Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

広域分散ファイルシステム Gfarm における ローカルストレージのファイルアクセスの高速化

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 は，クライアントの端末のストレージを束ねることで，巨大なストレージを提供する．さらに，Gfarm ファイルシステムは FUSE を利用してクライアントのファイルシステムにマウントできる．FUSE 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ためのフレームワークである．しかしながら，FUSE を用いて Gfarm ファイルシステムをマウントする場合，FUSE によるメモリコピーとコンテキストスイッチが I/O 性能のボトルネックとなる．そこで本研究では，マウントした Gfarm ファイルシステムへのアクセスが，クライアントのローカルストレージへのアクセスとなる場合に，メモリコピーとコンテキストスイッチを削減することで，ファイルアクセスを高速化する機構を提案する．本論文では，提案機構の実装について述べ，その性能評価の結果を示す．

Fast Speed File Access to a Local Storage 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 makes large storages by federating local file systems on client nodes. 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 has I/O bottlenecks due to many memory copies and context switches in case a Gfarm file system is mounted. In this paper, we propose a mechanism that achieves fast speed file access to a local storage for Gfarm by reducing memory copies and context switches. We describe the implementation of the mechanism and report the result of performance evalutation.

1. はじめに

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．分散ファイルシステムは，複数のノードのストレージにデータを配置し，それらのストレージ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)–8)]．Gfarm[2)] は，メタデータサーバとファイルサーバで構成される．ファイルのメタデータやファイルの位置情報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．ファイルサーバは複数あり，Gfarm ファイルシステムのクライアントと同じノードに配置することができる．Gfarm では他のいくつかの分散ファイルシステムと同様に，クライアントは専用の API を用いて Gfarm 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現するためのフレームワークである FUSE 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．マウント後は，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，Gfarm ファイルシステムを利用できる．

FUSE[9)] は，カーネルモジュールとユーザレベルのデーモンで構成される．カーネルモジュールは，ユーザレベルファイルシステムを利用するプロセスとデーモンの仲介を行う．デーモンは，ユーザレベルファイルシステムを実現するための実際の処理を担当する．たとえば，プロセスが read システムコールを発行すると，その要求はカーネルモジュールを経由してデーモンへ送られる．デーモンはバッファに適切なデータをコピーし，処理結果をカーネルモジュールを経由してプロセスへ返す．したがって，プロセスからユーザレベルファイルシステムへのアクセスが発生しデーモンへ処理が要求される際に，コンテキストスイッチやメモリコピーが生じる．また，FUSE を用いて Gfarm ファイルシステムをマウントしている場合，プロセスが read などでデータを要求すると，デーモンはファイルサーバへデータを取りにいく．もしクライアントと同じノードのファイルサーバに目的のデータが存在する場合は，デーモンはローカルストレージにアクセスすることになる．したがって，目的のデータがローカルストレージに存在する場合であっても，Gfarm ファイルシステムを利用するプロセスは，必ずデーモンを介してデータへアクセスしなければならない．前述

†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

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

の通り，カーネルモジュールとデーモンが通信を行う際には，コンテキストスイッチとメモリコピーが発生する．ゆえに，プロセスがローカルストレージに直接アクセスする場合と比べて I/O 性能が低下する．

そこで本研究では，カーネルモジュールからローカルストレージへ直接アクセスする機構を提案する．提案機構では，デーモンがローカルストレージへアクセスする場合には，カーネルモジュールからデーモンへ要求を行わずに，カーネルモジュールからローカルストレージへ直接アクセスする．提案機構により，コンテキストスイッチやメモリコピーを削減することで，ファイルアクセスの高速化が期待できる．

本論文の構成は以下の通りである．2 章では，Gfarm および FUSE について述べる．3 章で提案機構の設計と実装について述べ，4 章で評価結果を示す．5 章で関連研究について述べ，6 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，メタデータサーバとファイルサーバで構成される．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などの管理も行う．さらに，ファイルサーバの負荷状態や利用可能容量などの情報も監視する．ファイルサーバは，Gfarm ファイルシステム上のファイルを実際に保存するサーバである．ファイルの複製を作成し，それぞれ別のサーバに保存することも可能であり，耐故障性を向上させたり，同一ファイルの並列アクセスによりスループットを向上させることができる．

