Vol.2016-OS-137 No.2
2016/5/30

データベース管理システムと協調した仮想マシン移送

河村 裕太[†1,a)]　山田 浩史[1,†1,b)]

概要：データセンタやクラスタ環境で仮想化技術の利用が広がっている．こうした環境では，仮想マシン（VM）を稼働させたまま異なるホスト間で移動する，VM Live Migration という技術を用いることができる．VM を退避したうえでの物理マシンのメンテナンスや，高負荷な VM を移送させて負荷分散が容易になるメリットがある．しかし，現在広く用いられている Pre-copy 方式は，メモリの全ページを転送し，変更の生じたページを逐次再送するため，VM の移送に時間がかかってしまう．また，DBMS を稼働させている VM の移送では事態が深刻化する．DBMS は大容量のデータをキャッシュし，トランザクションによるページの書き換えがあるため，ページの再送が頻発してしまう．本研究では DBMS のキャッシュのページ転送を削減し，仮想マシン移送を高速化させる手法を提案する．提案手法は DBMS のキャッシュのページ転送を省略し，共有ストレージから復元することでページ転送量を減らす．提案手法を Xen 4.4.2, Linux 3.18.20, MySQL 5.6.26 に対して実装した．読み取り専用なワークロードを実行する DBMS を稼働させている VM を移送したところ，最大で約 20%の総移送時間を削減できた．

Database-Assisted Live Migration of Virtual Machine

Yuta Kawamura[†1,a)]　Hiroshi Yamada[1,†1,b)]

1. はじめに

データセンタやクラスタ環境で仮想化技術が利用されている．仮想化技術によって，1 台の物理マシン上で複数の異なる OS を仮想マシン（VM）として並列に実行できるようになる．仮想化技術を用いることで，物理的リソースを効率よく使えるようになるため，ハードウェアや消費電力の削減につながる．実際に Amazon EC2[1] や，Google Cloud Platform[2] といったクラウド環境では，仮想化技術が採用されている．

仮想化技術の応用に VM Live Migration がある．VM Live Migration とは，仮想マシンを稼働させたまま，異なるホスト間で移動する技術である．これにより，VM を退避したうえでの物理マシンのメンテナンスや，高負荷な VM の移送による負荷分散が容易になる．実際にオープンソースの仮想化ソフトウェアである，Xen[3]，KVM[4]，VirtualBox[5] でサポートされており，Google のデータセンタでは，VM Live Migration を用いて運用されている．

Database-as-a-Service（DaaS）では，データベース管理システム（DBMS）を仮想マシン上で稼働させており，クライアントに DBMS を提供している．1 台の物理マシン上で複数の DBMS を稼働させることでリソースを有効活用している．実際に Amazon RDS[6] や Microsoft SQL Azure[7] などのサービスで利用されている．

しかし，大容量のメモリの VM の移送には時間がかかってしまう．現行の仮想マシン移送で広く用いられている Pre-copy 方式 [8] では，メモリの全ページを転送し，変更の生じたページを逐次再送している．特に DBMS はキャッシュサイズが大きいほど性能が出るため，メモリを多く使う．またライトトランザクションによるメモリの書き換えも激しい．このようにしてメモリのサイズが大きくなればなるほど，移送中に変更が生じるページが多くなり，移送が長期化してしまう．実際に 8GB の VM と 16GB の VM

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，東京農工大学
Presently with Tokyo University of Agriculture and Technology
[a)] yutak@asg.cs.tuat.ac.jp
[b)] hiroshiy@cc.tuat.ac.jp

の移送時間には 2 倍の差がある．移送中は CPU やネットワークなどのリソースを使用するため，VM のパフォーマンスが落ちてしまうため，移送は可能な限り短く完了することが望ましい．

本研究では，DBMS が稼働する，メモリサイズの大きい VM を高速に Live Migration する手法を提案する．提案手法では，DBMS のメモリに含まれるページは共有ストレージに存在することに着目し，DB キャッシュは共有ストレージから復元する．移送元ホストでは，本来 DB キャッシュの情報は DBMS 自身にしかわからないが，それを移送プロセスに伝えることで DB キャッシュのページ転送を省略する．移送先ホストでは，転送されなかった DB キャッシュを共有ストレージから復元する．

提案手法を Xen 4.4.2，Linux 3.18.20，MySQL 5.6.26 上に実装を行った．また，提案手法の性能の評価実験を行った．読み取り専用のワークロードとして Sysbench を実行しながら仮想マシン移送を実行した結果，最大で約 20%総移送時間を短くすることができた．

2. 背景

仮想化技術とは，一台のマシンの OS を含む全ての機構を仮想化し，仮想マシン（VM）を物理マシン上で動作させる技術である．仮想化技術によって 1 台の物理マシン上で複数の VM を同時に動作させることが可能になりリソースが効率的に使用され，ハードウェアや消費電力の削減につながる．

クラウド環境では仮想化技術が運用されている．管理者は高性能なサーバマシンを仮想化し，一方クライアントは VM の機能を利用している．管理者はリソースを効率的に使用してハードウェアや消費電力を削減できるほか，VM Live Migration という技術によってマシンの負荷分散やメンテナンスが容易になる．クライアントは必用な分の資源を利用できる．

IaaS の形態の 1 つとして Database-as-a-Service（DaaS）がある．クライアントはサーバマシン上の VM の上で稼働するデータベースを利用する．DaaS を提供しているサービスに Amazon RDS[6] や Microsoft SQL Azure[7] などがある．Amazon RDS は，MySQL や PostgreSQL，MariaDB などを提供しており，Microsoft SQL Azure は，SQL Database を提供している．

2.1 VM Live Migration

VM Live Migration[8] は VM 上の全てのサービスを動作させたまま異なるホスト間を移動させる技術である．この技術により，高負荷な VM の移送を利用した負荷分散やマシン上の VM を退避したうえでのマシンメンテナンスが容易になった．

現在広く用いられている Pre-copy 方式は，あらかじめできるだけメモリのページを転送しておいて，VM をサスペンドしたときの転送量を少なくする手法である．ページを複数回転送するためページ転送量が多く，総移送時間は長くなってしまう．しかしダウンタイムが小さく，移送後に VM の性能劣化がない，移送中にエラーが起きても VM を復元できるなどの利点がある．

