Vol.2017-OS-141 No.8
2017/7/26

キャンセラブルなポストコピー VM 移送

小川 遥加[†1,a)]　山田 浩史[†1,b)]　十文字 優斗[†1,c)]　阿部 芳久[†2,d)]

概要：
ライブ仮想マシン (VM) 移送は, VM を稼働させたまま別の物理マシンへ移送する技術である. ライブ VM 移送を用いることにより, VM の再配置による物理マシンのメンテナンスや負荷分散が容易化する. ライブ VM 移送の実現手法として, ポストコピー方式がある. ポストコピー方式は, 負荷分散効果が早期に得られるなど強力なライブ VM 移送の実現手法であるが, 移送の中断が発生すると VM が破壊されてしまうという欠点がある. 本研究では, キャンセラブルなポストコピー VM 移送方式を提案する. 提案手法では, ポストコピー方式での移送途中に移送先 VM と移送元 VM のメモリや VCPU の状態を同期しておくことで, 移送を中断しても移送元で VM を稼働を可能にする. 提案手法の性能を見積もるため, ポストコピー方式による移送手法と VM 同期手法の性能について, 移送 VM に対するパフォーマンスへの影響と移送メカニズムにかかる物理ホストに対するオーバヘッドを計測した. 提案手法は, ポストコピー方式の移送に対して移送中の VM の性能が 10%程度低下が発生し, 実行が移送先 VM へ移った後の移送元ホストに対して CPU 使用率 3%以下の負荷がかかる.

1. はじめに

仮想化技術とは, 仮想マシン (VM) によって 1 台の物理マシン上で複数の異なる OS を稼働させる技術である. VM は仮想マシンモニタ (VMM) によって管理される. 仮想化技術の発達により, VM を 1 台のマシンに集約することで, 物理マシンを減らすことができ, 物理資源使用率の向上や, 消費電力の削減につながる. そのため, Amazon EC2[1] や, Google Cloud Platform[2] といったサービスが仮想化環境下で運用されている.

ライブ VM 移送は, ある物理マシン上の VM を稼働したままの状態で, 異なるホストへ移送する技術である. クラウド環境やデータセンタなどの仮想化されたプラットフォームにおいては, ライブ VM 移送を用いてサービスを稼働させたまま VM の再配置をすることにより, 高負荷の物理マシンから別の物理マシンへ VM を移送することによる負荷分散 [10][11][12] や, VM を集約することによる省電力化 [13][14][15], VM を退避させた物理マシンのメンテナンスが容易になる. 実際, Google のデータセンタでは, 実際にライブ VM 移送を用いた運用がされている.

ライブ VM 移送の効率的な実現方式として, プレコピー方式 [16] とポストコピー方式 [17] が提案されている. 現在のライブ VM 移送手法としてプレコピーは主流であり, 多くの VMM によってサポートされている [3][4][5]. プレコピー方式では, 移送元 VM を稼働させたまま, 移送先 VM にメモリ内容をすべて転送する. その後, 転送の間に更新のあったメモリページをすべて転送する. この工程を繰り返し, 更新のあったページが閾値より少なくなったら VM を停止し, 残りのメモリページと仮想 CPU の状態を転送し, 移送先 VM を再開する. しかし, プレコピー方式は VM 上でメモリの更新が多いワークロードが稼働している場合, メモリページの再送量が増加し, 移送時間が長期化してしまうという問題が存在する. ポストコピー方式では, 最初に仮想 CPU 状態を転送し, 移送先 VM で稼働させる. 未転送メモリページを移送元から逐次転送をするとともに, 転送先で未転送メモリへのアクセスがあった場合, ページフォルトを VMM が検知し, 移送元 VM から該当ページを優先的にフェッチする. プレコピー方式と比較すると, ポストコピー方式は重複送信が起きないため VM 移送時間の長期化を防ぐことができる. また, 最初に VM の稼働を移送先 VM に移行するため, 移送元ホスト上の負荷分散効果を即座に得ることができる.

ポストコピー方式は強力なライブ VM 移送方式であるが, VM 移送を中断した場合, VM が破壊されてしまうと

†1 現在，東京農工大学
Presently with Tokyo University of Agriculture and Technology
†2 現在，Nokia Bell Labs
a) ogaharu@asg.cs.tuat.ac.jp
b) hiroshiy@cc.tuat.ac.jp
c) jumonji@asg.cs.tuat.ac.jp
d) yoshihisaabe@gmail.com

いう問題点がある．VM 移送を中断する状況として, ネットワーク障害発生時や, ホスト上の負荷の変動などにより管理者が VM 移送を任意で中断する場合などが考えられる．プレコピー方式では, 移送元にすべての必要な情報が保持されているため, 中断が起きても移送元 VM で稼働し続けることができる．一方ポストコピー方式では, メモリ状態を転送しないまま移送先 VM で稼働してしまうので, 必要な情報が移送元と移送先に分断されてしまう．移送先では未転送のオリジナルのメモリ状態を保持せず, 移送元では仮想 CPU 状態や更新が起きたメモリの状態を保持しないため, 移送元, 移送先ともに再開できなくなり, VM が破壊されてしまう．

本研究では, キャンセラブルなポストコピー VM 移送方式を提案する．提案手法では, ポストコピー方式での移送途中に移送を中断しても, 移送元 VM で稼働を可能にする．これにより, ポストコピー方式が有する迅速な負荷分散効果や移送時間の短縮を達成し, かつプレコピー方式と同様に移送を中断しても VM 稼働可能にできる．ポストコピー方式の移送途中に中断しても VM を稼働可能にするため, 移送先 VM で再開後, 移送元に対してスナップショットを作成し, 中断時移送元の持つ最新のスナップショットから再開する．

本論文の貢献は次の通りである．

- キャンセラブルなポストコピー VM 移送方式を提案した．ポストコピー方式を中断可能にすることで, ネットワーク障害や管理者の任意の中断発生時に VM を続行可能にする．
- 提案手法を実現するために必要なメカニズムを明らかにし, 設計方針を提示した．Remus[18] や MicroCheckpointing[6] などの複製 VM を作成する手法を応用し, ポストコピー方式による移送中の移送元 VM と移送先 VM の同期を行う．中断が発生した場合, 同期された移送元 VM から再開する．この際, I/O view の一貫性を保持するために, 出力をバッファリングし, 同期後にデバイスへと解放する．これにより, 移送元 VM での再開時に厳密には実行が多少戻るもののI/O view の整合性が取れ，外部からは VM が連続稼動しているように見える．
- 予備実験を行い, 提案方式の有効性の見積もりを行った．Qemu 2.6.1, Linux 4.8.6 上で, ポストコピー方式のライブ VM 移送と MicroCheckpointing 手法に対する性能評価を行うことで, 提案手法の性能の見積もりを行った．従来のポストコピーに比べて移送中の VM の性能は 10%程度低下し, 移送メカニズムにかかる物理ホストに対するオーバヘッドは移送元ホストの CPU 使用率に対して 3%以下であると推測した．

本論文では, 第 2 章では, 本研究で取り組む問題とその解決手法について述べる．第 3 章では提案手法の設計を述べ, 第 4 章では提案手法の詳細な実装について述べる．第 5 章では, 予備実験を行い, 結果より提案手法の有効性について見積もりを行う．第 6 章では関連研究について紹介し, 現在の移送方式の問題点について述べる．最後に第 7 章で本研究のまとめと今後の課題について述べる．

