RDMAを用いた仮想マシンイメージストレージに向けて

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

概要：大量のデータを読み書きするデータインテンシブサイエンスの分野ではHadoopと呼ばれるデータ解析プラットフォームが利用される．Hadoopは大量の計算機を利用して分散処理を行う．そのため，管理コストの観点から，物理的な計算機の代わりに，多数の仮想マシン（VM）を利用する事例が存在する．事実，Amazon EMRクラスタと呼ばれるサービスでは，ユーザは自分が使う分だけのVMを立ち上げて利用することができる．ここで，Hadoopを高速化するため，VMのディスクI/Oを高速化することは重要である．Hadoopではデータを HDFSと呼ばれる分散ファイルシステムに格納するが，計算結果の中間ファイルはそれぞれのVMのディスクに直接保存される．そのため，ディスクI/Oの高速化はHadoopのジョブの高速化につながる．本研究では，大量のVMインスタンスを格納でき，かつ，高速なディスクI/Oを実現するVMイメージ向けストレージを提案する．特徴は，VMのディスクI/Oの転送にInfiniBandのRDMAによるゼロコピー通信を利用していることにある．これにより，CPUの負荷が少なく，かつ，高速なVMのディスクI/Oを実現できると期待できる．本稿では，プロトタイプの実装と評価を行ったので報告する．

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

Abstract: The field of data-intensive science that handles a large amount of data, employs the Hadoop data analysis platform. Since Hadoop uses many computers to analyze data, many Virtual Machines (VMs) are utilized from the aspect of the computer management instead of using physical machines. In fact, the Amazon EMR cluster generates just as much VMs as a user wants to use Hadoop. It is important to speed up disk I/O of VM. Although Hadoop utilizes HDFS that is distributed file system to store the input data, the intermediate data is stored into VM disks directly. Therefore, if we can increase VM disk I/O speed, it also increase speed Hadoop jobs. We propose a new VM storage which allows us to store many VM instances and offer high speed VM disk I/O. Since our VM storage has a characteristic that utilizes InifniBand RDMA Zero-copy, we can achieve low CPU-loading and high-speed VM disk I/O. In this paper, we report examinations and an implementation of the VM storage.

1. はじめに

大量のデータを読み書きするアプリケーションを利用する，データインテンシブサイエンスの分野では，使い勝手の良いデータ解析プラットフォームが要求される．

Hadoop[1] はこのような要求にこたえるためのデータ解析プラットフォームの一つである．並列に処理を行う`map`という並列スケルトンと，結果を一つにまとめる`reduce`という並列スケルトンという二つのプリミティブを組み合わせることで，プログラマは複雑な並列プログラミングを容易に行うことができる．

Hadoopの利用環境には仮想化技術が大いに活用される．大量のマシンを利用するHadoopでは，必要に応じて物理マシンと同等の能力を持つ仮想マシンを作り出したり，また破棄することができるVMM（Virtual Machine Monitor）を利用すると便利である．事実，仮想化を利用したHadoopクラスタにはいくつかの事例がある．例えば，Amazon EMRクラスタでは，必要な台数分だけHadoopがインストールされたVMクラスタを起動することができるサービスであり，従量課金制で利用することが可能である．また，Cloudera社はHadoop環境がすぐに使えるVMイメージを配布している．

ここで，Hadoopの計算ノードが動作するVMのディスク

1 電気通信大学
The University Electro-Communications
2 独立行政法人科学技術振興機構, CREST
JST, CREST
a) kazushi@inf.uec.ac.jp
b) sasashin@ol.inf.uec.ac.jp
c) oyama@inf.uec.ac.jp

I/O を高速化を考えることは極めて重要である．Hadoop には専用の HDFS[2] と呼ばれるファイルシステムが存在し，解析対象のデータと出力結果はこの HDFS 上に保存されるのが一般的である．しかし，途中の計算結果は HDFS には保存されず，Hadoop が動作するノードのディスクドライブに直接保存されるのが一般的な構成である．そのため，Hadoop の計算結果を高速化するためには，HDFS の高速化のみならず Hadoop の計算ノードが動作する VM のディスク I/O 自体も高速化することが必要である．

まとめると，Hadoop のための VM ストレージには，VM に対して高速なディスク I/O を提供するのと同時に，大量の VM イメージを格納することができる大容量のクラスタファイルシステムを考える必要がある．

われわれは RDMA (Remote Direct Memory Access) を用いたスケーラブルな VM イメージ格納用クラスタファイルシステムを提案する．想定する利用シーンとしては，Hadoop クラスタを構成する大量の VM イメージはすべてここに格納され，本システムから直接 VM イメージをマウントして起動することを想定している．類似の VM 用クラスタファイルシステムとしては，Sheepdog[3] などがあげられる．これは大量のマシンを TCP/IP を使用して相互接続し，巨大な VM ストレージプールを提供することができるファイルシステムである．一方，提案システムでは，VM 上でリード/ライトが行われるたびに InfiniBand の RDMA でデータが転送される点が特徴である．そのため，TCP/IP を用いた既存の VM 用クラスタファイルシステムとくらべて，RDMA のゼロコピー通信によりホストマシンの CPU に負荷をかけることなく，高速にデータを転送することも期待できる．