2.2 Xen における VM Live Migration

Xen においても Pre-copy 方式の仮想マシン移送が採用されている．仮想マシン移送を行う関数は xc_domain_save() と xc_domain_restore() であり，Dom0 上で実行される．移送元のホストは SSH で移送先ホストに接続し，移送元ホスト上で xc_domain_save()，移送先ホスト上で xc_domain_restore() を実行する．

xc_domain_save() の主な機能は，メモリのページの転送である．これからページ転送の流れを説明する．まず一度に転送できる最大数を batch とし，VM 上の物理ページであるページフレームナンバー（Page Frame Number：PFN）とそのページが確保されている，壊れているなどを示す属性を順番に batch 分読み取る．次に batch の数，PFN の属性を転送する．先にページの属性を転送することで，転送する必要のないページの転送を避けることができる．最後に読み取られた VM の PFN の先頭アドレスにポインタを合わせ，実際にページを転送する．ページの属性を見て，そのページが壊れていたり，ただメモリ確保されているだけの領域は転送する必要がない．残りページ数や転送回数が閾値を超えたら VM をサスペンドし，残りページや CPU レジスタなどを転送する．

xc_domain_restore() の主な機能は，転送されたページを受け取り，そのページを VM に当てはめていくことである．xc_domain_save() 側でページ転送の順序は保たれているので，xc_domain_restore() は順番に読み取って，適切な場所に当てはめていくだけである．

2.3 問題点

既存の仮想マシン移送に用いられている Pre-copy 方式はページ転送量が多く，総移送時間が長くなってしまうという問題を抱えている．メモリサイズが大きくなればなるほど移送に時間がかかるため，ページ転送中に書き換わるページも多くなってしまう．ワークロードによってはページの書き換わる速度が転送速度を上回り，総移送時間が長期化し，ダウンタイムが大きくなってしまうこともある．

また DaaS のような環境を想定すると事態は深刻化する．DBMS は大容量のデータをメモリをキャッシュしている．しかし DBMS が実行するトランザクションが DB キャッシュに入りきらない場合，DB キャッシュのスワップアウトが起こり，再度スワップアウトされた PFN のページを転送しなくてはいけない．またライトトランザクションの

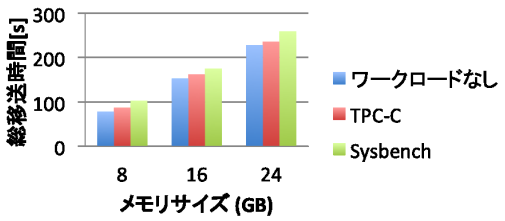

図 1 総移送時間
Fig. 1 Migration time

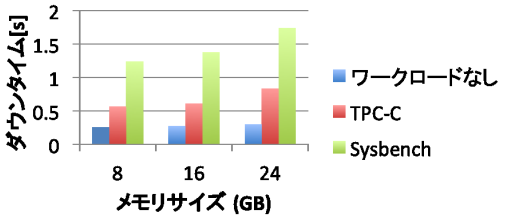

図 2 ダウンタイム
Fig. 2 Downtime

場合はスワップアウトされなくても DB キャッシュのページが書き換わり，再転送の対象となる．

DBMS を稼働させている仮想マシンの移送について，次の方法で予備実験を行った．ワークロードをなにもかけていない VM の移送と，TPC-C 及び Sysbench というベンチマークを実行している VM の移送にかかる総移送時間とダウンタイムを計測する．TPC-C と Sysbench は DB キャッシュが書き換えられるように実行する．VM のメモリサイズは 8GB，16GB，24GB とし，DB キャッシュのサイズは 6GB，12GB，18GB とする．

実験結果の総移送時間とダウンタイムを**図 1** と**図 2** に示す．VM のメモリサイズが大きくなるほど，総移送時間とダウンタイムが大きくなっている．またベンチマークを実行している VM の総移送時間はワークロードなしよりも長く，ダウンタイムの場合は数倍大きくなった．総移送時間が長くなったのはベンチマークによって DB キャッシュが書き換えられ，その DB キャッシュを再度転送したためであると考えられる．ダウンタイムが大きくなったのは，ベンチマークによって Stop-and-copy フェーズの直前までメモリが書き換えられ，Stop-and-copy フェーズの転送量が多くなってしまったためであると考えられる．

以上の結果から DBMS を稼働してトランザクションを処理している，メモリサイズの大きい VM の移送には総移送時間が長くなってしまうことが確認できた．故にメモリサイズの大きい VM を効率的に移送する方式が必要であるといえる．

3. 関連研究

VM Live Migration に移送方式に関する研究は数多くされてきた．現行の Pre-copy 形式はページ転送量が多く，総移送時間が長くなってしまう．Java Aware VM Migration（JAVMM）[10] は Java Virtual Machine（JVM）と連携した仮想マシン移送手法である．JAVMM では，使われていないページの転送はしないことと，頻繁に更新されるページの転送は最後にすることで，余分なページ転送を削減している．Efficient VM Live Migration[11] は，共有ストレージを利用してカーネルキャッシュのページ転送を削減し，総移送時間を短くする手法である．

VM の Live Migration と同様にデータベースの Live Migration も数多く研究されている．これから示す研究は Shared Process Model を対象としている．Shared Process Model はホスト上で稼働する 1 つの DBMS プロセスが複数のテナントセルを管理している．Albatross[12] は，ストレージが NAS で共有されている環境の低負荷なデータベース移送を提供する．移送元ホストではセルの snapshot を作成し，移送先ホストへ転送する．移送元ホストは snapshot をもとに共有ストレージから DB キャッシュの復元を行う．Zephyr[13] は，ストレージが共有されていない環境のデータベース移送を提供する．Zephyr は移送元ホストのデータベースのページを移送先ホストへ逐次転送する．Madeus[14] はストレージが共有されていない環境で，効率的なデータベース移送を行うミドルウェアである．Lasy Snapshot Isolation Rule（LSIR）という転送ルールを実装し，最低限のトランザクションを転送をする．Madeus によって commit はまとめて実行され，ディスクへの書き込み回数を減らしている．

Remus[15] は，VM が障害発生してもすぐに復旧させる技術である．定期的に checkpointing と呼ばれる VM の複製を行い，障害発生後すぐにキャッシュがウォームアップされている状態で VM を再開することができる．RemusDB[16] は，Remus を DBMS 向けに最適化した技術であり，checkpointing 時のページ転送量を抑える．

3.1 課題

仮想マシン移送，データベース移送について様々な研究がなされてきたが，DBMS が稼働する VM の効率的な移送については研究されていない．JAVMM では，JVM の情報を利用した仮想マシン移送をしているが，DBMS の情報を利用した仮想マシン移送は提案されていない．また Efficient Live Migration では，共有ストレージを利用してカーネルキャッシュのページ転送を削減したが，DB キャッシュは DBMS に管理されているため，この手法を適用できない．またデータベース移送は，VM レベルの移送に対応していないため，DBMS を稼働させている VM の移送に適用できず，DaaS 環境では利用できない．

