Vol.2021-OS-151 No.9
2021/3/2

ライブマイグレーションにおける サブページ書き込み保護の評価

小澤 洋介[1]　品川 高廣[1]

概要：ライブマイグレーションでは，仮想マシンを動作させたままメモリの内容を移行元から移行先へと同期させるため，移行元で仮想マシンが書き込んだメモリ領域を動的に検知して再転送する必要がある．従来の書き込み検知はページ単位でおこなわれているが，Intel はまもなくサブページ書き込み保護と呼ばれるページより細かい粒度でのメモリ保護が出来る機構を導入予定であり，これをライブマイグレーションに応用することでメモリ転送量の削減やマイグレーション時間の短縮が期待できる．一方，この機構は書き込み時にダーティビットをセットする代わりにページフォルトを発生させるため，実行時の CPU オーバーヘッドの増加が予想される．本研究では，様々なワークロードにおける書き込み操作のトレースからメモリ転送量の削減効果や CPU オーバーヘッドの大きさを分析し，ライブマイグレーションにおけるサブページ書き込み保護機構の有効性を評価した．エミュレータを用いた評価の結果，メモリ転送量の削減によりマイグレーション時間は多くのワークロードにおいて従来方式よりも短縮できることや，あるワークロードではサブページ書き込み保護のみがマイグレーションを完了させられることが分かった．また，CPU オーバーヘッドは，ページのゼロクリア命令を検出して書き込み検知の最適化をおこなうことにより 8.3%〜54.5%程度に抑えられることが分かった．

1. はじめに

ライブマイグレーションは，あるホストで実行中の仮想マシンを停止することなく別のホストへと移行する技術であり，負荷分散やハードウェア維持管理，電力管理，フォールトトレランスなどに有用である [1]．ライブマイグレーションでは，移行元と移行先で仮想マシンのメモリ内容を同期する必要があり，その実現にはプレコピー方式 [2] が広く使われている．この方式では，移行元で仮想マシンを動作させたまま移行先にメモリ内容を転送し，転送中に新たに書き込まれたメモリ内容を再度転送するという処理を，差分が十分に小さくなるまで繰り返す．従って，転送メモリ量を削減することは，マイグレーション時間やネットワーク負荷，電力消費 [3] の削減に有効である．

ライブマイグレーションにおいて，転送中に新たに書き込まれたメモリ内容を検知するためには，ネステッドページングに基づく方式が広く使われている．この方式では，仮想マシンがゲスト物理ページに書き込みをおこなうと，CPU のハードウェアがネステッドページテーブルの対応するエントリのダーティビットをセットするため，仮想マシンモニタはダーティビットを走査することによって，新たに書き込みがおこなわれたゲスト物理ページを検出することが出来る．この方式は CPU のハードウェアが書き込み検知をおこなうため実行時のオーバーヘッドを低く抑えられるが，書き込まれた領域の検知がページ単位になるため，ページの一部だけが書き込まれた場合でもページ全体を転送する必要があり，結果としてメモリ転送量が必要以上に大きくなる可能性がある．

ソフトウェア的に書き込み検知をページより細かい粒度でおこなう手法としては，転送済みページからの差分をとる手法がある [4,5]．また，ページより細かい粒度でハッシュを計算する手法 [6,7] や，ページを圧縮することで転送量を削減する手法 [8,9] も提案されている．しかし，これらの手法は，CPU やメモリのオーバーヘッドがかかるため，ホストの負荷が高い場合には，仮想マシンの性能に大きな影響を与える可能性がある．転送ページ数を削減するアプローチ [10–18] も多数提案されているが，ページより細かい粒度での書き込み検知は考慮していない．

本研究では，CPU のサブページ書き込み保護機構をライブマイグレーションにおける書き込み検知に応用した場合の有効性を評価する．サブページ書き込み保護とは，ネステッドページングに追加される機能であり，通常のページをより細かい単位に分割したサブページという単位で書き込み保護をおこなうことが可能である．従って，サブペー

[1] 東京大学
The University of Tokyo

ジ書き込み保護をライブマイグレーションに応用することにより，ページより細かい粒度での書き込み検知をハードウェア的におこなうことが可能である．

サブページ書き込み保護は，指定したページの各サブページあたりに書き込み保護をおこなうかどうかの情報を保持するだけでよいので，追加のメモリ消費を少なく抑えることが出来る．一方，サブページ書き込み保護にはダーティビットの機能が無く，書き込み保護がされたサブページへの書き込みがおこなわれるたびにページフォルトが発生する．従って，頻繁にサブページへの書き込みが発生する場合には，ページフォルトのオーバーヘッドが大きくなる可能性がある．

そこで本研究では，様々なワークロードにおけるライブマイグレーション時の挙動を擬似的に測定し，サブページ書き込み保護によるメモリ転送量の削減効果や CPU のオーバーヘッドを分析した．測定環境としては，Intel がまもなく導入予定の Sub-Page Write Permissions for EPT (SPP）[19] を搭載した CPU を対象とした．本稿の執筆時点では SPP を搭載した CPU の実機は入手できなかったため，SPP をエミュレート可能な IA-32 エミュレータである Bochs [20] をベースにして，一部のバグ修正を施して実験環境として使用した．また，QEMU/KVM をベースにして SPP による書き込み検知を実装し，上記のエミュレータ上で動作させて性能評価をおこなった．

実験の結果，多くのワークロードにおいて，メモリ転送量の削減によりマイグレーション時間は従来のページ単位の方式よりも短縮することが可能であり，QEMU に搭載された差分圧縮方式である Xor Binary Zero Run Length Encoding (XBZRLE) [5] を組み合わせた手法と同程度のマイグレーション時間をより少ないメモリ消費で実現出来ることが分かった．また，SPEC CPU 2017 のベンチマークの一つである 531.deepsjeng_r においては，サブページ書き込み保護のみがライブマイグレーションを完了させられることが分かった．また，CPU オーバーヘッドは，単純にサブページ単位で書き込み検知をおこなうと 8.3%〜80.0%程度であったが，ページのゼロクリア命令を検知して最適化をおこなうことにより 8.3%〜54.5%程度に削減できることが分かった．

2. 背景

2.1 ライブマイグレーション

ライブマイグレーションの方式としては，プレコピー方式 [2]，ポストコピー方式 [21]，及びそれらを併用するハイブリッド方式 [22] などがあるが，本研究では主に標準的に使われているプレコピー方式を対象とする．プレコピー方式では，移行元から移行先にメモリコピーを複数回繰り返してメモリの内容をある程度同期させた後，移行元で仮想マシンの実行を停止して最後の同期をおこなう stop-and-copy フェーズを経て，移行先で実行を再開することでライブマイグレーションが実現される．仮想マシンを移行元で停止してから移行先で実行を再開するまでがダウンタイムとなるため，stop-and-copy にかかる時間を数百ミリ秒単位まで削減することによって，ユーザーは仮想マシンが継続的に動作しているかのように感じることができる．なお，ディスクに関しては基本的には共有ストレージを用いるため，データ転送を必要としないケースが多い．従って，ライブマイグレーションにおいてはメモリ転送のコストが最もコストの大きい処理となる．

