Vol.2019-OS-147 No.14
2019/7/25

インメモリデータベースと連動する仮想マシンライブ移送

真島 大輝[†1,a)]　山田 浩史[1,†1,b)]

概要：仮想マシン（VM）ライブ移送は，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．VM ライブ移送を用いることでサービスを停止せずに VM の再配置が可能となり，メンテナンスや負荷分散が容易になる．VM ライブ移送は強力であるが，インメモリ DB のようなメモリ使用量が多いアプリケーションが稼働する VM の移送では，その送信しなければならないメモリの増加分の送信時間以上に移送完了までの時間が伸びてしまう．移送中は移送プロセスがマシンのリソースを使用するため，VM 上で稼働するインメモリ DB の性能が低下してしまう．本研究では，インメモリ DB のための VM ライブ移送手法を提案する．提案手法では，インメモリ DB のアクセスに局所性があることを活用し，メモリ空間をアクセスの多いものと少ないものに分け, 移送直前にアクセスの少ない領域を削除することでメモリ転送量を削減する．これにより，移送時間を短縮して VM 上で稼働するインメモリ DB の性能劣化を抑えることを可能にする．提案手法を Redis 5.0.4, QEMU 3.0.93 上に実装し，その性能を計測した．提案手法は，ポストコピー方式と比べて転送量を削減し，総移送時間を 1 秒削減した．

キーワード：仮想化，仮想マシンライブ移送，インメモリ DB

1. はじめに

現在，インメモリ DB はキャッシュサーバや高い応答性が必要なサービスの提供に広く使われている．インメモリ DB とは全てのデータアイテムをメモリ上に展開するデータベースであり，従来のデータベースと比べてデータの管理方法が単純でデータアイテムへのアクセスが速く，安定して高いパフォーマンスを発揮することができる．近年物理マシンに搭載されるメモリが大幅に増加し， Amazon EC2 [1] や，Google Compute Engine [2] といったクラウドサービスでも多くのメモリを搭載した仮想マシンを借りることが可能になったため，インメモリ DB は各種ソーシャルネットワークサービスなどで，広く使われるようになった [3, 4].

仮想マシン（VM）ライブ移送は，ある物理マシン上の VM を稼働したままの状態で，異なるホストマシンへと移送する技術である．この技術はデータセンタなどの仮想化されたプラットフォームで大きな利益を生み出す． VM ライブ移送を用いることでサービスを稼働させたまま VM の再配置が可能になるため，負荷が集中する物理マシンから別の物理マシンに移送するという手法を用いて負荷分散を行うことができる [5]. また物理マシンから VM を退避させることで物理マシンやホスト OS のメンテナンスが容易になる．実際に，Google Cloud Platform のデータセンタでは VM ライブ移送を用いた運用が行われており，サーバメンテナンスや障害の解消のための VM 停止を減らし, サービスを長期間稼働させ続けることに成功している [6, 7].

VM ライブ移送は強力であるが，インメモリ DB が稼働する VM の移送は長い時間を要するため, インメモリ DB の性能低下を引き起こし, 移送を用いた柔軟な運用の足かせとなる．インメモリ DB は通常のデータベースではストレージに保存されていたデータアイテムをメモリ上に置いているため，従来のものと比べてメモリ使用量が多い．VM ライブ移送では VM が持つ全てのメモリを移送先のマシンへと送信する必要があるため，VM で稼働しているアプリケーションが保持するメモリ全てを転送しなければならない．そのため，多くのメモリを使用するインメモリ DB が稼働している場合は送信するメモリが多くなり，転送にかかる時間が増加する．現在主流の移送方式であるプレコピー方式では，1 度のメモリ転送ラウンドの間に起こるメモリページの更新部分を追加で送信しなければならない．そのため, 転送ラウンドが長期化し, メモリページの

[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)] dmashima@asg.cs.tuat.ac.jp
[b)] yamada@asg.cs.tuat.ac.jp

