情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 2　152–165 (July 2009)

仮想計算機遠隔ライブマイグレーションのための透過的なストレージ再配置機構

広渕崇宏[†1]　小川宏高[†1]　中田秀基[†1]
伊藤智[†1]　関口智嗣[†1]

計算機センタやデータセンタの計算機資源の運用効率を高めるために，仮想計算機とそのライブマイグレーション技術が注目されている．仮想計算機を遠隔拠点に対して動的に再配置できれば，資源運用の柔軟性向上や省電力化に寄与すると考えられる．しかし，LAN 環境を想定した既存の仮想計算機技術ではストレージアクセス手法に問題をかかえており，WAN 環境においてライブマイグレーションを効率的に行うことが困難である．そこで本研究では，周辺機器を含めた仮想計算機の実行環境全体の移動を可能にするために，仮想計算機ストレージの透過的な再配置機構を提案する．提案機構はブロックレベルのストレージ I/O プロトコルに対するターゲットサーバとして振る舞い，ライブマイグレーションの際には仮想計算機の動作に支障を与えることなく透過的に仮想ディスクの拠点間再配置を完了する．その評価として，提案手法の基本的な I/O 特性を明らかにしたうえで，実アプリケーションを想定した実験を行った．提案手法においては，オンデマンドなブロック再配置機構のみでは未再配置ブロックの読み込みが遅く，大量のファイルを対象としてランダムに同期 I/O を行うメールサーバ型アプリケーションにおいてはボトルネックとなりうる．しかし，同時並行的に動作する先読み的なブロック再配置機構により，可用 WAN 帯域を利用して効率的にデータが再配置され性能低下が緩和されることが分かった．

A Transparent Storage Relocation Mechanism for Wide-area Live Migration of Virtual Machines

Takahiro Hirofuchi,[†1] Hirotaka Ogawa,[†1]
Hidemoto Nakada,[†1] Satoshi Itoh[†1]
and Satoshi Sekiguchi[†1]

Live migration of virtual machines is one of the important features provided by virtual machine monitors, which enables users to move a virtual machine to another physical computer without stopping its operating system. It is very useful for hardware maintenance and load balancing of hosting nodes. The practical use of this feature, however, is limited in LAN environments. In WAN environments, network latencies cause inevitable performance degradation of storage devices shared between source and destination host nodes, which are required to continue storage access of VMs before/after live migration. We propose a transparent, relocatable I/O mechanism for VM migration, which enables VM disk images to be completely migrated to remote nodes without any modification of virtual machine monitors. It works as a target and proxy server of a block-level I/O protocol, and transparently copies virtual disk blocks among source and destination sites. Experiments showed the proposed system achieved feasible I/O performance for applications. Although its on-demand remote block fetch mechanism involves I/O latencies including RTT of WAN environments, this drawback is alleviated by its background block copy mechanism; it prefetches not-yet-cached blocks in an efficient manner by exploiting an available WAN bandwidth.

1. はじめに

計算機センタやデータセンタの資源運用の効率性を高めるために，仮想計算機（VM）とそのライブマイグレーション技術が注目されている．仮想計算機技術によって計算機資源を抽象化して論理的に分割・共有できる．さらに仮想マシンモニタ（VMM）が備えるライブマイグレーション機能によって，VM をいっさい停止することなく異なる物理ノード上に再配置可能になる．

たとえば，VM を介したホスティングサービスにおいては，物理ノードの負荷に応じた VM の再配置によって，一定のサービス提供品質を維持できる．また，VM を一部の物理ノード上に集約して過剰な物理ノードの電源を停止することで，データセンタ全体の省電力化をはかれる．さらには，サービス提供を継続しながら，物理計算機ハードウェアの点検や交換などを円滑に進めることもできる．

このようにライブマイグレーション機能は非常に有用な機能であるものの，その実用化は単一拠点内での運用にとどまっている．遠隔拠点にまたがって利用できれば，計算機センタやデータセンタの運用柔軟性を飛躍的に高められる．拠点間をまたがった負荷バランスや省電力化が可能になるほか施設メンテナンスも容易になる．しかし，それらの実現は LAN 環

†1 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

境を前提とした現状のライブマイグレーション技術では困難である．

遠隔拠点間にわたる VM の動的再配置をとりわけ困難にしている理由は，ライブマイグレーションに対応した VM のストレージアクセス手法にある．一般的に VM をライブマイグレーションする際には，移動元・移動先ホスト計算機両方からアクセスできる共有ストレージが必要とされる．しかし，広域ネットワークを介して共有ストレージを構築すると，LAN 環境よりも大きなネットワーク遅延により I/O 性能が低下してしまい実用性を失う．また，仮に異なる拠点に VM を再配置できたとしても，依然としてストレージアクセス先を変更することができず柔軟性に乏しい．

そこで，本研究では，VM の遠隔ライブマイグレーションに対応したストレージアクセス手法を提案する．VM の実行メモリイメージの再配置にともなって，VM のストレージ領域も透過的に遠隔拠点に対して再配置される．VM 内部の OS は移動前後においても引き続いてストレージ I/O を継続することができる．また VMM や VM 内部の OS に対していっさい特別な変更を加える必要がない．提案手法はネットワークを介したストレージアクセスプロトコルに対する透過的なプロキシおよびキャッシュ機構として実装される．

本稿では 2，3 章で遠隔拠点間の VM 動的再配置における問題を明らかにし，4 章で既存研究を概観する．5 章で提案手法とその実装について述べる．6 章で評価を行い，7 章でまとめる．

2. VM 動的再配置とストレージアクセス

一般的に VM のライブマイグレーションを実現するためには，VMM および周辺機器 I/O 機構両方において対応が必要になる．VMM は VM メモリイメージを動的にノード間で再配置できる必要がある．VM に提供されるストレージ領域は何らかの方法で移動元ホスト・移動先ホスト双方からアクセス可能である必要がある．また VM 内部で実行中のネットワーク接続を維持するためには，移動元・移動先ホスト双方において同一ネットワークセグメントを VM に提供する必要もある[⋆1]．

このときストレージアクセス手法としては，VM のディスクイメージを移動元・移動先双方のホストからアクセス可能な領域に作成する．NFS[17] などのネットワークファイルシステムや OCFS[8] や Lustre[7] などのクラスタファイルシステム上に仮想ディスクイメージを作成し，移動元・移動先双方のホストからマウントしておく．あるいは，iSCSI[14] などのブロックレベルの I/O プロトコルを用いて，双方のホストから仮想ディスク領域をあらかじめ接続しておく．また VM 内部のシステムを NFS Root あるいは iSCSI Root によるディスクレス[25] な構成とし，ストレージサーバに VM 内部から直接アクセスする場合もある．

ライブマイグレーション時には VMM は次のように動作することで，実行メモリイメージを再配置する[6]．

(1) 移動元ホストの VMM は，移動先ホストに対して VM の実行に必要なメモリ領域を予約する．
(2) 予約したメモリ領域に対して移動元ホストで実行している VM メモリイメージのコピーを開始する．
(3) メモリイメージのコピー中にも VM 内の OS は動作しており，この間更新されたメモリ領域もさらに移動先ホストへコピーすることを繰り返す．
(4) 移動元ホストで VM を停止し，CPU 状態や残りのメモリイメージを移動先ホストへ転送する．
(5) すべての実行状態が転送できたら，移動先ホストで VM の実行を再開する．

このとき第 3 段階までは，移動元ホストを介して周辺機器 I/O が行われ，第 5 段階以降は移動先ホストを介して周辺機器 I/O が行われる．

3. 遠隔ライブマイグレーションにおける問題

以上の仕組みを遠隔拠点間のライブマイグレーションに適用することを考える．VMM による実行メモリイメージのコピーは，移動元・移動先ホスト双方の VMM 間で通信接続性があれば基本的には可能である．VM のメモリサイズやメモリページの更新頻度などに依存するものの，今日の WAN において 1 Gbps 程度の帯域を確保できる場合もあることを考えれば，実用的な時間内でコピー可能であると考える．コピーされるメモリページの大半は，第 2 段階開始直後におけるチェックポイント時のものであり，バースト的な転送で処理できる．したがって，ネットワーク遅延に対して TCP のバッファサイズを十分多くなるように最適化しておけば，広帯域に見合うだけの速度で迅速にコピーが完了するはずである．また VM に対して割り当てられたメモリ領域のうち実際に使用しているページのみをコピーすればよい．キャッシュ領域として使用しているメモリページはマイグレーション直前に解放することで，データ転送量をさらに小さくできる．