ライブマイグレーションにおける繰り返し処理の各イテレーションでの具体的な操作は以下のとおりである．第一イテレーションでは，仮想マシンのメモリ全体のコピーをおこなう．第二イテレーション以降では，直前のイテレーション中に書き込みがおこなわれたメモリ領域を検知し，その内容のみコピーをおこなう．この作業を繰り返すことにより，各イテレーションにおける転送メモリ量および転送時間が徐々に減少し，最終的に転送メモリ量が予め指定した閾値を下回った時に stop-and-copy がおこなわれる．

イテレーションの開始から移行先での仮想マシンの起動までにかかった時間を総マイグレーション時間と呼ぶ．また，全イテレーション及び stop-and-copy フェーズでの転送メモリ量の合計を総転送メモリ量と呼ぶ．

2.2 書き込み検知の細粒度化

プレコピー方式においては，各イテレーションでメモリを転送中に新たに書き込みがおこなわれたメモリ領域を検知して，それを次のイテレーションで再転送する必要がある．書き込みがおこなわれる場所や頻度はアプリケーションやワークロードによって異なるため，転送メモリ量を必要最小限に抑えるためには，仮想マシンモニタが書き込みがおこなわれた範囲を適切に捕捉する必要がある．各イテレーションでの転送メモリ量の削減は総転送メモリ量の削減につながり，それはマイグレーション時間及びネットワーク負荷の削減に直結する．また，イテレーションを重ねる毎に転送メモリ量が減らなければ，転送メモリ量が閾値を下回ることができず，マイグレーションを完了させることが出来なくなる．

現在実用的に使われているライブマイグレーション機構においては，基本的には書き込み検知の粒度はページ単位となっている．これは，書き込み検知が主に CPU のネステッドページングの付随機能であるダーティビットを用いて実装されているためである．しかし，アプリケーションは必ずしもページ全体に書き込みをおこなうとは限らない．例えば，4KiB ページのうちの 1 バイトだけ書き込んだ場合でも，ダーティビットだけではページの中の実際に書き込まれた部分だけお検知することが出来ないため，4KiB ページ全体を転送する必要がある．このようなページへの

部分書き込みが多い場合には，各イテレーションで転送するメモリ量は，実際に書き込まれたメモリ量と比べて数百倍〜数千倍に及ぶ可能性がある．

3. サブページ書き込み検知

本章では，CPU のハードウェアを利用した書き込み検知の細粒度化の手法として，サブページ単位での書き込み検知の手法について述べる．本章では，まず section 3.1 でサブページ書き込み保護による書き込み検知について説明し，次に section 3.2 でそれを用いたサブページ単位での書き込み検知システムについて述べる．

3.1 サブページ書き込み保護による書き込み検知

サブページ書き込み保護とはページング機構の追加機能であり，従来のページングよりも細かい粒度で書き込み保護をおこなえる機能である．この機能では，従来の通常サイズの「ページ」を，より小さくて同じサイズの「サブページ」に等分割し，個々のサブページに対して個別に書き込み保護を設定することが出来る．書き込み保護が設定されたサブページに書き込みが発生した場合，CPU はサブページフォルトを発生する．

サブページ書き込み保護を書き込み検知に応用した場合，従来の方式と比べて新たなオーバーヘッドが発生する．従来のページングによる書き込み検知では，CPU のハードウェアが自動的に設定するダーティビットを活用できるため，極端にメモリ帯域が逼迫しているケースを除いては，仮想マシンからの書き込みに対するオーバーヘッドはほとんど無いと考えられる．一方，サブページ書き込み保護では，仮想マシンが書き込み保護を設定したサブページに書き込み保護をおこなうたびにページフォルトが発生する．ページフォルトが発生すると，仮想マシンモニタへのコンテキストスイッチ，イベントの特定，ページフォルトが発生したアドレスの取得，書き込み保護設定の更新などの多数の操作が必要となり，ソフトウェア的なオーバーヘッドがそれなりに必要である．従って，このオーバーヘッドを抑えるための仕組みが必要になると考えられる．

3.2 サブページ単位での書き込み検知システム

図 1 に，本研究で想定するサブページ書き込み保護を利用したサブページ書き込み検知システムの概要を示す．仮想マシンモニタは，サブページ単位での書き込み保護を設定できるように拡張されたネステッドページテーブルを管理する．まず仮想マシンモニタは，書き込み検知をおこなう対象の仮想マシンに対して，メモリを割り当てているゲスト物理アドレスの全領域について書き込み保護を設定する．対象の仮想マシンがゲスト物理メモリの書き込み保護が設定されたサブページに対して書き込みをおこなうと，CPU はサブページフォルトを発生させて，制御を仮想マシンモニタに渡す．仮想マシンモニタは，サブページフォルトハンドラにおいて，書き込みがあったサブページを記録し，仮想マシンの実行を再開する．この操作を繰り返した後，仮想マシンモニタは書き込みがあったサブページを列挙することにより，仮想マシンから書き込みがおこなわれたメモリ領域をサブページ単位で検出することが出来る．

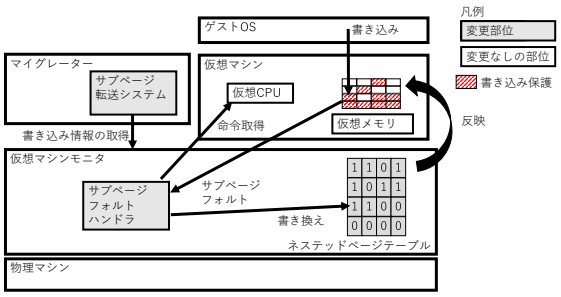

図 1 サブページ書き込み検知システムの概要

ライブマイグレーションをおこなう場合は，第二イテレーション以降では，仮想マシンモニタと連動するマイグレーターのサブページ転送システムが書き込みがあったサブページのみを取得して，その内容を移行先に転送する．転送メモリ量が一定の閾値を下回った際には stop-and-copy フェーズに移行し，移行先のホストで仮想マシンの動作を再開する．

サブページフォルトによるオーバーヘッドを抑えるためには，サブページフォルトの発生回数を抑える仕組みが必要である．まず，ライブマイグレーションにおいては，各イテレーションにおいてあるサブページに複数回の書き込みがあったとしても，その書き込みは全てまとめて転送することになる．従って，最初にサブページフォルトが発生したら，そのサブページへの書き込みはいったん許可するように設定し，次のイテレーションが始まるときに再度書き込み保護を設定すればよい．これによって，各イテレーションで同一サブページで複数回のサブページフォルトが発生することによるオーバーヘッドを抑えることが出来る．