クライアントは Gfarm ファイルシステムを利用するために，Gfarm ライブラリを使用する．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 の実装である．クライアントはファイルを open する際には，まずメタデータサーバへ問い合わせる．これ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の位置を知る．続いてクライアントは，ファイルサーバに対して open を要求し，以降ファイルを close するまではファイルサーバのみに対して read，write などの要求を行う．ファイルを close 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 close を行う．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．

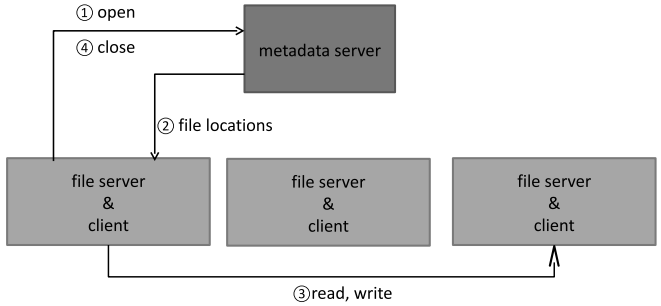

図 1 Gfarm のアーキテクチャ
Fig. 1 The architecture of Gfarm

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からなる．このデーモンを実装することにより，ユーザレベルのファイルシステムを作成できる．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，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，まず 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．デーモンは FUSE モジュールの要求にしたがい処理を行い，その結果を FUSE モジュールへ返す．FUSE モジュールはその結果を受け取り，システムコールの処理結果としてその内容を返す．なお，デーモンでは多くの場合，ext3 などのローカルファイルシステムにアクセスしたり，ネットワーク通信を行う．FUSE を利用したファイルアクセスの例を**図 2** に示す．

以下に，ユーザレベルファイルシステムをマウントする例を示す．

```
$ ./userfs /tmp/user
```

`userfs` がデーモンの実行ファイルであり，`/tmp/user` がマウントポイントである．マウントが完了すると，`/tmp/user` 以下に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．例として，プロセスがユーザレベルファイルシステムに対して 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．そして 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

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

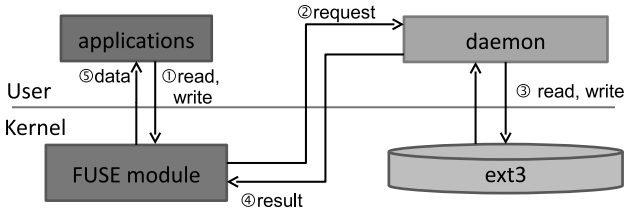

図 2 FUSE を利用したファイルアクセス
Fig. 2 File access using FUSE

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 や seek などの処理についても同様の通信が行われる．

FUSE モジュールがデーモンと通信する際には，コンテキストスイッチとメモリコピーが発生する．コンテキストスイッチは，ユーザレベルファイルシステムにアクセスするプロセスとデーモンのプロセスの間で発生する．コンテキストスイッチは FUSE モジュールとデーモンが通信する度に 2 回行われる．1 回目は，FUSE モジュールがデーモンへ要求を行いデーモンに処理が移るときに，システムコールを発行したプロセスからデーモンのプロセスへコンテキストスイッチが行われる．2 回目は，デーモンが処理を完了して FUSE モジュールへ結果を返し，FUSE モジュールに処理が移るときに，デーモンのプロセスからシステムコールを発行したプロセスへコンテキストスイッチが行われる．一方メモリコピーは，FUSE モジュールとデーモンの間で行われる．リクエストの種類などの情報の他に，read や write などのリクエストではバッファの内容のコピーが行われる．こうしたコンテキストスイッチやメモリコピーの影響から，ユーザレベルファイルシステムの動作速度はカーネル内で動作するファイルシステムと比べて遅くなる．

2.3 Gfarm ファイルシステムのマウント

Gfarm ファイルシステムをマウントするための FUSE のデーモンは，gfarm2fs である．クライアントは gfarm2fs を利用して Gfarm ファイルシステムを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でも，Gfarm ファイルシステムにアクセスさせることができる．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 ファイルシステムへのアクセスは，まず FUSE モジュールがその要求を受け取り，gfarm2fs へ処理を依頼する．gfarm2fs は，Gfarm