更新回数が増加すると, 送信するメモリが増えた分以上に総移送時間が延びてしまう．移送中は負荷が移送元のマシンから取り除かれない上に, 移送プロセスがネットワークやメモリなど物理マシンのリソースに負荷をかけるため, VM 上で稼働するインメモリ DB の性能が低下してしまう．よって, インメモリ DB が稼働している VM の移送にかかる時間を短く抑える手法が必要である．既存の VM ライブ移送を高速化する手法は移送の処理を改良するものが多く, 送信するデータの削減は行われない．送信するメモリ領域を制限し，送信するデータを削減する手法もあるが, インメモリ DB を対象としているものは存在しない．

本論文では，インメモリ DB のための VM ライブ移送手法を提案する．問題を根本的に解決するためにはインメモリ DB が持つメモリを削減する必要がある．提案手法では, インメモリ DB へのアクセスに局所性があることに着目し, アクセスが少ないデータアイテムの送信を行わないことでメモリ転送量を削減する．提案手法は, メモリ更新の総移送時間への影響が少ないポストコピー方式をベースにして，移送を行う直前にインメモリ DB のアクセス頻度が少ないデータアイテムを削除することでメモリ転送量を削減する．これにより総移送時間が短縮され VM 上で稼働するインメモリ DB の性能劣化を抑えることが可能になる．この動作を円滑に行うため，仮想マシンモニタ（VMM）はインメモリ DB と連動して移送を行う．

本論文の貢献は以下の通りである．

- インメモリ DB と連動する仮想マシンライブ移送を提案した．アクセスの局所性を活用したインメモリ DB が持つアイテムの削減によって移送するメモリ量を抑えて総移送時間を短縮し，VM ライブ移送の長期化によって発生するインメモリ DB の性能劣化を抑える．
- 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．これまでの分析で明らかになっているインメモリ DB のアクセス特性を利用して, アクセスの頻度が低いデータアイテムを移送開始直前に削除することでメモリ転送量を削減する．また，転送量を確実に削減するために，削除するデータアイテムをメモリ上にまとめて配置してメモリの断片化を抑える．
- 実験を行い，提案方式の有効性を確認した．Redis 5.0.4, QEMU 3.0.93 上に提案手法を実装し, ポストコピー方式の VM ライブ移送と提案手法に対して比較実験を行った．アイテムの削除によって転送量が削減された結果，総移送時間が 1 秒短縮されることがわかった．また，現状の実装の問題点を洗い出した．

本論文では，第 2 章で本研究の背景にあたる VM 移送技術について述べる．第 3 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 4 章では提案手法の設計と実装について述べる．第 5 章では，実験を行い提案手法の有効性を確認する．最後に第 6 章で本研究のまとめと今後の課題について述べる．