提案システムを実現するために，本研究では Gfarm[4] に対して拡張を行い，プロトタイプ実装を行った．より具体的には，Gfarm を InfiniBand の RDMA に対応した上で，仮想マシンモニタである KVM/QEMU[5], [6] が RDMA を用いて直接 Gfarm 上にある VM イメージに対してリードライトできるように改造を行った．Gfarm とは，多数の計算機を束ねることで巨大な単一のファイルスペースを提供することができる分散ファイルシステムである．計算機を追加した分だけ，容量をスケールアップすることができるため大量かつ巨大なファイルサイズの VM イメージを格納する必要がある Hadoop の仮想マシンクラスタ環境には適切な選択である．

評価を行うにあたって，RDMA 通信で Gfarm と KVM/QEMU を接続したものと，TCP/IP を使って Gfarm と KVM/QEMU を接続したものとの二種類を比較した．評価の結果，RDMA 通信によるゼロコピー通信の効果を確認することが出来，RDMA を用いた VM 向けのファイルストレージの実装は，VM のディスク I/O を向上できる可能性があることを期待できる結果となった．

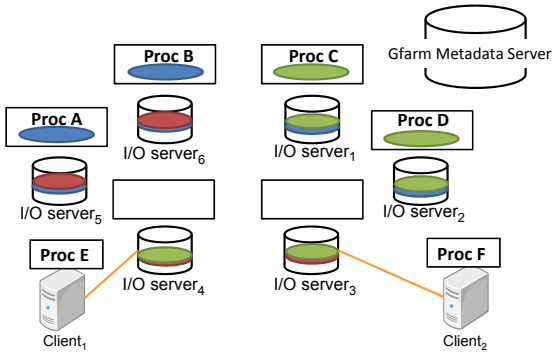

図 1　Gfarm のアーキテクチャ
Fig. 1　Gfarm architecture

2. Gfarm ファイルシステム

Gfarm 分散ファイルシステムは，Lustre ファイルシステム [7] などとは異なり，計算を行うアプリケーションと分散ファイルシステムのデータを保存するサーバを同一のノードで動作させることにより，アプリケーションがデータに効率的にアクセスできるという利点を持つ分散ファイルシステムである．そのため，Gfarm では，クライアントとディスクを束ねるノードを同一にすれば，それらの間を接続する高速なインタフェイスを用意する必要がないと言った利点が存在する．

図 1 に Gfarm のアーキテクチャ図を示す．この図が示すように，ディスクを束ねるノード（I/O server）は計算ノードになることもでき，計算ノード上に配置されたジョブ（図中の四角）はデータ（図中の円盤）に直接アクセスすることができる．もちろん，ディスクを束ねるノードとデータを利用するクライアントは別々に用意して運用することも可能である．

Gfarm は，ファイルの open 状態や，どの I/O server にファイルが実際に格納されているかと言ったメタデータを管理するサーバ (gfmd: 図 1 の Gfarm Metadata Server) とファイルを実際に格納する I/O server (gfsd) と，Gfarm ファイルシステムを利用するクライアントの三つから構成される．Gfarm は専用の Gfarm API で open, close, read, write, seek, stat, rename, unlink と言ったファイル操作を行うことができる．また，gfarm2fs と呼ばれるツールを用いることで，FUSE [8] 経由でマウントを行い，通常のアプリケーションで Gfarm を利用することも可能である．

クライアントがファイルを open する時は，まずはじめに gfmd にたいして open リクエストが送られる．この時，open 対象のファイルがどの gfsd に存在しているかが gfmd から送られてくる．次に，gfmd が返してきた gfsd に対して改めて open を行う．gfmd にアクセスするのは open と close 時のみである．read, write 時には gfsd へのアクセスのみが発生し，gfmd へのアクセスは発生しない．そのため，ファイルを実際に読み書きしている時には gfmd への

負荷はかからない．

Gfarm は libgfarm と呼ばれるライブラリを提供しており，gfsd, gfmd, gfarm2fs はこれを利用している．libgfarm には Gfarm を利用するための入出力インタフェイスがすべて実装されており，このライブラリを利用することで，容易に Gfarm を利用することができる．後に詳述するが，われわれはこの libgfarm のリードライト内で行われている TCP/IP ソケット通信を InfiniBand の RDMA に置き換えることで，提案システムを実現した．

3. InfiniBand のアーキテクチャ

InfiniBand[9] は高い信頼性，可用性，保守性を持つ高速なインターコネクトであり，HPC（ハイパフォーマンス・コンピューティング）や基幹系で主に使われる．計算機に対して HCA（InfiniBand host channel adapters）と呼ばれるインタフェイスカードを接続し，これを InfiniBand 専用ケーブルで接続することで計算機同士を接続する．この HCA は Ethernet でいうところの NIC（ネットワークカード）に相当するものである．