2. 提案

本研究では, キャンセラブルなポストコピー VM 移送を提案する．

2.1 想定するキャンセル状況

ポストコピー方式による移送中のキャンセルが起きる状況として, ネットワーク障害発生時と管理者の任意のキャンセルを対象とする．移送元 VM から移送先 VM への転送にはネットワークを介して行われるため, ネットワーク障害が発生した場合, 移送を続けることができなくなってしまう．このような場合, 移送の中断処理を行うことで VM を続行する必要がある．また, 管理者の任意のキャンセルが起きる可能性がある．サーバシャットダウンなど管理中のキャンセルや, 多数のサーバを管理しているデータセンタなどでは, 負荷の変動などにより移送中に別のサーバに移送先を変更したい状況が考えられる．本研究では, これらのキャンセル状況において中断可能にすることで, ポストコピー方式の信頼性と利便性を向上させる．

2.2 アプローチ

提案手法では, ポストコピー方式による移送中に中断があった場合, 移送元 VM で続行するために, 移送中の移送元 VM と移送先 VM の状態を同期するアプローチを取る．ポストコピー方式による移送の中断時発生時に, 移送先 VM と同期済みの移送元 VM で再開することで, VM 稼働を可能にする．これにより, ポストコピー方式が持つ迅速な負荷分散効果や移送時間の短縮を達成しながら, プレコピー方式と同様に移送中断時に VM の稼動を可能にする．

提案手法の概略図を図 1 に示す．ポストコピー方式の移送中断時, 移送元 VM で稼働を続行させるために, ポストコピー方式による移送中に移送元 VM に対して移送先 VM のスナップショットを定期的に作成する．スナップショットの作成には, Remus や MicroCheckpointing といった VM の複製を高速に作成する手法を応用し, 移送元 VM に対して再開に必要なメモリや VCPU 情報を転送する．すべてのメモリ転送が終わり, ポストコピー方式による移送が終了したら, 同期を停止し, 移送元 VM の領域を解放する．

移送の実行を中断された場合, 移送元 VM の持つ最新のスナップショットから稼働を継続する．移送先 VM は停止し, VM の領域を解放する．たとえば, ネットワーク障害を検知した場合, 移送を続行不可能と判断し移送が中断される．中断されると, 移送先 VM は実行を停止し, 移送先 VM

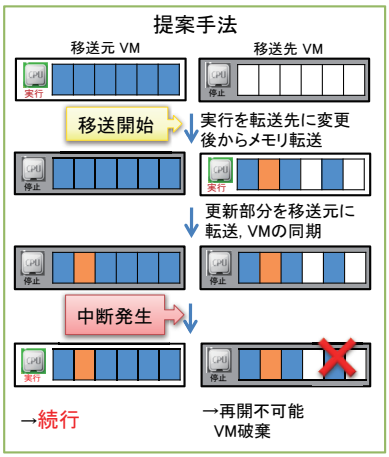


図 1: キャンセラブルなポストコピー VM 移送

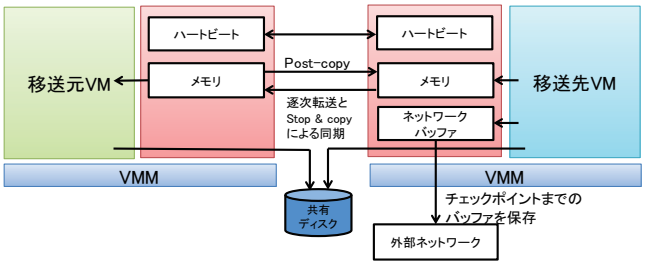


図 2: 提案手法の設計

で保持する最新のスナップショットから VM が再開される. 移送先 VM は破棄する. これにより, オリジナルのページと移送先で更新されたメモリ情報や仮想 CPU 情報を保持する移送元 VM で稼働を再開することができ, ネットワーク障害などによる移送の中断が発生した場合でも VM の実行を続けることができる.

3. 設計

本章では提案手法の実現に向けた設計を示す. 効率よく移送元 VM と移送先 VM を同期するために, Remus や Microcheckpoitning を応用する. 提案手法は, 移送先・移送元 VM の同期, I/O view の保証, dirty bitmap の管理, 移送元 VM での再開の 4 要素で構成される. 提案手法の概要図を図 2 に示す. 移送元 VM から移送先 VM のポストコピーによるメモリ移送とともに, 移送先 VM で更新があったメモリを dirty bitmap として保存しておき, dirty page と VCPU の状態をスナップショットとして移送元 VM へ転送する. この時, ネットワークの出力はすべてバッファリングしておき, チェックポイントが完成し次第バッファを解放する. また, 中断を検知するためにハートビートを行い, お互いの VM の状態を定期的に確認する.

3.1 ポストコピー方式の移送中の VM 同期

効率的にメモリイメージをスナップショットとして転送するために, 移送先ホストで書き換えが起きたメモリのみを転送する. dirty bitmap を用いることで, dirty page を検知する. 移送先ホストで書き換えが起きたメモリと VCPU の情報を定期的に移送元 VM に転送を行う. ポストコピー方式では, 移送が開始されると再開に必要なデータを移送先 VM に転送後, メモリを保持しないまま移送先 VM で実行が再開される. 一方, 移送元 VM は移送前のメモリ状態をすべて保持している. そのため, 移送先 VM で書き換えが起きたページのみを転送することで, 移送元 VM は再開に必要な情報をすべて取得することができ, 移送元 VM と移送先 VM の同期が可能である. 移送先 VM で書き換えられたメモリだけを転送するために, 更新のあったページの検知を行う. 移送先 VM を再開する際にページの更新の検知を開始し, 更新があったページを dirty page として記録する. スナップショットを作成する際に, 以前のスナップショットまでの間に記録された dirty page と VCPU の情報を移送先 VM へ転送する. この手順を定期的に繰り返すことで, 移送元 VM に対して同期を行うことができる.

3.2 I/O view の保証

移送先 VM は稼働しながらスナップショットを転送するため, 移送元 VM の保持するスナップショットは移送先 VM の状態より少し古い状態となる. したがって, 移送元 VM で VM を再開した際, 厳密には動作が少し戻ることになる. この間に発生した出力がネットワークやディスクに解放されてしまうと, 移送元 VM でスナップショットから処理のやり直しが起きた際に, スナップショットとの I/O の整合性が取れなくなってしまう. I/O view の保証をするためには, 移送元 VM へ転送されたスナップショットと I/O の出力の一貫性を保つ必要がある. I/O view を保証するために, ネットワークやディスクに対する出力をバッファリングしておき, スナップショット転送のタイミングで各デバイスに解放をする. 移送中断発生時, 移送元 VM で再開されたスナップショット以降にバッファに保存された出力はすべて破棄する. スナップショットまでの間の I/O は破棄されるため, VM 上のネットワークの処理は, 障害としてクライアント側に認識されることになる. これらは TCP などのプロトコルにより再送機能により改めて行われる. これにより, スナップショットから再開した際の処理は戻るが, I/O の整合性が取れ, 外部からは VM が稼働し続けているように見える.

3.3 Dirty bitmap の管理

提案手法では, ポストコピー方式による移送中に移送先 VM で更新があったページを移送元 VM に転送する. 通常, 更新のあったメモリは dirty bitmap を用いて探索され, 転送が行われる. Dirty bitmap は KVM により管理され, ゲスト領域のページに対して書き換えが起きた際にそのページに対応する dirty bit が立つようになっている. そのた

め, ポストコピー方式によるメモリ転送によるゲスト OS のメモリ領域への書き込み時にも dirty bit が立ってしまう. これにより, ゲスト OS 上で書き換えがあったページだけでなく, 移送元 VM から転送されたページも dirty bit として扱われてしまうため, 移送元 VM に対してスナップショットを転送する際に, 移送元 VM が保持しているページも転送されてしまう問題点がある.

この問題を解決するために, ゲスト OS 上で書き換えがあったページのみを dirty page として扱い, 移送元から転送されたページに対する dirty bit を削除する. 移送元から受け取ったページをメモリへ書き込む際, 該当の領域に対してロックを取り, ページの書き込み後に KVM 側の管理する dirty page から該当の領域の dirty bit を削除してからロックを解放することで, ページへの書き込み操作と領域の dirty bit の削除がアトミックに行われる. これにより, KVM から dirty bitmap 取得した際に, 移送元からのメモリ転送による書き込みには dirty bit が立っておらず, 移送元の保持していないゲスト OS で書き換えのあったページのみを転送することができる.