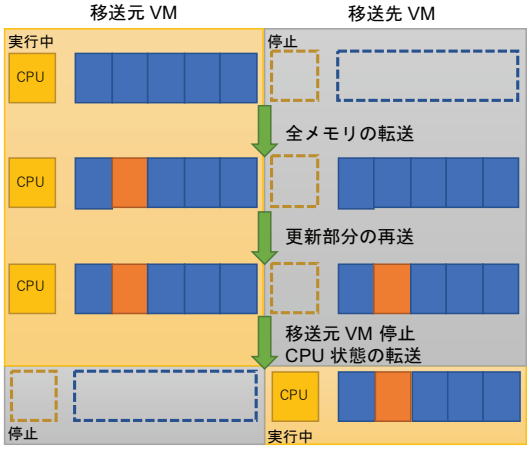

図 1 プレコピー方式による移送

2. 背景

プレコピー方式は VMware [8] や Xen [9]，KVM [10] などの VMM に実装されており, 現在主流の VM ライブ移送の実現方法である [11]．プレコピー方式の概略図を図 1 に示す．この方式は，移送中に起こったメモリなどの変更は全て移送元が保持している．そのため，VM 移送を中断しても移送元で VM を再開することができる．ネットワーク障害が発生して必要な情報を得られなくなった場合や，移送中の物理ホストに対する負荷の変動などにより管理者が任意で移送を中止したい場合にも，VM の稼働を保証できるため，信頼性が高い．一方で，送信すべきメモリが多い場合やメモリ更新が頻繁に発生するワークロードの場合，1 回のメモリ転送ラウンド中に起こるメモリページの更新が増加し，転送の繰り返し回数が増えてしまうため，メモリ転送量が多くなり，総移送時間が増加する．

ポストコピー方式は，実行を先に移送先 VM へ移し，その後に移送元のマシンから移送先のマシンへ必要なメモリを転送する VM ライブ移送の実現手法である [12]．ポストコピー方式の概略図を図 2 に示す．ポストコピー方式では，VM 移送が開始すると，移送先の VMM が VM 再開に必要な領域の準備を行う．次に, 移送元の VM を停止し, 移送先の VMM に仮想 CPU の情報を転送する．その後, 移送先で VM を再開する．VM が再開した後, 移送元のマシンから移送先のマシンへ VM が保持していたメモリの転送が行われる．この転送の間に, 移送先で稼働する VM で未転送メモリへのアクセスが発生した場合, ページフォルトが発生し, そのメモリ内容は移送元のマシンから優先的に送信が行われる．移送元のマシン上にある VM のオリジナルのメモリを送信し終わると移送プロセスを終了し, 移送元のマシン上にある VM を破棄する．

このポストコピー方式では同じメモリページは 1 度しか

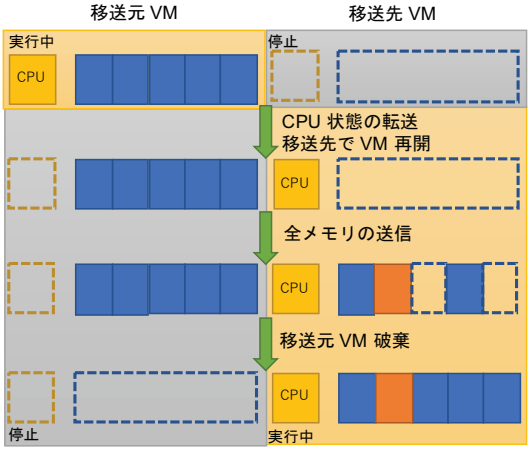

図 2 ポストコピー方式による移送

送信されないため，メモリの更新が頻繁に発生しても再送が起こらない．そのため, 従来の手法のように転送ラウンドが繰り返されて移送が長期化するということはない．また，メモリの送信前に VM の稼働場所が移送先マシンへと移るため，移送元のマシンから CPU やネットワークの負荷を迅速に取り除くことができる．一方で，VM の再開には CPU の状態以外に一部のメモリ内容が必要なためダウンタイム中に送信する内容が増加する．また, 移送中に起こった VM の状態変化は移送先のマシンにしか存在しないため, 移送が中断されると VM が破壊されてしまう．これを防ぐためには別の機構を用意する必要がある [13].

これら移送を効率化する手法はこれまでにも行われている．Xiang Song らの研究 [14] では送信するデータの探索や取得をマルチスレッドで処理することで効率化し，プレコピー方式の 1 度の転送ラウンドを短縮する事で総移送時間を短くしている．Changyeon Jo らの研究 [15] ではメモリと共有ディスクで重複している内容を共有ストレージを通じて送信する事で送信経路を増やし，メモリの送信を並列化し高速に行うことで総移送時間を短縮する．これらの手法では移送されるメモリの量は減らないため，送信すべきメモリがそもそも多い場合に移送が長期化してしまう．また，Kai-Yuan Hou らの研究 [16] では Java アプリケーションのヒープ領域のうち，更新の多い領域の送信をスキップすることでメモリ転送量を削減し，プレコピー方式で発生するメモリの再送を抑えることが可能とした．この手法では送信されるメモリの量は削減されるが，Java アプリケーションにしか適用することができない．Jui-Hao Chiang らの研究 [17] ではメモリの重複排除や空きメモリの送信を防ぐことでメモリ転送量を削減し，移送の高速化を図っている．この手法はインメモリ DB にも使用可能だが，使用メモリが多い場合は空きメモリの送信を防ぐことによる転送量削減が難しく，メモリ内容がどの程度重複しているかによって減らせる転送量が変わってしまう．