InfiniBand はキューベースモデルの通信方法を採用している．InfiniBand の RDMA で通信を行う際には Queue Pair（QP）と呼ばれるものをローカル側とリモート側に作る必要がある．これは TCP/IP におけるソケットに相当する．QP には Queue Pair Number（QPN）と呼ばれる 24 ビット幅のユニークな番号が割り当てられ，これで QP を識別することができる．QP は Send Queue（SQ）と Receive Queue（RQ）を一つづつ持っており，ここに Work Request（WR）と呼ばれるメッセージが積まれる．送信側の SQ に積まれた WR は受信側の RQ に入れられることになる．InfiniBand ではこの WR を QP に積むことで，データの送信を行うことになる．WR はメモリポインタの配列が含まれており，不連続な複数のメモリ領域を WR に含めて一度に送信することができる．

転送の完了やエラー通知は Completion Queue（CQ）と呼ばれるもので管理される．QP に積まれた WR の転送が完了すると，CQ に WR の完了通知が Completion Queue Entries（CQE）というデータ構造で積まれる．また，エラーの場合では当該 WR のエラー通知が CQ に積まれる．プログラマーはこの CQ を見ることで，QP に積んだ WR の転送が完了したか，また，エラーがでて送信出来なかったかを知ることができる．

また，InfiniBand には，Reliable service と Unreliable service の二つがあり，送信時いつ CQ に対して CQE が積まれるかが異なってくる．Unreliable service の場合は，HCA が送信要求の WR を受信側に投げた時点で CQ に完了の WR が積まれる．一方で，Reliable service の場合はリモートから Ack が帰ってきた時点で CQ に対して WR が積まれる．これは TCP/IP における UDP と TCP の関係に似ている．

InfiniBand にはいくつかの転送モデルがあり，どれを使うかはプログラマの選択に委ねられている．われわれはそのうち本研究に関わりの強い SEND オペレーション，RDMA WRITE オペレーションと RDMA READ オペレーションについて解説することにする．

はじめに，SEND オペレーションについて解説する．プログラマーは送信時には `ibv_post_send()` と呼ばれる関数を用いて WR を QP の SQ に積む．この時 WR のメモリ領域を指定する場所（`sg_list`）に転送したいデータがあるメモリのポインタ領域を指定する．そして，`ivb_poll_cq()` によって送信が完了したか，エラーが出たかのステータスを CQ から取り出す．受信側は `ivb_post_recv()` という関数を用いて，QP の RQ に WR を積み続ける．そして `ivb_poll_cq()` によって受信が完了したのか，エラーが出たのかのステータスと送信されたデータを CQ から取り出す．本オペレーションではリモートの受信側は `ibv_post_recv()` で WR を積み続けないと，ローカルの送信側がエラーになってしまうことに注意しなければならない．これが SEND オペレーション の基本的な流れである．

次に，RDMA WRITE オペレーションについて解説する．この転送モデルでは，リモート側の特定のメモリアドレス空間をピンダウンし，その上で，ローカル側がリモート側のピンダウンしたメモリ領域に対して，直接データを RDMA で流し込むモデルである．ローカル側が `ivb_post_send()` で送信したいデータのメモリ領域を指定する．この時，QP の SQ に積む WR 内部に書き込みを行いたいリモート側のピンダウンしたメモリアドレスを指定する．このリモートのメモリアドレスは事前に何らかの手法でリモート側とローカル側で共有する必要がある．RDMA WRITE オペレーションの大きな特徴はリモート側は `ibv_post_recv()` を呼んで QP の RQ に WR を積み続ける必要がないということにある．リモート側は何もする必要はない．ローカル側が勝手にデータを流し込んでくれるモデルである．

最後に，RDMA READ オペレーションについて解説する．この転送モデルでは，上述した RDMA WRITE オペレーションとは逆であり，リモート側でピンダウンした領域をローカル側が読みに行くモデルである．ローカル側が `ibv_post_send()` で，データを受信するメモリ領域を指定しデータを受信する．リモート側は何もする必要はない．ここでの注意は，ローカル側は `ibv_post_recv()` ではなく，`ibv_post_send()` でデータを受信するメモリ領域を指定する点である．

以上が InfiniBand の基本的なアーキテクチャである．

4. 設計

提案手法は，VM のリードライトを VMM 側でフック

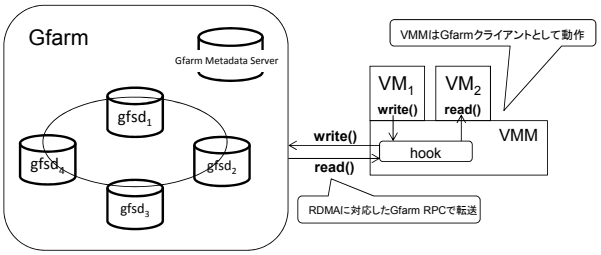

図 2　提案システムの全体図
Fig. 2　System overview

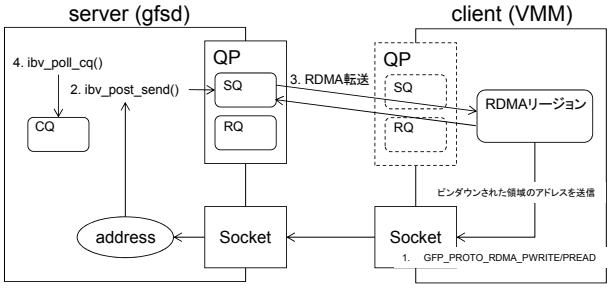

図 3　RDMA を使った VMM 側からリードライト
Fig. 3　read/write from VMM using RDMA

し，その上で，フックしたリードライトを RDMA に対応した Gfarm に対して送り込むという手法をとっている．これを実現するために，われわれは，VMM のリードライトのフックと Gfarm に対して RDMA でデータを送受信できる拡張機能を持たせた．本章ではそれらの設計について述べていく．