また，サブページフォルトの要因となった命令を調べることで，次にサブページフォルトが起こる領域を予想する手法が考えられる．例えば，サブページフォルトを起こした命令が，ページ全体への書き込みを繰り返すような命令だった場合には，個々のサブページフォルトが発生するたびに書き込みを許可する代わりに，最初のサブページフォルトが発生した際に，連続するサブページが全て書き込まれたとみなして，予めそれらのサブページの書き込み許可を設定しておくといった最適化が考えられる．

4. 実装

本研究では，サブページ書き込み保護を提供する CPU の

アーキテクチャとして Intel SPP を使用した．動作環境には SPP のエミュレーションが実装された x86 エミュレーターである Bochs [20] を使用し，仮想マシンモニタとしては KVM に SPP 対応パッチ [23] を適用したものをベースにサブページ書き込み検知を実装し，サブページ単位のマイグレーションに必要な機能の一部を QEMU に実装した．

4.1 Intel SPP

Intel SPP は，Intel CPU のネステッドページング機構である Extended Page Table (EPT) の拡張機能であり，通常の 4KiB ページを 128 バイトのサブページ 32 枚に分割して，各サブページ毎に書き込み保護を設定することが出来る．SPP を使うためには，まず Virtual Machine Control Structure (VMCS) の VM-execution control で SPP の機能を有効にした上で，EPT の末端のページテーブルのエントリで SPP ビットを 1 にセットする．また，SPP はページテーブルと同様の構造体で構成されており，ルート SPP テーブル（SSPL4）と呼ばれる最上位のページテーブルへのポインタを VMCS に予め設定しておく．

CPU は EPT のエントリの SPP ビットが 1 にセットされていた場合，SSPL4 から順番に SPP ページテーブルのページウォークをおこない，SPP ベクターテーブル（SSPL1）と呼ばれる末端のテーブルを特定する．SSPL1 は 64 ビットのエントリが 64 個で構成されており，各エントリが 1 枚の 4KiB ゲスト物理ページに対応する．各エントリの 64 ビットは 32 組の 2 ビットから構成され，各 2 ビットが 1 枚のサブページに対応する．この 2 ビットのうち上位ビットが書き込み許可ビットになっており，このビットが 0 であるサブページへの書き込みがあった場合には，ページフォルトに相当する EPT violation が発生する．このとき，ソフトウェアは対応する EPT エントリの SPP ビットを調べることで，EPT と SPP のどちらで書き込み保護違反が発生したのかを特定することが出来る．

4.2 Bochs の修正

Intel SPP は第 10 世代の Intel Core プロセッサである Ice Lake から実装されるとされていたが，現在入手可能な Ice Lake 世代の CPU はモバイル用しかなく，SPP が実装された CPU を入手することは出来なかった．そこで，動作環境として SPP のエミュレーションが実装された x86 のエミュレータである Bochs を使用した．Bochs のバージョン r13850 を使用したが，このバージョンの Bochs の SPP エミュレーションには以下に述べるようなバグがあったため，それを修正したうえで使用した．

Bochs は TLB を実装しており，EPT のページテーブルのエントリもキャッシュするようになっている．一方，Intel Software Developers Manual によると，SPP のエントリも TLB にキャッシュされる仕様となっているが（28.3.1 項 “Information That May Be Cached” 参照），Bochs は SPP に関する情報をキャッシュしない実装となっていた．その結果，あるゲスト物理ページの 32 枚のサブページのうち 1 枚でも書き込みが許可されると，そのゲスト物理ページのエントリが書き込み許可の状態で TLB にキャッシュされてしまい，ページ全体が書き込み許可されたものと判断されて SPP による書き込み保護が機能しなくなるバグがあった．そこで，SPP に関する情報も TLB に正しくキャッシュするように修正することで，書き込み保護が正しく機能するようにした．

4.3 KVM の修正

サブページ書き込み検知を実装する仮想マシンモニタのベースとしては，Linux 5.5.7 の KVM に SPP 対応のパッチを適用したものを使用した．このパッチにより，KVM は SPP の有効化や SPP ページテーブルの構築，EPT violation 及び SPP 関連イベント発生時のハンドリングをおこなうことができる．この KVM をベースとして，section 3.2 で述べた書き込み保護違反を起こした命令に基づく最適化を実現するために，EPT violation が発生したときの命令を取得する機能を実装した．命令の取得は，EPT violation 時に仮想 CPU の命令ポインタの値を取得し，仮想マシンのメモリからそのアドレスのメモリの値を読み取ることで実現した．

4.4 QEMU の修正

ライブマイグレーションのためのサブページ書き込み検知を実装するために，QEMU 4.2.50 をベースとして使用した．書き込みがおこなわれたサブページを QEMU で取得するために，KVM と連携して指定したサブページの書き込み許可ビットの値を ioctl で取得する仕組みを実装した．また，QEMU ではライブマイグレーションのメモリ転送時に，転送するページの物理アドレスをメタデータとして一緒に送るようになっているが，サブページ単位での物理アドレス記述に対応するため，メタデータとして新たに 1 バイトの値を追加した．

5. 評価

ライブマイグレーションにおけるサブページ書き込み保護を用いた書き込み検知の有効性を評価するために，(1) 従来の 4KiB ページ単位の書き込み検知手法（4KiB ページ書き込み検知），(2) 4KiB ページ書き込み検知と QEMU に標準搭載の XBZRLE を組み合わせた手法（4KiB ページ書き込み検知＋ XBZRLE），及び Intel SPP によるサブページ書き込み検知手法（128B サブページ書き込み検知）の 3 つの手法で性能評価をおこなった．

実験環境は以下の通りである．実機としては，CPU が Intel Core i7-4790K，メモリが 16GiB のマシンを使用し

た．Bochs が提供する仮想マシンの構成は，仮想 CPU が 1 個，メモリが 2GiB であり，その仮想マシン上で Debian 10 (buster) と Linux 5.5.7 に SPP patch を当てたカーネルで QEMU/KVM を動作させた．QEMU/KVM が提供する仮想マシンの構成は，仮想 CPU が 1 個，メモリが 1GiB で，その仮想マシン上で Debian 10 (buster) と Linux 5.5.7 を動作させた．ライブマイグレーションにおけるネットワーク帯域幅は 1Gbps を仮定した．また，XBZRLE で転送ページの一時保存に用いるメモリサイズは 512MiB とした．