3. 提案

本研究ではインメモリ DB のための VM ライブ移送手法を提案する．インメモリ DB は大量のメモリを使用するため，従来通りの移送方式では総移送時間が長期化してしまう．特にプレコピー方式では増えたメモリを送信する分だけ各転送ラウンドにかかる時間が増加し, 1 度の転送ラウンド中に更新されるメモリページが増える．更新されたメモリページは再度送信しなければならないため, 総移送時間はメモリの増加分の送信時間以上に長期化してしまう．移送中は移送プロセスがネットワークやメモリなど物理マシンのリソースに負荷をかけるため, VM 上で稼働するインメモリ DB の性能が劣化してしまう．これを根本的に解決するにはインメモリ DB が持つメモリを削減する必要がある．そこで，インメモリ DB へのアクセスに局所性があることに着目する．Rajesh Nishtala らの研究 [3] から，インメモリ DB へのアクセスの 9 割が多くとも 3 割のデータアイテムに集中している，つまりデータアイテムの 7 割へのアクセスは全体の 1 割となる．そこでアクセスが少ないデータアイテムの送信を行わないことでメモリ転送量の削減が可能である．この考えから，インメモリ DB のデータの一部を移送時に送信しないデータとして削除することで, メモリ転送量を削減して総移送時間を短縮する手法を提案する．これによりインメモリ DB の性能が劣化する時間の短縮が可能になる．移送を行う直前にアクセスの少ないデータを削除することで，インメモリ DB は大きな影響を与えずに，メモリ転送量を大きく削減できる．

インメモリ DB のデータアイテムの一部を削除するという処理を行うためには，実際に移送を行う VMM がインメモリ DB の持つメモリ領域を把握し，送信せずに削除するデータがどこにあるかを知っていなければならない．この削除するデータを把握しているのはインメモリ DB 自身のみである．そのため，提案手法では VMM の移送プロセスが移送を開始する前に，インメモリ DB に対して削除するデータアイテムを削除する命令を出し，データアイテムが削除されるまで移送プロセスを待機させる．また，インメモリ DB でのデータアイテム削除した後，送信するメモリが確実に減っていなければならない．削除によってメモリ断片化が起きると VM の使用するメモリが減らず，メモリ転送量が変わらなくなってしまう．そこで提案手法では，インメモリ DB はアクセスの多いホットなアイテムとアクセスの少ないコールドなアイテムを分け，削除されるコールドなアイテムをまとめて管理する．これによってメモリ断片化を抑え，送信されるメモリの量が確実に減るようにする．

