Vol.2014-ARC-210 No.3
Vol.2014-OS-129 No.3
2014/5/14

GPUを活用した仮想マシンマイグレーション

直井 由樹[†1,a)]　山田 浩史[†1,b)]

概要：クラウドコンピューティング環境において仮想化技術の利用が一般的になり，柔軟なサービス提供がより容易に行えるようになった．VM マイグレーションとは異なる物理マシン間で VM を移送する技術であり，サーバ上の VM 管理における非常に強力な機能の一つである．VM 管理において強力な役割を果たす VM マイグレーションであるが，移送処理にかかるコストが大きいという問題点が存在する．転送するデータのうち，VM のメモリ上のデータはデータサイズが特に大きい．それらの抽出処理における CPU 負荷の増加やデータ転送時のネットワーク負荷の増大が，VMM 上の他の VM のパフォーマンスにまで影響を与えてしまう場合もある．また，大容量のメモリを搭載した VM が使用される場合も多くなり，VM マイグレーションによるシステム負荷のさらなる増大が考えられる．本論文では演算アクセラレータとして Graphics Processin Unit(GPU) を対象とし，VM マイグレーション処理時に転送が必要となるメモリ上のデータを GPU 上で圧縮・伸張し，転送する手法を提案する．転送データのデータサイズを圧縮・伸張処理によって小さくすることで，VM マイグレーションのデータ転送時におけるネットワーク負荷の軽減を図る．提案するシステムを Xen-4.2.1 上に実装した．実装したシステムの評価は実行時間や CPU 使用率の面から行った．圧縮処理によるデータ転送時間の減少がみられたが，同時に圧縮処理によるオーバヘッドが生じ，VM マイグレーションの実行時間はやや増加する結果となった．CPU 使用率には大きな変化は生じなかったが，GPU 上の処理の実行時間の推測に基づくスリープ処理を併用した場合に，CPU 使用率減少の効果が得られた．

1. はじめに

仮想化技術の発達によって，物理資源の管理や柔軟な対応が可能なシステム構築が可能になり，クラウドコンピューティングなどの場においても仮想化技術が積極的に用いられている．ハードウェアの大容量化・高性能化が進み，一つのハードウェア上に一つのシステムを構築するだけでは処理能力に余剰が生じることも多くなった．仮想化技術によって，一つのハードウェア上で複数のシステムをそれぞれ独立した仮想マシン (VM) として稼働させることが可能になり，余剰リソースの有効活用や，電気代や設置面積の削減が期待できる．VM は仮想マシンモニタ (VMM) とよばれる仮想マシン管理ソフトウェア上で管理される．管理対象が実マシンからソフトウェア上に作成される VM になるため，それらのマシン環境の管理や設定変更が非常に容易になるといった利点も存在する．

VM マイグレーションは異なる物理マシン間で VM を移送する技術であり，サーバ上の VM 管理に置ける非常に強力な機能の一つである．VM マイグレーションは，VMM の機能によって抽出した，移送対象 VM の実行状況の記録データを移送先の物理ホストへ転送し，そのデータをもとに移送先の VMM 上に移送対象 VM の複製を構築することで実現される．転送するデータ内容は主に VM のメモリ上のデータの状態や CPU や各種デバイスの状態に関するもので構成される．VM マイグレーションの利点として，ハードウェメンテナンス時に別の物理ホストへ稼働中の VM を退避させたり，複数の物理ホスト間での負荷分散などが可能になるといったものが挙げられる．

VM 管理において協力な役割を果たす VM マイグレーションであるが，移送処理にかかるコストが大きいという問題点を抱えている．VM マイグレーションにおいては VM のメモリ上のデータをすべて転送する必要があるため，そのデータの抽出処理による CPU 負荷やデータ転送時のネットワーク負荷が増大してしまう．VM のメモリサイズが巨大な場合には VM の移送にかかる時間が長期化しやすく，システムに与える影響が大きくなりやすい [1]．移送時間の長期化は，負荷増大によるパフォーマンスの低下が長引くだけでなく，管理者にとって柔軟な VM 管理を行う上での障壁となる．昨今ではメモリサイズが十数 GB にも及ぶ VM が使用される場合も多くなり，VM マイグ

[†1] 現在，東京農工大学
Presently with Tokyo University of Agriculture and Technology
[a)] naoi@asg.cs.tuat.ac.jp
[b)] hiroshiy@cc.tuat.ac.jp

レーションが数分間にも及ぶ場合が増えてきている．ネットワーク環境への依存が強いサービスを提供している場合には，データ転送時のネットワーク負荷の増大も無視できない問題である [2]．

そこで本研究では，VM マイグレーションを拡張し，演算アクセラレータ上で転送データの圧縮・伸張処理を行う手法を提案する．本研究では演算アクセラレータとして Graphics Processing Unit (GPU) を対象とし，GPU 上で転送データを圧縮・伸張するプロセスを追加することで，VM マイグレーションのデータ転送時におけるネットワーク負荷の軽減と圧縮・伸張処理時の CPU 負荷の増加を抑える．また同時に GPU へのオフロードによって生じるオーバヘッドを検証し，VMM 上での GPU 活用の可能性を考察していく．

実装は Xen-4.2.1 の VM マイグレーションプロセスを拡張する形で行い，実装したシステムの評価は，実行時間や CPU 使用率に関して行った．GPU 上での圧縮処理によってデータ転送にかかる時間は減少した．しかしながら，圧縮処理によるオーバヘッドのほうが大きく，VM マイグレーションの実行時間はやや増加する結果となった．また，CPU 使用率はデフォルトの場合とほとんど差が生じなかったが，GPU 上の処理の実行時間の推測に基づくスリープ処理を実装することで，CPU 使用率減少の効果が得られた．