仮想マシンのワークロードは以下のものを使用した．

idle　仮想マシン上で特定のワークロードを実行しない．

kernel compile　仮想マシン上で linux 5.5.7 のコンパイルを実行する．

redis+YCSB　仮想マシン上で redis 5.0.3 [24] を実行し，外部から YCSB [25] 0.17.0 [26] を実行する．YCSB のプリセットワークロード（**workloada〜workloadf**）を用い，各レコードのフィールドサイズを 100 バイト，フィールド数を 10，レコードサイズを 1KiB，レコード数を 256K とし，redis は約 500MiB のメモリを使用するものとする．

SPECCPU　SPEC CPU 2017 v1.0.1 [27] に含まれるもののうち 13 のワークロード（**xxx.*_r**）を用いる．

5.1　疑似マイグレーション

実機と著しく性能の異なる Bochs を用いて性能評価をおこなうために，時間経過ではなく実行命令数を基準としてメモリ転送量を測定する疑似マイグレーションをおこなった．疑似マイグレーションの流れを以下に示す．

(1) 転送メモリ量の取得: 第一イテレーションではゲスト物理メモリ全体，それ以降は書き込みがおこなわれたページ（サブページ）の総バイト数を転送メモリ量とする．XBZRLE を使用する場合は，差分圧縮後のデータサイズを転送メモリ量とする．

(2) 仮想マシンの実行: (1) で求めた転送メモリ量の転送にかかる時間を転送メモリ量/ネットワーク帯域幅として求め，その時間で実機上の仮想マシンが実行できる命令数を，転送時間 × ワークロード毎の Instructions Per Second (IPS) として求める．求めた命令数と同じ数だけ Bochs 上の仮想マシンで命令を実行する．ワークロード毎の IPS は，実機で動作する仮想マシン上で perf を用いて事前に計測した．

(3) マイグレーション: 転送メモリ量が 300 ミリ秒以内に転送可能なサイズとなったときに疑似マイグレーションを終了する．最大イテレーション回数は 20 回とし，それ以上かかる場合はマイグレーションが終了しないと見なす．

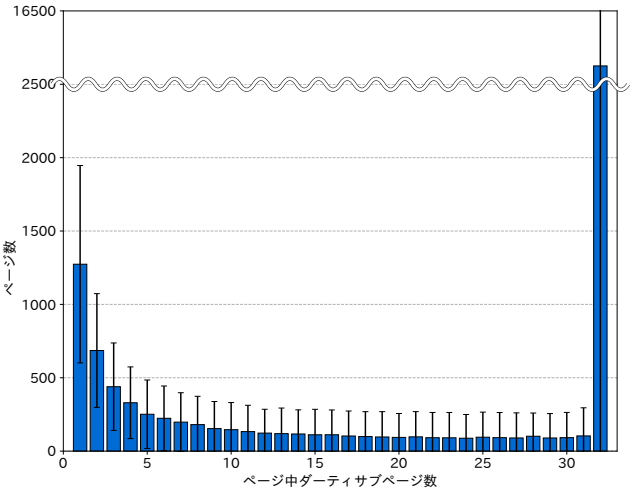

図 2　4KiB ページあたりの平均書き込みサブページ数

5.2　メモリ転送量削減

サブページ書き込み検知によるメモリ転送量の削減効果を確認するために，上記のワークロードにおいて第二イテレーション時に実行される命令数を計算して Bochs 上で再現し，その実行期間内において各 4KiB ページ内のうち書き込みがおこなわれたサブページ数の平均を求めた．図 2 に実験結果を示す．エラーバーは標準偏差を示す．

実験結果から，ワークロードによって偏りはみられるものの，平均して約 1,300 枚の 4KiB ページでは 1 枚のサブページにのみ書き込みがおこなわれており約 700 枚の 4KiB ページでは 2 枚のサブページのみ書き込みがおこなわれていた．従って，サブページ書き込み検知によって，このようなサブページだけを検知して転送することにより，転送メモリ量の削減が期待できる．

一方，4KiB ページ中の全てのサブページに書き込みがおこなわれるケースも約 16,000 枚ほどあった．このような書き込みをおこなうゲスト OS の命令を調べたところ，Linux カーネル内の `clear_page_rep()` というページのゼロクリアをおこなう関数内の `rep stosq` 命令が 9 割以上を占めるケースが多いことが分かった．

そこで，section 4.3 で述べた KVM の最適化用実装を用いて，命令ポインタから `rep stosq` 実行によるゼロクリアを検出し，予め該当する 4KiB ページの全てのサブページ書き込み保護を解除することで，ページフォルト（EPT violation）の回数を削減する最適化を実装した．以下の実験では，この実装を用いて転送メモリ量の削減をおこなっている．また，この手法のオーバーヘッドについては，section 5.4 で説明する．