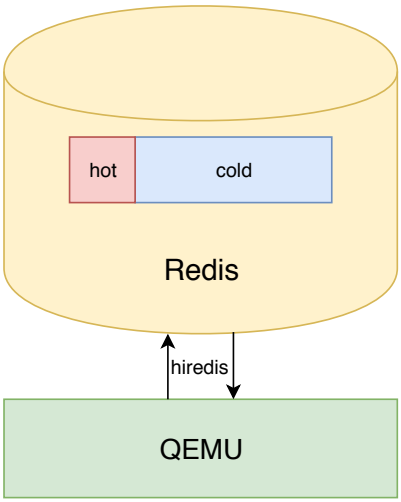

図 3 実装の概略図

4. 設計

本章では提案手法の設計について述べる. 実装は Redis 5.0.4 [18]，QEMU 3.0.93 [19] 上に行った．実装の概略を図 3 に示す．

4.1 インメモリ DB の設計と実装

提案手法でのインメモリ DB ではアイテムのアクセス数を集計し，アクセスの多いホットなアイテムとアクセスの少ないコールドなアイテムに分け，コールドなアイテムをまとめておく必要がある．提案手法を実現するためにまず，インメモリ DB が起動する際に，ホットなアイテムを持てるメモリの大きさとコールドなアイテムの領域の大きさを決定する．これは前述の研究の結果から，そのインメモリ DB で使用される予定のメモリの 3 割をホットなアイテムを持てるメモリの大きさとし，7 割をコールドなアイテムの領域の大きさとする．その後，アイテムが登録されると最初はコールドでない領域に配置される．そして，定期的にコールドでない領域にあるアイテムの大きさの合計がホットなアイテムを持てるメモリの大きさを超えているかを確認し，もし超えていたら全てのアイテムのアクセス数を確認して，アクセス数の少ないアイテムから順に ホットなアイテムを持てるメモリの大きさを切るまでコールドな領域へ移動し，またアクセスが多いコールドな領域にあるアイテムをコールドでない領域に戻す．このアクセス数は定期的に 0 に初期化される．

Redis はイベントループで駆動するため，1 つのイベントで長い時間がかかることを避けなければならない．アイテムのコールドな領域への移動，つまりコールドな領域にメモリをコピーし，そうでない領域にあるデータを解放するということを一度に全て行う処理には長い時間がかかり，他の命令をブロックしてしまう．そこで，命令ごとにコールドでない領域の大きさや命令で変化したアイテムのアクセス数を確認し，一部のアイテムだけを移動するように実装している．

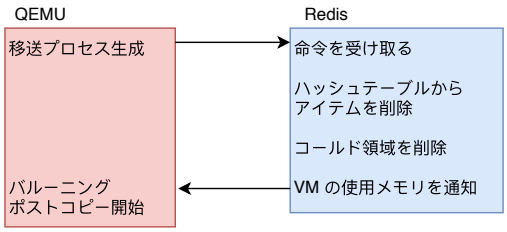

図 4 提案手法での移送の流れ

4.2 移送の流れ

移送の流れを図 4 に示す．移送が開始されると，まず VMM がインメモリ DB に移送が開始したことを知らせる命令を送信する．それを受け取ったインメモリ DB はコールドなアイテムを削除する．メモリの解放する回数が多いと命令を処理するのにかかる時間が増えるため，Redis 内部でハッシュテーブルからコールドなアイテムを削除した後に，コールドな領域を一括で解放する．そして，現在の VM の使用メモリを取得し，それを QEMU に返信する．QEMU は返信された値をもとに virtio を使用してバルーニングを行い [20]，その後ポストコピー方式でのメモリの移送を開始する．バルーニングとは VM が持つ使用していないメモリ領域をホストに返却する機構である．これは QEMU の移送プロセスが 1 度確保したメモリをそれが移送時に解放され使われていなかったとしても送信してしまうため，強制的に使われていないメモリを返却させるために使用される．

5. 実験

本章では，提案手法の評価のために行った実験について述べる．

5.1 実験環境

ネットワークに接続された 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．それらとは別の物理マシンに 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，移送元ホストと共有した．物理マシンは，CPU6core，メモリ 32GB，ギガビットイーサネットアダプタを搭載し，Linux 4.20.0 の ubuntu server 18.04 が稼働している．仮想化環境に qemu–kvm に対応した QEMU 3.0.93 を使用した．移送対象となる VM は，メモリを 5GB, VCPU を 4 コア