4.1　システム全体像

図 2 に提案システムの全体図を示す．提案システムを実現するにあたって重要なのは VM のリードライトを VMM 側でフックし，その上で，フックで得られたリードライトを Gfarm に対して RDMA で転送することである．2 章でも述べたとおり，Gfarm には gfarm2fs というユーティリティがあり，これを使えば，FUSE 経由で Gfarm ファイルシステムをローカルのファイルシステムとしてマウントして使うことができ，VMM 側でのフックは不要になる．しかし，われわれは，FUSE が介在した時に発生するオーバーヘッドを考慮して，VM のディスク I/O のアクセスを VMM 側でフックしそのアクセスをそのまま Gfarm API で転送する手法を取った．つまり，VMM 自体が Gfarm のクライアントになるアーキテクチャを取った．

4.2　Gfarm の RDMA 対応

次に，Gfarm の RDMA 対応について述べる．

Gfarm と VMM（Gfarm クライアント）の通信には二つのコネクションを使用する．一つ目がギガビットイーサネットのコネクションであり，この通信にはリード要求，ライト要求，その他各種情報交換といった，Gfarm と VMM 間のコマンドをやりとりするためのコネクションである．このやりとりは通常の TCP/IP 通信として通信が行われる．

二つ目が，InfiniBand で RDMA を通信するためのコネクションである．3 章でも述べたとおり，InfiniBand には Reliable service と Unreliable service という二つの手法がある．これらは更にコネクションが必要なタイプである Reliable Connection, Unreliable Connection と，コネクションが不要な Reliable Datagram, Unreliable Datagfarm の 4 つに分類される．今回，われわれは Reliable Connection を使用した．そのため，事前にコネクションを確立する必要がある．接続には LID, QPN, PSN と言った情報が必要である．これらの情報を交換するため，上述した TCP/IP 通信を利用する．具体的には新たに `GFS_PROTO_RDMA_EXCH_INFO` という RPC を追加した．クライアント（VMM）が gfsd に接続してきた時，InfiniBand RDMA とは別に，イーサネットの通信を使って RDMA のコネクションを確立するために必要な各種情報（LID, QPN, PSN）を gfsd とクライアント（VMM 側）で交換する．次に，RDMA によるリードライトがどのように VMM と gfsd 間で行われるかについての解説を行う．

4.2.1　RDMA を使った VMM 側からのライト

図 3 に，RDMA を使った VMM 側からのライトについて解説する．上述したように，サーバ（gfsd）側とクライアント（VMM）側にそれぞれ一つずつ QP を作って InfiniBand のコネクションを作る．コネクションを張る際に VMM 側にも QP が必要である，そのため，VMM 側にも QP を作成する．また，それとは別に，TCP/IP によるソケット通信のコネクションも作る．

Gfarm のライト要求である `GFP_PROTO_RDMA_PWRITE` が VMM 側から gfsd 側へ来たとする．この通信は通常の TCP/IP 通信で行われる．この時，VMM は自分が書き込もうとするデータのあるアドレス (図中の RDMA リージョン) を `GFP_PROTO_RDMA_PWRITE` に含めて転送する（図中の 1）．サーバ側は受けたアドレスを元に WR を作り，`ibv_post_send()` 関数を使って，その WR を QP の SQ に積む．この時，転送のタイプは RDMA READ リードオペレーションとする．すると，VMM 側の RQ には積まれず，gfsd は VMM 側の RDMA リージョンを直接リードすることが可能になる．VMM 側は `ibv_post_recv()` を呼び続ける必要がなく，この転送において一切の CPU パワーを消費しない．最後に，転送が完了したかどうかを gfsd 側が `ibv_poll_cq()` を使いポーリングする．**少しややこしいが，VMM にとってのライト要求が来た時，それは gfsd にとってのリードになる点に注意する必要がある．**転送が完了した時は，ポーリングしていた `ibv_poll_cq()` から完了をしらせる CQE を得ることができるので，その時点