2. 背景

2.1 VM マイグレーション

VMM が提供する機能の一つとして，VM マイグレーションが挙げられる．この機能は異なる物理マシン間での VM の移送を可能にするものである．サーバ運用などにおいて，複数マシン間での負荷分散などの柔軟な資源管理を可能にするだけでなく，非常時に備えた VM 退避による可用性向上システム [3] などにも利用され，仮想化技術によって提供されるものの中でも強力な機能の一つである．

図 1 は VM マイグレーションの概略図である．VM マイグレーションは，移送対象 VM に関するすべての情報を移送元の VMM から転送し，その情報をもとに移送先の VMM 上で移送対象 VM の完全な複製を新たに構築することで実現される．引き継がれる情報はメモリ上のデータや CPU などの各種デバイスの実行状態に関するものである．ディスク上の情報に関しては共有ストレージなどを利用する場合が多い．VM の実行状態を引き継ぐ方法は VMM によって微妙に異なるが，移送先の物理ホスト上で VM の複製を生成する点に関しては同じである．移送終了後は移送先で構築された複製が稼働し，移送元の VM は解放される．

Xen でもこの機能は搭載されていて，共有ストレージを利用した VM マイグレーションが可能である．Xen-4.2.1 ではその時点の VM の実行状況を保存・復元する，スナップショット保存・復元プロセスを利用して VM マイグレーションが行われる．まず，スナップショット保存プロセスによって抽出した VM の実行状況が，共有ストレージを通じて移送先ホストへ転送される．それと同時に移送先ホストでは転送されてきた情報をもとに VM の構築が行われる．

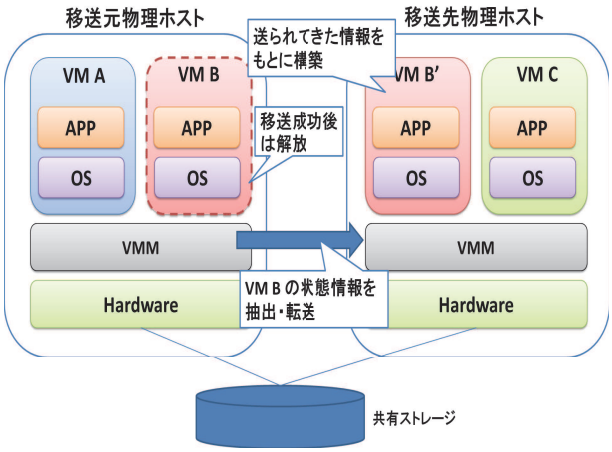

図 1　VM マイグレーションの概略図

2.2 クラウドサービスにおける GPU の提供

クラウド環境における GPU コンピューティングを可能にするサービスが始まりつつある．GPU メーカー NVIDIA は，Amazon Web Services や NIMBIX などのクラウドコンピューティングを提供する企業との提携を発表しており，GPU を活用したクラウドコンピューティングが浸透してきていることがうかがえる．Amazon Web Service は Amazon EC2 G2 インスタンス [4] を発表し，強力な GPU を利用可能なクラウドコンピューティング環境の提供を始めている．ユーザはクラウド上の GPU インスタンスを利用することで，自身のハードウェア環境に依存することなく 3D グラフィックや GPGPU を利用したアプリケーションを配信するサービスの構築を容易に行うことができる．NIMBIX ではサーバでの運用を目的とした NVIDIA 製のハイエンドな GPU を利用可能な，Nimbix Accelerated Compute Cloud (NACC)[5] と呼ばれるサービスの提供が開始され，映像や画像処理だけでなく科学モデルの演算などのクラウドコンピューティングが可能なサービスを展開している．このようなサービスの普及に従って，データセンタなどのマシンに高性能な GPU が搭載される機会が増え，それらを利用したアプリケーションの重要性が高まっている．近年では GPGPU プログラミングの開発環境の充実から GPGPU の活躍の場が増えつつあることから，さらなる重要性の向上が期待される．

Vol.2014-ARC-210 No.3
Vol.2014-OS-129 No.3
2014/5/14

3. 提案

3.1 VM マイグレーションにおける GPU の活用

本論文では，VM マイグレーションにおける GPU の活用を提案する．クラウド環境のおける仮想化技術や GPU の利用が普及しはじめ，VM 上のアプリケーションレベルにおける GPU の利用が行われているが，VMM レベルでの利用はまだ行われていない．VMM が提供する機能の一部を，従来の CPU だけによる実行から GPU を併用した実行へと対応させることで，CPU の負荷の軽減と同時に機能の高速化を図る．

VM マイグレーションは VM のバックアップ作成による可用性向上システムにおいても使用され，仮想化技術によって提供される機能を支えるプロセスの一つである．しかしながら，VM マイグレーションは CPU の使用量が大きく，VMM 上で動作する他のゲスト VM のパフォーマンスに影響を及ぼすことも起こりうる．

VMM マイグレーションの処理の一部において GPU を活用するという選択肢を設けることで，GPU の余剰リソースを有効に利用する一つの手段として確立させる．GPU を搭載したシステムによるサービスの普及が進めば，システムが提供しているサービスの内容によっては GPU リソースに余剰を抱える状況が生じることも考えられる．それらのリソースを VMM 上でも利用できるようにすることでさらなるリソースの有効活用が可能になる．