3.4 移送元 VM での再開

移送元 VM で移送の中断やネットワーク障害を検知するために, 移送先 VM と移送元 VM で定期的にお互いの状態を通知し合うハートビートを行う. ハートビートによって, ネットワークの障害や VM 移送中断の合図を検知を行う. ネットワーク障害時, または管理者から中断のコマンドが入力された場合, 移送の中断が発生する. 移送を中断すると, 移送先 VM は実行を停止し, VM が破棄される. 移送先元 VM では, ポストコピー方式によるメモリ移送を中断し, 移送元 VM の持つ最新のスナップショットから VM の再開を行う. 移送元でバッファリングされた I/O は破棄され, 移送先 VM での再開後, TCP などの再送機能などにより改めて実行される.

4. 実装

本章では提案手法の実装について述べる. 実装は Qemu 2.6.1 , Linux 4.8.6 上に行う. 提案手法では, 移送元 VM と移送先 VM を同期するために, ポストコピー方式による移送中に, 移送元 VM に対して移送先 VM のスナップショットを定期的に作成する. 概略図を図 3 に示す. 図中の Cancellable Post-copy Migration が本提案手法の機構である. 提案手法では, ポストコピー方式の移送を行う Post-copy process と, VM の同期機構である Checkpointing process の 2 つの機構に分かれる.

4.1 ポストコピー方式の変更

提案手法のベースとして Qemu の PostCopyLiveMigration[7] を用いる. Qemu の PostCopyLiveMigration は, プレコピー方式とポストコピー方式をハイブリッドした移送を想定した設計になっている. まずプレコピー方式による移送を開始し, 管理者の任意のタイミングで移送先 VM へ実行が移され, ポストコピー方式による移送に移行する実装がされている. これは, HSG-LM と同様の手法であり, プレコピー方式によってメモリを一通り転送しておくことで, 移送先 VM に実行が移った際に未転送のメモリを減らし, VM の性能の低下を防ぐためである.

PostCopyLiveMigration では, 通常プレコピー方式による移送が行われており, 管理者による任意の migrate_start_postcopy コマンドが入力されることで, 移送の状態を保持する構造体が持つ要素である start_postcopy が true になると, ポストコピー方式の移送に変更される. ポストコピー方式の移送が開始すると, 移送元 VM を停止し, すでに送信済みのページで dirty page となったページを未転送メモリとして扱うために移送先 VM から削除する. その後, 移送先 VM で再開され, 移送元 VM から未転送メモリの転送が開始される仕組みになっている. migrate_start_postcopy コマンド入力によるポストコピー方式への切り替えは, プレコピー方式の移送が 1 ラウンド経過以降を推奨とされている.

提案手法では, ポストコピー方式中の移送先 VM と移送元 VM の同期が目的であるため, プレコピー方式によるメモリ転送を行わず, すぐにポストコピー方式による転送が開始するように変更を加えた. そこで, 提案手法の実行開始時に start_postcopy の値をあらかじめ true にしておくことで, 即座にポストコピー方式の移送を開始するようにした.

4.2 ポストコピー方式の移送中の VM 同期と I/O view の保証

Post-copy process では, ポストコピー方式による移送を行う. 前項で述べたように, 移送開始後すぐに移送先 VM での稼働に移り, ポストコピー方式の移送のみを行うように変更を加えた PostCopyLiveMigration を用いる.

Checkpointing process では, Microcheckpointing をベースとして用い, VM の同期を行う. Microcheckpointing は Qemu 2.3 上で開発が行なわれている, 複製 VM を作成する機能である. 一定時間毎に dirty page の探索と VCPU の情報を転送し, スナップショットを同期先 VM に転送する. この時, I/O をバッファリングしておき, スナップショットとして dirty page と VCPU の状態をすべて転送し終わったら蓄積したバッファを解放することで, スナップショットとの I/O view を保証している. 移送元 VM と移送先 VM を同期する際に, この機能を用いることにより, I/O の一貫性を保ちながらスナップショットを作成することができる. 提案手法で Microcheckpointing の機能を利用するために, PostCopyLiveMigration が実装されている

Qemu 2.6.1 上へ機能の移植を行った．

提案手法による，ポストコピー方式と，移送中の I/O view を保証した移送先 VM と移送元 VM の同期の手順を 4 に示す．VM 移送が開始すると，移送先 VM の稼働前に，ポストコピー方式による移送時に移送先 VM での再開に必要な情報の転送と checkpointing process による VM 同期の準備が開始される．ポストコピー方式による移送が開始すると，はじめに移送元 VM と移送先 VM で通信路が確保される．この通信路を用いてお互いの VM の状態を取得し，移送が可能であるかの確認がされる．その後，移送先 VM に対して VCPU の転送後，移送先 VM での再開命令を送る．移送先 VM では再開の命令を受けると，VM 移送用の通信路とは別に，新たに VM 同期用の通信路を確保する．これは，ポストコピー方式によるページの移送と，同期によるページ移送の混合が起きないようにするためである．移送先 VM では，checkpointing process により，出力のバッファリングと dirty page の転送を準備をする．移送元 VM では，checkpointing incoming process を起動し，スナップショットを受け取れる状態にしておく．同期の準備が終了すると，移送先 VM で稼働が再開する．

移送先 VM での稼働が開始すると，移送元 VM からポストコピーによるメモリの逐次送信と，移送先 VM から移送元 VM へのスナップショットの転送が開始される．移送先 VM では，post-copy incoming process により，メモリの受け取りと，未転送メモリへのアクセス時に Linux から発生する userfaultfd[8] の検知を行う．Userfaultfd が検出されると，未転送メモリへのアクセスが中断され，移送元 VM に対してページの取得要求が行われる．また，VM 稼働と同時に checkpointing process も実行される．Checkpointing process では定期的に VM を一時停止し，プレコピー方式の移送で用いられる，dirty page 探索用の関数を使用し，転送する必要のあるメモリを slab という形でまとめる．収集したメモリと VCPU の状態をスナップショットとして転送する際に，出力が保存されているバッファをデバイスへ解放する．転送後，VM を再開する．この動作を 100ms の間隔をあけて繰り返す．これにより，dirty page 探索中の短時間の VM 停止と 100ms の VM 稼働が繰り返される．これをポストコピーによる移送が終了するまで続ける．

ポストコピー方式の移送時，移送先 VM ではページフォルトの検知を行う以外は通常通り VM を稼働させている．そのため Microcehckpointing の短時間の VM 停止や ditry page の探索は VM の動作に大きな影響を与えない．また，ポストコピー方式の移送と Microcheckpointing による同期には共有するデータを持たないためそれぞれのプロセスを同期せずに行うことができる．

4.3 中断の検知と移送先 VM での再開

中断の検知は，checkpoint process で用いられる通信経路のネットワーク障害を検知することで行う．通常のポストコピー方式では，ネットワーク障害やキャンセル要求を検知した場合，移送元 VM，移送先 VM ともに異常を検知し終了してしまう．ポストコピー方式の障害発生時に終了してしまうと，最新のチェックポイントから再開することができなくなってしまう．そこで，移送元 VM に対して異常発生時に VM が終了せず，移送が中断された状態を保持したまま post-copy process を終了するように変更を加えた．

中断発生後，移送元 VM では checkpointing incoming process により VM 再開フェイズに入る．VM の状態を確認し，ポストコピー中断状態であれば，VM を再開する．移送先 VM では，VM の稼働を停止し，VM の保持する領域を解放する．この時，移送元 VM のバッファに保存されていた，最新のスナップショット以降の I/O は同時に破棄される．破棄された間の VM 上で稼働するアプリケーション