で GFP_PROTO_RDMA_PWRITE を戻り値とともに，VMM 側へ返す．これでライトの処理は完了する．

4.2.2 RDMA を使った VMM 側からのリード

次に，図 3 を使って，RDMA を使った VMM 側のリードについて解説する．

Gfarm のリード要求である GFP_PROTO_RDMA_PREAD が VMM 側から gfsd 側に来たとする．この通信は TCP/IP 通信で行われる．この時，VMM 側はリードに使用するバッファのアドレス，すなわち gfsd からのデータを受け取るバッファのアドレス（図中の RDMA リージョン）を Gfarm の RPC である GFP_PROTO_RDMA_PREAD に含めて転送する（図中の 1）．サーバは受け取ったアドレスを元に，WR を作り，ibv_post_send() 関数を使ってそのアドレスを QP の SQ に積む．この時，転送のタイプは，RDMA WRITE オペレーションとする．すると，VMM 側の RQ には積まれず，gfsd は VMM 側の RDMA リージョンに直接データを書き込むことができる．この転送モデルにおいて VMM 側は ibv_post_recv() を呼び続ける必要はなく，転送に置いては一切の CPU パワーを消費しない．最後に，転送が完了したかどうかを gfsd 側が ibv_poll_cq() を使って CQ をポーリングする．**同様に少しややこしいが，VMM にとってのリードは，gfsd にとってのライトになる点に注意する必要がある．**転送が完了した時は，ポーリングしていた ibv_poll_cq() から完了をしらせる CQE を得ることができるので，その時点で GFP_PROTO_RDMA_PREAD を戻り値とともに，VMM 側に返す．これでリードの処理は完了である

リードもライトも，VMM 側は一切の CPU パワーを消費せず全くの受け身で，gfsd 側が RDMA を利用してデータを転送している点について注目する必要がある．これにより VMM 側が重い処理を行っていたとしても，高速に VMM 側はデータをリードライトすることができると期待できる．

5. 実装

実装は libgfarm と KVM/QEMU をそれぞれ改造することで行った．KVM/QEMU の実装はよく構造化されており，KVM/QEMU に対してブロックデバイスドライバを追加することで独自の VM イメージフォーマットのサポートを追加することができる．われわれは，KVM/QEMU のブロックデバイスドライバを開発することで VM のディスクへのリードライトをすべてフックし，Gfarm 側へ転送するように実装を行った．

また，Gfarm の RDMA 対応に関しては，すべて libgfarm 側を改造することで実現した．Gfarm は libgfarm と呼ばれるライブラリを提供しており，gfsd, gfmd, gfarm2fs はこれを利用している．したがって，libgfarm 側を RDMA 対応することにより Gfarm の主要コンポーネントすべてを RDMA に対応させることが可能になる．われわれは，RDMA 対応済み libgfarm を KVM/QEMU にリンクさせることで QEMU のブロックデバイスレイヤーから直接 Gfarm へアクセスすることを実現した．

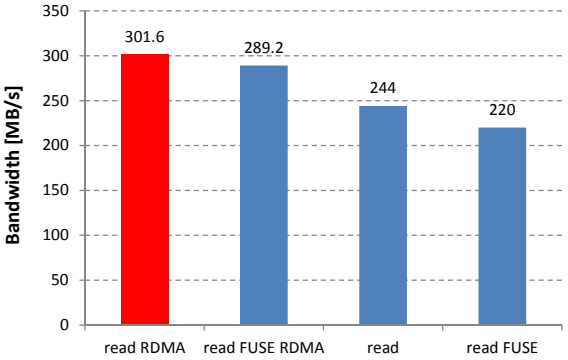

図 4 dd によるシーケンシャルリード性能（gfsd キャッシュ有り）
Fig. 4 Sequential read speed using dd (Cached)

6. 実験

RDMA 対応済み Gfarm を利用した場合，どの程度 VM ディスク I/O の速度向上が見込めるのかの評価を行うため実験を行った．評価は VM 上にて，リードに関しては 4GB のデータを dd を利用して 4KB 単位で/dev/null へとコピーすることで行った．ライトに関しては，VM 上にて，4GB のデータを 4KB 単位で/dev/zero から dd を使って読み出すという手法を取った．すべての評価は 5 回行ったうちの平均値である．Gfarm は gfmd 一つと gfsd を一つ立ち上げ使用した．なお，評価に使った計算機の詳細な環境については**表 2** に示した．

6.1 read の評価 (gfsd 上の Cache あり)

はじめにリード性能から述べる．**図 4** に dd によるシーケンシャルリード性能の評価実験結果を示す．評価を行う前にファイルを一度すべて読み込み，gfsd 上のキャッシュに載せた上で評価を行った．なお，VM 上でのキャッシュは echo 3 > /proc/sys/vm/drop_caches を行うことで完全にクリアしている．そのため，図 4 に示した評価は gfsd 上に乗ったキャッシュから VM 上に対してデータが転送されたものとみなせる．

表 1 にグラフの凡例を示す．図中赤いバーで示された read RDMA が提案手法であり，QEMU のブロックデバイスレイヤーから RDMA 対応済みの libgfarm を直接呼び出し Gfarm にアクセスした時のものである．read FUSE RDMA は，RDMA 対応済み libgfarm を使用した gfarm2fs を使ってホスト OS のファイルシステム上に Gfarm ファイルシステムをマウントしそこから VM イメージを起動したものである．したがって QEMU のブロックデバイスレイヤーから Gfarm API を呼び出してはいない．read は QEMU のブロックデバイスレイヤーから直接 Gfarm API