3.2 GPU 上での転送データの圧縮・伸張

VM マイグレーション中に転送するデータを GPU 上で圧縮して転送し，転送先で転送データを伸張することで，VM マイグレーションのネットワーク負荷を低減する．

VM マイグレーションでは VM メモリ上のすべてのデータを転送する必要があり，そのファイルサイズは数 GB にも及ぶことも少なくない．特に昨今では巨大なシステムにおいては一つの VM に割り当てられるメモリ量が数十 GB 以上にも及ぶ場合も増加してきている．転送するデータ量が多いほど，VM マイグレーション時のダウンタイムは大きくなりやすく，VM マイグレーションの負荷は増大する．VM マイグレーションにおけるデータ転送量を減らすことで，大量のメモリを搭載した VM を移送する場合であっても，管理者が VM マイグレーションをより実行しやすくすることが可能である．

3.3 CPU と GPU の処理の協調的なスケジューリング

VM マイグレーションにおける VM のメモリ転送時に必要となる各処理は並列実行可能であることがわかっている [2]．そこで，VM マイグレーションにおける各処理を並列実行させるのに加えて，GPU 上での転送データの圧縮処理も平行して実行するようなスケジューリングを提案する．CPU 上での処理と GPU 上での処理は適切に同期を行うことで，並列実行が可能である．CPU と GPU が互いに協調してそれぞれの処理を実行することで，VM マイグレーションをより効率的に行うことができる．

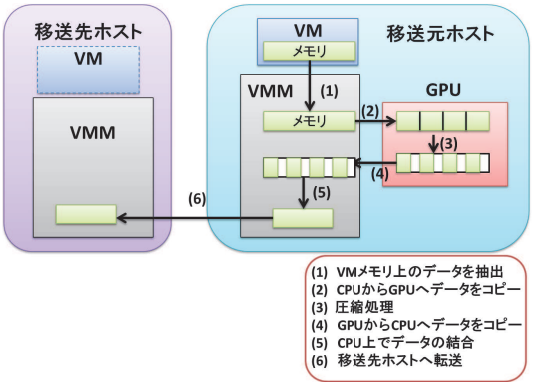

図 2 VM のメモリ上のデータの圧縮と転送処理の設計

4. 設計

4.1 移送元ホスト上での処理の設計

本手法では VM マイグレーション中に転送される転送データのうち，VM のメモリ上のデータにだけ圧縮処理を加える．その他のデバイスの状態や管理用のメタデータに関する部分には適用しない．それらの部分のデータサイズはメモリ上のデータに比べて非常に小さいため，圧縮による恩恵が十分に望めないためである．

VM マイグレーションにおける VM のメモリ上のデータの抽出やデータの転送は，VMM が提供するマイグレーションプロセスによって行われる．本手法では VM のメモリ上のデータの抽出と転送処理を拡張し，データ転送の前に GPU 上での圧縮処理を追加する．VM のメモリ上のデータに関する部分のみを拡張するため，その他のデータの転送時には GPU 上での圧縮処理は実行されない．

図 2 は，VM のメモリ上のデータの圧縮と転送処理の設計の概略図である．VMM のマイグレーションプロセスによって抽出した移送対象の VM のメモリ上のデータは，一度 GPU 上へコピーされ，分割・圧縮処理が行われる．圧縮処理は GPU 上の複数のスレッドで並列に実行される．圧縮処理が行われた転送データは CPU 上に再びコピーされ，分割・圧縮されたデータの結合処理が行われる．結合処理によって転送データは連続したデータのとなり，移送先ホストへと転送される．

4.2 移送先ホスト上での処理の設計

移送先ホストの VMM 上でも VM マイグレーションプロセスが動作し，移送元ホストが転送したデータを受信し，移送対象の VM を再構築していく．転送データの伸張処理は，圧縮処理と同様に，VM のメモリ上のデータに関する

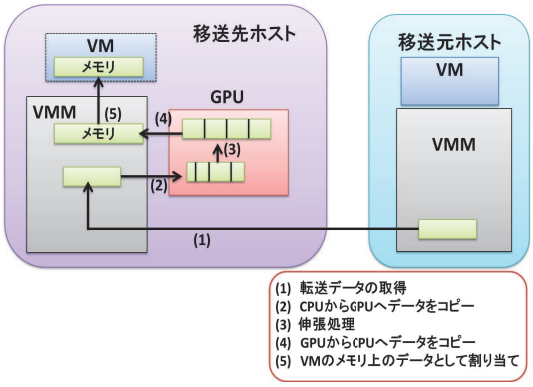

図 3 取得したデータの伸張処理の設計

部分にだけ適用される．移送先ホストでは，転送されてきたデータがメモリ上のデータであることが分かった場合には，VM へデータの割り当てを行う前に，圧縮された転送データの伸張処理を行う．

図 3 は圧縮処理の設計の概略図である．VMM は VM マイグレーションプロセスを通して転送データを取得し，VM へ取得したデータの割り当てを行う．取得したデータが VM のメモリ上のデータである場合には，一度そのデータを GPU 上へとコピーし，伸張処理を行う．伸張処理も圧縮処理と同様に GPU 上で複数スレッドによって並列に処理される．伸張処理が施されたデータは CPU 上へと再びコピーされ，VMM によって VM のメモリ上のデータとして割り当てられる．