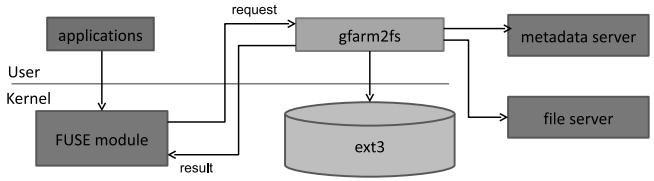

図 3 FUSE を利用した Gfarm ファイルシステムへのアクセス
Fig. 3 Access to Gfarm file system using FUSE

ライブラリを利用してメタデータサーバやファイルサーバと通信し，FUSE モジュールからの要求に応じて，適切な処理を行う．その結果を FUSE モジュールが受け取り，最終的に Gfarm ファイルシステムへアクセスしたプロセスに結果が返される．

gfarm2fs は要求されたファイルがローカルのファイルサーバに存在する場合，そのファイルサーバを優先してアクセスを要求する．Gfarm では，ファイルのアクセスがローカルのファイルサーバへのアクセスとなる場合，クライアントのプロセスはそのファイルサーバへ要求を行わずに，ファイルサーバが管理しているファイルを直接読み書きするようになっている．したがって gfarm2fs のプロセスは，ローカルのファイルサーバが管理しているファイルにアクセスする際には，実際には直接そのファイルへアクセスしている．これによりローカルのファイルサーバに配置されたファイルへのアクセスは高速化されている．しかし，FUSE の仕組みにより，マウントされた Gfarm ファイルシステムを利用するプロセスは，必ず gfarm2fs を介してファイルアクセスを行うことになる．ゆえに，FUSE 自体のオーバーヘッドが I/O 性能に影響する．

3. 提 案 機 構

2.3 節で述べた通り，目的のファイルがローカルストレージに存在していても，Gfarm ファイルシステムのマウントに FUSE を利用しているため，マウントされた Gfarm ファイルシステムを利用するプロセスは gfarm2fs を介してそのファイルを読み書きすることとなり，そのファイルを直接読み書きすることはできない．そこで提案機構では，gfarm2fs がローカルストレージのファイルを直接読み書きするときには，FUSE モジュールから直接そのファイルを読み書きする．これにより，gfarm2fs と FUSE モジュールが通信しないためコンテキストスイッチやメモリコピーを削減することができ，I/O 性能を向上させることができる．さらに，ローカルストレージで利用される ext3 などのローカルファイルシステムは，

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

ファイルの内容をキャッシュとして保持しているが，一方で FUSE でもファイルの内容をキャッシュしている．したがって，gfarm2fs がローカルストレージのファイルにアクセスする際にはローカルファイルシステムでのキャッシュと FUSE でのキャッシュの 2 つのキャッシュが存在しており，無駄にメモリを消費している．提案機構では，FUSE ではキャッシュせず，直接ローカルファイルシステムへアクセスするためメモリが節約されている．

3.1 設　　計

提案機構は，gfarm2fs がローカルストレージへアクセスする場合では，read システムコールおよび write システムコールが発行された場合に，FUSE モジュールからローカルストレージのファイルを直接読み書きする．提案機構の動作を**図 4** と**図 5** に示す．提案機構は，open システムコールが発行されたとき，および read システムコールと write システムコールが発行されたときに動作する．まず，open システムコールが発行されると，通常どおり gfarm2fs に open リクエストが要求される．gfarm2fs は，メタデータサーバへアクセスし，目的のファイルが配置されたファイルサーバの位置を知る．次に gfarm2fs は，ファイルサーバへアクセスする．ここで，ファイルサーバがローカルの場合，gfarm2fs はローカルストレージのファイルを直接 open する．そして，そのファイルのファイルディスクリプタを open リクエストの結果とともに FUSE モジュールへ返す．FUSE モジュールは，そのファイルディスクリプタを利用して，gfarm2fs が open したファイルの構造体を取得する．さらに，その構造体をシステムコールを発行したプロセスが open を要求したファイルの構造体と関連付けておく．

read および write システムコールが発行されたときは，そのシステムコールの対象となるファイルを調べて，もし gfarm2fs が open しているファイルと関連付けられていれば，gfarm2fs が open しているファイルに対して直接 read や write の処理を行い，その結果を返す．関連付けられていない場合は，元のファイルに対する read および write の処理を行う．なお，マウントされた Gfarm ファイルシステムに対して，read システムコールと write システムコール以外のシステムコールが発行された場合は，通常の FUSE と同じ動作をする．