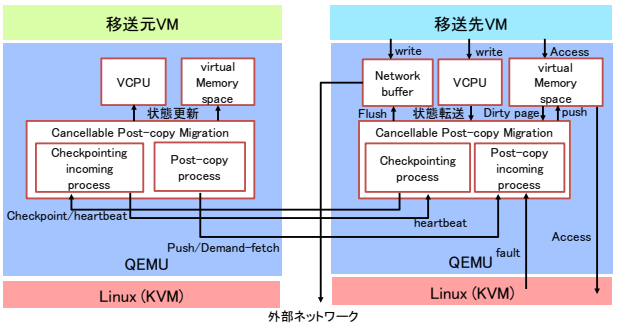


図 3: キャンセラブルなポストコピー VM 移送の概略図

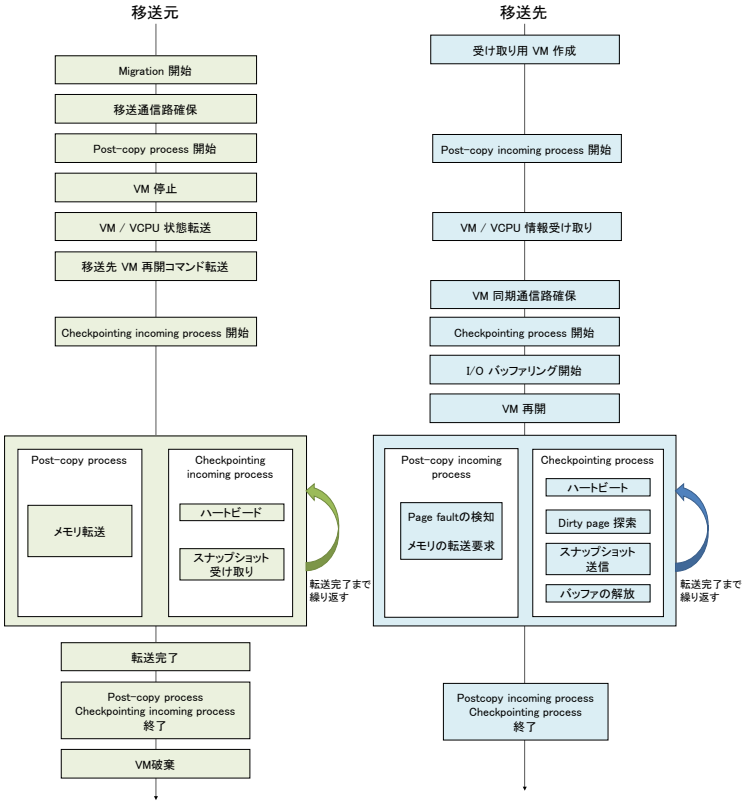


図 4: 提案手法の実行フロー

に対してネットワークを介したリクエストは, クライアントにはネットワーク障害として扱われる. これらは TCP の再送機能などに頼り, 改めて実行を行う.

4.4 今後の課題

上記の設計と実装に基づき, 中断時復帰可能なポストコピー移送方式についてプロトタイプの実装を行ったが, 設計した挙動にならず, 2 つの問題点が見つかった. 実際の処理の流れを図 5 に示す. 1 点目の問題点は, dirty page の探索中に未転送メモリへのアクセスが発生してしまい, 探索が終わらない問題がある. 従来の microcheckpointing では, プレコピー方式の移送時に用いられる dirty page 探索の関数によって, dirty page を一通り探索し移送先に転送される. しかし, ポストコピー方式による移送中に microcheckpointing による同期を行った場合, dirty page 探索時に未転送ページへのアクセスが発生してしまい, post-copy process によって userfaultfd が検知され, アクセスが中断し移送元ホストに対してページの要求が行われてしまう. そのため, すべてのページが揃うまでメモリの探索が終了しないという問題が発生する. これにより, 初回のスナップショット転送がポストコピーによるメモリ終了後になってしまう. 提案手法では, ポストコピー方式による移送中に移送先 VM と移送元 VM が同期されている設計にしているため, 要件を満たさない. この問題を解決するために, プレコピー方式の移送で用いられてる関数について, dirty page の探索範囲を転送済みページに絞るなどの手法により, 未転送ページへのアクセスを回避するように修正を加える必要がある.

2 点目の問題点として, ポストコピー方式の移送により受け取ったページが dirty page として認識されてしまう問題がある. これは, ゲスト OS のメモリ空間への書き込みを検知して dirty page として認識するため, ポストコピー方式により受け取ったメモリ内容を書き込む際にも dirty page として認識してしまうためである. その結果, 本来すでに移送元 VM が保持しているメモリをすべて移送元 VM に転送し直すことになってしまう. これにより, 最大 2 倍のネットワーク負荷が掛かるとともに, VM 同期に掛かる時間が長期化してしまうという問題点がある. この問題を解決するために, dirty page の取得方法をゲスト OS からのアクセスによって発生した dirty page のみを探索できるように変更する必要がある.

5. 予備実験

本章では，提案手法の評価を見積もるために行った, ポストコピー方式による移送と VM 複製についての実験について述べる.

5.1 実験環境

仮想化環境に qemu-kvm 2.6.1 を用いた. ネットワークで接続された 2 台の同じスペックの物理マシンを使用し, 1 台を移送元ホスト, もう 1 台を移送先ホストとした. 移送元ホストに対して VM のイメージファイルを保存し, ネットワークを介して移送先ホストと共有した. 物理マシンは, CPU16core, メモリは 32GB で, Linux4.8.6 の Debian が稼働している. 移送対象となる VM は, メモリを 8GB, VCPU を 1 コア割り当てをしている.

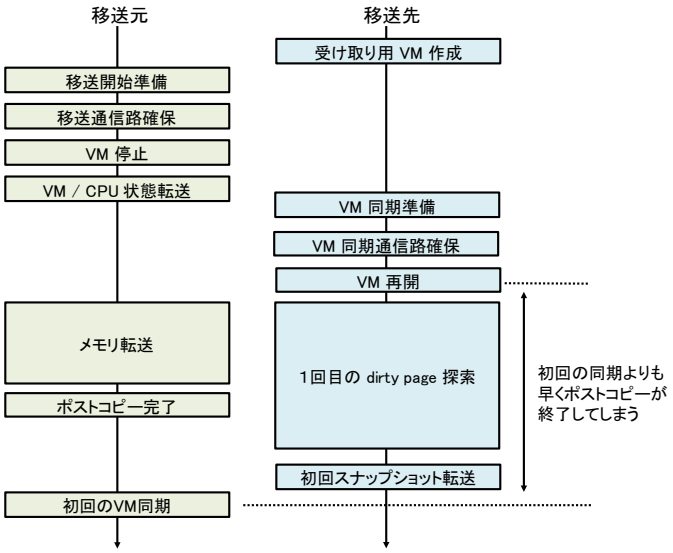


図 5: 実際の挙動

5.2 実験目的

提案手法の性能の見積もりを行うために, 関連する技術であるポストコピー方式の移送と MicroCheckpointing について性能評価を行う. 提案手法では, ポストコピー方式による移送を行うとともに, 移送元 VM と移送先 VM の同期を行う. 様々なワークロードが実行された VM を移送や複製した場合の VM の性能への影響を求めることで, 提案方式による移送中の VM の性能への影響を求める. また, 物理マシン上でそれぞれの機能がどれだけの資源を必要としているかを調査し, 提案手法の物理マシンへの影響を求める.

5.3 実験 1：VM 上のアプリケーションの性能評価

VM 上で様々なワークロードが実行された状態で, 移送や複製を行った場合の VM の性能への影響を求める.