5. 実装

本章では提案手法の実装に関して述べる．実装は Xen-4.2.1 の VM マイグレーションプロセスおよび，スナップショット保存・復元プロセスをベースにして行った．GPU 上での処理は Gdev を介した CUDA プログラムの実行によって実装し，実装した圧縮・伸張アルゴリズムには RLE 法をもとにしたものを用いている．また，VM マイグレーション時に実行スレッドを三つ用意し，VM のメモリ上のデータの抽出，GPU 上での圧縮，データの転送の三つの処理をそれぞれ別のスレッドで並列処理させることで，VM マイグレーションの処理効率を向上させる実装も行った．

5.1 Xen 上での実装

Xen-4.2.1 の VM マイグレーションプロセス中では，スナップショット保存・復元プロセスを利用して VM マイグレーションを行う形をとっている．VM に関する情報の抽出や転送処理にはスナップショット保存プロセスを，取得したデータの VM への割り当て処理にはスナップショット復元プロセスをそれぞれ利用している．そのため，VM マイグレーションの拡張はそれらのプロセスに手を加えることで行う．各プロセスの拡張に当たっては，移送対象の VM のメモリ上のデータに関する部分にのみ手を加える．

移送元のホストでは管理ドメイン上で VM マイグレーションプロセスが実行される．VM マイグレーションプロセスにおける，移送対象 VM のメモリ上のデータの抽出と移送先ホストへの転送処理は，スナップショット保存プロセスを利用することで行われる．GPU 上でのデータ圧縮処理は移送対象 VM のメモリ上のデータの抽出と転送の間に行われる．抽出されたデータは GPU 上に一度コピーされ，圧縮された後に再び CPU 上にコピーされる．圧縮処理は GPU 上の複数スレッド上で並列処理されるため，圧縮直後のデータは結合処理が必要な状態である．そのため，圧縮データは CPU 上で結合処理を施した後に移送先ホストへと転送される．一方で，移送先ホスト上の VM マイグレーションプロセスでは，スナップショット復元プロセスを利用し，転送されてきたデータをもとに VM を再構築していく．スナップショット復元プロセスに対しては，転送されてきたデータが VM のメモリ上のデータに対応するものだった場合に，GPU 上で伸張処理を行うような拡張を行う．転送されてきた圧縮データは圧縮時と同様に一度 GPU 上にコピーされ，伸張処理が施された後に CPU 上へと再びコピーされる．伸張処理が終了したデータは VM のメモリ上のデータとして，スナップショット復元プロセスによって VM のメモリに割り当てられる．なお，GPU 上での圧縮・伸張処理は各プロセス中から GPU 上にデータをコピーした後に，圧縮・伸張処理を記述した CUDA プログラムを実行することで行う．

5.2 CUDA と Gdev による GPGPU

GPU に関する処理は CUDA プログラムを Gdev[6] を通して実行することで実現される．Xen 上では NVIDIA 製の純正ドライバが正常に機能しないため，CUDA プログラムを Xen のプロセスから実行するために Gdev によるサポートを利用している．Gdev とは OS レベルで GPU プログラム用の API を提供するシステムである．Gdev では linux の nouveau ドライバを拡張することで CUDA 用の API が実装されているため，NVIDIA の純正ドライバを用いずに CUDA による GPGPU プログラミングが可能である．GPU 上のメモリ確保やスレッド数の設定，GPU 上での処理の開始や同期処理は GdevAPI を利用して行う．

通常の CUDA プログラムでは，CPU 上での処理と GPU 上で処理を両方とも CUDA プログラム用のソースファイルに記述するが，Gdev を用いた実装では，GPU 上での実行処理を記した CUDA プログラムを CPU 上で実行される C プログラム中でロードすることで行う．Xen-4.2.1 のスナップショット保存・復元プロセスは C プログラムで構成されているため，スナップショット保存・復元プロセス中で GPU 上での圧縮・伸張処理を実装した CUDA プログラムをロードし，実行する形で本手法は実装されている．

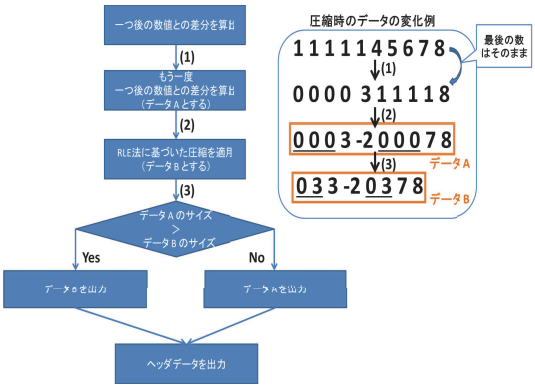

図 4　圧縮処理の流れ

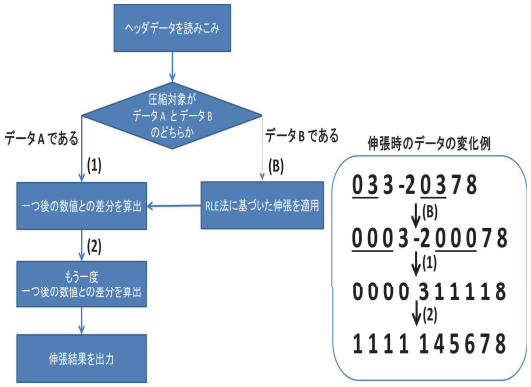

図 5　伸張処理の流れ

5.3　圧縮・伸張処理

5.3.1　RLE 法をベースとした圧縮・伸張アルゴリズム

圧縮アルゴリズムには RLE 法 [7] をベースに用いている．RLE 法とは非常にシンプルな圧縮手法の一つであり，ある数値の連続をその数値とその個数の二つの数値に変換することでデータの圧縮を図るものである．