3.2 実　　装

提案機構は，Gfarm ライブラリ，gfarm2fs，FUSE に実装した．FUSE に関しては，FUSE モジュール及び FUSE のデーモンを実装する際に利用する FUSE ライブラリを拡張した．

3.2.1 Gfarm ライブラリ

Gfarm ライブラリを利用するプロセスが直接 open するローカルストレージのファイルのファイルディスクリプタは，Gfarm ライブラリによって隠蔽されている．したがって，

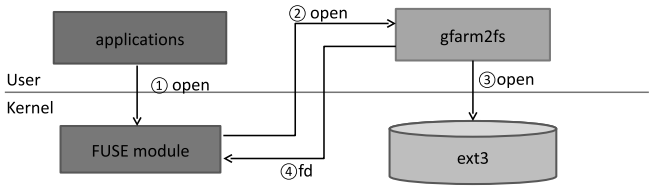

図 4　open 時の処理
Fig. 4　Processing for an open operation

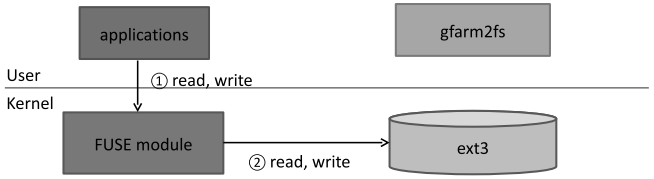

図 5　read と write 時にローカルファイルシステムに直接アクセスする様子
Fig. 5　Direct access to local file system for read and write operations

gfarm2fs の実装ではそのファイルディスクリプタを直接参照することはできない．そこで Gfarm ライブラリに，指定したファイルのファイルディスクリプタを返す API を追加した．gfarm2fs はこの API を利用することにより，直接 open しているローカルストレージのファイルのファイルディスクリプタを取得できる．なお，リモートのファイルサーバへのアクセスとなる場合は，Gfarm の仕様によりファイルディスクリプタには −1 が設定される．

3.2.2 gfarm2fs

gfarm2fs では open リクエストの処理を拡張した．通常，gfarm2fs では open リクエストを受けとると，メタデータサーバのみと通信し結果を返す．したがって，ファイルサーバと通信しない．しかし，ファイルディスクリプタの値が設定されるのは，初回のファイルサーバとの通信時である．そこでメタデータサーバとの通信のあとに，ファイルサーバと通信することにより，ファイルディスクリプタが設定されるようにする．ファイルサーバとの通信では，ファイルサーバに対してファイル位置を先頭にセットする seek 要求を行う．これによりファイルディスクリプタが設定される．そして設定されたファイルディスクリプタの値を，新たに追加した Gfarm ライブラリの API を用いて取得する．

FUSE デーモンの open リクエストの処理では，FUSE モジュールへ情報を伝えるための構造体へのポインタが引数で与えられる．その構造体のメンバに値を設定することにより，

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

ファイルのアクセス権限などの情報を FUSE モジュールへ伝えることができる．提案機構ではこの構造体にファイルディスクリプタの値を設定するためのメンバを追加して，そこにファイルディスクリプタの値を代入することにより，FUSE モジュールにファイルディスクリプタの値を返す．

3.2.3 FUSE

gfarm2fs が open リクエストの処理結果を FUSE モジュールへ返す際に，ファイルディスクリプタを一緒に渡すために，データを受け渡すために利用する構造体に，ファイルディスクリプタの値を保存するためのメンバを追加した．この変更は，FUSE モジュールおよび FUSE ライブラリに対して行った．