5.3　総マイグレーション時間

図 3 に各ワークロードにおける疑似マイグレーションの総マイグレーション時間を示す．エラーバーは標準誤差を示す．波線より大きい値はマイグレーションが完了しな

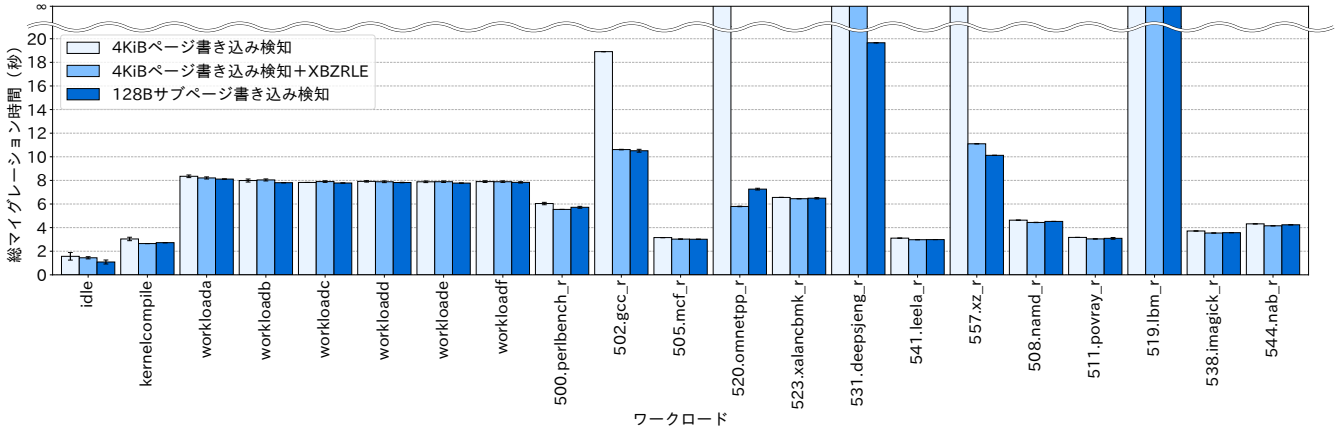

図 3 総マイグレーション時間

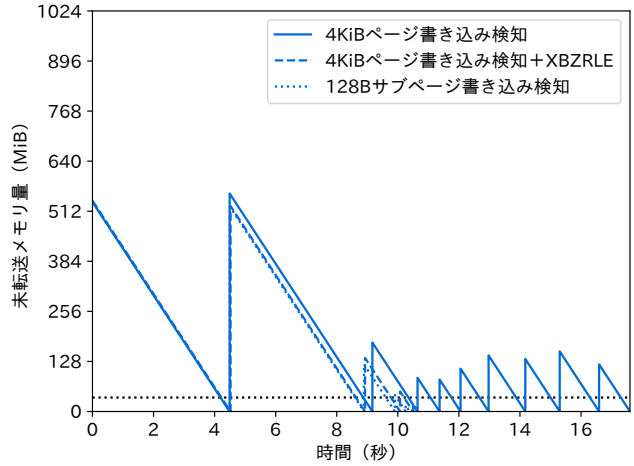

図 4 502.gcc_r の各イテレーション毎の未転送メモリ量

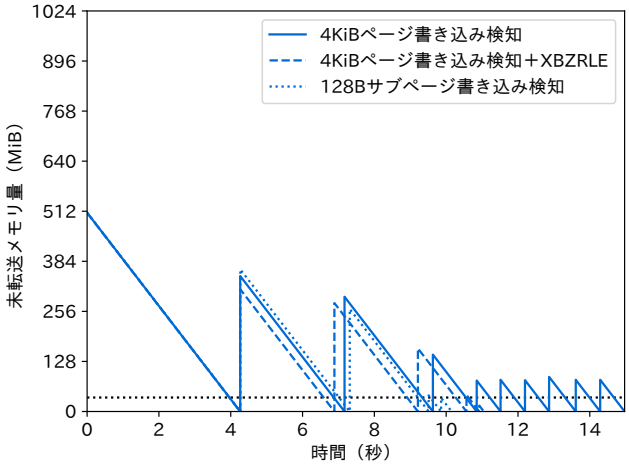

図 5 557.xz_r の各イテレーション毎の未転送メモリ量

かったことを示す．総マイグレーション時間は総転送メモリ量にほぼ比例するため，転送メモリ量の削減効果を示すと共に実際のライブマイグレーションにおける有用性も示すものとなっている．

図 3 の実験結果から，多くのワークロードにおいて 128B サブページ書き込み検知は，4KiB ページ書き込み検知＋ XBZRLE と同等のマイグレーション時間の短縮が実現できることが分かった．また，SPECCPU の 4 つのワークロード（520.omnetdp_r, 531.deepsjeng_r, 577.xz_r, 519.lbm_r）では，4KiB ページ書き込み検知はマイグレーションを完了出来なかったのに対し，XBZRLE を併用するとそのうちの 2 つ（520.omnetdp_r, 577.xz_r），128B サブページ書き込み検知を用いるとさらに 1 つ（531.deepsjeng_r）でマイグレーションを完了させられることが分かった．

典型例として，502.gcc_r というワークロードにおける，各イテレーション毎の未転送メモリ量の推移を図 4 に示す．502.gcc_r では，最初のイテレーションに 4.5 秒，次のイテレーションに 4.4 秒ほどかかっているが，その後のイテレーションでは 4KiB ページ書き込み検知では転送メモリ量の削減が十分におこなわれていないのに対し，4KiB ページ書き込み検知＋ XBZRLE や 128B サブページ書き込み検知では，差分を十分に小さくすることができるため，いずれも 5 イテレーションほどでマイグレーションを完了させられることが分かった．

また，一部のワークロードでは，128B サブページ書き込み検知は，4KiB ページ書き込み検知＋ XBZRLE よりもマイグレーション時間を短縮できることが分かった．例えば 557.xz_r というワークロードでは，図 5 に示すように，第二イテレーションでは 128B サブページ書き込み検知の方が差分が大きかったものの，その後に逆転して結果的に短い時間でマイグレーションが完了していることが分かる．これは，メモリに書き込まれた内容が XBZRLE による差分圧縮が効きにくいものであったためと考えられる．

また，128B サブページ書き込み検知だけがマイグレーションを完了させられた 531.deepsjeng_r における，各イテレーション毎の未転送メモリ量の推移を図 6 に示す．グラフから，4KiB ページ書き込み検知ではイテレーションが進んでも転送メモリ量がほとんど削減できていないことや，4KiB ページ書き込み検知＋ XBZRLE であっても

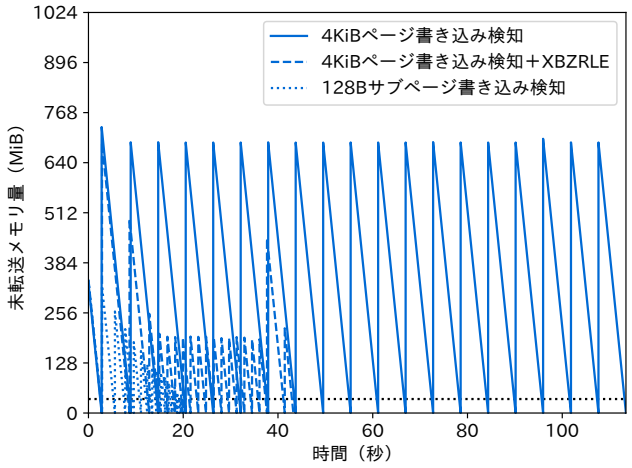

図 6 531.deepsjeng_r の各イテレーション毎の未転送メモリ量

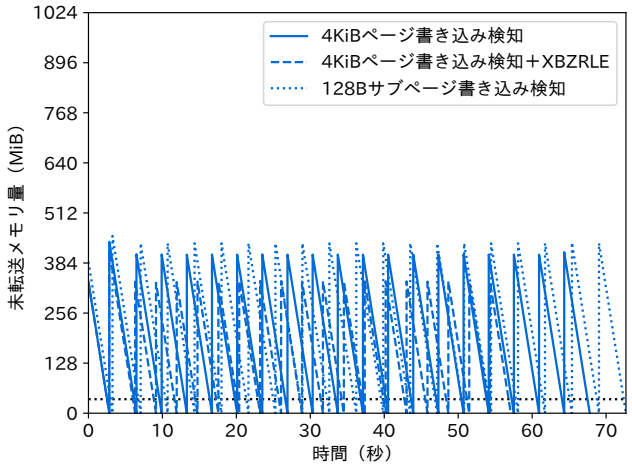

図 7 519.lbm_r の各イテレーション毎の未転送メモリ量

192MiB 前後から転送メモリ量を削減できておらず，20 回のイテレーションではマイグレーションを完了させられないことが分かった．一方，128B サブページ書き込み検知では，17 回程度でマイグレーションを完了させられることが分かった．これは，531.deepsjeng_r が非常に書き込み頻度の高いワークロードであり，差分圧縮も効きにくい値を書き込んでいてるのに対して，128B サブページ書き込み検知では，回数はかかるもののイテレーションを重ねる毎に着実に転送メモリ量を減らしていってマイグレーションを完了させられていることが分かった．