表 1　評価グラフの x 軸の凡例
Table 1　X-axis legend of experiment result graphs

X 軸のラベル	意味
read RDMA	**提案手法**．VMM が read を hook して RDMA 対応済み Gfarm に直接 API を発行
read FUSE RDMA	VMM は read を hook せず，RDMA 対応済み Gfarm FUSE 経由で VM イメージをマウント
read	VMM が read を hook して，RDMA 未対応の Gfarm に直接 API を発行
read FUSE	VMM は read を hook せず，RDMA 未対応の Gfarm FUSE 経由で VM イメージをマウント
write RDMA	**提案手法**．VMM が write を hook して RDMA 対応済み Gfarm に直接 API を発行
write FUSE RDMA	VMM は write を hook せず，RDMA 対応済み Gfarm FUSE 経由で VM イメージをマウント
write	VMM が write を hook して，RDMA 未対応の Gfarm に直接 API を発行
write FUSE	VMM は write を hook せず，RDMA 未対応の Gfarm FUSE 経由で VM イメージをマウント

を読んだものである．ただし，RDMA は使っていない．最後の read FUSE が FUSE を使って Gfarm ファイルシステムをマウントして，そこから VM イメージを起動したものである．ただし，RDMA は使っていない．

結果を見ると，提案手法の read RDMA が 301.6MB/s と最も速い．FUSE を使わず QEMU のブロックデバイスレイヤーから直接 RDMA 対応の Gfarm API を使いデータをリードしているためである．ついで，read FUSE RDMA が 289.2MB/s と次番手で高速である．約 12.3MB/s 程度の速度差が観察されるが，read RDMA は FUSE を使わずに直接 QEMU のブロックデバイスレイヤーからデータを読んでいるため，FUSE のオーバーヘッド分だけ高速化していると考えられる．次に，RDMA 未対応でかつ，QEMU のブロックデバイスレイヤーから直接 Gfarm API を呼んでいる場合は 244.0MB/s 程度の速度であり，先ほどの read RDMA と比べると，約 57MB/s 程度の速度差が見られることがわかる．この結果は RDMA によるゼロコピー通信が高速化に寄与したためである．最後に，最も遅いのは read FUSE であり，220.0MB/s 程度の速度が出ている．われわれの改良を入れずに Gfarm 上の VM イメージを VMM（KVM/QEMU）で利用するためには，Gfarm ファイルシステムを FUSE でマウントする以外に方法はない．したがって，既存手法 (read FUSE) とわれわれの提案手法である FUSE 無しかつ RDMA による通信と比べると，約 1.37 倍の性能向上があることがわかる．

なお，VMM や VM を介さずに，RDMA 対応済み Gfarm を利用して，FUSE でマウントした上でデータを dd でリードした場合 700MB/s 程度の速度が出ることを確認した．この結果は，今回の提案手法と比べると 2 倍以上高速である．これは，VM を使った時のディスクドライブエミュレーションのオーバーヘッドにより速度低下が起こっていると考えられる．後述するが，QEMU のディスクドライブエミュレーションを行わない PV ドライバ（virtio）[10] を使うことでより提案手法をより高速にできるのではないかと考えている．これについては今後の課題として後述する．

表 2　実験に使用したクラスタマシンのスペック
Table 2　Experiment Environment

CPU	Intel(R) Xeon(R) CPU E5645 @ 2.40GHz
Memory	49 GBytes
HDD	Seagate SAS 600GB (15000 rpm)
OS	CentOS release 6.3
Network 1	InfiniBand MT26428 QDR×4
Network 2	Gigabit Ethernet

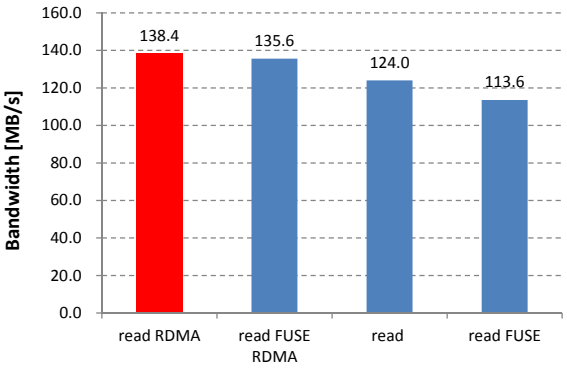

図 5　dd によるシーケンシャルリード性能（gfsd キャッシュ無し）
Fig. 5　Sequential read speed using dd (Noncached)

6.2　read の評価 (gfsd 上の Cache なし)

次に gfsd 上のキャッシュを完全にクリアした場合での転送速度の評価を示す．キャッシュをクリアした場合でも，最も高速なのは read RDMA 提案手法の 138.4MB/s である．次番手で read FUSE RDMA であり，その次に read，最後に read FUSE が続く．上述したように，われわれの提案手法なしに Gfarm 上から VM イメージをマウントするためには，FUSE 経由でマウントした上でそれを QEMU に読ませるしか方法 (read FUSE) がない．われわれの提案手法では，既存手法と比べると，キャッシュなしの場合でも約 1.21 倍の速度向上があることがわかる．

6.3　write の評価

次に，dd を使ったシーケンシャルライトの性能評価について述べる．表 1 にグラフの凡例を示す．write RDMA が提案手法であり，KVM/QEMU ブロックデバイスレイヤー