5.3.1 実験方法

実験条件に合わせて確保したサイズの配列に対してメモリへの書き込みを繰り返し,1 秒間に書き込みが完了したデータサイズをスループットとして 60 秒間出力し続けるベンチマークソフトを作成した. 配列の大きさは 1KB, 256MB, 1GB に変更する. このベンチマークソフトを稼働させた VM に対して, ポストコピー方式の移送時, Microcheckpointing による VM 複製を行うことで, VM のス

ループットを計測した．比較対象として移送や複製を行わない状態の VM 上でのスループットを計測した．また, apache[9] のコンパイルを行い, 掛かった時間を time コマンドにより計測した．全実験において, ポストコピーの移送は, 計測開始から 10 秒後に移送を開始した．それぞれの条件を 5 回ずつ行い, それらを平均した値を実験結果として用いた．

5.3.2 実験結果

配列の大きさを変えたベンチマーク実行中の VM における 1 秒あたりのスループットを図 6 に示す．赤 (default) が移送を行わない場合のスループット, 青 (postcopy) が計測開始から 10 秒後にポストコピー方式による移送を開始したもの, 緑 (microcheckpoitning) はすでに複製を開始している VM に対して実行した場合のスループットである．また, apache の make に掛かった 5 回の平均時間を図 7 に示す．これらの結果をもとに, 提案方式の結果の見積もりの計算を行った．見積もった値は expected として各図中に表記する．

CPU インテンシブなワークロード：図 6 (a) より, CPU のキャッシュに入る 1KB の小さなメモリを更新した場合, VM での再開直後はスループットが 1 割以下まで減少するが, 時間経過とともに増加し続け 10 秒後には移送前と同程度の性能まで回復した．Microcheckpointing による VM 複製による VM のスループットは, 通常稼働させた VM に対して平均して 4%程度低下した．

メモリインテンシブなワークロード：図 6 (b) と (c) より, ポストコピーによる移送時の移送先 VM での稼働開始後, スループットが移送開始前の水準まで回復するまでに, 配列のサイズが 1KB のとき 9 秒, 256MB のとき 12 秒, 1GB のとき 18 秒間掛かった．これにより, 配列のサイズが大きくなるに連れて VM の性能回復までに要する時間が長期化していることがわかる．特に, 1GB の配列の場合は, 通常稼働させた VM に対してスループットが 20%以下の状態が 15 秒間続いている．したがって, メモリの更新範囲が広い場合, スループットが大きく減少することが確認できた．Microcheckpointing によるによる VM のスループットは, 通常稼働させた VM に対して平均して 10%程度の性能低下が起きた．CPU インテンシブなワークロード時の VM 同期による性能低下が 4%程度であるのに対して, メモリインテンシブなワークロードの場合スループットが減少していることが言える．このことより, 1 回のスナップショット転送あたりに発生する dirty page が増加すると, Microcheckpointing による VM への性能低下は大きくなると言える．

apache のコンパイル：図 7 より, 通常稼働させた VM 上でのコンパイル時間に対して, ポストコピー方式による VM 移送中は約 18%, Microcheckpointing による VM 同期を行った場合は 14%実行時間が長くなった．実行時間の逆数より, Microcheckpointing による性能は 12%程度低下している．提案方式の結果の見積もりは, ポストコピー方式の移送が完了するまでの時間を 20 秒と仮定し, ポストコピー移送時の結果に対して 20 秒間 12%の性能低下が起きたとして計算した．

5.4 実験 2：ホスト上の移送プロセスの負荷評価

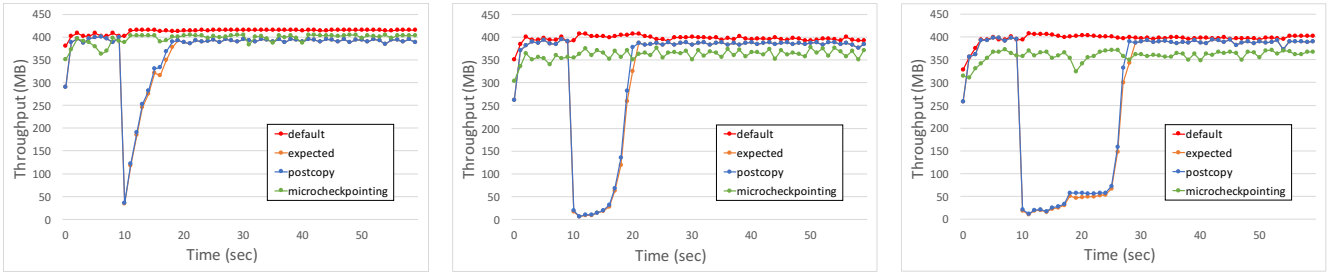
5.4.1 実験方法

条件に合わせた配列のサイズを確保し, 配列を更新し続けるプログラムを作成した．配列のサイズは 1KB, 256MB, 1GB に変更した．このプログラムを稼働させ続けた VM をポストコピー方式による移送と Microcheckpointing による VM 複製を行い, 物理ホストに対する CPU 負荷とネットワーク負荷を計測した．負荷の計測には sar コマンドを用い, 1 秒間の平均 CPU 使用率と 1 秒あたりの送信バイト数を 60 秒間取得した．ポストコピー方式による移送は計測開始から 10 秒後に開始した．ポストコピー方式の移送の場合は, 移送元ホストと移送先ホスト, Microcheckpointing の場合は, 複製元ホストと複製先ホストに対して計測を行った．比較対象として VM で同様のワークロードを稼働させ, 移送や複製を行わない場合の物理ホストに対する CPU 負荷とネットワーク負荷を計測した．

5.4.2 実験結果

実験に用いる配列のサイズを変更し, CPU 使用率を計測した結果を図 8,9 に示す．赤 (default) が移送を行わない場合の CPU 使用率である．ポストコピー方式による移送中の結果の場合, 青が移送元ホスト, 緑が移送先ホストに対する CPU 使用率である．VM 移送は計測開始から 10 秒後に開始した．Microcheckpointing による VM 複製中の結果の場合, 青が稼働中の VM が存在する同期元ホスト, 緑は同期状態で待機している同期先ホストに対する CPU 使用率である．ネットワークの送信状況を計測した結果を図 10,11 に示す．ポストコピー方式による移送中の結果の場合, 青が移送元ホスト, 緑が移送先ホストに対する 1 秒あたりのネットワーク出力バイト数である．Microcheckpointing による VM 複製中の結果の場合, 青が稼働中の VM が存在する同期元ホスト, 緑は同期状態で待機している同期先ホストに対する 1 秒あたりのネットワーク出力バイト数である．

CPU 負荷：図 8,9 より, VM を通常稼働させた場合の物理ホストに対する CPU が 6%以上であるのに対し, ポストコピー方式の移送開始後, 移送元 VM の CPU 負荷が 2%以下になっている．また, Microcheckpointing による複製先ホストへの負荷は, どの条件においても 1%未満である．今回の実験では VCPU を 1 コアのみ割り当てたが, CPU の割り当てを増加させることで, VM 稼働に必要な CPU のリソースに対して無視できる程度の負荷になると考えられる．同期元ホストでの CPU 使用率は, Microcheckpointing による同期中と通常稼動中の CPU 使用率の差は最大でも



(a) 1KB (b) 256MB (c) 1GB

図 6: VM のスループット

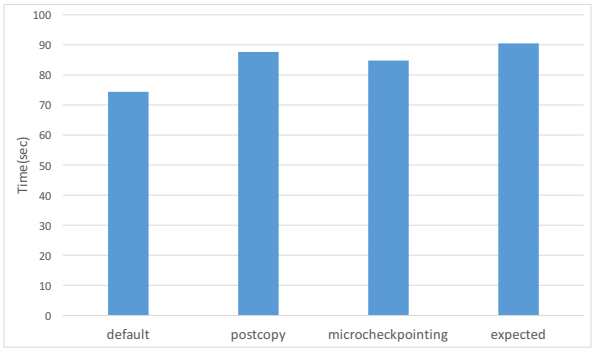


図 7: apache のコンパイルに要した時間

1%未満の差しか現れなかった. 図 9(a) の結果では, 通常稼動させた場合の CPU 使用率に対して Microcheckpointing による同期中の CPU 使用率は平均 0.2%高くなっている. しかし, 図 9(c) の結果では, Microcheckpointing による同期中の CPU 使用率が平均 0.08%低くなった. これは, dirty page を探索している期間に短時間 VM が停止するため, リソースを多く使うワークロード稼動中においては CPU 使用率が下がったのではないかと考えられる. したがって, Microcheckpointing による同期中の CPU 使用率は, 通常稼動させた場合と大きく差が出ないと言える.