移動元・移動先双方で同一のネットワークセグメントを提供するためには，イーサネット

⋆1 VM 内部において再配置を検知し OS やアプリケーションが独自に対応するのであればこの限りではない．異なるネットワークセグメントに移動しても，DHCP による IP アドレスの更新とアプリケーションレベルでのネットワーク接続の再確立を行ったり，Mobile IP を使って同一 IP アドレスを維持したりすることもできる．

VPN が使用できる．我々が開発しているマルチサイト仮想クラスタ[24),26)]においては，複数拠点にまたがる仮想クラスタを構築し大規模ノード群を管理するために，イーサネット VPN によって分散仮想ノードを単一クラスタとしてのビューの元に容易に取り扱えるようにしている．ゆえに我々の実装においては開発済みのコンポーネントを利用できる．

しかし，ストレージアクセスについては大きな問題があり，既存技術での解決は難しい．第 1 に，移動元・移動先拠点間で WAN を介して VM のストレージ領域を共有した場合，ストレージサーバに対するネットワーク遅延が増大してしまい，WAN 経由で遠隔ストレージアクセスする際に十分な I/O 性能を維持できない[⋆1]．一般に大きな I/O バッファサイズでのシーケンシャル I/O が行われる場合は，高遅延環境下でも TCP のバッファサイズを最適化することにより十分な I/O 性能を発揮できることが知られている[23)]．しかし，VM のストレージ領域として使われた場合，必ずしもシーケンシャルな読み込みや書き込みを VM が要求するだけではなく，内部の OS やアプリケーションに依存して，ランダムな読み込みや書き込みも数多く要求される．また 1 度に発行される I/O バッファサイズは，内部の OS のストレージドライバや上位のアプリケーション，VMM に付随するブロックデバイスドライバなどに依存し，高遅延環境下でも十分な I/O 性能を発揮できる I/O バッファサイズで要求をつねに発行するわけではない．ゆえに，これらの手法においては遠隔ライブマイグレーションにともなって VM のストレージ I/O 性能を維持することが困難である．

第 2 に，VM のストレージデータは移動元の拠点に残り続けてしまう．移動元の拠点のストレージサーバがつねに維持される必要があり，また WAN 経由での通信接続性も継続的に保持しなければならない．マイグレーションによって移動元拠点の設備を柔軟にメンテナンスすることが難しくなり，また移動後の VM を運用するうえでの信頼性も損なう可能性がある．

また，別の手法として遠隔拠点間で VM のストレージデータを同期的につねにミラーしておく手法が考えられるものの，実用面でのコストを考慮すれば適用が難しい．ある拠点がホストするすべての VM ストレージを同期的にミラーするためには，すべてのデータを格納できるだけの大容量のストレージが移動先拠点においても確保される必要がある．またミラーリングのために WAN を介したデータ転送量がつねに発生してしまう．計算機センタやデータセンタのように VM を大規模に運用する環境において一般的に利用するにはコスト面において問題がある．

⋆1 たとえば，WAN を介した遠隔ストレージアクセスによる I/O 性能の劇的な低下は，後述する 6 章の評価実験結果においても確認できる．特にランダムアクセスにおいて顕著であり，LAN 内のアクセスと比べ 2%程度にまで落ち込んでいる．

4. 関 連 研 究

遠隔拠点間における VM のマイグレーションに取り組んだ研究は少ない．文献 3) においては，Xen において特殊なバックエンドストレージドライバを作成し，先にストレージデータのすべてを移動先ホストへコピーし，その後メモリの再配置を開始する．2 章で述べた第 3 段階終了までは，移動元ホストでは VM が動作し続けており，次々に新しいデータがストレージに書き込まれ続ける．これらのデータも移動先ホストへ転送される．このときバックエンドストレージドライバにおいて意図的に VM の書き込み速度を低下させることで，有限時間内にデータ転送が完了するようにしている．文献 11) では，この再配置中の動作に改良を加えて，ストレージ移動にともなう VM 停止時間の短縮を試みている．いずれの手法も，メモリ再配置の完了と移動先での VM の起動以前に，ストレージの移動が完了しているという特徴を持つ．ライブマイグレーション完了後には I/O 性能の低下がいっさい存在しない．一方で，マイグレーション開始から移動先ホストで VM が起動するまでの時間が長く，意図的な書き込み速度低下による性能低下も存在する．またストレージデータ移動中に更新されたブロックは再度繰り返しコピーしなければならないため，完了までには仮想ディスクサイズよりも大きなデータ転送が発生してしまう．先にストレージを移動するので VM の起動ホストをすばやく切り替えることができず，きめの細かい負荷バランスや省電力化には不向きであると予想する．

また文献 13) では，あらかじめ VM の雛形となるディスクイメージを移動元・移動先双方に用意しておき，マイグレーション時にはその変更差分のみを移動元ホストからオンデマンドに取得する．VMware GSX Server のサスペンド・レジューム機能を利用して静的なマイグレーションのみに対応し，システムコールの上書きを用いて実装されている．あらかじめ雛形ディスクイメージを用意することでデータ転送量を軽減させることができる．しかし移動先となりうる拠点すべてに対してあらかじめ共通の雛形イメージをコピーしておかなければならず，あらかじめ移動先拠点が知られていない場合や共通となるべき雛形イメージが存在しない場合には不向きである．また雛形イメージを各拠点にコピーしておくことは，ソフトウェアライセンス上制限される場合もある．

文献 10) では，VMware Workstation のサスペンド・レジューム機能を利用した静的なマイグレーションを，広域分散ファイルシステムである Coda[15)] との組合せにおいて実現している．VM の仮想ディスクを特殊なブロックデバイスドライバによって Coda サーバ

上に小さく分割したファイルとして保存する．そしてその分割ファイルごとに移動元・移動先ホストであらかじめキャッシュしておいたり，オンデマンドな更新差分の取得を可能にしたりしている．しかし Coda がディスコネクト・オペレーションに対応しているとはいえ，VM が拠点外に存在するかもしれない特定の Coda サーバにつねに依存し続けるのは，運用における柔軟性や信頼性を損なってしまう．さらに VM のディスク I/O がさまざまなソフトウェアコンポーネントを経ることになりそのオーバヘッドによる性能低下が危惧される．

VMware 社による Storage VMotion [20] は，VM を起動したまま仮想ディスクを異なるストレージサーバ上に透過的に移動する技術である．仮想ディスクのスナップショット機能やロギング機能を利用して実装されている．単一拠点内で用いられることを想定したものであり，遠隔拠点間のストレージ再配置は対応していない．

また，以上に述べたいずれの手法も特定の VMM に依存した実装である．さまざまな VMM が活発に開発されている現状においては，特定の VMM のみに対応した手法は望ましくない．

5. 提 案 手 法

以上の問題点をふまえて，遠隔ライブマイグレーションを実環境において実用化するためには，我々は VMM が備えるメモリイメージの動的再配置だけではなく，VM のストレージ領域の透過的な動的再配置が不可欠であると考える．VM が異なる拠点のホストに移動することと同時に，その VM がアクセスするストレージ領域も異なる拠点に移動させる．VM が依存する資源を移動元に残さずにすべて移動先に移すことで，VM 運用の柔軟性と信頼性を向上させ，WAN を経由した遠隔資源アクセスによる性能低下を避けることができる．また以上の仕組みは VM に対して透過的に実装されて，VM 内部のシステムが動作を中断することなく動的移動を完了できる．

5.1 提案手法とその基本動作

我々が提案する VM ストレージの動的再配置機構の概要を**図 1** に示す．提案システムは VMM に対しては標準的なブロックデバイスレベルのストレージサーバとして振る舞いながらも，遠隔拠点間の提案システムどうしの間でメモリイメージの再配置にあわせてストレージデータのコピーを行う．

VM の移動元・移動先ホストはそれぞれ提案ストレージサーバに接続し，ホスト OS 上のブロックデバイスとして VM のストレージ領域を提供する．たとえば，図 1 において提

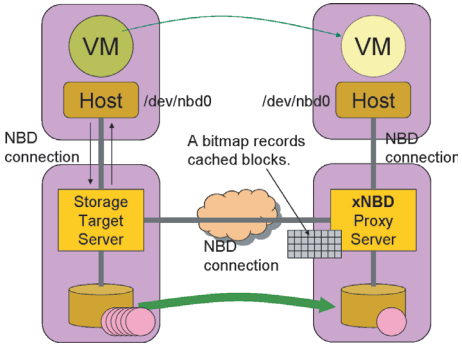

図 1　提案アーキテクチャ概要
Fig. 1　Architecture overview.