本研究では，DBMS を稼働している VM の効率的な移送を目標とする．この目標を達成するために，大量の DB

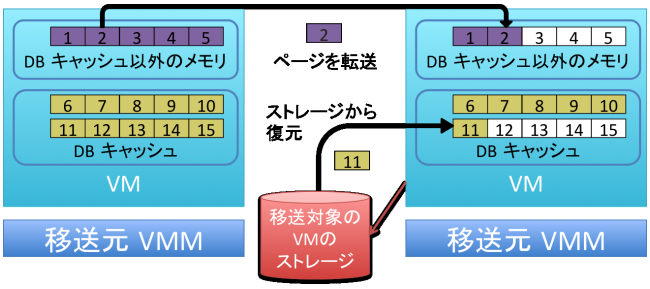

図 3 提案手法の概略図
Fig. 3 Outline of our proposal

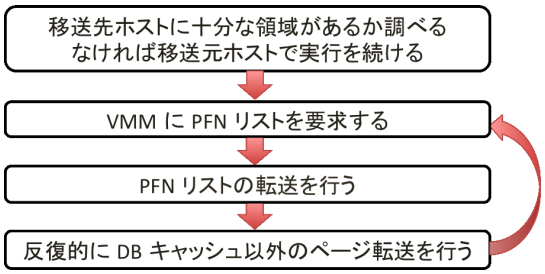

図 4 移送元ホストの移送アルゴリズム
Fig. 4 Migration algorithm at source host

キャッシュのページ転送を削減することを解決すべき課題とする．

4. 提案

本研究では，DB キャッシュの情報を利用し，DB キャッシュのページ転送を削減する仮想マシン移送を提案する．提案手法によりボトルネックとなるページ転送を削減し，仮想マシン移送の高速化が可能となる．本研究では，DB キャッシュのページ転送を削減するために，移送先ホストで DB キャッシュを共有ストレージから復元するアプローチをとる．

4.1 想定する環境

本研究では，DaaS 環境のような DBMS を稼働させている VM の移送を対象とする．また読み取り専用のワークロードのみを実行する DBMS を想定する．クラウド環境では，ワークロードの負荷を分散をさせるための Read Replica と呼ばれる読み取りのみを実行するデータベースが存在する．実際に Amazon RDS[6] などで利用されている．ライト命令によってデータセットが書き換えられることがないため，一度 DB キャッシュに読み込んだページは書き換えられることがない．なお，書き込みを含むワークロードを実行する DBMS を稼働させている VM の移送は今後の課題とする．

4.2 アプローチ

本研究では，DBMS のキャッシュに含まれるページは共有ストレージに存在することに着目し，DB キャッシュのページ転送を削減するために，移送先ホストで DB キャッシュを共有ストレージから復元する．提案手法の概略図を**図 3** に示す．

移送元ホストでは，DB キャッシュのページ転送を省略する．本来 DB キャッシュの情報は DBMS にしかわからない．本研究では，DBMS と VMM を連携させ，DB キャッシュのページ情報を VMM に伝える．移送プロセスは VMM から DB キャッシュのページ情報を受け取り，そのページの転送を省略する．図 3 では，1 から 5 の DB キャッシュ以外のページを転送し，DB キャッシュである 6 から 15 のページは転送していない．

移送先ホストでは，DB キャッシュを共有ストレージから復元する．VM のストレージに保存されている DBMS のデータの位置は，移送プロセスにはわからない．本研究では，VM の DBMS のデータがあるディレクトリを NFS で共有する．移送プロセスは共有された DBMS のデータから DB キャッシュを復元する．図 3 では，転送されなかった DB キャッシュである 6 から 15 のページを移送対象の VM の共有ストレージから復元している．

このアプローチをとることで，ボトルネックとなるページ転送量の削減ができる．またページ転送と DB キャッシュの復元は並行して行えるため，仮想マシン移送の高速化が可能になる．

5. 設計

本研究で提案する仮想マシン移送は，DB キャッシュのページ転送量を削減するために移送アルゴリズムの変更を行う必要がある．移送アルゴリズムを変更するにあたって，移送元ホストでの DBMS と VMM の連携と DB キャッシュの共有ストレージからの復元が必要になる．

5.1 移送アルゴリズムの変更

本研究では，DB キャッシュのページ転送を省略するために Push フェーズのアルゴリズムに変更を加える．Pre-copy 方式の Push フェーズでは，最初のページ転送ではメモリの全ページを転送し，以降のページ転送では，ページ転送中に変更があったページを繰り返し転送していた．

5.2 移送元ホストの移送アルゴリズムの変更

移送元ホストは，batch 分のページを転送する前にどのページを転送しないかを移送先ホストへ伝えてから，DB キャッシュのページ転送を省略する．**図 4** に本研究の移送元ホストの移送アルゴリズムを示す．まずどのページが転送を省略されたかが移送先ホストにもわかるように移送先ホストへ転送しない DB キャッシュのページを伝える．DB キャッシュの情報と PFN のリストを PFN リストと呼ぶ．移送プロセスは PFN リストを管理していないため，

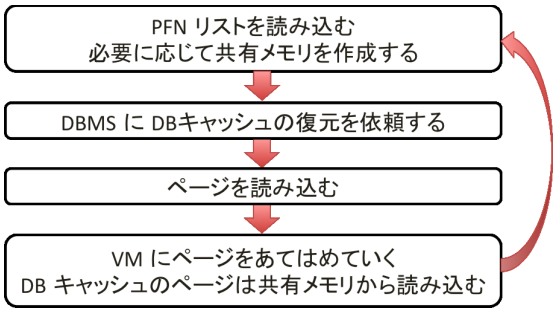

図 5 移送先ホストの移送アルゴリズム
Fig. 5 Migration algorithm at destination host

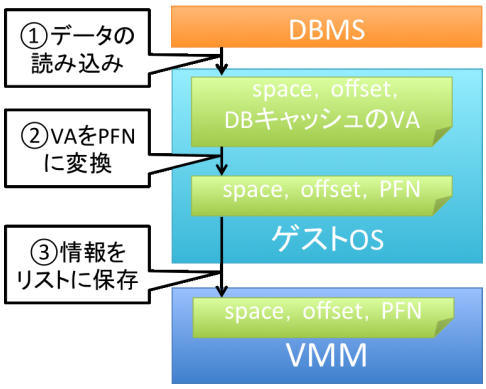

図 6 DBMS と VMM の連携
Fig. 6 Cooperation between DBMS and VMM

VMM にハイパーコールで PFN リストを要求する．そしてその PFN リストをページ転送の前に移送先ホストへ転送する．PFN リストについては第 6.3.1 項で説明する．