ネットワーク負荷：図 10 , 11 より, ポストコピー方式の移送中は移送元ホストで 1 秒あたり 120MB 程度のデータが送信されていることがわかる. 一方, 移送元ホストでは 1 秒あたり 300KB 程度データ送信量である. また, Microcheckpointing による同期は, 配列のサイズが 1KB の場合は 1 秒あたり 8MB 程度であった. 一方, メモリに対して広範囲へのアクセスが起きる 256MB , 1GB の条件においては, 同期期間中 1 秒あたり 120MB 近くのデータを送信し続けていた. これらより, CPU インテンシブなワークロードの場合, 同期元 VM のネットワーク負荷は小さいが, メモリインテンシブなワークロードの場合ネットワーク負荷が大きくなることが確認できた. また, 同期先ホストに対するネットワーク負荷は 100KB 程度であった.

5.4.3 提案手法の物理ホストへの負荷

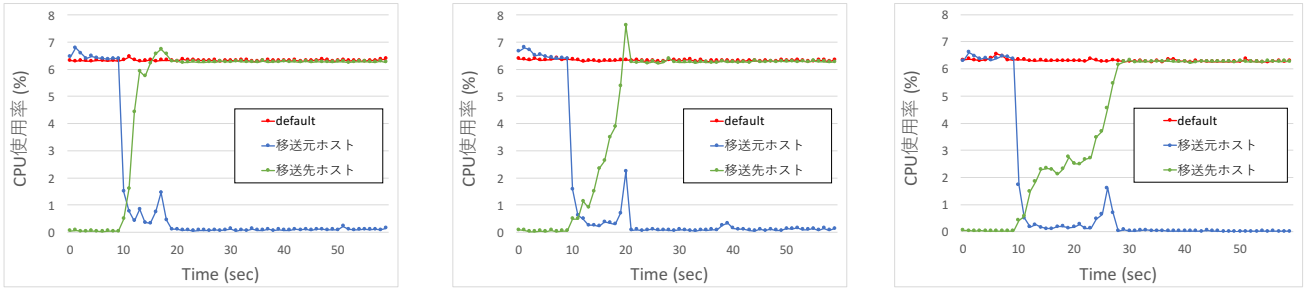
実験結果より見積もりを行った提案手法の CPU 使用率とネットワーク負荷を図 13 , 14 に示す.

CPU 使用率に関しては, ポストコピー方式による移送時の CPU 使用率の結果を元に見積もりを行った. 移送先ホストに対しては, ポストコピー方式による移送実行中の期間の結果に, VM 同期に掛かる同期先ホストの CPU 使用率を足し合わせた. 移送が行なわれている期間は, 図 10 , 11 のネットワーク負荷のグラフの転送量を元に推測を行った. 移送先ホストの CPU 使用率に関しては, 実験結果より同期元ホストに対する CPU 使用率への影響が少ないことが確認されたため, 今回の見積もりでは考慮しなかった. 見積もり結果より, 通常の VM が 6%程度の負荷であるのに対して, ポストコピー方式による移送が開始すると移送元ホストの CPU 使用率は 3%以下になる. これより, 移送元ホストに対する同期の負荷は, 移送元での稼働時の負荷に比べて十分低いと考えられる. したがって, 移送元ホストに対するポストコピー方式による迅速な負荷分散効果は維持されると考えられる.

ネットワーク負荷に関しては, ポストコピー方式による転送に掛かるネットワーク負荷の結果を元にし, ポストコピー方式にによる移送実行中の結果に, 移送先ホストの結果に対しては VM 複製元ホストにかかるネットワーク負荷を, 移送先ホストの結果に対しては VM 複製先ホストにかかるネットワーク負荷を足しあわせた. 見積もり結果より, 移送元ホストに対しては, VM 同期の際に送信するデータがポストコピー方式によるメモリ移送量に比べて大幅に少ないため, 通常のポストコピーによる移送にかかるネットワーク負荷とほとんど変わらないと思われる. 移送元ホストでは, ポストコピー方式によるメモリ転送量は VM 複製に掛かる送信量に比べてかなり少なく, Microcheckpointing による VM 同期時と同程度のネットワーク負荷がかかると思われる. また, 提案方式では, ポストコピー方式による移送によりすべてのメモリを 1 通り送り終わるまでの期間であるため, ネットワーク負荷が大きくなる期間は限定的だと考えられる.

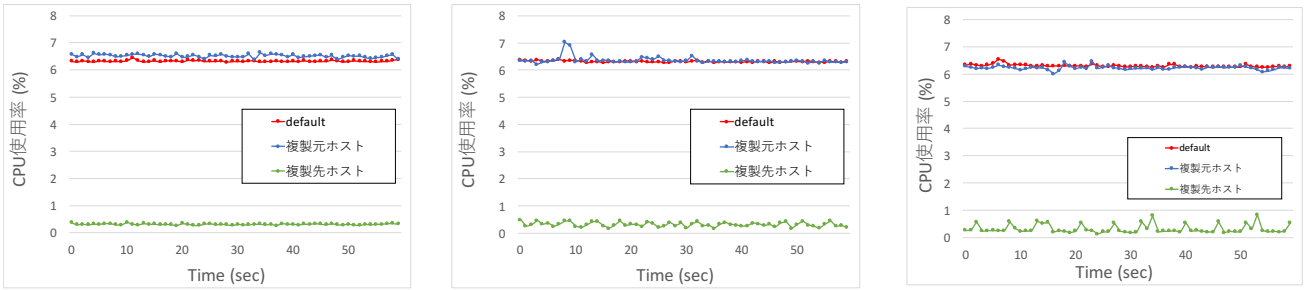
6. 関連研究

ポストコピー方式のライブ VM 移送の最適化手法が存在する. Enlightened Post-Copy[19] は, ポストコピー方式の



(a) 1KB　(b) 256MB　(c) 1GB

図 8: VM 移送中の CPU 使用率



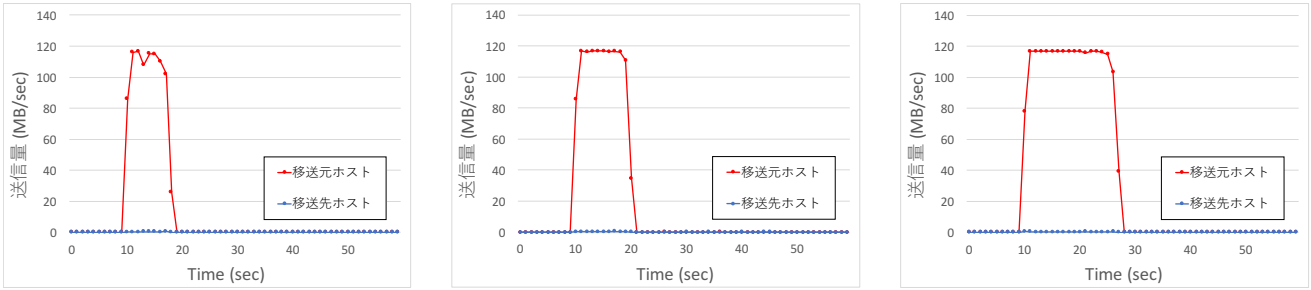
(a) 1KB　(b) 256MB　(c) 1GB

図 9: VM 複製中の CPU 使用率

移送時間の短縮を行うとともに, マシン上に存在する VM 全体のスループットを改善する手法である. Enlightend Post-copy では, アクセスされる可能性が高いメモリから先に転送することで, 未転送のメモリへのアクセスの抑制を行う. これにより, ポストコピー方式の高い負荷分散効果を得ながら, 移送時間の短縮を達成できる. HSG-LM[20] は, VMM に依存しない VM 移送方式を用い, プレコピー方式とポストコピー方式の利点を生かした Hybrid-copy 手法によるメモリ移送によって最小のダウンタイムを維持しながら総転送時間の短縮を実現する. HSG-LM では, VMM に移送プロセスを依存しないことで総転送時間を短縮することによりホストマシンのリソース解放を早める. これらはポストコピー方式の最適化手法について述べており, ポストコピー方式による中断が起きた場合, VM が破壊されてしまう点については言及されていない.