案システムは NBD [4] サーバとして振る舞いながら，ホスト OS 上ではそれぞれ`/dev/nbd0`として見えている[⋆1]．

VM が移動元ホストで動作しているときには，一般的なブロックデバイスレベルのストレージサーバ同様に振る舞う．VM 内部の OS が発行した I/O リクエストは，ローカルのストレージデバイスに対して処理される．

VM がメモリイメージの再配置を開始すると，2 章で述べた第 3 段階まではすべての I/O リクエストは移動元ホストにおいて引き続き実行されるので，移動元拠点に存在するストレージデバイスに対して読み書きを続ける．しかし，その後，移動元ホストでの VM の停止を経て，第 5 段階において移動先ホストで VM が開始されると，すべての I/O リクエストは移動先ホストにおいて発行される．この開始時点では移動先拠点のストレージデバイスには，直前まで VM が読み書きしていた移動元のデータは存在していない．

移動先における提案システム（図 1 中 xNBD Proxy Server，xNBD プロキシ）は，VM のリクエストに対して不足しているデータをオンデマンドに移動元拠点から取得する．同時に移動先のストレージデバイスにデータをコピーし，最終的にはすべての I/O が移動先拠点内で完結するようにする．移動元拠点のストレージサーバは複数のクライアント接続を同時に受け付けるように実装する．xNBD プロキシは標準的な NBD プロトコルを使用して移動元拠点のストレージサーバに接続し，NBD クライアントとして不足しているデータを

⋆1 現在はプロトコルが単純な NBD を用いているものの，将来的には iSCSI による実装を視野に入れている．

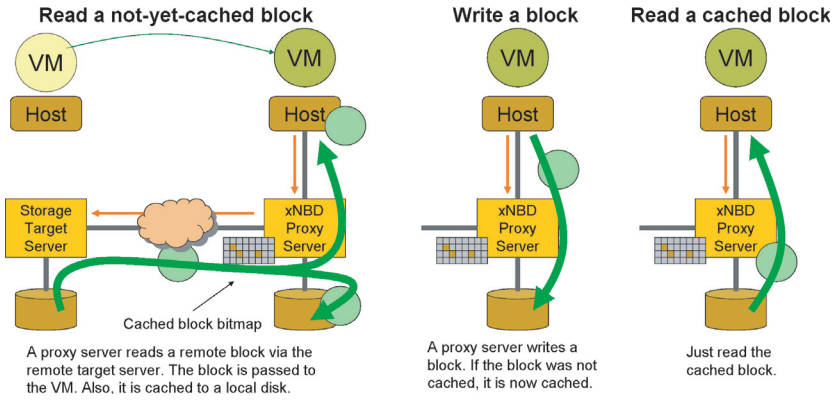

図 2　基本動作概要
Fig. 2　On-demand block fetch mechanism.

取得する[⋆1].

このときの詳しい振舞いを**図 2** に示す.

- VM が移動先に存在しないブロックを読み込もうとした場合，移動元拠点のストレージデバイスから遠隔読み込みを行う．読み込んだデータを VM に返すと同時に，移動先拠点のストレージデバイス（以降本章では**ローカルディスク**[⋆2]）にも保存する．対象ブロックがキャッシュされた（移動先に存在済み）としてビットマップテーブルに記録する[⋆3].
- VM がデータを書き込もうとした場合，対象ブロックがすでにローカルディスクに存在するしないにかかわらず，ローカルディスクの対象ブロックにデータを保存する．移動元拠点のストレージデバイスは更新しない．この操作前に対象ブロックがキャッシュされていなかったならば，新たにキャッシュされたとしてビットマップテーブルに記録する[⋆4].
- VM がすでに移動先に存在済みのブロックを読み込もうとした場合，ローカルディスクの対象ブロックからデータを読み込む．移動元ストレージデバイスから読み込む必要はない.

VM が移動先で I/O 処理を行うたびに，徐々に移動先にキャッシュされるディスクブロックが増えていく．以上の動作に並行して，残りのブロックを移動元拠点からローカルディスクにコピーすることで，最終的にすべてのブロックが移動先拠点にキャッシュされた状態になり，VM のストレージ領域の再配置が完了する．xNBD プロキシは移動元拠点のストレージサーバに対する接続を終了する．以降 xNBD プロキシは標準的なストレージサーバとして振る舞う．さらに同様の仕組みで他の拠点に移動できる.

5.2　バックグラウンドコピー

上述した VM の I/O に応じたオンデマンドなデータ再配置に加えて，未キャッシュなブロックはバックグラウンドでもコピーされる．最終的に仮想ディスクのすべての領域がキャッシュされると，仮想ディスクの再配置は完了し，プロキシサーバから移動元拠点ターゲットサーバへのストレージ I/O プロトコル接続は終了する．バックグラウンドコピーは，VM が特定のブロックに初めてアクセスする前にあらかじめブロックをコピーしてキャッシュしておくことで，移動元への遠隔読み込みを防ぐ側面もある.

5.3　実　　装

現在我々は比較的単純な NBD プロトコルに対して提案機構を実装している．ネットワーク遅延がある WAN 環境でも性能低下を軽減させるために I/O リクエストの多重キューイング機能を備え，図 1 のターゲットサーバにあるように複数のクライアントからの同時接続をサポートする．またプロキシサーバもクライアントからは通常のターゲットサーバとして見えるので，さらに次の拠点に再配置することもできる．仮想ディスクファイルやキャッシュファイルに対しては `mmap()` を用いることでメモリコピーを最小限に抑えている．バックグラウンドコピーはオンデマンドな I/O 処理を妨げないように低い優先度で処理する.

5.3.1　I/O リクエスト処理

図 3 にあるように，多重キューイングのサポートおよび低い優先度のバックグラウンドコピーを実装するために，プロキシサーバとして動作するときには 4 つのスレッドが動作

⋆1 移動元拠点のストレージサーバは移動元ホストおよび xNBD プロキシの両方と接続する．しかし，2 章で述べたメモリイメージの再配置処理における第 3 段階と第 5 段階で I/O パスが移動元ホストから移動先ホストへと切り替わるため，同時に I/O リクエストを受けることはない．分散ロック機構を利用しなくてもデータの一貫性が保たれる.

⋆2 すでに VM は移動先で起動しており，VM が起動している拠点を基準に表現している.

⋆3 便宜的にキャッシュという表現を用いているが，移動先拠点のストレージデバイスは一時的なデータの保存場所ではない点に注意されたい.

⋆4 書き込みリクエストの対象領域の始端オフセットや終端オフセットがブロック単位でない場合は，移動元拠点から始端ないし終端ブロックを先に取得する必要がある.

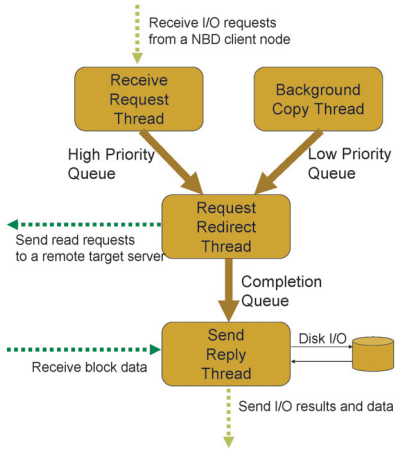

図 3　提案機構におけるプロキシサーバ実装
Fig. 3　Proxy server implementation.