一方，128B サブページ書き込み検知を用いるとかえって転送メモリ量が増えるワークロードもあった．図 7 に，519.lbm_r の各イテレーション毎の未転送メモリ量の推移を示す．グラフから，4KiB ページ書き込み検知と比べて 4KiB ページ書き込み検知＋ XBZRLE では転送メモリ量をやや削減出来ているのに対し，128B サブページ書き込み検知では転送メモリ量がやや増えていることが分かる．これは，転送時にサブページごとにメタデータを付与したため，4KiB ページに比べ転送内容が増えたことが理由として考えられる．ただし，このワークロードでは，いずれの手法でもマイグレーションを完了させることは出来なかった．

このように，128B サブページ書き込み検知の効果はワークロードによって異なるものの，概ね 4KiB ページ書き込み検知＋ XBZRLE と同等の総マイグレーション時間削減を達成することが出来ると考えられる．

5.4 実行時オーバーヘッド

5.4.1 CPU のオーバーヘッド

各手法における CPU オーバーヘッドを見積もるために，各ワークロード毎に，1 つの 4KiB ページを転送する準備として QEMU で使用するフォーマットに整形するのに要した命令数を測定した．4KiB ページ書き込み検知の場合は，書き込みがおこなわれたページの情報を取得する ioctl の実行や，ページの値がすべてゼロであるかの判定，ページ全体を所定のフォーマットに整形する処理などにかかった命令数である．4KiB ページ書き込み検知＋ XBZRLE の場合は，それに加えてページの一時保存や圧縮にかかった命令数が含まれる．

図 8 に測定結果を示す．エラーバーは標準誤差を示す．4KiB ページ書き込み検知の場合と比べて，4KiB ページ書き込み検知＋ XBZRLE では必要な命令数が 1.33〜175.38 倍に増加した．これは，XBZRLE がバイト単位での値の比較を含む圧縮操作に多数の命令を要するためであると考えられる．128B サブページ書き込み検知も 1.22〜15.84 倍に命令数が増加しているが，これはサブページを扱うためにページ毎の処理が増加したためと考えられる．しかし，いずれも場合でも XBZRLE の場合と比べて命令数を低く抑えられている．

5.4.2 メモリのオーバーヘッド

メモリに対するオーバーヘッドを見積もるために，ライブマイグレーションの第二イテレーション開始時における，ホストマシン（Bochs）全体におけるメモリ使用量を free コマンドを用いて測定した．図 9 に測定結果を示す．エラーバーは標準誤差を示す．

測定結果から，4KiB ページ書き込み検知の場合と比べて，4KiB ページ書き込み検知＋ XBZRLE では，最大 512MiB のメモリをページの一時保存に使用するように設定しているため，ホストマシンにおけるメモリ使用量が 1.21〜1.85 倍に増加することが分かった．一方，128B サブページ書き込み検知によるメモリ使用量の増加はほとんど見られなかった．

5.4.3 サブページ書き込み保護のオーバーヘッド

また，128B サブページ書き込み検知による実行時のオーバーヘッドと，ゼロクリア命令を検知する最適化の効果を見積もるために，疑似マイグレーション中に発生した EPT violation などの SPP 関連イベントの回数と，事前に計測した 1 回のイベントハンドリングにかかる命令数をかけた