RLE 法による圧縮に加えて，一つ後の数値との差分を二度にわたって算出する処理を加えている．この処理によって，単調増加または同じ数値の連続のデータが，連続した 0 へと変換される．変換後のデータは 0 の連続にだけ RLE 法を適用し圧縮をかける．圧縮・伸張時には 0 にだけ注目すればよく，処理を単純化することで，条件分岐を苦手とする GPU でも処理を行いやすくしている．また，単純な RLE 法を実装しているため，圧縮後のサイズが元のサイズを超えてしまう場合がある．例えば，0 が一つだけ出現した場合にはその部分を 01 と表現しなくてはならず，一つの 0 を表現するために二つの数値が使用されることになり，サイズの増大を招く要因となる．そのような場合には RLE 法を適用する前のデータを結果として出力することで，出力データのサイズが元のサイズを超過しない処置を施している．

図 4，**図 5** は，圧縮・伸張処理の大まかな流れを示したものである．圧縮処理時には，11111 や 45678 といった，同じ数値が連続する数列や単調増加がみられる数列が二度にわたる差分算出によって連続した 0 へと変化している．差分算出は圧縮時には先頭の数値から行っていき，伸長時には後方の数値から行っていく．このとき一番最後の数値だけはそのままの値を保ち，伸張処理時に値を復元する際の手掛かりとして用いられる．連続した 0 の数列は RLE 法にしたがって，二つの数値へと変換される．RLE 法による変換前のデータと変換後のデータのうち，よりデータサイズが小さいほうを最終的な変換データとして出力する．どちらのデータを出力したかどうかは，圧縮データのヘッダデータに記録され，伸張処理時はそのヘッダデータをもとにどの処理を行うかが判断される．

5.3.2　圧縮後データのヘッダ

GPU 上では圧縮時と伸張時で同数のスレッドが動作し，それぞれが同じデータ領域に対して処理を行うために，圧縮データにヘッダ情報を記録している．ヘッダにはそのスレッドが処理を行うデータの先頭の位置が保存されている．RLE 法によってサイズ超過が起きた部分はこの位置データをビット反転して記録することで，処理開始位置とともに行うべき処理を伸張時に判断できるように施してある．

5.4　マイグレーション時の各処理のスケジューリング

Xen-4.2.1 のデフォルトの VM マイグレーション処理におけるメモリ上のデータ転送部分の中心部は，以下の手順を反復的に実行することで実装されている．

(1) 転送するべきページのチェック (dirty bitmap のチェック)
(2) 転送が必要な各ページに関するメタデータの取得
(3) メモリ上の転送が必要なページのデータを取得
(4) メタデータの転送
(5) メモリ上のデータの転送

説明の簡略化のため，上記の処理群を“ループ A”と以後呼ぶことにする．Xen-4.2.1 のデフォルトの VM マイグレーションでは，移送対象 VM のメモリ上のデータ全体に対してループ A を一度に実行するのではなく，1024 ページ (4MB) ごとに複数回にわたってループ A が実行される．したがって，移送対象 VM のメモリ上のデータは小分けにされて転送されていく．例えば，1GB のメモリ上の全データを転送する場合には，ループ A 一回当たりの転送量は 4MB なので，ループ A は 256 回実行されることになる．

本手法では，VM のメモリ上のデータの転送時に GPU による圧縮処理が拡張されるため，ループ A の処理内容は以下のような手順になる．

(1) 転送するべきページのチェック (dirty bitmap のチェック)
(2) 転送が必要な各ページに関するメタデータの取得
(3) メモリ上の転送が必要なページのデータを取得
(4) CPU 上から GPU 上へ転送データをコピー
(5) GPU 上で転送データを圧縮

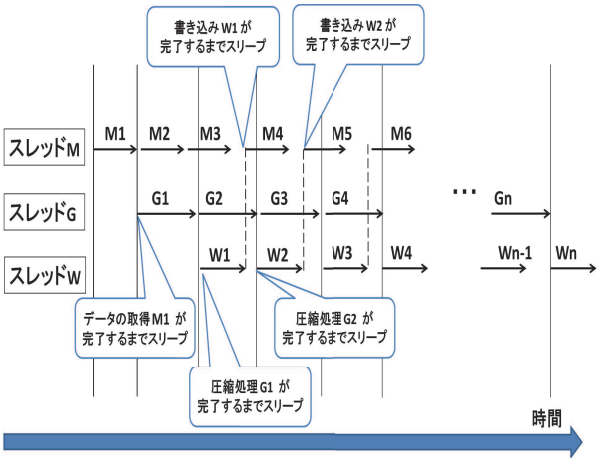

図 6　各スレッドの処理の時系列図

(6) GPU 上から CPU 上へ転送データをコピー
(7) メタデータの転送
(8) メモリ上のデータの転送

このとき，これらの処理を以下の三つのグループに分けることができる．

- 転送データのマッピングに関する処理
  (1) 転送するべきページのチェック (dirty bitmap のチェック)
  (2) 転送が必要な各ページに関するメタデータの取得
  (3) メモリ上の転送が必要なページのデータを取得
- GPU 上での圧縮処理に関する処理
  (1) CPU 上から GPU 上へ転送データをコピー
  (2) GPU 上で転送データを圧縮
  (3) GPU 上から CPU 上へ転送データをコピー
- 移送先へのデータ転送に関する処理
  (1) メタデータの転送
  (2) メモリ上のデータの転送

本手法では，これら三つの処理グループをそれぞれ担当するスレッドを用意し，三つのスレッドが同期をとりながら処理を並列実行させることで，VM マイグレーションの各処理のスケジューリングをより効率的なものにしている．