次にページ転送量を削減するために DB キャッシュのページ転送を省略する．DB キャッシュのページ転送をするときは，PFN リストを参照して DB キャッシュのページ転送を省略する．PFN リストのサイズはページサイズの 4KB よりも小さいため，ネットワークを使用した転送量を削減できる．以上の流れを batch 分の転送ごとに繰り返す．

5.2.1 移送先ホストの移送アルゴリズムの変更

移送先ホストでは，batch 分のページを読み取る前に移送元ホストから送られた PFN リストを読み取り，転送が省略された DB キャッシュを共有ストレージから復元する．**図 5** に本研究の移送先ホストの移送アルゴリズムを示す．まず転送された PFN リストを読み取る．その PFN リストの情報から DBMS に DB キャッシュの復元を依頼し，DB キャッシュを共有メモリに保存する．DB キャッシュの復元についての詳細は第 5.4 項で説明する．先に DB キャッシュを復元しておくことで，DB キャッシュのページを VM にあてはめるときの DB キャッシュの復元がボトルネックになることを防ぐ．次に転送が省略されていない DB キャッシュ以外のページを読み込む．最後に VM にページをあてはめていく．このとき DB キャッシュのページについては，共有メモリから読み込んであてはめる．以上の流れを batch 分の読み取りごとに繰り返す．

5.3 移送元ホストでの DBMS と VMM の連携

移送元ホストでは，移送プロセスが DB キャッシュの情報を利用するために DBMS と VMM の連携が必要になる．本研究では，移送元ホストで DBMS と OS，OS と VMM を連携させ，DB キャッシュの情報を VMM に伝える．**図 6** に DB キャッシュの情報と PFN がどのように VMM に渡されるかを示す．まず DBMS はワークロードを実行するときに，DB キャッシュにそのデータが存在するかを確認する．もし存在していなければ，ディスクからデータを読み取る．本研究では，ディスクからデータを読み取るときにシステムコールを発行する．そのシステムコールには通常の読み取りに必要なファイルディスクリプタ番号やオフセットといった情報だけでなく，DBMS で独自に管理しているストレージ上のデータを特定できる情報を渡す．

次に OS はシステムコールで受け取った DB キャッシュの仮想アドレスを物理アドレスに変換する．その物理アドレスから PFN を求め，PFN と DB キャッシュの情報をハイパーコールで VMM に渡す．

最後に VMM は，ハイパーコールで PFN と DB キャッシュの情報を受け取る．受け取ったものを PFN リストに保存する．移送プロセスは移送が開始されたらハイパーコール保存された PFN リストを要求して，DB キャッシュのページ転送を省略できるようになる．

5.4 DB キャッシュの共有ストレージからの復元

本研究では，移送先ホストの移送プロセスで DB キャッシュを復元するために，移送対象の VM で使用されている DBMS と同じ DBMS を稼働させ，DBMS が共有メモリに DB キャッシュを復元する．第 4.2 節で述べたように，移送対象の VM の DBMS のデータがあるディレクトリは NFS で共有する．その DBMS のデータを読み取るために移送先ホストにおいても DBMS を起動する．DBMS は読み込んだ DB キャッシュを共有メモリに書き込むことで，移送プロセスは復元された DB キャッシュを利用することができる．

図 7 に DB キャッシュの共有ストレージからの復元の流れを示す．まず移送プロセスは，DB キャッシュを保存するための共有メモリを作成しておく．次に移送プロセスは PFN リストに含まれている DBMS の情報と共有メモリ番号を DBMS に送り，DB キャッシュの復元を依頼する．DBMS は，送られてきた情報から DB キャッシュを復元し，共有メモリに DB キャッシュのデータを書き込む．このように DB キャッシュを共有メモリに保存すれば，移送プロセスは共有メモリから DB キャッシュを読み取ることが可能となる．

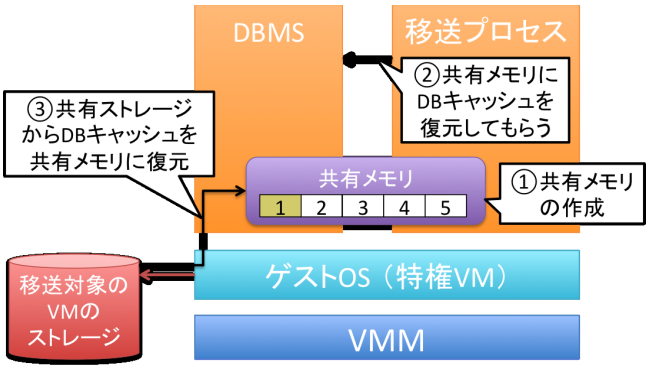

図 7 DB キャッシュの共有ストレージからの復元
Fig. 7 Restoring DB caches from shared storage

```
ssize_t sys_my_pread(unsigned fd, void __user *buf,
 unsigned int count,loff_t pos,
 unsigned long space_id, unsigned long block_offset);
```

図 8 DomU の Linux に定義したシステムコール
Fig. 8 Systemcall in Linux of DomU

```
struct mm_struct *mm = current->mm;
address = (unsigned long)buf;
pgd = pgd_offset(mm, address);
pud = pud_offset(pgd, address);
pmd = pmd_offset(pud, address);
pte = pte_offset_kernel(pmd, address);
pfn = pte_pfn(*pte);
```

図 9 PFN を求める流れ
Fig. 9 Flow of getting PFN

```
struct pfn_struct{
 unsigned long space_id;
 unsigned long block_offset;
 unsigned long pfn;
 unsigned int number;
 struct hlist_node list;
};
```

図 10 Xen に実装した pfn_struct 構造体
Fig. 10 Definition of pfn_struct

6. 実装

本章では，提案手法の実装について述べる．仮想化環境に Xen 4.4.2，Dom0 と DomU に Linux 3.18.20，DBMS に MySQL 5.6.26 を利用した．

6.1 移送元ホストの DBMS と Xen の連携

移送元ホストでは，MySQL でデータを読み取るときに DB キャッシュの情報である space_id と block_offset と PFN を Xen に渡す．これにより，移送プロセスはどのページの移送を省略すればよいかがわかる．

6.1.1 DomU の MySQL 側の実装

6.1.1.1 MySQL が管理している DB キャッシュ情報を Linux カーネルに提供

MySQL はデータを読み取るときに DB キャッシュを特定する情報である space_id と block_offset を Linux カーネルに渡す．space_id と block_offset は，ファイルの読み書きを行う関数の fil_io() において，ファイルディスクリプタ番号とデータのオフセットに変換される．fil_io() は，データを読み取るとき，最終的にはシステムコールの pread() を呼んでいる．pread() に渡す引数は，ファイルディスクリプタ番号，データを格納するバッファへのポインタ，読み取るサイズ，データのオフセットである．