している．

まず VM が I/O リクエストを発行すると，クライアントからのリクエスト受信スレッドがリクエストを受信する．移動元拠点からの遠隔データ読み込みの必要性を一定のキャッシュブロックサイズ（4 KB）[⋆1] ごとに判定したうえで，優先リクエストキューに中間リクエストをキューイングする．一方，バックグラウンドコピースレッドは外部プログラムから先読みすべきブロック番号を受け取り，非優先リクエストキューに中間リクエストをキューイングする．

中間スレッドは優先リクエストキューから先に中間リクエストを取り出し，必要があれば移動元拠点のターゲットサーバに読み込みリクエストを送信し，完了スレッドへ中間リクエストをキューイングする．優先リクエストキューが空であれば非優先リクエストキューから中間リクエストを取り出し，未キャッシュブロックであることを確認して，同様に処理する．

完了スレッドは中間リクエストを取り出し，必要があれば移動元拠点のターゲットサーバから先ほど投げた読み込みリクエストのデータを受信する．書き込みや未キャッシュブロックの読み込みの場合は，ディスクへデータを保存する．最後に，I/O リクエストの結果（と読み込みリクエストの場合には読み込みデータ）をクライアントに返信する．

⋆1 任意の値に設定可能であるものの，6.1 節で述べるように，ファイルシステムのブロックサイズと同値にすることが望ましい．

5.3.2　バックグラウンドコピー処理

バックグラウンドコピーにおいては，任意の未キャッシュブロックの取得を外部プログラムから命令可能にしている．仮想ディスクの先頭から未キャッシュブロックを順にコピーするという単純な動作だけではなく，利用頻度の高いブロックから先にコピーすることも可能である．

実際，その一例として，我々は仮想ディスク上に作られた ext3 ファイルシステム[5),19)] を解析することによって，利用頻度が高い領域を判別し優先的にコピーする仕組みを実装している．ext3 はディスク領域を複数のグループに分割して，関連性の高いブロックを同じグループ内に保存することで，キャッシュ効率を高めディスクヘッド移動を減らすよう試みる[⋆2]．同じディレクトリに属するファイルは同じグループに，同じファイルに属するブロックは同じグループに属することが多い．また使用中のディスクブロックを管理するために，inode ブロックビットマップおよびデータブロックビットマップとしてディスク上にビットマップを保存している．

そこで，ターゲットサーバで I/O リクエスト領域の統計をとっておくことで，直近にアクセス頻度の高いグループからバックグラウンドコピーにとりかかれるようにしている．さらにターゲットサーバにおいて残された ext3 イメージのビットマップブロックを解析して，データが保存されているブロックから優先的にコピーを開始できる．

6. 評 価 実 験

実験環境を**図 4** に示す[⋆3]．WAN を介して VM 移動元・移動先の 2 拠点が存在する状況を想定する．拠点内は LAN 環境を想定して Gb Ethernet によりノード間を接続する．拠点間は帯域 100 Mbps を設定しネットワークエミュレータ（Linux カーネルの netemu 機能）によりネットワーク遅延（RTT20 ms）を発生させる．我々は先行研究[26)] 以降，複数拠点のデータセンタを仮想化技術によって柔軟に統合運用することに取り組んでいる．そこで，本章では適用先環境の 1 つとして東京および大阪に存在するデータセンタ間を意識した設定とした．また，異なるネットワーク環境については 7.3 節において議論する．

⋆2 一般的に，ext3 のブロックサイズが 4 KB である場合，各グループのサイズは 128 MB 程度になる．
⋆3 VM ホスト計算機：Intel Xeon 2.8 GHz，メモリ 2 GB ストレージサーバ計算機：AMD Opteron 250，メモリ 4 GB

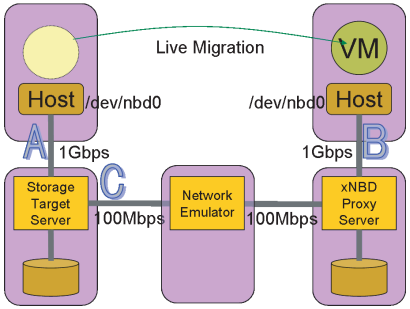

図 4　実験環境
Fig. 4　Evaluation environment.

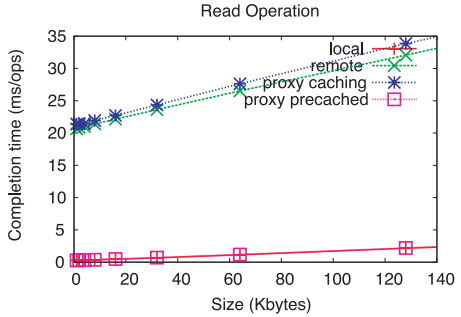

図 5　Direct I/O `read()` 完了時間．local および proxy precached はグラフ下部でほぼ重なっている
Fig. 5　Direct I/O `read()` completion time.

最初に NBD ドライバに直接的に I/O 命令を発行して提案機構の基本的な性能特性を確認する．次に VM 内部における I/O 性能を検証する．特に一般的に I/O レイテンシが大きい場合に不向きとされるメールサーバを想定した実験を中心に行う．

6.1　基本的な I/O 性能特性

最初にカーネルによるバッファリングやキャッシングの影響を排して提案機構自体の基本的な性能特性を確認する．一般的にカーネルは読み込みに際してはデータの先読みを行ったり，書き込みデータをいったん内部でバッファリングしたりして，それらのデータをメモリ上でキャッシュする．またブロックデバイスに発行されるリクエストがハードウェアのブロックサイズ区切りとなるよう調整している．しかし，Linux の Direct I/O 機能を用いるとこれらの処理を経ることなく直接的にブロックデバイスに I/O 命令を発行できる．

ホスト OS に対して接続された NBD のデバイスファイルに対して，リクエストサイズを変えながら以下の条件のもとで，Direct I/O による `read()`/`write()` に要する時間を計測した．Linux カーネルが NBD に対して発行する上限サイズ 128 KB までで計測している．

- **local**：移動元ホストでターゲットサーバに対する性能を計測する．LAN 環境での利用に相当する．
- **remote**：移動先ホストで提案機構を用いずに移動元拠点のターゲットサーバに遠隔アクセスした場合の I/O 性能を計測する．WAN 経由での遠隔アクセスとなる．図 4 のリンク B をプロキシサーバを経由せずに直接ネットワークエミュレータにつなぐ．
- **proxy caching**：移動先ホストで提案機構を経由して I/O 性能を計測する．未キャッシュ状態のブロックに対する I/O 性能を計測するために実装を改変しており 1 度アクセスされたブロックをキャッシュ済みとしてマークしない[⋆1]．同じ領域に何度アクセスしてもつねに未キャッシュ状態のブロックとして動作する．
- **proxy precached**：移動先ホストで提案機構を経由して I/O 性能を計測する．キャッシュ済み状態のブロックに対する I/O 性能を見るために，あらかじめすべてのブロックをキャッシュ済みとしておく．

図 5 において，`read()` に要する時間に対して LAN 環境（**local**）および事前キャッシュした場合（**proxy precached**）に差は見られない．1 度キャッシュされたブロックは，WAN を経由して移動元ターゲットサーバにアクセスすることなく取得できる．そのため，LAN 環境でのアクセスとまったく同様の I/O 性能を得られ，遠隔アクセスする場合（**remote**）よりも大幅に性能が向上している．**remote** が示すように移動元ストレージへの遠隔アクセスは RTT 分さらに完了時間を要してしまう．また多くの場合 LAN に比べて小さい帯域を利用してデータ転送しなければならない．**proxy caching** では未キャッシュのブロックを取得するために遠隔アクセスとほぼ同様の完了時間を要する．プロキシサーバ内の処理において，遠隔取得したブロックを再度クライアントに転送するので若干完了時間が増加している．

図 6 において，`write()` に要する時間に対しては **local**，**proxy precached** および **proxy caching** についてほとんど差はない．プロキシサーバは，対象ブロックがキャッシュ済みかどうかにかかわらず直接キャッシュファイルに書き込む．原則遠隔アクセスが発生せ

⋆1 データの一貫性が保証されないため本実験のみで用いる．

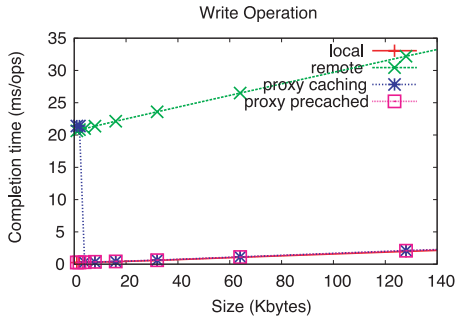

図 6　Direct I/O `write()` 完了時間．local，proxy caching および proxy precached はグラフ下部でほぼ重なっている

Fig. 6　Direct I/O `write()` completion time.

