのキャッシュが複数作成される問題を解決した．実験の結果，提案機構を用いると `kernel_cache` オプションを指定して Gfarm ファイルシステムをマウントしても，close-to-open コンシステンシが保証されることがわかった．さらに，ファイルアクセスの速度も提案機構を用いない場合とほぼ同じであった．キャッシュによるメモリの消費量については，複数のマウントポイントから Gfarm ファイルシステム上の同一ファイルを読んだ場合についてもキャッシュが一つしか作られないため，メモリが節約されることがわかった．

今後の課題としては，クライアントが複数の端末に存在する状態で，close-to-open コンシステンシを保証しつつ，キャッシュを有効活用するキャッシュ機構が挙げられる．提案機構では，プロセスがキャッシュアクセスのみ行うと Gfarm 上のファイルに変更があった

© 2011 Information Processing Society of Japan

場合にそれに気づかない．したがって，提案機構を介せずにファイル更新があった場合は close-to-open コンシステンシを保証できない．異なる端末間でキャッシュ機構を利用するには，そのキャッシュが最新であるかどうかに常に注意を払う必要がある．そのためには，ファイルが更新されたらキャッシュを無効化するなどの手法が必要となる．

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) Pawlowski, B., Shepler, S., Callaghan, B., Eisler, M., Noveck, D., Robinson, D. and Thurlow, R.: The NFS Version 4 Protocol, *Network Working Group RFC 3010* (2000).
2) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *In Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp.20–43 (2003).
3) Sage 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) Schwan, P.: Lustre: Building a File System for 1,000-node Clusters, *In Proceedings of the 2003 Linux Symposium* (2003).
5) Babu, A.: GlusterFS. `http://www.gluster.org/`.
6) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New General Computing*, Vol.28, No.3, pp.257–275 (2010).
7) Szeredi, M.: FUSE: Filesystem in Userspace. `http://fuse.sourceforge.net/`.
8) Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *In Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies* (2010).
9) Dun, N., Taura, K. and Yonezawa, A.: An Ad Hoc and Locality-Aware Distributed File System by using SSH and FUSE, *In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing ant the Grid*, pp.188–195 (2009).
10) Taura, K.: GXP: An interactive shell for the grid environment, *In Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.59–67 (2004).
11) Szeredi, M.: SSHFS: SSH Filesystem. `http://fuse.sourceforge.net/sshfs.html`.

© 2011 Information Processing Society of Japan