そこで Linux カーネルへ space_id と block_offset を渡すため，Linux カーネルに新たにシステムコールを定義し，MySQL の pread() 文の部分をそのシステムコールに書き換える．またそのシステムコールに space_id と block_offset を渡すため，fil_io() からそのシステムコールを呼ぶまでの関数の引数に spac_id と block_offset を追加する．

6.1.2 DomU の Linux 側の実装

6.1.2.1 システムコールの定義

MySQL から space_id と block_offset を受け取るために，新たに**図 8** のシステムコールを定義した．

sys_my_pread() は，通常の pread() と同じ動作をした後，次のような動作を行う．

6.1.2.2 DB キャッシュから PFN を求める

DB キャッシュの仮想アドレスから PFN に変換する．**図 9** に PFN を求める流れを示す．Linux のページング機構は，ページグローバルディレクトリ（pgd），ページアッパディレクトリ（pud），ページミドルディレクトリ（pmd），ページテーブルエントリ（pte）の 4 段構えになっている．それぞれ仮想アドレスと，pgd は mm_struct 構造体，pud は pgd，pmd は pud，pte は pmd があれば求められる．PFN は pte_pfn を用いれば pte から変換して求められる．DB キャッシュのサイズは 16KB であり，4 つのページからなる．そのため図 9 で示した流れを，仮想アドレスを 4KB ずらして 4 回行い，4 つの PFN を求める．

6.1.2.3 DB キャッシュ情報と PFN を Xen に提供

Linux カーネルから PFN，space_id，block_offset，先に求めた PFN が DB キャッシュの何番目のページにあたるかを示す数を Xen に渡す．1 つの DB キャッシュから 4 つの PFN が求められるため，Xen には DB キャッシュ情報と PFN だけでなく，PFN が DB キャッシュの何番目のページにあたるかを示す数を渡す必要がある．そのために Xen に新たにハイパーコールを定義し，sys_my_pread() はそのハイパーコールを呼ぶ．

6.1.3 Xen 側の実装

6.1.3.1 pfn_struct 構造体の定義

図 10 に DB キャッシュ情報と PFN を格納する pfn_struct 構造体を示す．これを用いて Xen は DB キャッシュ情報と PFN を関連付ける．

```
int do_reg_pfn(unsigned long pfn, unsigned long space_id,
 unsigned long block_offset, unsigned int number);
```

図 11 DB キャッシュ情報と PFN を登録するハイパーコール
Fig. 11 Hypercall registering DB cache infomation and PFN

```
int do_copy_skip_pfn(void* skip_pfn);
```

図 12 移送プロセスに pfn_struct 構造体を渡すハイパーコール
Fig. 12 Hypercall sending migration process pfn_struct

```
int load_cache(UDF_INIT *initid, UDF_ARGS *args,
 char *is_null, char *error);
```

図 13 ユーザ定義関数
Fig. 13 User defined function

6.1.3.2 DB キャッシュ情報と PFN を登録するハイパーコールの定義

Linux カーネルから PFN，space_id，block_offset，number を受け取るために，**図 11** のハイパーコールを定義する．このハイパーコールは DB キャッシュ情報と PFN を関連付けて，ハッシュリストに保存する．図 10 の pfn_struct 構造体を xmalloc() で確保し，受け取った DB キャッシュ情報と PFN をその構造体に代入し，ハッシュリストに保存する．

6.1.3.3 移送プロセスに pfn_struct 構造体を渡すハイパーコールの定義

移送プロセスに pfn_struct 構造体を渡すために**図 12** のハイパーコールを定義する．これにより，移送プロセスが Xen のハッシュリストに保存された pfn_struct 構造体を利用でき，どのページの転送を省略すればよいかがわかる．このハイパーコールでは，ハッシュリストに登録された pfn_struct 構造体のエントリがあれば，ユーザ空間にコピーして，エントリを削除している．

6.2 DB キャッシュの共有ストレージからの復元

移送先のホストでは，DB キャッシュを移送対象の VM の共有ストレージから復元する．

6.2.1 Dom0 の MySQL 側の実装

6.2.1.1 DB キャッシュを読み取るユーザ定義関数の定義

DB キャッシュを読み取るユーザ定義関数を定義することで，MySQL にユーザ定義関数のクエリを送信し，DB キャッシュを読み取る関数を実行することが可能になる．**図 13** にそのユーザ定義関数を示す．ユーザ定義関数では，引数から DB キャッシュ情報と共有メモリの id（shmid），ハッシュ値を読み取り，my_buf_read_page_async() を呼ぶ．

6.2.1.2 shmat によるキャッシュの共有

my_buf_read_page_async() は，内部で my_buf_read_page_low() を呼び，DB キャッシュを読み取っている．**図 14** に共有メモリである，shared_db_cache 構造体を示す．my_buf_read_page_low() で DB キャッシュを読み取った後，**図 15** に示すように，DB キャッシュを共有メモリに書き込む．まず shm に共有メモリのポインタを当て，それをハッシュ値分ずらす．その後そのポインタの位置に DB キャッシュをコピーし，共有メモリをデタッチする．これにより，移送プロセスから同じ id の共有メモリを参照することで，DB キャッシュを利用できる．

```
struct shared_db_cache {
unsigned long space_id, block_offset;
char is;
char db_cache[DB_PAGE_SIZE];
};
```

図 14 shared_db_cache 構造体
Fig. 14 Definition of shared_db_cache

```
shm = (struct shared_db_cache *)shmat(shmid, 0, 0);
p = shm+hvalue;
memcpy(p->db_cache, ((buf_block_t*) bpage)->frame,
 UNIV_PAGE_SIZE);
shmdt(shm);
```

図 15 shmat で DB キャッシュを共有するコード
Fig. 15 Code sharing DB caches using shmat

6.3 移送アルゴリズムの変更

6.3.1 移送元ホスト側の変更

6.3.1.1 pfn_struct 構造体を移送先ホストへ転送

移送元ホストでは，ページ転送を始める前に pfn_struct 構造体を Xen から読み取り，移送先ホストに転送する．まず do_copy_skip_pfn() ハイパーコールを実行する．Xen に保存されている pfn_struct 構造体が移送プロセスにコピーされるので，それをハッシュリストに追加する．次に転送を省略するページ数の pfn_count を移送先ホストに転送する．最後に全ての pfn_struct 構造体を全て移送先ホストに転送する．