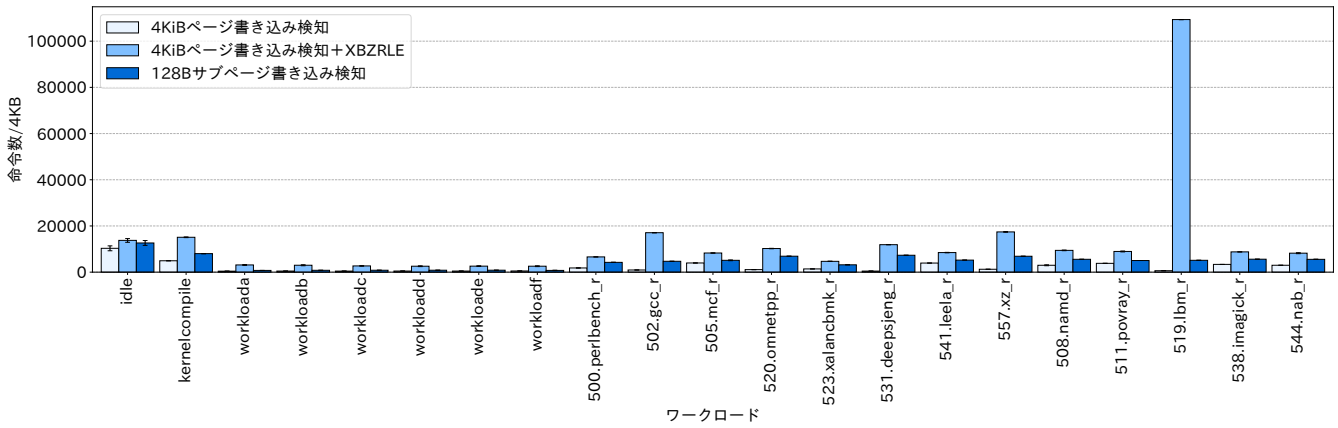

図 8 4KiB ページ転送準備にかかる命令数

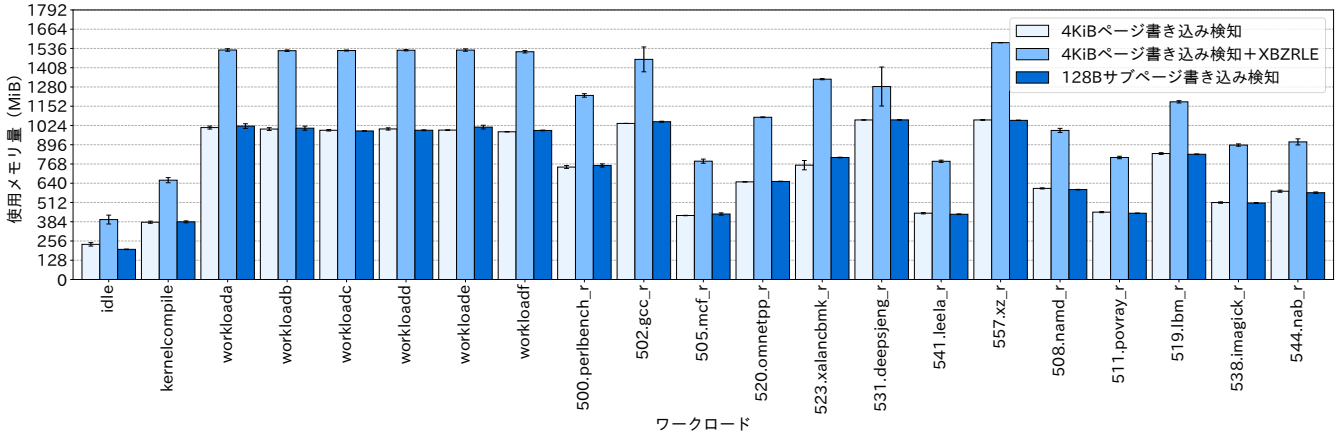

図 9 ライブマイグレーション中のメモリ使用量

ものを計測した．

図 10 に SPP がない場合と比べた相対的なオーバーヘッドを示す．エラーバーは標準誤差を示す．idle 状態の場合には，オーバーヘッドが 200.74%と大きい値を示したが，もともと負荷のかかっていない状態であるため，このオーバーヘッドはあまり問題にならないと考えられる．他のワークロードを見ると，主に 4KiB ページ書き込み保護ではマイグレーションが終了しなかったワークロードにおいて，比較的オーバーヘッドが大きくなっていることが分かる．すなわち，書き込み頻度の高いワークロードでは，SPP のイベントハンドリングのオーバーヘッドが大きくなることを示している．

また，ゼロクリア命令の検知による最適化の効果としては，単純に 128B サブページ単位で書き込み検知をおこなうとオーバーヘッドが 8.3%〜80.0%程度であったのに対し，最適化をおこなうことにより 8.3%〜54.5%程度に削減できることが分かった．従って，ゼロクリア命令の検知は一定の効果があるものの，さらなる最適化の余地もあると考えられる．

6. 関連研究

ページより細かい粒度で書き込み検知をおこなう手法としては，既に転送したページからの差分をとる手法がある．Svärd ら [4] は，転送したページの一部を一時的に保存しておき，差分圧縮により転送メモリ量を抑える手法を提案している．この手法では，差分圧縮のアルゴリズムとして XOR binary RLE (XBRLE) を用いており，ページ間で XOR を取った後にランレングス圧縮をおこなう．Shribman ら [5] は，XBRLE を改良した XBZRLE を提案しており，QEMU でも採用されている．このアルゴリズムでは，ページ間で XOR を取った後にゼロとなる領域のみをランレングス圧縮をおこなうことで圧縮効率を改善している．しかし，差分圧縮では元のページを保存しておく必要があるため，書き込みページ数が多いとメモリ消費量が増大する．また，圧縮率がメモリの値に依存するため，差分が少なくても十分に圧縮できない場合がある．

ハッシュ値を計算して書き込み検知をおこなう手法もある．Wood ら [6] は，WAN 向けのライブマイグレーションとして，1 つのページを 4 つのサブページに分割してハッシュ値を計算する手法を用いている．Deshpande ら [7]

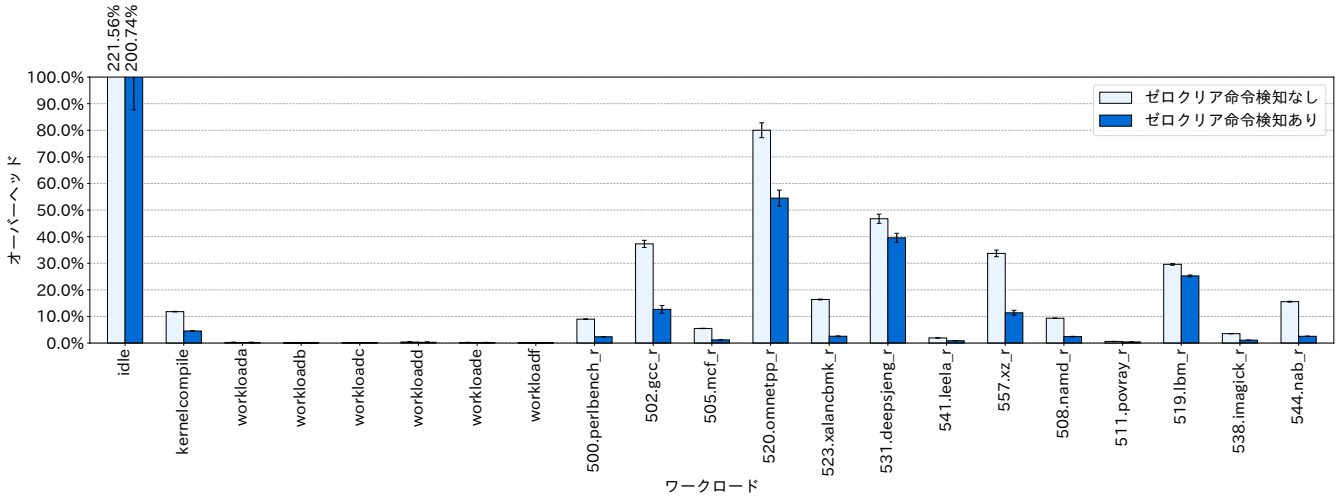

図 10 SPP によるオーバーヘッド

は，クラスタ環境で複数の仮想マシンを同時にライブマイグレーションする際に，仮想マシン間での重複排除のためにサブページのハッシュ値を計算する手法を提案している．Li ら [10] は，メモリの値を変更しない書き込み（write-not-dirty）に着目し，dirty ではあるが値は変更されていない fake-dirty ページの転送をハッシュにより回避する手法を提案している．これら手法は転送量の削減は期待できるが，サブページを小さくするほど多くのメモリを消費するほか，ハッシュ衝突に対処するためにバイト単位での比較処理も必要となる．

これらのソフトウェア的な手法は，ホストマシンに潤沢な資源があれば，並列処理によってオーバーヘッドを隠蔽することも可能であるが，現実のユースケースでは十分な資源があるとは限らない．例えば，Birke ら [28] はプライベートクラウド上で分析をおこなって，50%以上の物理マシンで実際のメモリと同程度のメモリが仮想マシンに割り当てられていることを示している．このように，仮想マシンがメモリをほぼ占有している状態でライブマイグレーションのためにさらにメモリを消費すると，仮想マシンの性能劣化につながる可能性がある

ライブマイグレーションにおける転送メモリ量の削減手法は様々なアプローチから提案されている．Jin ら [8] は，転送するページの内容に応じて効果的な圧縮アルゴリズムを選択する手法を提案している．Li ら [9] はネットワークバンド幅に応じて動的に圧縮率を変更する手法を提案している．また，QEMU/KVM にも Zlib を用いたメモリ圧縮が実装されている．しかし，メモリ圧縮は一般に圧縮率が高いほど CPU 時間を消費するほか，圧縮対象のデータや圧縮アルゴリズムによって効果が大きく変わる．メモリ圧縮はサブページ書き込み検知と併用することは可能であるが，サイズが小さくなることで圧縮率が低下することが考えられるため，複数サブページをまとめて圧縮するなどの工夫が必要である．