ず LAN 環境でのアクセスと同様の I/O 性能を得られ，遠隔アクセス（**remote**）に比べ大幅に性能が向上している．**proxy caching** において書き込みサイズが 1 KB および 2 KB のときに遠隔読み込みが発生しているのは，キャッシュブロックサイズとして設定した 4 KB よりも小さいサイズの書き込みであるため，プロキシサーバが対象ブロックの 4 KB 分のデータを先に読み込んだうえで，内部で書き込みデータで上書きしてディスクに書き戻しているからである．仮想ディスク上にファイルシステムを作成するのであれば，ファイルシステムのブロックサイズをあらかじめ 4 KB にしておくことで，この遠隔読み込みを防ぐことができる．

以上より，提案機構は，未キャッシュのブロックを読み込む場合を除けば，LAN 環境でのストレージアクセスとまったく同様の I/O 性能を有し，遠隔アクセスに比べて大幅に性能を改善することができることが確認された．

6.2　VM 内部での性能

次に VM で動作する OS 上で性能評価を行った．Xen 3.0 [1] を用いてメモリ 512 MB を割り当てた VM を作成し Linux をインストールした．Linux をインストールした仮想ディスクとは別に，実験用のファイルを作成する仮想ディスク (4 GB) を VM に追加した．ext3 によりフォーマットおよびマウントしている．

6.2.1　Bonnie++

ベンチマークプログラム Bonnie++1.03c [2] により計測したシーケンシャル I/O 性能を **表 1** に示す．遠隔アクセス（**remote**）の場合においては，ネットワーク遅延（RTT20 ms）

表 1　シーケンシャル I/O 性能（KBytes/s）
Table 1　Sequential I/O performance (KBytes/s).

	Read	Write
local	71,435（0%）	83,403（23%）
remote	4,501（0%）	12,879（3%）
proxy precached	72,320（0%）	82,224（23%）

括弧内はゲスト OS における CPU 使用率

表 2　ランダムシーク性能（ops/s）
Table 2　Random seek performance (ops/s).

	Random
local	7,644（1%）
remote	159（0%）
proxy precached	7,198（1%）

括弧内はゲスト OS における CPU 使用率

および狭い帯域（100 Mbps）が原因で低い I/O 性能（読み込み 4.5 MBytes/s，書き込み 12.9 MBytes/s）しか達成できていない．しかし提案機構によりキャッシュされた状態（**proxy precached**）においては，LAN 内での利用（**local**）と同様に高い I/O 性能（読み込み 72.3 MBytes/s，書き込み 82.2 MBytes/s）を得られる．

local および **proxy precached** において，ホスト OS においてプロセスおよびカーネルスレッドの CPU 使用率を観察すると，Xen のバックエンドブロックデバイスドライバ（xvd）がこの実験中は 90%を超えていた．実験環境においては xvd におけるメモリコピー処理がボトルネックとなっている．

表 2 に示したランダムアクセス性能において，提案機構は遠隔アクセス（**remote**）における値（159 ops/s）よりもはるかに良い値（7,198 ops/s）を示す．Bonnie++におけるランダムアクセス性能の計測は，対象となるファイルをランダムに `lseek()` したのち 90%の割合で `read()` および 10%の割合で `write()` を発行する．対象となるファイルは OS のメモリサイズに対して 2 倍の大きさ（1 GB）で作成されるために，カーネルキャッシュだけでなくディスクへのアクセスも頻発する．したがってネットワーク遅延を介した遠隔アクセスにおいては非常に性能が悪くなる一方，提案機構による改善効果が顕著に現れている．

表 3 では，メールサーバプログラムを想定したファイル操作性能を示す．Bonnie++のファイル操作ベンチマーク機能を用いている．1 つあたり 10 KB の 102,400 個のファイル

表 3　ファイル操作性能（ops/s）
Table 3　File operation performance (ops/s).

	Create	Access	Delete
local	699（1%）	3,856（1%）	426（1%）
remote	15（0%）	65（0%）	21（0%）
proxy precached	674（1%）	4,107（1%）	425（1%）

括弧内はゲスト OS における CPU 使用率

表 4　プロキシサーバ経由ファイル操作性能（ops/s）
Table 4　File operation performance via I/O proxy (ops/s).

		Access
proxy	**bgcopy off**	44.6
proxy	**bgcopy seq**	145.2
proxy	**bgcopy ext3**	537.8
proxy	**precached**	1,963.6

を 100 個のディレクトリの下に均等になるように作成（`creat()`，`write()`，ファイルとディレクトリの `fsync()`，`close()`）する．すべてのファイルをランダムな順番にアクセス（`stat()`，`open()`，`read()`，`close()`）する．その後すべてのファイルをランダムな順番で削除（`unlink()`，ディレクトリの `fsync()`）する．一般に小さなデータサイズの読み書きを大量に同期的に行うアプリケーションは，高遅延環境下での遠隔ストレージアクセスに不向きとされる．作成されるファイルの総量（10 KB*102,400 個）はゲスト OS に割り当てたメモリサイズ（512 MB）の約 2 倍となるよう設定している．そのため，約半分のファイルはカーネルメモリ内にキャッシュされず，実際にディスクにアクセスして読み込まれる．

ファイルの作成および削除時には 1 つのファイルを処理するごとにディスクへの同期的な書き込みが発生し，ランダムなアクセス時にもディスクからの読み込みが多くの場合発生する．遠隔アクセス（**remote**）においては，それらの処理ごとにネットワーク遅延に起因する少なくとも 20 ms の完了時間を要するために，操作性能が大きく低下している．しかし提案機構においては 1 度キャッシュされたブロックであれば LAN 環境と同様の操作性能を得られている．

6.2.2　未キャッシュブロック読み込みとバックグラウンドコピー

次に未キャッシュブロック読み込みの影響およびバックグラウンドコピーの効果を確認した．6.1 節において確認したように，プロキシサーバで未キャッシュ状態であるブロックを読み込む際には遠隔アクセス同様の長い完了時間を要する．このことが影響して，ライブマイグレーション以後にデータがプロキシサーバにキャッシュされるまでに，VM 内で動作するアプリケーションの実行性能が低下してしまう可能性がある．特に，アプリケーションが I/O インテンシブなプログラムであって，カーネルキャッシュに収まりきらないほどの大量のデータに対して継続的にアクセスし続ける場合に顕著であると考えられる．さらにランダムアクセスにおいてはカーネルによる先読みが動作せず，より非効率であるといえる．一方，これらの問題に対して，バックグランドコピー機能がブロックを事前にキャッシュすることで，性能低下を緩和できると期待される．

まず 6.2.1 項のファイル操作性能測定時と同様のファイル群をあらかじめ移動元ターゲットサーバ上に作成しておく[⋆1]．次に移動先ホストからプロキシサーバに接続して，バックグラウンドコピーの条件を変えながら，ファイル群のランダムなアクセス性能を 6.2.1 項のファイル操作測定時と同様に計測した．