5.5 GPU 上での処理時間の推測を利用したスレッドスリープ

GPU 上での圧縮処理を行うスレッドは，GPU との同期時にビジーウェイトする必要がある．このビジーウェイトは，GPU へのオフロードによって CPU 負荷の軽減を図る本手法にとって障壁となりうる．そこで，GPU 上での処理の実行とともにその処理時間をサンプリングし，そこから推測される時間分だけビジーウェイトではなくスリープさせることで，CPU 負荷の軽減を行う仕組みを実装した．

本実装では，繰り返し実行される圧縮処理のうち，はじめから二十回目までの実行の処理時間をサンプリングする．

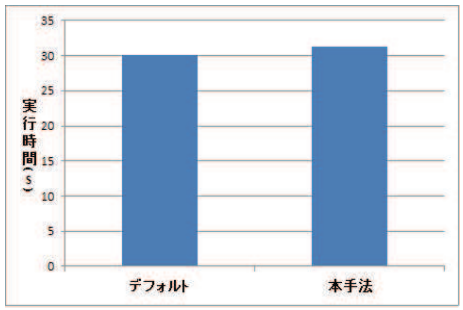

図 7　本手法適用による実行時間の変化

そして，サンプリングした処理時間のうち，最も短かった時間をスリープさせる時間として用いる．はじめから二十回目までの実行の処理時間をサンプリングすることで，GPU の使用率がピークに達する前にサンプルを得ることが可能である．ピーク前の処理はピーク時に比べて比較的短時間で終了する傾向がある．その時の処理時間をスリープ時間として用いることで，スリープ時間が実際の処理時間を上回ってしまうことで生じるオーバヘッドを少なくすることも可能であると考えられる．

6. 実験

実装したシステムのパフォーマンスの評価にあたって，VM マイグレーションの実行時間，CPU 使用率を測定した．また，GPU 上の処理時間の推測によるスレッドのスリープを使用した場合についても実行時間と CPU 使用率を計測し，その有用性を調査した．

6.1 実行環境

提案手法を Xen-4.2.1 上に実装した．ドメイン 0 とドメイン U のカーネルには linux 3.6.1 を用いた．マシンは DELL PowerEdge T320 を用いた．メモリは 16GB，CPU は Intel Xeon CPU E5-2470 2.30GHz を 8 × 2 コア (ハイパースレッディング) 搭載している．GPU は NVIDIA Quadro6000 を使用した．

6.2 VM マイグレーションの実行時間

メモリ 1GB の VM をマイグレーションしたときのプロセスの実行時間を計測した．**図 7** は，Xen-4.2.1 のデフォルトの VM マイグレーションと，本手法による VM マイグレーションの各実行時間をグラフにしたものである．デフォルトの VM マイグレーションに比べて，本手法による VM マイグレーションはやや実行時間がやや長くなることがわかった．GPU 上での転送データの圧縮処理や複数スレッドの並列処理による実行時間の減少量に比べて，GPU 上での圧縮処理によるオーバヘッドが大きいことが原因であると考えられる．

表 1 は，VM マイグレーションにおける，VM のメモリ上のデータの転送部分の実行時間と，そのうちの各処理ご

表 1　メモリ上のデータ転送部分における各処理の実行時間 (s)

処理内容	デフォルト	本手法
処理全体	22.25	23.65
メモリ上のデータの抽出	4.69	4.70
CPU・GPU 間のメモリコピー	0	1.27
GPU 上での圧縮処理	0	21.97
データの転送	17.17	1.61

との実行時間を示している．デフォルトの VM マイグレーションではデータの抽出と転送は逐次的に実行されるため，二つの処理時間の合計がメモリ上のデータの転送処理全体の処理時間におおよそ匹敵する．二つの処理時間の合計が全体の処理時間に満たないのは，同時に転送されるメタデータに関する処理やマイグレーションの続行の判定などの，その他の処理にかかる時間が全体の処理時間に含まれているためである．一方で，本手法による VM マイグレーションでは，メモリ上のデータの抽出と，GPU 上での圧縮処理，データの転送は並列に行われる．したがって，メモリ上のデータの転送処理全体の処理時間において，最も実行時間のかかる処理が大きな支配項となる．転送データの圧縮によって転送処理にかかる時間は減少しているが，本来の転送処理にかかっていた時間よりも多くの時間が圧縮処理に割かれているため，結果的に VM マイグレーションの実行時間の増加が生じている．なお，CPU・GPU 間のメモリコピーにかかった時間は圧縮処理による時間に比べると小さく，CPU・GPU 間のメモリコピーによるオーバヘッドは，支配項としてはあまり大きくないことがわかる．

GPU 上の圧縮処理によるオーバヘッドが大きな理由として，GPU プログラム内のループ処理や条件分岐が多いことから，GPU の並列性を生かすことができていないためであると考えられる．GPU 上では実行命令がスレッド間で分岐する場合には，それらのスレッドを並列実行できず，他方のスレッドの実行が終了してからまた他方のスレッドの実行が行われる．今回の実装では比較的単純な圧縮・伸張アルゴリズムを用いているが，ループ処理や条件分岐は少数ながら存在しているため，対象のデータによって実行命令が分岐しやすい．単純なアルゴリズムを用いている場合でもこのように差が出ているため，より GPU に適した圧縮アルゴリズムの実装の必要性がうかがえる．