6.3.1.2 DB キャッシュのページ転送を省略

pfn_struct 構造体のもつ PFN のページ転送のときは，ページの転送を省略する．通常のページ転送では，VM のページをコピーし，バッファに書き込む．提案手法では，転送の前に PFN をチェックし，ハッシュリストに保存されている pfn_struct 構造体の PFN と一致した場合には，ページ転送を省略する．

6.3.2 移送先ホスト側の変更

6.3.2.1 pfn_struct 構造体をハッシュリストに保存

移送先ホストでは，まず転送された pfn_struct 構造体をハッシュリストに保存する．まず移送元ホストから転送された pfn_count，及び pfn_struct 構造体を，バッファから読み取る．その後第 sec:source 項で説明したように，pfn_struct 構造体をハッシュリストに保存する．

6.3.2.2 MySQL に DB キャッシュの読み込みを依頼

次に MySQL に対して共有メモリに DB キャッシュを読み込んでもらう必要がある．DB キャッシュは 16KB でページサイズは 4KB であるため，1 つの DB キャッシュは 4 つのページからなる．4 つのページのうち，1 回だけ読み取ればよいので，読み取った pfn_struct 構造体の number が 0 のときのみ，MySQL に DB キャッシュの読み込みを依頼する．共有メモリのポインタをハッシュ値分ずらした後，共有メモリに DB キャッシュ情報を書き込む．その後，図 13 で定義したユーザ定義関数のクエリを，mysql_query() で MySQL に送る．MySQL では第 6.2.1 項で述べた動作をし，共有メモリに DB キャッシュを読み込んでいる．

6.3.2.3 共有メモリから DB キャッシュのページをコピー

通常は，バッファから読み取ったページを VM にコピーしているが，提案手法は，DB キャッシュのページは共有メモリからコピーする．先ほど共有メモリに DB キャッシュを読み込んだように，共有メモリのポインタをハッシュ値分ずらし，その space_id と block_offset が一致していたら，共有メモリのポインタからページサイズである 4KB × number 分ずらしたポインタから，ページをコピーする．

6.4 VM Live Migrarion の実行フロー

最後に提案手法の実行フローの詳細を述べる．第 5.1 説では，移送アルゴリズムの変更の概要を述べた．**図 16** に実行フロー図を示す．移送元ホストでは，まず Xen に pfn_struct 構造体の要求をし，ハッシュリストに保存する．その後ページ数である pfn_count と pfn_struct 構造体の転送を先に転送する．PFN とページの属性の転送は従来通りに行うが，ページを転送するときは DB キャッシュのページ転送を省略する．

移送先ホストでは，まず pfn_count と pfn_struct 構造体を読み取り，ハッシュリストに保存する．次に得られた pfn_struct 構造体から MySQL に DB キャッシュの復元を依頼する．PFN とページの属性の読み込みとページの読み込みは従来通りに行う．読み込んだページを DomU にコピーするときは，DB キャッシュのページの場合は共有メモリから読み込む．

7. 実験

7.1 VM Live Migration の性能評価

本研究では，読み込み専用のワークロードを実行している DBMS を稼働させている VM の Live Migration を高速化する手法を提案し，実装している．提案手法と既存手法を比較し，Live Migration が高速化できているかを確認するため，評価実験を行った．

7.1.1 実験環境

仮想化環境に Xen 4.4.2，DBMS に MySQL 5.6.26 を利用する．3 台の同じスペックの物理マシンを使用し，1 台を移送元のホスト，1 台を移送先のホスト，1 台を VM の共有ストレージ及びワークロードを送るクライアントとして使用する．移送元ホストと移送先ホストはギガビットイーサネットで接続されており，ワークロードを送るクライアントのパケットと競合が起きないようにしている．使用した物理マシンは，CPU が 8 コアの Intel Xeon E3-1286 v3，メモリが 32GB，ストレージが 2GB の HDD である．VM は，VCPU4 コアで，仮想メモリと MySQL の DB キャッシュのサイズは必要に応じて変更する．ベンチマークには Sysbench を使用し，読み取り専用のワークロードを実行させる．

表 1 比較する実験設定

設定の略称	設定の内容
デフォルト	提案手法を実装しない標準の設定
Dom0 の DB キャッシュなし	提案手法の移送開始時に DB キャッシュを読み込み始める設定
Dom0 の DB キャッシュあり	提案手法の移送開始前に DB キャッシュを読み込んでおく設定

7.1.2 実験方法

Sysbench を実行している MySQL を稼働させている VM の移送を行った．Sysbench は読み取り専用のワークロードで，データは全て DB キャッシュに収まるものとする．VM のサイズは 8GB，DB キャッシュのサイズは 6GB とし，データサイズを 512MB，1GB，2GB，4GB と変化させて計測した．また，提案手法では移送が開始したら DB キャッシュの読み込みを開始しているため，Dom0 の DB キャッシュはない状態で移送が始まる．

表 1 に比較する実験設定を示す．本実験では，提案手法の移送開始時に Dom0 の DB キャッシュがある場合とない場合，従来手法を比較し，総移送時間とダウンタイムを計測した．

7.1.3 実験結果

7.1.4 ダウンタイム

図 17 にダウンタイムを示す．提案手法と従来手法は，どれも数百ミリ秒のダウンタイムで差はない．これは Sysbench のデータセットが全て DB キャッシュに収まるため，ページが変化することがないからであると考えられる．ワークロードによって DB キャッシュがスワップアウトされる場合のダウンタイムの計測は今後の課題とする．

7.1.5 総移送時間

図 18 に総移送時間を示す．提案手法の Dom0 の DB キャッシュありの場合，データサイズが大きくなるに連れて，デフォルトと比べて大幅に総移送時間が長くなってしまった．これは MySQL に DB キャッシュを読み込むときに，ディスクアクセスが発生してしまっているからだと考えられる．