バックグラウンドコピーを無効にしてオンデマンドなブロック再配置機能のみでアクセスした場合（**bgcopy off**）と，それに加えてバックグランドコピーも有効にした場合を比較する．この際，バックグランドコピーにおいて，単純にディスクの先頭から未キャッシュブロックをコピーした場合（**bgcopy seq**[⋆2]）や，ext3 ファイルシステム上で使用中のブロックを分析し，それらを順次プロキシサーバへコピーした場合（**bgcopy ext3**）を実験した．また，あらかじめすべてのブロックがキャッシュ済みであった場合（**precached**）も計測している．いずれも計測前にゲスト OS およびホスト OS のカーネルメモリ中のキャッシュを破棄している．

表 4 にベンチマーク結果を示す．さらに，**表 5** に WAN 経由での転送ブロック数を，**表 6** にキャッシュヒット率を記す．WAN 経由での転送ブロック数には，オンデマンドな I/O 処理にともなう遠隔ブロック取得とバックグラウンドコピーの両方による転送ブロック数に相当する．オンデマンドな I/O 処理において，ターゲットサーバへの遠隔ブロック読み込みが発生した場合はキャッシュミスとしてカウントしている．すでにプロキシサーバ上でキャッシュされているブロックの読み込みにおいては，キャッシュヒットとしている．

バックグラウンドコピーを無効にした場合，ランダムアクセスは 44.6 ops/s しか実行で

⋆1 プロキシサーバに接続した移動先ホスト上 VM で実験に用いるファイルを作成すると，そのブロックは読み込みテスト前からキャッシュ済みとなってしまう．そこで Bonnie++を改変して，計測用ファイルの作成とその読み書き動作を別々に実行可能にしている．
⋆2 sequential の略．

表 5　WAN 経由での転送ブロック数
Table 5　Transferred Blocks over WAN.

		Transferred Blocks
proxy	**bgcopy off**	310,972
proxy	**bgcopy seq**	999,944
proxy	**bgcopy ext3**	384,666
proxy	**precached**	0

表 6　キャッシュヒット率（%）
Table 6　Cache Hit Ratio (%).

		Hit Ratio
proxy	**bgcopy off**	27.3
proxy	**bgcopy seq**	97.1
proxy	**bgcopy ext3**	98.1
proxy	**precached**	100.0

きず，6.2.1 項の遠隔アクセス同様に非常に遅い操作となってしまう．

しかし ext3 ファイルシステムを解析したバックグラウンドコピーによって同時並行的にブロックをキャッシュすれば 537.8 ops/s（**bgcopy ext3**）となり大きく改善することができる．また，このときファイルシステムを解析せずに単純に未キャッシュブロックを先頭からコピーした場合（**bgcopy seq**）は，145.2 ops/s にとどまり改善の幅は小さい．仮想ディスク全体（4 GB）をバックグラウンドコピーの対象としてしまい，仮想ディスクの総ブロック数（1,000,001）にほぼ近い数（999,944）のブロックを WAN 経由で転送する必要がある．未使用のブロックも含めて転送しなければならず効率的ではない．一方，**bgcopy ext3** においては，ファイルシステムイメージ中のビットマップから推定される使用中ブロック 384,999 個のみをバックグラウンドコピーの対象とする．仮想ディスク全体をコピー対象とするのに比べて，必要なブロックのみを迅速にキャッシュできている．実際には，メタデータの更新などのために VM の書き込みによるキャッシュも発生し，それらキャッシュ済みのブロックはターゲットサーバから取得する必要はない．ゆえに，WAN 経由で転送されたブロック数は 384,666 と若干小さい値になる．

しかし，バックグラウンドコピーを行っても，事前キャッシュ済み状態（**precached**）における読み込み（1,963.6 ops/s）とは性能に開きがある[⋆1]．**bgcopy ext3** の場合，ベンチマーク開始後からおおよそ 150 秒間はバックグラウンドコピーが発生しており，その間のベンチマーク処理が事前にすべてのブロックをキャッシュした場合に比べて遅いからである．この点は，7.2 および 7.3 節で議論するようにブロックを圧縮したうえでバックグラウンドコピーすることにより改善することができると考えられる．また，対象となるデータセットが小さい場合には，バックグラウンドコピーによって対象ブロックをすべてキャッシュするのに要する時間が短くなり，事前にすべてのブロックをキャッシュした場合との差は小さくなる．

キャッシュのヒット率は，**bgcopy seq** および **bgcopy ext3** ともに高い値（それぞれ 97.1%および 98.1%）となっている．ベンチマークの実行中に発生した VM によるブロック読み込み処理は，その大半がキャッシュ済みブロックへのアクセスであった．ベンチマークの開始当初はバックグラウンドコピーによってキャッシュされたブロックはごくわずかであり，オンデマンドな I/O 処理においては未キャッシュブロックの遠隔読み込み（キャッシュミス）が頻繁に発生する．しかし，遠隔読み込みはキャッシュ済みブロックのアクセスよりもはるかに大きな時間を要するので，この時点ではベンチマーク処理（ファイル群に対するランダムアクセス）はゆっくりと進行する．その後，バックグランドコピーによりキャッシュ済みブロックが増えてくると，ベンチマーク処理は迅速に進むようになる．その結果，ベンチマーク処理全体としてみれば，バックグランドコピーによって先にキャッシュされたブロックを，VM がアクセスする割合がきわめて高くなっている．

未キャッシュ状態の読み込みにおいて，バックグラウンドコピー無効時には WAN を経由する遠隔読み込み方向のトラフィックは 200–600 KBytes/s 程度を推移する．カーネルの先読み機能はランダムアクセスに対しては有効ではなく，また NBD ドライバのリクエストキューイングサイズは最大でも 128 KB であり高遅延環境下で可用帯域を埋めるには十分な大きさではない．一方，バックグラウンドコピー動作時には，継続的に 12 MBytes/s 程度のトラフィックが発生しており，可用帯域を利用して事前キャッシュが効率的に行われていることが確認できる．

6.2.3　カーネルコンパイル

メールサーバプログラムよりは I/O レイテンシに対する要求が緩和される場合として，Linux カーネルコンパイル時間を計測した．あらかじめ移動元ホストで展開しておいたカーネルソースコードに対して，移動先ホスト上の VM でコンパイルを行う[⋆2]．ソースファイル

⋆1 事前にメモリキャッシュを破棄しているためキャッシュ済み状態（**precached**）に対する結果は 6.2.1 項よりも低下している．

⋆2 Linux 2.6.24 を用い `make allmodconfig && make` と実行．

表 7　Linux カーネルコンパイル時間（s）
Table 7　Linux kernel compilation time (s).

		Elapsed Time
remote		6,572
proxy	**bgcopy off**	5,393
proxy	**bgcopy ext3**	5,012
proxy	**precached**	4,958

はゲスト OS のメモリ内にあらかじめキャッシュされておらず，必ず 1 度はディスクから読み込まれる．仮想ディスクに対して，提案機構を用いず遠隔アクセスした場合（**remote**），提案機構を用いた場合（**proxy**）を比較する．

結果を**表 7** に示す．提案機構（**bgcopy off**）は，オブジェクトファイルのディスク書き出しがローカルディスクになるため，遠隔アクセス（**remote**）よりも所要時間を短くできている．バックグラウンドコピーが有効であれば（**bgcopy ext3**），事前にすべてのブロックがキャッシュされていた場合（**precached**）と同等の所要時間で完了している．しかし，バックグラウンドコピーの有無による所要時間の差は，メールサーバプログラムで見られた性能差ほど大きくない．また，遠隔アクセスの所要時間が悪化する度合いも相対的に大きくない．これは，

- ソースファイルの読み込みはシーケンシャルアクセスでありカーネルによる先読みが有効である．未キャッシュブロックの読み込みが相対的に問題となりにくい，
- オブジェクトファイルのディスクへの書き出しは `write()` とは非同期にカーネルによって行われる．アプリケーションが書き込み待ちでブロックされる時間が少ない，
- CPU インテンシブなアプリケーションであるため I/O レイテンシによる影響は相対的に小さい，

という理由によると考えられる．

6.3　評価のまとめ

提案機構におけるプロキシサーバはブロックデバイスの特性として，書き込みおよびキャッシュ済みブロックの読み込みは LAN 環境と同等に高速，未キャッシュブロックの読み込みは WAN 経由の遠隔読み込みが発生するゆえに低速，という特徴を持つ．しかし 1 度読み込めばプロキシ上でキャッシュされ次回の読み込みは LAN 環境と同等に高速である．