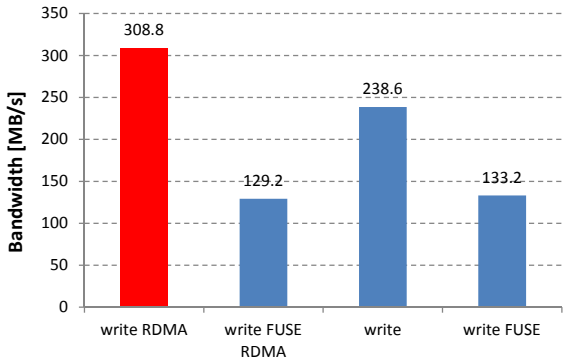

図 6　dd によるシーケンシャルライト性能
Fig. 6　Sequential write speed using dd

から，FUSE を介さずに，直接 RDMA 対応済み Gfarm API を呼び，RDMA でデータをライトしているものである．write FUSE RDMA は RDMA 対応済み Gfarm を FUSE でマウントして，その上 VM イメージを KVM/QEMU から読み込んでいる状態である．write は，RDMA 未対応の Gfarm に対して QEMU のブロックデバイスレイヤーから直接 Gfarm API を呼んで，ライトを実現しているものである．最後の，write FUSE は RDMA 未対応の Gfarm に対して FUSE を経由してマウントし，その上で KVM/QEMU が VM イメージを読み込んでいるものである．

シーケンシャルライトに関しても，提案手法である write RDMA が最も高速であり 308MB/s の速度が出ている．これはキャッシュありのリード性能である read RDMA の 301.6MB/と比べてほぼ同等の速度であり，妥当な性能であると言える．次に，write FUSE RDMA の性能は 129.2MB/s 程度の速度しか出ていないことがわかる．提案手法の write RDMA と比べると約 2 倍の性能差があり，詳しい解析はできていないが，これは FUSE のオーバーヘッドであると考えられる．そのため，FUSE を経由せず，VMM がリードライトをフックしたほうがよい性能が出るという本手法の正当性を示す結果となった．これは，RDMA を使っていない write（238.6 MB/s）と write FUSE（133.2 MB/s）の性能差でも観察され，FUSE のオーバーヘッドが大きいことが確認された．上述したように，われわれの手法無しで Gfarm 上の VM イメージを QEMU/KVM で利用するためには，FUSE を利用するよりほかなく，われわれの手法は，既存手法（write FUSE）と比べると約 2.3 倍の性能が出ていることがわかる．

7. 関連研究

Sheepdog[3] は KVM/QEMU 専用の分散ファイルシステムであり，クラスタマシンを束ねて巨大なストレージプールを構築することができる．実装としては，われわれが今回とった手法と同様に，KVM/QEMU のブロックデバイスレイヤーに直接 Sheepdog 用のドライバを導入し，VM のディスク I/O をフックする形式をとっている．Sheepdog にはストライピングの機能があり，VM ディスクを細かいブロックに分割して多数のクラスタマシンに分散して保存することで高速化を図っている．Sheepdog は TCP/IP 通信を使ってすべてのデータ転送を行っている．一方，われわれは TCP/IP と RDMA を使ったゼロコピー通信による VM 用の分散ファイルシステムを提案している点が異なる．

GlusterFS[11] は中央集権的なメタデータサーバを持たない汎用の分散ファイルシステムである．Gfarm や Sheepdog 同様 PC クラスタを増やした分だけ，容量が増加する．KVM/QEMU の試験的なブランチでは，GlusterFS 用のブロックデバイスドライバが存在しており，FUSE を介さずに直接 GlusterFS に対して VM のリードライトを発行することができる．また，RDMA によるゼロコピー通信にも対応しており，高速な VM ストレージアクセスが可能である．

GlusterFS との本質的は違いは現在のところない．しかし，われわれは今後，カーネルレベルでのゼロコピー通信を実現することで，この GlusterFS との差別化を図ろうと考えている．具体的には，VM 内部に PV ドライバ（virtio）を導入し，ホストカーネルレベルで直接 gfsd と通信を行うアーキテクチャを今後導入することを考えている．現在の実装では，VMM のホストカーネル側から，ユーザランドで動作する QEMU のブロックデバイスレイヤーに一度データがコピーされるというアーキテクチャになっている．そのため，ユーザ空間とカーネル空間側でコピーが行われるため，速度低下につながる．われわれは，gfsd 側から，VMM 側のカーネル空間へと直接データ転送を RDMA で行う実装を今後行っていく予定である．6.1 節でも述べたとおり，VMM や VM を介さずに，RDMA 対応済み Gfarm を利用して，FUSE でマウントした上でデータを dd でリードした場合 700MB/s 程度の速度が出ることを確認している．一方で，現在実装による評価結果ではリードライトともにこの 700MB/s には到底達していない．この速度差の原因は QEMU のエミュレーションレイヤのオーバーヘッドにあると考えられるため，PV ドライバを使うことでこの速度差が縮まると考えている．そして，PV ドライバを導入した上で，ユーザ空間とカーネル空間のコピーをなくすことで，更に速度差が縮まるのではないかと考えている．