一方提案手法の Dom0 の DB キャッシュありの場合，データサイズが小さいときはデフォルトと差がないが，

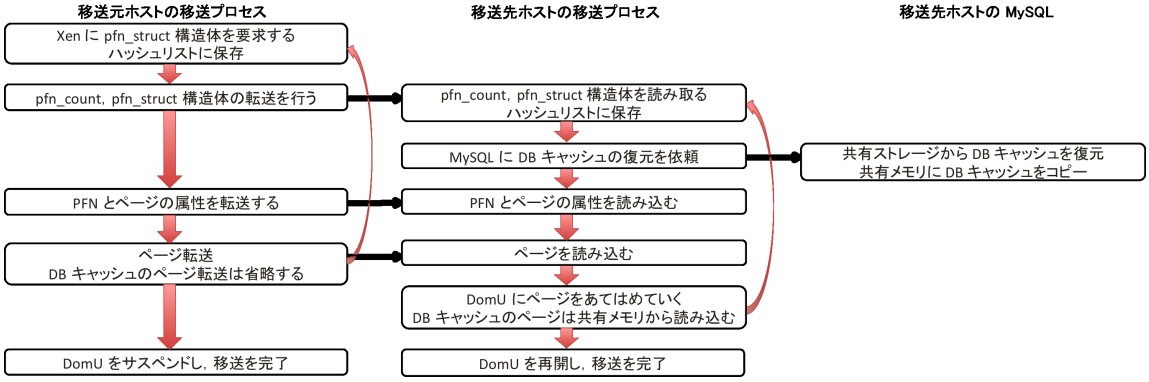

図 16 VM Live Migrarion の実行フロー
Fig. 16 VM Live Migration flow

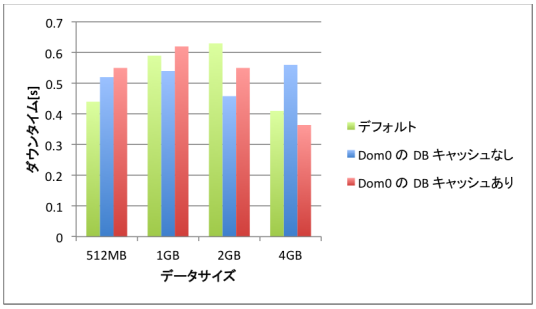

図 17 ダウンタイム
Fig. 17 Dowintime

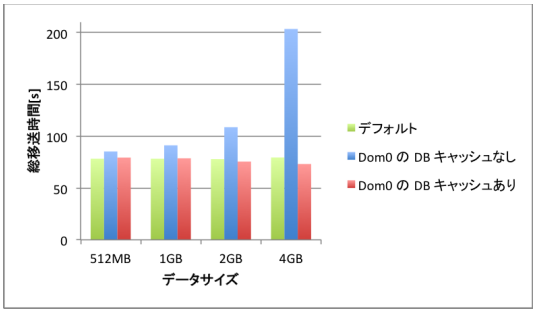

図 18 総移送時間
Fig. 18 Migration Time

データサイズが大きくなるとデフォルトよりも総移送時間が短くなった．ディスクアクセスをなくし，データ転送量を減らすことができたため，総移送時間を短くすることができたと考えられる．しかし，データサイズが小さいと共有メモリの作成などのオーバヘッドによりデフォルトよりも総移送時間が長くなってしまった．

この実験により，提案手法の Dom0 の DB キャッシュありの場合は，仮想マシン移送の総移送時間を短くすることができると考えられる．

Dom0 の DB キャッシュなしを高速化する方法として，DomU のストレージを SSD に換装することが挙げられる．この手法では，ディスクアクセスがボトルネックになっている．SSD のディスクアクセスは HDD に比べて高速であるため，MySQL が DB キャッシュを読み込む速度が上がれば，性能は改善されると考えられる．また，提案手法の MySQL の DB キャッシュの読み込みは，単一スレッドで行っている．これを複数スレッドで読み込みをさせ，並行化することによっても高速化できると考えられる．これらの最適化は今後の課題とする．

7.1.6 追加実験

Dom0 の DB キャッシュありの性能評価をより詳しく検証するために追加実験を行った．Dom0 の MySQL の DB キャッシュサイズを変化させ，移送開始前に DB キャッシュをできるだけ読み込んでおく．移送開始時に DB キャッシュがあることによる総移送時間への影響を計測する．VM のメモリサイズを 4GB，8GB，12GB とし，DB キャッシュサイズを 3.2GB，6.4GB，9.6GB と変化させる．デフォルトと Dom0 の DB キャッシュがないとき，及び Dom0 の DB キャッシュサイズが 1GB，3.2GB，6.4GB，9.6GB で，データが読み込まれているときを比較する．

VM のメモリサイズが 4GB，DB キャッシュサイズが 3.2GB のときの実験結果を**図 19** に示す．Dom0 の DB キャッシュサイズが VM の DB キャッシュサイズに近づくほど，総移送時間が短くなっている．しかし，Dom0 の DB キャッシュサイズが 3.2GB になっても，総移送時間は 40 秒であり，デフォルトと差がなかった．この実験の結果により，移送対象の VM の DB キャッシュサイズがそれほど大きくないと，提案手法の恩恵が受けられないと言える．移送アルゴリズムを改善することで，提案手法の総移送時間を短くすることができると考えられる．

VM のメモリサイズが 8GB，DB キャッシュサイズが 6.4GB のときの実験結果を**図 20** に示す．Dom0 の DB キャッシュサイズが VM の DB キャッシュサイズに近づくほど，総移送時間が短くなっている．Dom0 の DB キャッシュサイズが VM の DB キャッシュサイズと同じの 6.4GB になったとき，提案手法の総移送時間は 63 秒であった．デフォルトの総移送時間は 80 秒であるため，約 20%総移送時間を短くすることができた．この実験の結果により，提

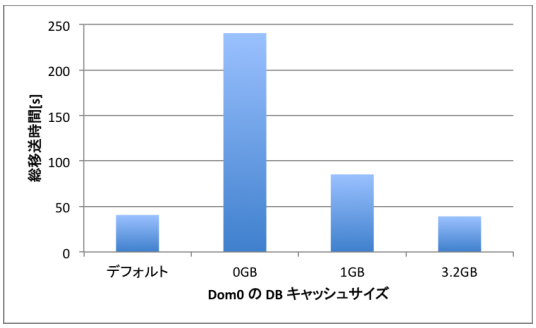

図 19 VM のメモリサイズが 4GB，DB キャッシュサイズが 3.2GB のときの総移送時間
Fig. 19 Migration time with 4GB VM and 3.2GB MySQL Buffer

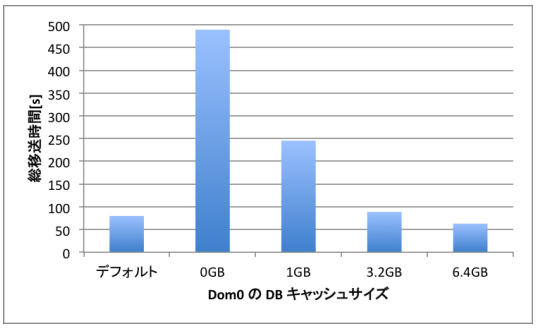

図 20 VM のメモリサイズが 8GB，DB キャッシュサイズが 6.4GB のときの総移送時間
Fig. 20 Migration time with 8GB VM and 6.4GB MySQL Buffer