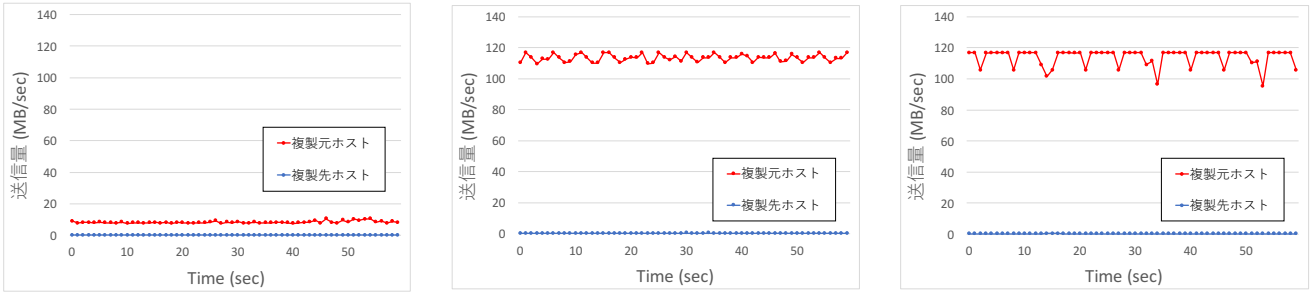
VM が保持するページ内容を認識する技術をライブ VM 移送に応用した手法が存在する. JAVMM[21] は,Java Virtual Machine(JVM) のヒープにおける, 短命オブジェクトが配置され更新度の高い young 領域の転送をスキップする. メモリ移送終了直前に強制的にガベージコレクションを行い, young 領域中で生存している部分のみを, stop & copy 時において転送する. これにより, 転送後のページが dirty page となることによるメモリの再送を抑制する. Memory Buddies[22] は, データセンタでの利用を想定した環境下におけるライブ VM 移送を用いた Content-based page sharing (CBPS) の最適化技術である. 提案方式では, ページを共有する可能性の高い VM を同じホストに配置することで, 共有ページを増加させ, CBPS の最適化を行う. Bootstrapping VMI (BVMI)[23] は, ブートストラップ方式の Virtual machine introspection (VMI) 手法である. 提案手法では, カーネルのバージョンに関係なく空ページ情報を取得し, メモリ転送をスキップすることで VM 移送の短縮を行う. これらの手法はプレコピー方式を対象とした手法であり, ポストコピー方式を用いた移送については対象とされていない.

ライブ VM 移送の範囲を従来の LAN 環境から WAN 環境に拡張する手法が存在する. CloudNet[24] は, ネットワークを仮想化することで, 地理的に離れた他データセンタとの接続をシームレスにし, VM 移送による柔軟なリソース管理を可能とする手法である. 提案手法では, 遅延によってメモリ転送時に dirty rate が収束せず, 再送が繰り返されることを防ぐために, 1 回あたりの転送量が今までのメモリ転送量から減らない場合プレコピー方式による転送を打ち切り, stop & copy による転送に切り替える. これにより, 低帯域幅かつ高遅延の環境でも短いダウンタイムを維持したままライブ VM 移送を可能にする. XvMotion[25] は, ライブ VM 移送を LAN 環境と WAN 環境のどちらでも利用可能な VM 移送方式である. 提案手法では, メモリとストレージの移送を行う. WAN 環境に対応するため, 不安定なネットワーク上でも安定してメモリやストレージ転送を行うことのできる TCP を介した移送フレームワークを提供する. また, メモリ移送時に dirty rate が転送レート



(a) 1KB (b) 256MB (c) 1GB

図 10: VM 移送中のネットワーク負荷



(a) 1KB (b) 256MB (c) 1GB

図 11: VM 複製中のネットワーク負荷

を超えた場合に VCPU に対して遅延を入れることで, dirty rate を調整可能にし, ネットワーク遅延により dirty rate が収束せずダウンタイムが長期化することを防ぐ. これらの手法はプレコピー方式を対象とした手法である.

7. まとめ

本章では，本研究のまとめおよび今後の課題について述べる．本研究では，ポストコピー方式による移送時に中断が発生すると VM が破壊されてしまうという問題点に対して, キャンセラブルなポストコピー VM 移送手法を提案した．提案手法では, ポストコピー方式による移送中に移送元 VM と移送先 VM を同期しておくことで, 移送の中断時には移送元 VM で再開を可能にする. ポストコピー方式による移送と VM 同期に対する性能調査を行い, 提案手法の性能の見積もりを行った. VM を同期することよる VM の性能低下は 10%程度であった. さらに, VM の稼働が移送先に移行した後, 移送元 VM での同期に必要な物理マシンに対するコストは, 移送前の VM の稼働に対して少ないことが確認できた. これにより, ポストコピー方式の移送中に VM の同期を行っても, VM の性能を維持しながら負荷分散効果を得られると推測できた.

7.1 今後の課題

本研究では，ポストコピー方式による移送時に中断が発生すると VM が破壊されてしまうという問題点に対して, キャンセラブルなポストコピー VM 移送手法を提案した．提案手法では, ポストコピー方式による移送中に移送元 VM と移送先 VM を同期しておくことで, 移送の中断時には移送元 VM で再開を可能にする. ポストコピー方式による移送と VM 同期に対する性能調査を行い, 提案手法の性能の見積もりを行った. VM を同期することよる VM の性能低下は 10%程度であった. さらに, VM の稼働が移送先に移行した後, 移送元 VM での同期に必要な物理マシンに対するコストは, 移送前の VM の稼働に対して少ないことが確認できた. これにより, ポストコピー方式の移送中に VM の同期を行っても, VM の性能を維持しながら負荷分散効果を得られると推測できた.

7.1.1 実装時の問題点の解消

本論文のプロトタイプの実装では, 4.4 項で述べたように, dirty page の探索中に未転送メモリへのアクセスが発生してしまい探索が終わらない問題と, 移送元 VM から転送されたメモリに対して dirty page と認識してしまう問題が発生した. これらの問題を解決し, 実際に提案手法を稼働させることで, VM の性能への影響や物理マシンへの負荷を改めて評価する必要がある.