FUSE モジュールは，open システムコールが発行されると，通常どおり gfarm2fs へ open リクエストを送り，結果が返るまで待機する．gfarm2fs の結果は，`/dev/fuse` に gfarm2fs が書き込みを行うことによって FUSE モジュールに伝えられる．`/dev/fuse` に書き込みが行われると，FUSE モジュールの関数が呼び出され，そこで gfarm2fs が`/dev/fuse` に書き込んだ内容が，FUSE モジュールのバッファへコピーされる．この処理は，gfarm2fs のプロセスのコンテキストで行われるので，この関数の実行中に gfarm2fs のプロセスの構造体を取得する．結果の転送が完了すると，再び FUSE モジュールの open の処理が再開される．そして FUSE モジュールは，受け取った結果からファイルディスクリプタの値を取得する．ここで，ファイルディスクリプタの値が −1 である場合は，gfarm2fs がリモートのファイルサーバへアクセスすると判断する．そうでない場合は，gfarm2fs が`/dev/fuse` に書き込んだ際に取得したプロセスの構造体とファイルディスクリプタの値を利用して，gfarm2fs が open しているローカルストレージのファイルの構造体を取得する．プロセスの構造体を利用するのは，この構造体にはそのプロセスが現在 open しているファイルの構造体が登録されているためである．

ファイルの構造体の型は `struct file` であるが，この構造体は `void *`型の `private_data` というメンバを持っており，これはカーネルモジュールで自由に利用してよいことになっている．FUSE では，FUSE が扱うファイルの構造体の `private_data` に，`struct fuse_file` 型の構造体へのポインタを設定している．この値の設定は，FUSE モジュールの open 処理時に行われる．そこで提案機構では，`struct fuse_file` 型の構造体に `struct file *`型のメンバを追加し，ここに gfarm2fs が open しているファイルの構造体へのポインタを設定することにより，FUSE で扱うファイルの構造体から gfarm2fs が open しているファイルの構造体を参照できるようにする．もし gfarm2fs がリモートのファイルサーバへアクセスする場合は，FUSE が扱うファイルの構造体へのポインタを設定する．この様子を**図 6** に示す．

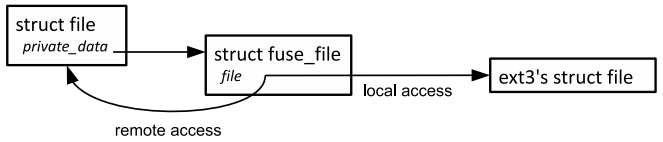

図 6 ファイル構造体の管理
Fig. 6 Management of data structures for files

FUSE は自身を fuse ファイルシステムとしてカーネルに登録しており，fuse ファイルシステムのファイルに対するオペレーションは，`struct file_oprations` 型の構造体に登録されている．FUSE では read システムコールと write システムコールが発行されたときは，`generic_file_read` 関数や `generic_file_write` 関数などの汎用的なカーネル関数を呼び出す．提案機構では，read システムコールや write システムコールが発行された場合にこれらの汎用関数を直接呼び出さず，提案機構で用意した関数を呼び出すようになっている．さらに，ローカルストレージで利用されるローカルファイルシステムである ext3 や ext4 では，read システムコールや write システムコール発行時には，FUSE と同様の汎用のカーネル関数を呼び出す．そこで提案機構で用意した read や write のオペレーションを担当する関数では，引数で渡されたファイルの構造体から gfarm2fs が open しているファイルの構造体を取得し，取得したファイルの構造体に対して改めて汎用関数を呼び出す．read システムコール発行時の処理の例を以下に示す．

```
ssize_t fuse_generic_file_read(struct file *filp, char __user *buf,
    size_t, len, loff_t *ppos){
    struct fuse_file *ff;
    ff = filp->private_data;
    return generic_file_read(ff->file, buf, len, ppos);
}
```

`filp` が FUSE が扱うファイルの構造体へのポインタであり，そのメンバである `private_data` を参照することにより，`struct fuse_file` 型の構造体をたどり，最終的に `ff->file` として，gfarm2fs が open しているファイルの構造体を取得している．なお，gfarm2fs がリモートのファイルサーバへアクセスする場合は，`ff->file` は `filp` と同様である．これにより，

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

gfarm2fs がローカルストレージのファイルを直接読み書きする場合でも，リモートのファイルサーバにアクセスする場合でも動作できる．write の場合も同様に，FUSE が扱うファイルの構造体から gfarm2fs が open しているファイルの構造体を取得し，write 用の汎用関数を呼び出す．

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．一つはファイルアクセス速度を計測するためのベンチマーク測定であり，もう一つはキャッシュの使用量の測定である．