案手法は VM の DB キャッシュサイズが十分に大きければ有効であると言える．

VM のメモリサイズが 12GB，DB キャッシュサイズが 9.6GB のときの実験結果を**図 21** に示す．Dom0 の DB キャッシュサイズが大きくなると総移送時間が一旦短くなっていくが，さらに大きくなると総移送時間が長くなってしまった．これは，Dom0 の DB キャッシュと共有メモリがメモリサイズを超え，メモリのスワップアウトが頻繁に起きてしまったからだと考えられる．この実験により，VM の DB キャッシュサイズが大きすぎると，Dom0 のメモリが不足し，提案手法の利用が困難であると言える．

以上の実験結果から，提案手法の Dom0 の DB キャッシュありの場合は，Dom0 のメモリサイズが VM の DB キャッシュサイズよりも十分に大きく，また VM の DB キャッシュサイズが大きい場合に有効であると考えられる．

8. おわりに

8.1 まとめ

本研究では，メモリサイズの大きい VM の総移送時間が長期化してしまうという課題に対し，DBMS を稼働している VM の移送を高速化する手法を提案した．提案手

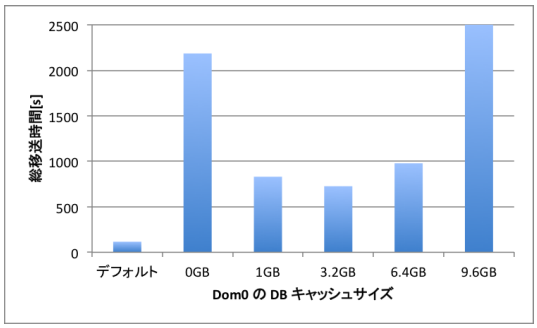

図 21 VM のメモリサイズが 12GB，DB キャッシュサイズが 9.6GB のときの総移送時間
Fig. 21 Migration time with 12GB VM and 9.6GB MySQL Buffer

法は，ページ転送量を削減するために，移送元ホストで DB キャッシュのページ転送を省略し，移送先ホストで DB キャッシュのページを共有ストレージから復元している．提案手法を実装したところ，移送先ホストであらかじめ DB キャッシュが復元されている場合に，最大で約 20%総移送時間の削減を達成できた．

8.2 今後の課題

8.2.1 DB キャッシュの復元高速化

提案手法では，第 7.1.3 項で述べたように，Dom0 の MySQL の DB キャッシュが読み込まれていない状態で移送を開始する場合，総移送時間がデフォルトの移送方式よりも長くなってしまう．なぜならディスクアクセス遅いことによって DB キャッシュの復元がボトルネックとなってしまっているからである．DomU のストレージを SSD に換装することや，複数スレッドによる DB キャッシュの復元の実装を行うことで，DB キャッシュの復元を高速化できると考えられる．これにより，Dom0 の MySQL の DB キャッシュが読み込まれていない状態で移送を開始する場合でも，総移送時間が短くすることができると考えられる．

8.2.2 DB キャッシュのスワップアウトへの対応

Dom0 の DB キャッシュありの提案手法では，VM のメモリサイズ及び稼働している MySQL の DB キャッシュサイズが大きすぎると，Dom0 とのメモリの競合が起きてしまう．これによってメモリのスワップアウトが頻繁に起きてしまい，仮想マシン移送の総移送時間が長くなってしまった．この課題を克服するには，DB キャッシュの復元高速化が有効だと考えられる．なぜなら Dom0 で必要となる MySQL の DB キャッシュサイズを減らすことができるからである．提案手法では，Dom0 の DB キャッシュありの場合にのみ，仮想マシン移送の総移送時間が短くなっている．しかし，この手法では MySQL の DB キャッシュと共有メモリの両方を確保しなくてはいけない．Dom0 の DB キャッシュなしの場合でも仮想マシン移送が高速化できればこの問題は解決できると考えられる．

8.2.3 DB キャッシュの変化への対応

本研究では，読み取り専用のワークロードを実行する DBMS を稼働している VM の移送を対象としていた．しかし，あらゆる DBMS ではライトトランザクションが実行され，DB キャッシュが書き換わってしまう．書き換わった DB キャッシュは，フラッシュリストに追加され，遅延的に書き込まれる．そのため，提案手法の共有ストレージから DB キャッシュを復元するアプローチを用いることができない．この課題を克服するために書き換わった DB キャッシュのページ転送の省略をキャンセルする方法が挙げられる．VM の DBMS のパフォーマンスの低下を抑えられる可能性があるが，ページ転送量が多くなってしまい，提案手法の効果が減る可能性がある．

参考文献

[1] Amazon EC2. https://aws.amazon.com/jp/ec2/.
[2] Google Cloud Platform. https://cloud.google.com.
[3] Xen Project and A Linux Foundation Collaborative Project. http://www.xenproject.org/.
[4] KVM (for kernel-based virtual machine). http://www.linux-kvm.org/.
[5] Oracle VM VirtualBox. https://www.virtualbox.org/.
[6] Amazon RDS. http://aws.amazon.com/jp/rds/.
[7] Microsoft sql azure. https://azure.microsoft.com/ja-jp/services/sql-database/.
[8] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. *Live migration of virtual machines.* In Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI '05), pp.273286, 2005.
[9] VMware, Inc. http://www.vmware.com/.
[10] Kai-Yuan Hou, Kang G Shin, and Jan-Lung Sung. *Application-assisted live migration of virtual machines with java applications.* In Proceedings of the 10th European Conference on Computer Systems (EuroSys '15), pp.15:115:15, 2015.
[11] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. *Efficient live migration of virtual machines using shared storage.* In Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '13), pp.4150, 2013.
[12] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, and Amr El Abbadi. *Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration.* Proceedings of the VLDB Endowment, Vol.4, No.8, pp.494505, 2011.
[13] Aaron J Elmore, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. *Zephyr: live migration in shared nothing databases for elastic cloud platforms.* In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11), pp.301312, 2011.
[14] Takeshi Mishima and Yasuhiro Fujiwara. *Madeus: Database live migration middleware under heavy workloads for cloud environment.* In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15), pp.315329, 2015.
[15] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. *Remus: High availability via asynchronous virtual machine replication.* In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08), pp.161174, 2008.
[16] Umar Farooq Minhas, Shriram Rajagopalan, Brendan Cully, Ashraf Aboulnaga, Kenneth Salem, and Andrew Warfield. *Remusdb: Transparent high availability for database systems.* The VLDB Journal, Vol.22, No.1, pp.2945, 2013.