また，圧縮アルゴリズムの問題に加えて，GPU ドライバによるリクロッキングサポートの問題 [8][9] も一つの要因として考えられる．この問題は，nouveau などの NVIDIA の純正ドライバ以外のドライバを使用した際に，純正ドライバを使用した際に比べて GPU のパフォーマンスの低下が生じるものである．本実装では linux が提供する nouveau ドライバを用いているため，リクロッキングサポートによる問題を受けていることが考えられる．GPU 上での圧縮処理は，本手法による VM マイグレーションの実行時間の大きな支配項となっている．したがって，リクロッキングサポートの問題の解決によって GPU 上の処理が高速化すれば，大きく性能を向上させることができると考えられる．

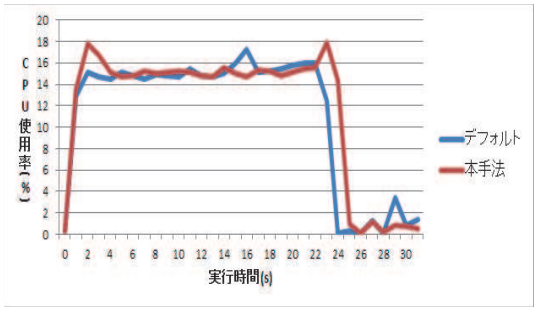

図 8　VM マイグレーション時の CPU 使用率

6.3　VM マイグレーション時の CPU 使用率

GPU へのオフロードによる CPU 使用率への影響を検証するため，VM マイグレーション中のドメイン 0 の CPU 使用率を計測した．図 8 は，Xen-4.2.1 のデフォルトの VM マイグレーションと，本手法によるマイグレーション実行時の CPU 使用率の変化をグラフにしたものである．このグラフから，双方の CPU 使用率には大きな差が生じていないことがわかる．これは，GPU 上でデータの圧縮を行うことで，移送先ホストとのデータ IO による CPU 負荷が小さくなるのと同時に，GPU との同期処理時に発生するビジーウェイトによる CPU 負荷の増加が起こっているためであると考えられる．今回の実装では，GPU に関する処理を担当しているスレッドが，GPU 上ので処理の実行中は常にビジーウェイトしているため，その影響を大きく受けてしまっている．CUDA が提供する API の仕様のため，この問題の根本的な解決は困難であるが，CPU リソースの使用量を抑えるという観点から，対処すべき重要な要素であると考えられる．

6.4　GPU 上での圧縮処理中のスリープによる効果

GPU との同期処理時のベジーウェイトによる CPU 使用率の増加という問題への対処法として実装を行った，GPU 上の処理の実行時間のサンプリングと推測に基づくスレッドのスリープによる効果を検証した．

図 9 は本手法による圧縮処理に加えて，処理時間の推測に基づくスレッドのスリープを行った場合と行わなかった場合の CPU 使用率を比較したものである．実行の初期段階では，GPU 上の処理の実行時間のサンプリングを行っているため，双方の CPU 使用率に差は生じていない．しかしながら，サンプリングが終了し，推測に基づくスリープ処理を開始した後は CPU 使用率が 9%前後まで低下している．実行時間によるサンプリングと推測に基づくスリープ処理によって，CPU 資源の使用が低く抑えられている

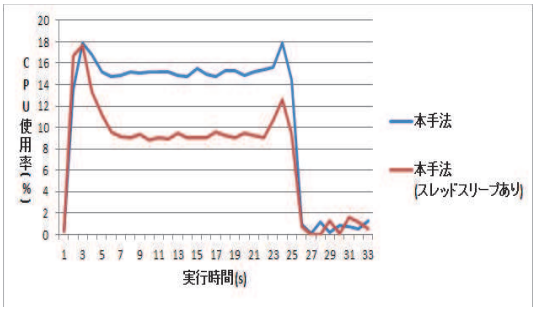

図 9　スレッドスリープによる CPU 使用率への影響

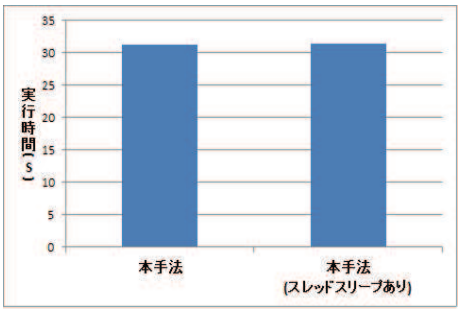

図 10　スレッドスリープによる実行時間への影響

ことがわかる．

図 10 は，本手法による圧縮処理に加えて，処理時間の推測に基づくスレッドのスリープを行った場合と行わなかった場合の実行時間を比較したものである．実験の条件は，6.2 項の評価実験と同様である．サンプリングした処理時間に基づく推測によるスリープを行う場合，スリープ時間が実際の処理時間を上回ってしまうことによる実行時間の増加が懸念される．しかしながら，双方の実行時間にはほとんど差が生じなかった．処理の早い段階でサンプリングした処理時間のうち，最も短かった時間をスリープ時間として使用することで，そのような問題を回避することができているものと考えられる．