ファイルアクセス速度の測定実験には，IOzone を用いた．実験では，100GB の一時ファイルに対する write と read のスループットを測定した．write と read について，シーケンシャルアクセスとランダムアクセスについてそれぞれ測定した．write 及び read は 8MB 単位で行った．測定は，提案機構を用いた場合と用いない場合およびローカルファイルシステムに対して行った．なおローカルファイルシステムは，ext3 である．

キャッシュの使用量測定では，ファイルを読み込むプログラムの実行前後のページキャッシュの使用量を測定した．プログラムは，1GB のファイルを read システムコールですべて読み込み，終了する．このプログラムを実行する前と実行した直後に，`free` コマンドでページキャッシュの使用量を測定した．測定は提案機構を用いた場合と用いない場合についてそれぞれ行った．

実験は，4 台のノードで構成されるクラスタで行った．うち 1 台がメタデータサーバのノードであり，残りの 3 台はファイルサーバおよびクライアントのノードである．残りの 3 台のノードから 1 つを選び，提案機構を導入する前と導入した後でファイルアクセス速度およびページキャッシュ使用量を測定した．各ノードは，CPU は Intel Xeon 2.40GHz が 2 つ，メモリは 48GB，HDD は 600GB 15000rpm，OS は Cent OS 5.6 64bit（kernel-2.6.18-238.el5），Gfarm のバージョンは 2.4.2，FUSE のバージョンは 2.7.4，gfram2fs のバージョンは 1.2.3 である．ファイルアクセス速度の測定結果を**図 7**，キャッシュ使用量の測定結果を**図 8** に示す．

4.2 考　　察

提案機構を用いた場合，提案機構を用いない場合と比べて，シーケンシャル read を除いてファイルアクセス速度が向上した．提案機構を用いた場合の結果は，ローカルファイルシステムに直接アクセスする場合とほぼ同じ程度となった．この結果から，gfarm2fs を介し

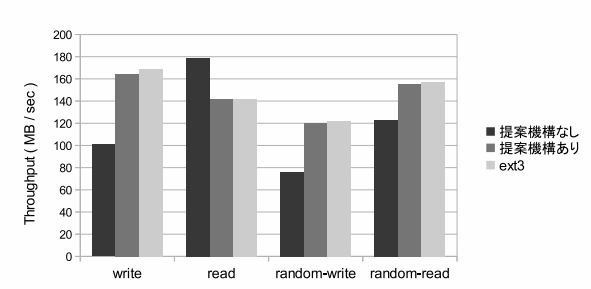

図 7 シーケンシャル write 及び read，ランダム write 及び read のスループット
Fig. 7 Throughput of sequential write and read, random write and read

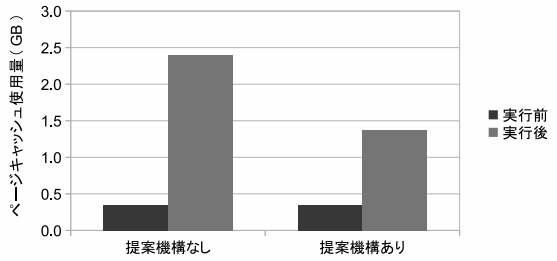

図 8 1GB のファイルを read するプログラムの実行前と実行後のページキャッシュ使用量
Fig. 8 Amount of page cache before and after reading a 1GB file

てローカルファイルシステムにアクセスする際には，大きなオーバーヘッドがあることがわかった．さらに，その主な原因としてコンテキストスイッチおよびメモリコピーがあることがわかった．なおシーケンシャル read のスループットが，提案機構なしの場合にもっとも高い理由としては，カーネルのファイル先読みアルゴリズムが，FUSE デーモンを経由する場合と経由しない場合で変化するためと考えられる[10]．

ページキャッシュについては，提案機構を用いた場合は用いない場合と比べて半分のメモリ使用量となった．これは，提案機構を用いた場合はローカルファイルシステムのキャッシュのみが利用されるのに対して，提案機構を用いた場合はローカルファイルシステムのキャッシュに加えて FUSE モジュールによるキャッシュが存在するためであると考えられる．

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

5. 関連研究

NFS[4] 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 は基本的に，ファイルシステムを提供する NFS サーバとマウントを行うクライアントは異なるノードであるが，Gfarm は同一ノードにサーバとクライアントの両方が存在できる．