提案機構を使用する VM 上では，データの書き込みや 1 度プロキシサーバでキャッシュされたデータの読み込みは LAN 環境と同等に高速である．特に多数のファイルにアクセスし同期的な I/O を頻繁に行うタイプの I/O インテンシブなアプリケーションにおいては遠隔アクセスするよりも改善は顕著である．ランダムアクセスによりメモリ上に有効にキャッシュされない場合も大きく改善される．

そのようなアプリケーションがライブマイグレーション後も移動元拠点で作成されたファイルに大量にアクセスし続けるのであれば，未キャッシュブロックの遠隔読み込みが遅いために，大きく性能低下してしまう．しかしバックグラウンドコピーによって可用帯域を十分利用しながら使用中のブロックを効率的にコピーすることで，性能低下を緩和できる．

7. 議　　論

7.1　既存手法との比較

本稿での提案手法と 4 章の文献 3)，11) による手法は，両者とも VM のライブマイグレーションと連動した仮想ディスクの再配置を提供しながらも，その実現手法は異なっている．我々の提案手法では，VM の起動ホストの切替え後に仮想ディスクの再配置を開始する．一方，既存手法では起動ホストの切替え前に仮想ディスク再配置を完了する．

提案手法の利点としては，仮想ディスクが大きな場合であっても，VM の起動ホストを迅速に切り替えられる点がある．既存手法では，仮想ディスクの再配置後に VM の起動ホストを切り替えるために，ディスクサイズが大きければ，その分 VM 起動ホストの切替えまでに要する時間も長くなる．ディスクおよび VM のメモリイメージ両方を移動先ホストに転送するまで，起動ホストを切り替えることができない．一方提案手法では，仮想ディスクの再配置は VM 起動ホストの切替え後であるため，VM のメモリイメージのみを起動先ホストに転送するだけでよい．

たとえば，6 章の評価実験環境において Linux カーネルコンパイルを行いながら起動ホストを切り替えると，我々の提案手法ではおおよそ 45 秒程度[⋆1]で完了する．一方，既存手法[3] において，評価実験環境における 4 GB の仮想ディスクを移動するならば，少なくとも 380 秒程度は[⋆2]は要するはずである．

また，我々の提案手法では，VM 起動ホストの切り替え直後に未キャッシュブロックのオンデマンドアクセスにより性能低下（I/O レイテンシの増加）が発生する恐れがある．一

⋆1 概算として，VM メモリイメージ 512 MB およびメモリ再配置中に更新されたページ群を，WAN 帯域 100 Mbps で転送することに相当する．
⋆2 仮想ディスク 4 GB および再配置中に更新されたブロック群，さらにメモリイメージ転送に要する分（512 MB + 更新されたページ群）を WAN 帯域 100 Mbps で転送する．

方，既存手法では，動作中の VM の仮想ディスクを有限時間で移動するために，VM 起動ホストの切替え前に書き込み処理の意図的な性能低下を行う可能性がある．

しかし，提案手法によるこの欠点は，必ずしもつねに問題となるものではない．一般的にカーネルは内部に読み込み頻度が高いディスクブロックのキャッシュを持つことによって，I/O 性能の向上を図っている．ライブマイグレーション直前に VM の内部で動作していたプログラムが使用するデータは，移動直後にもカーネルキャッシュ上に存在する可能性がある．そのためアプリケーションによっては，ディスクアクセスレベルの I/O レイテンシの増加が大きく影響しない場合もある．またカーネルコンパイルのように CPU インテンシブな処理であれば，I/O 性能の低下が実行性能に与える影響は相対的に小さい．さらに，メールサーバプログラムのように I/O レイテンシが実行性能に大きく影響を与えるアプリケーションに対しては，バックグラウンドでアクセス頻度が高いブロックから優先的に再配置することで徐々に性能低下を緩和できる．

今後，仮想ディスクの動的な移動手法を実際的な運用環境において適用していく．この際，両者の再配置手法は状況に応じて相補的に用いられることが望ましいと考えている．

7.2　ディスクブロック転送アルゴリズム

バックグラウンドコピーにおける転送アルゴリズムについては，VM のディスク利用形態やアクセスパターンに応じてさまざまな最適化手法が考えられる．本稿ではその一例として，ext3 ファイルシステムの構造に注目した転送手法について，そのプロトタイプを実装した．単純に仮想ディスクの先頭からコピーする場合よりも効率的にデータを再配置できる．今後，さまざまなワークロードにおける VM のディスクアクセスパターンをより詳しく解析して，転送アルゴリズムを改良しさらなる最適化を目指していく．

また，ファイルシステムの構造に注目する手法は，ext3 以外のファイルシステム（たとえば XFS [16] や ZFS [18]）に対しても，それぞれディスク上の構造は大きく異なるものの，基本的には適用可能であると考えている．ストレージサーバにおいて，仮想ディスクイメージ内のファイルシステムのメタデータを，何らかの手法で解析できればよい．ext3 に対するプロトタイプ実装では，ファイルシステムの一貫性をチェックするプログラム（`fsck.ext3`）のソースコードを一部利用することで，実装期間を短縮することができた．他のファイルシステムにおけるメタデータの解析においても，既存プログラムのソースコードがおおいに参考になると思われる．

仮想ディスク内部を LVM などで複数のパーティションで分割している場合であっても，（プロトタイプでは未実装ではあるが）本手法は適用可能である．ディスクイメージ上に存在する分割情報を読み取ることにより，各ファイルシステムの存在場所を得ることができる．

プロトタイプでは未実装ではあるものの，ディスクブロック転送に際してデータを圧縮することで，より迅速にバックグラウンドコピーを完了できる可能性がある．上述のメールサーバを想定した実験においては，プロキシサーバの CPU 負荷は常時数%程度と低い値を示しており，圧縮処理を追加できる余地がある．また，次節で述べるように，ネットワーク帯域が狭い場合においては，提案手法の性能低下を緩和できると思われる．

7.3　ネットワーク環境

6 章の評価実験においては，日本国内の遠隔データセンタ間を想定して，1 つのネットワーク環境（RTT20 ms および 100 Mbps）を取り上げた．我々はさらに異なるネットワーク環境においても同等の実験を行っており，同様の傾向を示すことを確認している．

利用できるネットワーク帯域が小さい場合，その分バックグラウンドコピーが利用できる帯域も減少する．未キャッシュブロックの取得を完了するまでに時間がかかり，遠隔アクセスが長期にわたって発生してしまう．たとえば，ネットワーク帯域を（RTT20 ms のまま）50 Mbps に設定した場合，6.2.2 項における **bgcopy ext3** の値は 370.4 ops/s となり，100 Mbps の場合と比べて低下する．同期型の I/O を頻繁に行うアプリケーションにおいては，バックグラウンドコピーによる先行キャッシュが性能改善に大きく寄与していた．それにもかかわらず帯域を狭くしたゆえに先行キャッシュの完了に時間を要してしまい，結果として性能が低下している．

一方，6.2.2 項のベンチマーク中において，ネットワーク遅延の増加による **bgcopy ext3** の値への影響は小さい．たとえば，日本と北米間を想定した RTT120 ms の場合においても，RTT20 ms の場合と同様に 100 Mbps 弱の帯域を利用しながらバックグラウンドコピーが進む．先行キャッシュの完了時間にはほとんど差がない．6.2.2 項で見たように，オンデマンドな I/O 処理にともなう遠隔アクセスによってブロックが取得される割合は非常に小さく，ネットワーク遅延の増加による影響は結果的に小さいものとなる．

8. おわりに

本稿では，VM ストレージの動的かつ透過的な再配置機構を提案した．遠隔ライブマイグレーションにおいて VM が I/O 動作を継続したまま，拠点間で仮想ディスクを完全に再配置する．VM の I/O 動作にともなうオンデマンドなブロック再配置機能によって透過性を提供し，同時並行的に動作するバックグラウンドでのブロック再配置機能により I/O 性能の低下を緩和しながらディスク全体の移動を可能にする．