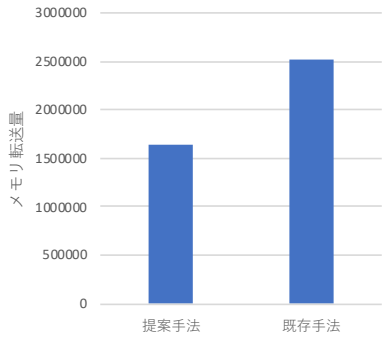

図 5 メモリ転送量

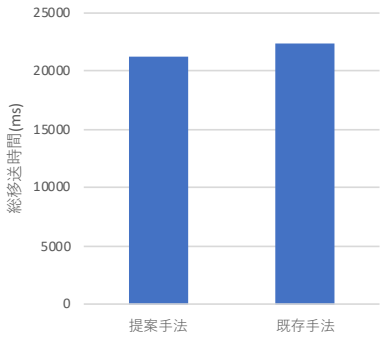

図 6 総移送時間

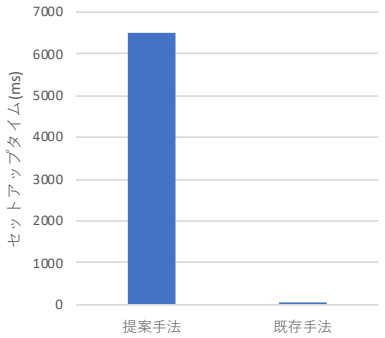

図 7 セットアップタイム

割り当てをしており，Linux 4.15.0 の ubuntu server 18.04 が稼働している．インメモリ DB には Redis 5.0.4 を使用している．

5.2 実験：VM の移送

5.2.1 実験方法

VM 上でホットなアイテムを持てる大きさを約 550 MB に設定した Redis に，キーの大きさが 1 から 8B，バリューの大きさが 100B のアイテムを 1000 万個登録した後に，その VM を提案手法と既存のポストコピー方式を用いて移送し，そのメモリ転送量と総移送時間，そして移送の命令が発行されてからメモリの転送が開始されるまでの時間であるセットアップタイムを計測する．

5.2.2 実験結果

図 5, 6, 7 に実験結果を示す．メモリ転送量は提案手法が 900MB 程少なく，大きく削減していると言える．しかし，アイテムがキーとバリューの他に幾つかのメタデータを持っていることを考えると，削除しているアイテムの割合に対してメモリ削減量が小さい．これは提案手法の実現のためにメタデータが追加したために同じ 3 割のデータでも提案手法の方が大きく，またメタデータが追加された構造体がデータアイテム以外の用途に Redis 内部で使われていることが原因だと考えられる．

総移送時間は提案手法が 1 秒ほど短くなった．メモリ転送量の削減量に対して非常に短い時間しか短縮できていないと言えるだろう．これは提案手法を実現するためにセットアップタイム中に行われている処理に時間がかかり，従来の手法と比べ非常に長い時間がかかっていることが原因と考えられる．

セットアップで行われている処理にかかる時間の内訳は図 8 のようになった．Redis 側の処理であるアイテムの解放と QEMU で行われる バルーニングに時間がかかっていることがわかる．アイテムの解放はコールドなアイテムを探しハッシュテーブルから削除するために全てのデータアイテムを検査しているために長い時間がかかっていると考えられる．

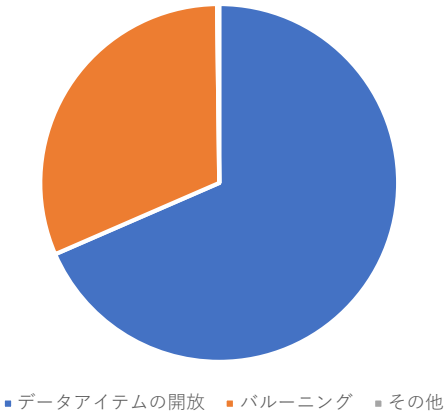

図 8 セットアップタイムの内訳

6. 終わりに

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．

6.1 まとめ

本研究では，インメモリ DB が稼働する VM の移送に長い時間がかかり，VM 上で動くインメモリ DB の性能が低下してしまうという問題点に対して，インメモリ DB と連動する仮想マシンライブ移送を提案した．提案手法ではポストコピー方式を応用し，メモリ転送前にインメモリ DB のアクセス特性を利用して，アクセスが少ない 7 割のデータを削除してメモリ転送量を削減し，総移送時間を短縮する．実験ではメモリ転送量を大きく削減し，総移送時間もわずかであるが短縮できることが分かった．