7. 関連研究

これまでさまざまな手法を用いて VM マイグレーションの高速化が図られてきたが，次のような課題が残されている．

PMigrate[2] ではマイグレーション処理の並列化と潤沢なリソースの活用によって処理時間を大幅に小さくしているが，一方で処理中の CPU・ネットワーク負荷の上昇が通常の場合と比べて非常に大きくなってしまっている．PMigrate では使用するリソース量を制限することで影響を抑える仕組みも同時に実装しているが，並列化による負荷の増加は無視できない問題である．また，サーバ環境に置いて GPU が積極的に用いられだしたのは最近の出来事であり，PMigrate では GPU をも利用した並列処理は提案されていない．並列化された処理のうち，GPU に適した処理をオフロードすることができれば負荷をより小さくすることも可能であると考えられる．

deltacompression[10] や MiG[11] では，転送データの圧縮を行うことでマイグレーション時間の短縮やネットワーク負荷の低減が図られている．しかしながら，圧縮処理自体による CPU 負荷が懸念される．転送データのサイズを小さくすることでマイグレーション時間の短縮やそのダウンタイムによるサービスへの影響を抑えることには成功しているが，圧縮処理による CPU 負荷の増加については着目されていない．

delta compression の手法で利用されている圧縮処理には，XOR 演算や RLE 法による圧縮が用いられている．これらの演算処理は原理的には複雑な処理があまり必要でないため，GPU が不得意な処理が少ないと考えられる．現在の GPGPU 環境を利用することで，GPU へのオフロードの実現も十分に可能であると考えられ，GPU へのオフロードによって CPU の負荷の軽減の可能性がある．

MiG においても，複雑なアルゴリズムの圧縮手法を用いている場面もあるが，メモリ上のデータの比較部分などでは GPU による大規模データを対象とした並列処理が生かせる部分も存在すると考えられる．用いられている圧縮手法の中で GPU に適した処理をオフロードすることができれば複雑なアルゴリズムによる CPU 負荷の増加を抑えることも可能であると考えられる．

VM マイグレーションの負荷の軽減は，Remus[3] などの可用性向上技術の性能を高めることにもつながる．Remus では定常的な VM の状態情報の転送によってネットワーク負荷が通常の場合よりも大きくかかり続けるため，ネットワークへの依存度が大きいサービスに対しては不向きである．VM マイグレーションをより低負荷で行うことができれば，そのようなサービスを提供するサーバでの Remus の導入も行いやすくなる．

8. おわりに

本論文では VM マグレーションにおける転送データを GPU 上で圧縮・伸張する手法を提案し，Xen-4.2.1 の VM マイグレーションプロセスの拡張を行った．実装したシステムによる，VM マイグレーションの実行時間と CPU 使用率への影響を評価したところ，圧縮処理によってデータの転送時間は減少したが，圧縮処理のオーバヘッドのほうがやや大きく，実行時間の増加がみられた．一方で，CPU 使用率にはほとんど差が生じなかったが，GPU 上の処理の実行時間の推測に基づくスリープ処理を併用した場合に，CPU 使用率減少の効果が得られた．

GPU 上での処理を実装することはできたが，同時にスレッドの同期処理による CPU 負荷の増大が起こることがわかった．GPU 上での処理時間が長期化すると，GPU 上の処理が終了するまで CPU が待機する必要が生じる．CUDA によって提供される GPU スレッドの同期関数で

は，GPU 上の全スレッドが終了するまで CPU がビジーウェイトするため，CPU 負荷が大きい．GPU 上の処理をできるだけ高速化するためにも，GPU 上の各種メモリの特徴 (アクセス速度，保存容量など) や，条件分岐などの GPU が苦手とする処理を考慮した GPU プログラムの作成が不可欠であると考えられる．

参考文献

[1] Koto, A., Yamada, H., Ohmura, K. and Kono, K.: Towards Unobtrusive VM Live Migration for Cloud Computing Platforms, *Proceedings of the Third ACM SIGOPS Asia-Pacific Conference on Systems*, APSys'12, pp. 7–7 (2012).

[2] Song, X., Shi, J., Liu, R., Yang, J. and Chen, H.: Parallelizing Live Migration of Virtual Machines, *Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '13)*, pp. 85–96 (2013).

[3] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N. and Warfield, A.: Remus: High Availability via Asynchronous Virtual Machine Replication, *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, pp. 161–174 (2008).

[4] Amazon: Announcing New Amazon EC2 (GPU Instance Type), http://aws.amazon.com/jp/about-aws/whatsnew/2013/11/04/announcingnewamazonec2-gpuinstancetype/.

[5] NIMBIX: Nimbix - NACC, https://nacc.nimbix.net/landing#tasks.

[6] Kato, S., McThrow, M., Maltzahn, C. and Brandt, S.: Gdev: First-class GPU Resource Management in the Operating System, *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*, pp. 37–37 (2012).

[7] Hiroi, M.: Algorithms with Python / 連長圧縮，Makoto Hiroi（オンライン），入手先 ⟨http://www.geocities.jp/m_hiroi/light/pyalgo29.html⟩（参照 2014-1-23）.

[8] Larabel, M.: [Phoronix] Nouveau vs. NVIDIA Linux Comparison Shows Shortcomings, Phoronix Media (online), available from ⟨http://www.phoronix.com/scan.php?page=article&item=nouveau_april_2013&num=1⟩ (accessed 2014-1-26).

[9] Larabel, M.: [Phoronix] Open-Source Nouveau Driver Remains Slow For NVIDIA's Fermi, Kepler, Phoronix Media (online), available from ⟨http://www.phoronix.com/scan.php?page=article&item=nvidia_nouveau_fermkep&num=1⟩ (accessed 2014-1-26).

[10] Svard, P., Hudzia, B., Tordsson, J. and Elmroth, E.: Evaluation of delta compression techniques for efficient live migration of large virtual machines, *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '11)*, pp. 111–120 (2011).

[11] Rai, A., Ramjee, R., Anand, A., Padmanabhan, V. N. and Varghese, G.: MiG: Efficient Migration of Desktop VMs Using Semantic Compression, *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*, pp. 25–36 (2013).