7.1.2 VM 同期頻度の最適化

提案手法では, 定期的に移送先 VM のスナップショットを移送元 VM へ転送している. 今回の設計では, スナップショットの取得のタイミングは 100ms 毎になっており, VM の稼働状況は考慮されていない. VM の稼働状況によっては, dirty page が頻繁に更新される場合や, 更新がほとんどない場合など様々な場合が考えられる. Dirty page

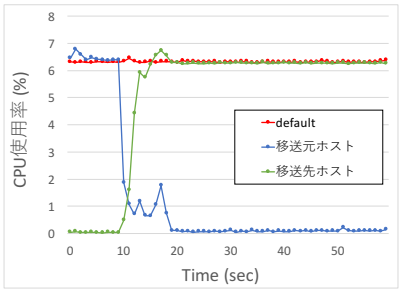
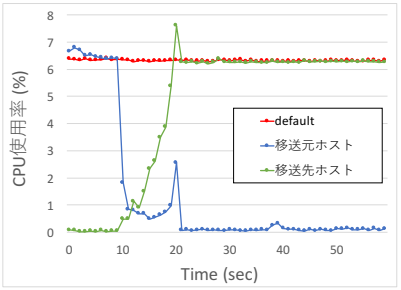
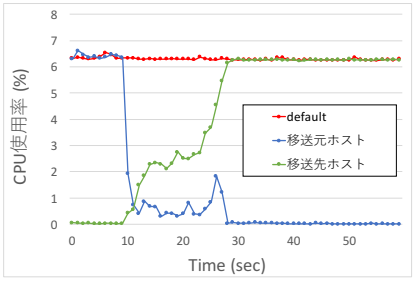


図 12: 1KB

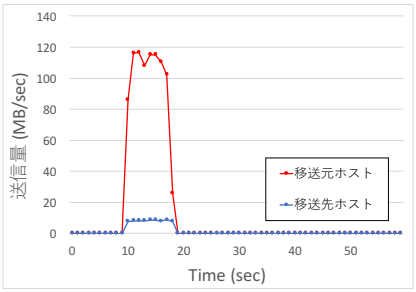


(a) 256MB

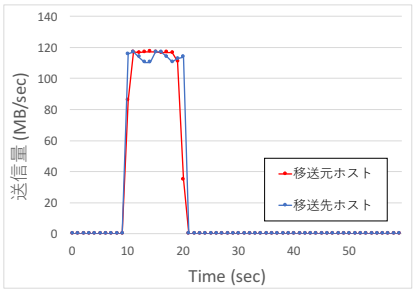


(b) 1GB

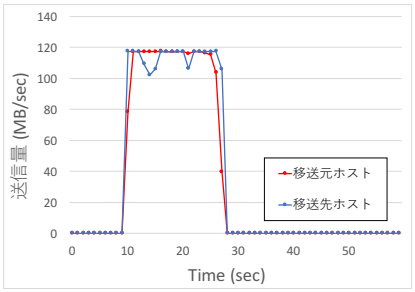
図 13: 提案手法の CPU 使用率



(a) 1KB



(b) 256MB



(c) 1GB

図 14: 提案手法のネットワーク負荷

の探索には VM が短時間停止するため, dirty page が少ない場合などは, スナップショットの間隔を長くすることで VM の性能低下を防ぐことができる. このように, VM の稼働状況によって, スナップショットを取得する際に適した間隔が変わってくると考えられる. そこで, VM 上のメモリの更新頻度やワークロードをもとに VM の同期頻度の調整を行う必要がある.

7.1.3 実践的なアプリケーション・ワークロードでの実験

本論文では, 提案手法の見積もりを行うために, メモリのみを更新するなど単純なワークロードを用いて実験を行った. 提案手法を用いた移送時の実際の性能をより正確に予測するためには, より実践的なアプリケーションを用いた実験をする必要がある. そこで, SPEC CPU や Memcached, Apache , MySQL などを用いた実験を実施する予定である.

謝辞

本研究の一部は総務省 SCOPE(152103004) の助成を受けたものである.

参考文献

[1] Amazon EC2, 入手先 ⟨https://aws.amazon.com/jp/ec2/⟩.
[2] Google Cloud Platform 入手先 ⟨https://cloud.google.com⟩.
[3] Xen, 入手先 ⟨https://www.xenproject.org/⟩.
[4] Kvm, 入手先 ⟨http://www.linux-kvm.org/⟩.
[5] Windows Server Hyper-V, 入手先 ⟨https://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx⟩.
[6] Qemu Features/MicroCheckpointing, 入手先 ⟨http://wiki.qemu.org/Features/MicroCheckpointing⟩.
[7] Qemu Features/PostCopyLiveMigration, 入手先 ⟨http://wiki.qemu.org/Features/PostCopyLiveMigration⟩.
[8] userfaultfd v4.3 [LWN.net], 入手先 ⟨https://lwn.net/Articles/656816/⟩.
[9] Apache HTTP Server Project, 入手先 ⟨https://httpd.apache.org/⟩.
[10] Timothy Wood and Arun Venkataramani and Mazin Yousif. *Black-box and Gray-box Strategies for Virtual Machine Migration*, Proc. of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07), pp 229–242, USENIX Association, 2007.
[11] Fabien Hermenier and Xavier Lorca and Jean-Marc Menaud and Gilles Muller and Julia Lawall *Entropy: a Consolidation Manager for Clusters*, Proc. of the 5th ACM International Conference on Virtual Execution Environments (VEE '09), 41–50, ACM, 2009.
[12] Hui Wang and Canturk Isci and Lavanya Subramanian and Jongmoo Choi and Depei Qian and Onur Mutlu *A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters*, Proc. of the 11th ACM International Conference on Virtual Execution Environments (VEE '15), 93–106, ACM, 2015.
[13] Sriram Govindan and Di Wang and Anand Sivasubramaniam and Bhuvan Urgaonkar *Leveraging Stored Energy for Handling Power Emergencies in Aggressively Provisioned Datacenters*, Proc. of the 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12) 75–86, ACM, 2012.
[14] Akshat Verma and Puneet Ahuja and Anindya Neogi *pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems*, Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware '08) 243–264, 2008.

[15] Meng Wang and Xiaoqiao Meng and Li Zhang *Consolidating virtual machines with dynamic bandwidth demand in data centers*, Proc. of the 30th IEEE International Conference on Computer Communications (INFOCOM '11) 71 -75, 2011.

[16] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. *Live migration of virtual machines*, Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2 (NSDI'05), 273–286, USENIX Association, 2005.

[17] Michael R Hines and Kartik Gopalan. *Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning*, Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '09), 51–60, ACM, 2009.

[18] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. *Remus: High availability via asynchronous virtual machine replication*, Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08), 161–174, San Francisco, 2008.

[19] Yoshihisa Abe, Roxana Geambasu, Kaustubh Joshi, and Mahadev Satyanarayanan. *Urgent Virtual Machine Eviction with Enlightened Post-Copy*, Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '16), 51–64, ACM, 2016.

[20] Yoshihisa Abe, Roxana Geambasu, Kaustubh Joshi, and Mahadev Satyanarayanan. Peng Lu, Antonio Barbalace, and Binoy Ravindran. *HSG-LM: Hybrid-copy Speculative Guest OS Live Migration Without Hypervisor*, Proceedings of the 6th International Systems and Storage Conference (SYSTOR '13), 2:1–2:11, ACM, 2013.

[21] Kai-Yuan Hou, Kang G. Shin, and Jan-Lung Sung. *Application-assisted Live Migration of Virtual Machines with Java Applications*, In Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15), 15:1–15:15, ACM, 2015.

[22] Timothy Wood, Gabriel Tarasuk-Levin, Prashant Shenoy, Peter Desnoyers, Emmanuel Cecchet, and Mark D Corner. *Memory buddies: exploiting page sharing for smart colocation in virtualized data centers*, Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '09), 31–40, ACM, 2009.

[23] Jui-Hao Chiang, Han-Lin Li, and Tzi-cker Chiueh. *Introspection-based Memory De-duplication and Migration*, Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '13), 51–62, ACM, 2013.

[24] Wood, Timothy and Ramakrishnan, K. K. and Shenoy, Prashant and van der Merwe, Jacobus. *CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines*, Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '11), 121–132, ACM, 2011.

[25] Ali Jose Mashtizadeh, Min Cai, Gabriel Tarasuk-Levin, Ricardo Koller, Tal Garfinkel, and Sreekanth Setty. *XvMotion: unified virtual machine migration over long distance*, In Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX ATC'14), 97–108 USENIX Association, 2014.