Lithium[12] は仮想マシン向けの VM イメージストレージであり，コモディティな計算機を複数束ねることで巨大なストレージプールを実現する分散ファイルシステムである．VM とストレージとの接続には iSCSI を使っているため本研究のように RDMA を利用したゼロコピー通信は利用していない．そのため，本研究とは異なる．

VMFS[13] は VMware 社が開発した VM イメージや VM のスナップショットを格納するためのストレージであり．同様にコモディティな計算機を複数束ねることで巨大なストレージプールを提供している．しかし，本手法のように，

VM と VMFS 間の接続を RDMA を利用するといったことはなされていない．そのため，本研究とは異なる．

8. おわりに

本研究では最終的に Hadoop で利用することを目的とした，大量の VM インスタンスを格納でき，かつ，高速なディスク I/O を実現する VM イメージ向けストレージを提案した．Hadoop では大量の計算機を使用して分散処理を行う．そのため，管理コストの観点から，物理的な計算機と同等の能力を持つ仮想マシンモニタを利用することもある．事実，Amazon EMR クラスタと呼ばれるサービスでは，ユーザは自分が使う分だけの VM を立ち上げて利用することができる．Hadoop のデータは通常 HDFS に格納されるが，計算結果の中間ファイルは各々の VM ディスクイメージに格納される．そのため，Hadoop の高速化のためには，HDFS の高速化だけでなく VM のディスク I/O そのものも高速化する必要がある．

本研究で実装した Gfarm を元にした VM ストレージの特徴は，VM のディスク I/O の転送に InfiniBand の RDMA によるゼロコピー通信を利用していることにある．これにより，高速な I/O を実現することが可能となる．評価の結果，RDMA 無しで FUSE を利用した上で VM をマウントしたものとくらべると，リードで最大 1.37 倍，ライトで最大 2.3 倍の速度向上が見られた．これは，われわれが導入した RDMA のによるゼロコピー通信の高速化によるものである．

今後の課題としては，以下のものがあげられる．まず，VM を介さずに RDMA を使った FUSE 経由でリードの性能を測った結果，gfsd 側にキャッシュありの状態で 700MB/s の速度が出ることが分かった．一方で，現在実装による評価結果ではリードライトともにこの 700MB/s には到底達していない．この速度差の原因は QEMU のエミュレーションレイヤのオーバーヘッドにあると考えられる．そこで，今後は PV ドライバを使用し，PV ドライバが動作するカーネル内部から直接 RDMA を行ってデータ通信を行う手法を検討中である．これにより，PV ドライバを導入することで，QEMU のエミュレーションレイヤのオーバーヘッドは消えることになるので，700MB/s の速度により近づくと考えている．

また，今回の評価は限定的なものにとどまってるため，Hadoop を用いて実際にどの程度 Hadoop のジョブが高速化するのかを比較するのも今後の課題である．特に GlusterFS や Sheepdog 上に VM イメージを置いた上で，これらと比較して，どれだけの性能差があるのかを確かめるのも今後の課題である

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，科学振興機構戦略的創造研究事業 (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] The Apache Software Foundation: Apache Hadoop. http://hadoop.apache.org

[2] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, Washington, DC, USA, IEEE Computer Society, pp. 1–10 (online), DOI: 10.1109/MSST.2010.5496972 (2010).

[3] Sheepdog Project: sheepdog. http://sheepdog.github.io/sheepdog/

[4] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, pp. 257–275 (online), DOI: 10.1007/s00354-009-0089-5 (2010).

[5] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: the Linux Virtual Machine Monitor, *Proceedings of the Linux Symposium*, pp. 225–230 (2007).

[6] Bellard, F.: QEMU, a fast and portable dynamic translator, *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005*, Berkeley, CA, USA, USENIX Association, p. 41 (online), available from ⟨http://portal.acm.org/citation.cfm?id=1247401⟩ (2005).

[7] Schwan, P.: Lustre: Building a File System for 1,000-node Clusters, *Proceedings of the Linux Symposium*, p. 9 (2003).

[8] FUSE Project: FUSE. http://fuse.sourceforge.net/

[9] InfiniBand Trade Association: InfiniBand Architecture Specification Volumne Release 1.2.1 (2008). Abruf am 09.06.2012.

[10] Russell, R.: virtio: towards a de-facto standard for virtual I/O devices, *SIGOPS Oper. Syst. Rev.*, Vol. 42, pp. 95–103 (online), DOI: http://doi.acm.org/10.1145/1400097.1400108 (2008).

[11] Red Hat, Inc: GlusterFS. http://www.gluster.org/

[12] Hansen, J. G. and Jul, E.: Lithium: virtual machine storage for the cloud, *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, New York, NY, USA, ACM, pp. 15–26 (online), DOI: http://doi.acm.org/10.1145/1807128.1807134 (2010).

[13] Vaghani, S. B.: Virtual Machine File System, *SIGOPS Oper. Syst. Rev.*, Vol. 44, No. 4, pp. 57–70 (online), DOI: 10.1145/1899928.1899935 (2010).