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，メタデータクラスタ，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．メタデータクラスタ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がファイルの内容を保持する．さらに，クラスタモニタがそれらのすべてのクラスタを監視する．Gfarm と異なる特徴として，メタデータを管理するメタデータサーバが複数存在することが挙げられる．現在 Ceph は Linux カーネルに組み込まれており，FUSE を利用せずにマウントできる．また，Ceph は計算ノード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 は計算ノードとストレージを同じノードにした場合に，ローカルストレージを積極的に活用することで性能を向上させている．

GFS[1] は Google が，自社のサービス提供や研究のために開発した分散ファイルシステムである．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．1 つのファイルは 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 つの複製が作成され，異なるチャンクサーバに配置される．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，クライアントへのチャンクの位置情報の提供を行う．クライアントは，マスタにチャンクの情報を問い合わせ，マスタから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 へのアクセスは専用 API を用いる必要があるが，GFS のオープンソースのクローンである HDFS は FUSE を利用してマウントできる．GFS ではクライアントとチャンクサーバのノードが異なるが，本研究ではクライアントとファイルサーバが同一ノードに存在する環境を想定している．

Lustre[5] 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．Luster はメタデータサーバと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ファイルの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．Gfarm とは異なり，オブジェクトストレージターゲットは単にストレージを提供するだけであり，クライアントのノードとは分かれている．

GlusterFS[6] 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へアクセスし，ボリュームを一つにまとめて利用する．GlusterFS ではメタデータサーバのような中央サーバは存在しない．Gfarm と同じく FUSE を用いてマウントできるが，ローカルファイルシステムへアクセスする場合も FUSE デーモンを経由するという点で，本研究と異なる．

GMount[8] 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[11] を利用して複数ノード上で SSHFS[12] 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構築できる．SSHFS を用いるため FUSE を利用するが，ローカルファイルシステムへのアクセスには FUSE デーモンの経由が不可欠なため，本研究と異なる．

6. おわりに

本研究では，マウントされた Gfarm ファイルシステムへのアクセスにおいて，FUSE のデーモンである gfarm2fs がローカルファイルシステムへ直接アクセスする場合，gfarm2fs を介さずに FUSE モジュールからローカルファイルシステムへアクセスする機構を提案した．提案機構を用いた結果，gfarm2fs がローカルファイルシステムへアクセスする場合で，ファイルアクセス速度が向上した．また，提案機構を用いない場合，gfarm2fs がローカルファイルシステムにアクセスすると，ローカルファイルシステムでのキャッシュと FUSE でのキャッシュの 2 つのキャッシュが存在していたが，提案機構を用いることにより FUSE ではキャッシュを作らないようにした．

今後の課題として，リモートのファイルサーバへアクセスする場合についても gfarm2fs を経由しない機構が挙げられる．現在，gfarm2fs がリモートのファイルサーバからデータを取得する場合は，そのデータはクライアントの OS カーネル内のバッファから gfarm2fs のメモリ領域にコピーされ，それがさらに FUSE モジュールのバッファにコピーされる．gfarm2fs を経由せず，カーネルモジュールから直接リモートのファイルサーバにアクセスできれば，カーネル内でのメモリコピーのみで済むので，メモリコピーの回数を減らせる．

Vol.2011-ARC-197 No.17
Vol.2011-HPC-132 No.17
2011/11/28

このために，FUSE を用いずに，Gfarm をマウントするためのカーネルドライバの開発を検討中である．

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) 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).
2) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New General Computing*, Vol.28, No.3, pp.257–275 (2010).
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) 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).
5) Schwan, P.: Lustre: Building a File System for 1,000-node Clusters, *In Proceedings of the 2003 Linux Symposium* (2003).
6) Babu, A.: GlusterFS. `http://www.gluster.org/`.
7) 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).
8) 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).
9) Szeredi, M.: FUSE: Filesystem in Userspace. `http://fuse.sourceforge.net/`.
10) Rajgarhia, A. and Gehani, A.: Performance and Extension of User Space File Systems, *In Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 206–213 (2010).
11) 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).
12) Szeredi, M.: SSHFS: SSH Filesystem. `http://fuse.sourceforge.net/sshfs.html`.