6.2 今後の課題

実験によって削減されたメモリ転送量に比べて総移送時間の短縮量が少ないことが示された．これを解決することが課題となる．アイテムの解放は全てのアイテムを検査することによって時間がかかっている．よって，コールドなアイテムがどれかを保持しておくことによって時間の削減が可能である．そして，アイテムを削除する代わりにコー

ルドなアイテムがあるメモリページの転送をスキップすることでバルーニングの手間を減らすことができると考えられる．

謝辞 本研究の一部は JSPS 科研費 JP17K00094 の助成を受けたものである．

参考文献

[1] Amazon EC2. https://aws.amazon.com/jp/ec2/. Accessed: 2019-04-23.

[2] 大規模クラウド コンピューティング プロダクト. https://cloud.google.com/products/compute/. Accessed: 2019-04-23.

[3] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling Memcache at Facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '13, pp. 385–398, Lombard, IL, 2013. USENIX.

[4] Mazdak Hashemi. The infrastructure behind twitter: Scale. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html. Accessed: 2019-04-23.

[5] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, pp. 17–17, Berkeley, CA, USA, 2007. USENIX Association.

[6] Google cloud platform blog: mgoogle compute engine uses live migration technology to service infrastructure without application downtime. https://cloudplatform.googleblog.com/2015/03/Google-Compute-Engine-uses-Live-Migration-technology-to-service-infrastructure-without-application-downtime.html. Accessed: 2019-04-23.

[7] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. VM Live Migration At Scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '18, pp. 45–56, New York, NY, USA, 2018. ACM.

[8] VMware. Vmware vmotion: Live migration of virtual machines without service interruption datasheet. https://www.vmware.com/pdf/vmotion_datasheet.pdf. Accessed: 2019-04-23.

[9] 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 & Implementation - Volume 2*, NSDI'05, pp. 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[10] Avi Kivity Qumranet, Yaniv Kamay Qumranet, Dor Laor Qumranet, Uri Lublin Qumranet, and Anthony Liguori. KVM: The Linux virtual machine monitor. *Proceedings Linux Symposium*, Vol. 15, , 01 2007.

[11] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pp. 25–25, Berkeley, CA, USA, 2005. USENIX Association.

[12] Michael R. Hines and Kartik Gopalan. Post-copy Based Live Virtual Machine Migration Using Adaptive Prepaging and Dynamic Self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pp. 51–60, New York, NY, USA, 2009. ACM.

[13] YaoZu Dong, Wei Ye, YunHong Jiang, Ian Pratt, ShiQing Ma, Jian Li, and HaiBing Guan. COLO: COarse-grained LOck-stepping Virtual Machines for Non-stop Service. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pp. 3:1–3:16, New York, NY, USA, 2013. ACM.

[14] Xiang Song, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. Parallelizing Live Migration of Virtual Machines. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '13, pp. 85–96, New York, NY, USA, 2013. ACM.

[15] 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. 41–50, New York, NY, USA, 2013. ACM.

[16] 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. 15:1–15:15, New York, NY, USA, 2015. ACM.

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

[18] Redis. https://redis.io/. Accessed: 2019-04-23.

[19] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pp. 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[20] Rusty Russell. Virtio: Towards a De-facto Standard for Virtual I/O Devices. *SIGOPS Oper. Syst. Rev.*, Vol. 42, No. 5, pp. 95–103, July 2008.