今後は提案機構を我々が開発中の仮想クラスタシステム[24),26)]に適用していく．その際，VM へのストレージ提供手法は，提案手法だけではなく既存のネットワークファイルシステムや雛形システムイメージに対する Copy-On-Write ディスク[13)]など[12)]，複数の手法を適切に組み合わせる予定である．本稿での提案手法は，ライブマイグレーションと連動して VM 固有のストレージデータを透過的に再配置するうえで重要な役割を担う．

謝辞　本研究は科研費（20700038）および CREST（情報システムの超低消費電力化を目指した技術革新と統合化技術）の助成を受けたものである．

参 考 文 献

1) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proc. 19th ACM symposium on Operating systems principles*, ACM Press (2003).
2) Bonnie++. http://sourceforge.net/projects/bonnie/
3) Bradford, R., Kotsovinos, E., Feldmann, A. and Schiöberg, H.: Live wide-area migration of virtual machines including local persistent state, *Proc. 3rd International Conference on Virtual Execution Environments*, pp.169–179, ACM Press (2007).
4) Breuer, P.T., Lopez, A.M. and Ares, A.G.: The enhanced network block device, *Linux Journal*, Vol.Issue 73 (2000).
5) Card, R., Ts'o, T. and Tweedie, S.: Design and Implementation of the Second Extended Filesystem, *Proc. 1st Dutch International Symposium on Linux* (1995).
6) Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live migration of virtual machines, *Proc. 2nd conference on Symposium on Networked Systems Design and Implementation*, pp.273–286, USENIX Association (2005).
7) Cluster File Systems, Inc.: Lustre: A scalable, high-performance file system, White Paper (2002).
8) Fasheh, M.: OCFS2: The Oracle clustered file system, version 2, *Proc. Linux Symposium*, Linux Symposium, pp.289–302 (2006).
9) Hirofuchi, T.: A relocatable storage I/O mechanism for live-migration of virtual machines over WAN, USENIX Annual Technical Conference (Poster) (2008).
10) Kozuch, M., Satyanarayanan, M., Bressound, T. and Ke, Y.: Efficient state transfer for Internet suspend/resume, Technical Report IRP-TR-02-03, Intel Research Pittsburgh (2002).
11) Luo, Y., Zhang, B., Wang, X., Wang, Z. and Sun, Y.: Live and incremental whole-system migration of virtual machines using block-bitmap, *Proc. Cluster 2008: IEEE International Conference on Cluster Computing*, IEEE Computer Society (2008).
12) Meyer, D.T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M.J., Hutchinson, N.C. and Warfield, A.: Parallax: virtual disks for virtual machines, *ACM SIGOPS Operating System Review*, Vol.42, No.4, pp.41–54 (2008).
13) Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S. and Rosenblum, M.: Optimizing the migration of virtual computers, *ACM SIGOPS Operating System Review*, Vol.36, No.SI, pp.377–390 (2002).
14) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: Internet small computer systems interface (iSCSI), RFC 3720 (2004).
15) Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H. and Steere, D.C.: Coda: A highly available file system for a distributed workstation environment, *IEEE Trans. Comput.*, Vol.39, No.4, pp.447–459 (1990).
16) Silicon Graphics Inc.: XFS filesystem structure, 2nd edition revision 1 (2006). http://xfs.org/
17) Sun Microsystems: NFS: Network file system protocol specification, RFC 1094 (1989).
18) Sun Microsystems: ZFS on-disk specification (draft) (2006). http://opensolaris.org/os/community/zfs/
19) Tweedie, S.C.: Journaling the Linux ext2fs Filesystem, *LinuxExpo'98* (1998).
20) VMware, Inc.: VMware Storage VMotion: Non-disruptive, live migration of virtual machine storage, Product Datasheet.
21) 広渕崇宏，小川宏高，中田秀基，伊藤　智，関口智嗣：仮想クラスタ遠隔ライブマイグレーションにおけるストレージアクセス最適化機構，情報処理学会研究報告（2008-HPC-116），pp.19–24, 情報処理学会 (2008).
22) 広渕崇宏，小川宏高，中田秀基，伊藤　智，関口智嗣：仮想クラスタ遠隔ライブマイグレーションにむけた仮想計算機ストレージの透過的再配置機構の評価，情報処理学会研究報告（2008-HPC-117），pp.7–12, 情報処理学会 (2008).
23) 山口実靖，小口正人，喜連川優：iSCSI 解析システムの構築と高遅延環境におけるシーケンシャルアクセスの性能向上に関する考察，電子情報通信学会論文誌 D-I，Vol.87, No.2, pp.216–231 (2004).
24) 中田秀基，横井　威，江原忠士，谷村勇輔，小川宏高，関口智嗣：仮想クラスタ管理システムの設計と実装，情報処理学会論文誌：コンピューティングシステム，Vol.48, No.ACS19, pp.13–24 (2007).
25) 小川宏高，中田秀基，広渕崇宏，伊藤　智，関口智嗣：仮想クラスタのステートレス化のための Rocks 5 ディスクレス化機構，情報処理学会研究報告（2008-HPC-116），pp.31–36, 情報処理学会 (2008).
26) 広渕崇宏，中田秀基，横井　威，江原忠士，谷村勇輔，小川宏高，関口智嗣：複数サイトにまたがる仮想クラスタの構築手法，第 6 回先進的計算基盤システムシンポジウム SACSIS 2008，pp.333–340 (2008).

(平成 20 年 10 月 3 日受付)
(平成 21 年 2 月 12 日採録)

広渕　崇宏（正会員）

2001 年京都大学理学部地球物理学教室卒業．2007 年奈良先端科学技術大学院大学情報科学研究科博士課程修了．同年独立行政法人産業技術総合研究所入所．グリッドコンピューティング，クラウドコンピューティング，Green IT の研究に従事．システムソフトウェア，ネットワーク，仮想化技術等に興味を持つ．博士（工学）．

小川　宏高（正会員）

1971 年生．東京大学大学院工学系研究科博士課程中退．1998 年より東京工業大学大学院情報理工学研究科助手．2003 年より産業技術総合研究所グリッド研究センター研究員．現在同所情報技術研究部門研究員．主な研究は，自己反映計算に基づいた拡張可能な Java 言語向け Just-In-Time コンパイラ，グリッド技術を用いてアプリケーションの実行をユーティリティサービスとして提供する GridASP 等．プログラミング言語処理系，仮想化技術，並列処理，グリッド技術等に興味を持つ．

中田　秀基（正会員）

1995 年東京大学大学院工学系研究科情報工学専攻博士課程修了．同年電子技術総合研究所入所．2001 年より独立行政法人産業技術総合技術研究所主任研究員．2001 年より 2005 年まで東京工業大学学術国際情報センター客員助教授．分散並列コンピューティング，プログラミング言語に興味を持つ．ACM 会員．博士（工学）．

伊藤　　智（正会員）

1982 年千葉大学理学部物理学科卒業，1987 年筑波大学大学院物理学専攻博士課程修了後，株式会社日立製作所に入社．中央研究所において材料シミュレーションの研究に従事し，以来ハイパフォーマンスコンピューティングに関する応用アルゴリズムの研究を推進．1999 年からは金融工学等を含むビジネス分野の研究にも従事．2002 年 6 月に独立行政法人産業技術総合研究所に入所．グリッド技術を中心に産業分野等への応用について研究開発を推進中．

関口　智嗣（正会員）

1959 年生．1982 年東京大学理学部情報科学科卒業．1984 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．情報アーキテクチャ部主任研究官．以来，データ駆動型スーパーコンピュータ SIGMA-1 の開発等の研究に従事．2001 年独立行政法人産業技術総合研究所に改組．2002 年 1 月より同所グリッド研究センターセンター長．2008 年 4 月より同所情報技術研究部門長．並列数値アルゴリズム，グリッドコンピューティング，IT による地球環境の理解に興味を持つ．市村賞，情報処理学会論文賞，文部科学大臣表彰科学技術賞（研究部門）賞受賞．地理情報システム学会理事，日本応用数理学会評議員，SIAM，IEEE 各会員．Open Grid Forum 諮問委員．