ページ単位でメモリ書き込みを予測し，ページの内容が再度書き換えられると予測される場合には，そのページの転送をスキップする手法も提案されている．Clark ら [2] や Lin ら [11] は，プレコピーの連続するイテレーションで両方ともダーティになったページを，その後のイテレーションでのコピー対象から除外するアルゴリズムを提案した．Hu ら [12] や Shi ら [13] は，複数イテレーションのメモリアクセスから，より複雑なアルゴリズムを用いてメモリアクセスを予測する手法を提案している．これらの手法は，サブページ書き込み検知にも応用できる可能性がある．

仮想マシンのメモリのうち，不要な領域を予め排除するアプローチも提案されている．Hines ら [14] は，メモリバルーニングを用いて動的に未使用メモリを排除する手法を提案している．Ma ら [15] は，ゲスト OS の情報をもとに実際に割り当てられている仮想マシンのページを取得し，それらのページのみ転送をおこなう手法を提案している．真島ら [29] は，Redis においてメモリアクセスの少ないデータの転送を排除する手法を提案している．これらの手法は，サブページ書き込み検知と併用できる可能性がある．

仮想マシンのワークロードを制限することにより，間接的にメモリ転送量を削減する手法も提案されている．Clark ら [2] はゲスト OS 上でページフォルトを多く発生させるプロセスを停止させることにより，ダーティページの数を削減する手法を提案している．また，Jin ら [16] は，vCPU の実行時間を削減する手法，Yiqiu ら [17] はスリープタスクをゲスト OS のタスクキューに追加する手法を提案している．また，QEMU/KVM にも，vCPU の動作を制限する手法が実装されている [18]．サブページ書き込み保護は，メモリ書き込みの頻度が上がるとページフォルトのオーバーヘッドが増加する特性があり，メモリ書き込み頻度を自動的に調整する機能を持たせられる可能性がある．

複数のマシン間でメモリを同期する古典的な手法として，Distributed Shared Memory (DSM) [30] がある．Zhang ら [31] は，複数の物理マシンで分散動作する仮想マシンを実現するために，EPT の書き込み権限による書き込み検知を用いた DSM を提案している．Zekauskas ら [32] は，コンパイラへの変更による細粒度での書き込み検知を提案し，最大で 44%のデータ転送量削減を達成した．Schoina ら [33] は，メモリインターフェースに改変を加え，メモリ書き込み命令の前にコードを挿入することで細粒度での書き込み権限を実装している．

7. まとめ

本研究では，サブページ書き込み保護機構を用いたサブページ書き込み検知のライブマイグレーションにおける有効性について評価した．サブページ書き込み保護は通常のページより小さいサブページ単位で書き込み保護の設定ができる機能であり，少ないメモリ消費でサブページ単位での書き込み検知を実現することが出来る一方，書き込み保護が設定されたサブページへのアクセスのたびにページフォルトのコストがかかる．

本研究では, サブページ書き込み保護を提供する Intel SPP を搭載したアーキテクチャを対象として，QEMU/KVM 上に Intel SPP を用いたサブページ書き込み検知と疑似マイグレーションを実現する機能を実装し，Intel SPP をエミュレート可能な Bochs 上で様々なワークロードを動作させることにより，メモリ転送量の削減による総マイグレーション時間の短縮効果や，CPU やメモリに対するオーバーヘッドを評価した．

実験の結果，サブページ書き込み保護を用いたサブページ書き込み検知は，QEMU に搭載されている 4KiB ページ書き込み検知＋ XBZRLE と同程度のメモリ転送量削減と総マイグレーション時間の短縮を実現しつつ，メモリ消費量の増加はわずかに抑えられることが分かった．また，SPECCPU の `531.deepsjeng_r` は，サブページ書き込み保護のみがマイグレーションを完了させられた一方，いずれの方式でもマイグレーションできないワークロードも存在することが分かった．

今後の課題としては，まずサブページ書き込み保護をサポートする CPU を搭載した実機マシン上で性能評価をおこなう必要がある．また，SPP によるページフォルトのオーバーヘッドをさらに削減するために，サブページ単位での書き込み予測によるページフォルトの回避などの最適化をさらに進める必要がある．さらに，複数コアをサポートした CPU では，サブページ書き込み保護の情報を仮想 CPU 間で一貫性を保つためのオーバーヘッドを考慮する必要がある．これは通常のページでも問題になるが，サブページではさらに頻度が増加することが予想されるため，そのオーバーヘッド増加に関する評価をおこなう必要がある．サブページ書き込み保護を用いて書き込み速度を調整することで，従来は完了させられなかったワークロードに対応することも考えられる．

参考文献

[1] Le, T.: A survey of live Virtual Machine migration techniques, *Computer Science Review*, Vol. 38, pp. 1–17 (online), DOI: https://doi.org/10.1016/j.cosrev.2020.100304 (2020).
[2] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *In Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation*, NSDI'05, pp. 273–286 (2005).
[3] Akiyama, S., Hirofuchi, T. and Honiden, S.: Evaluating Impact of Live Migration on Data Center Energy Saving, *In Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, CloudCom 2014, pp. 759–762 (2014).
[4] Svärd, P., Hudzia, B., Tordsson, J. and Elmroth, E.: Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines, *In Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, pp. 111–120 (2011).
[5] Shribman, A. and Hudzia, B.: Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications, *In Proceedings of the 18th International Conference on Parallel Processing Workshops*, Euro-Par 2012, pp. 539–547 (2013).
[6] Deshpande, U., Wang, X. and Gopalan, K.: Live Gang Migration of Virtual Machines, *In Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pp. 135–146 (2011).
[7] Wood, T., Ramakrishnan, K. K., Shenoy, P. and van der Merwe, J.: CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines, *SIGPLAN Not.*, Vol. 46, No. 7, pp. 121–132 (2011).
[8] Jin, H., Deng, L., Wu, S., Shi, X. and Pan, X.: Live Virtual Machine Migration with Adaptive Memory Compression, *In Proceedings of 2009 IEEE International Conference on Cluster Computing and Workshops*, CLUSTER 2009, pp. 1–10 (2009).
[9] Li, C., Feng, D., Hua, Y., Xia, W., Qin, L., Huang, Y. and Zhou, Y.: BAC: Bandwidth-Aware Compression for Efficient Live Migration of Virtual Machines, *In Proceedings of the 36th IEEE International Conference on Computer Communications*, INFOCOM 2017, pp. 1–9 (2017).
[10] Li, C., Feng, D., Hua, Y. and Qin, L.: Efficient Live Virtual Machine Migration for Memory Write-Intensive Workloads, *Future Generation Computer Systems*, Vol. 95, pp. 126–139 (2019).
[11] Lin, C., Huang, Y. and Jian, Z.: A Two-phase Iterative Pre-copy Strategy for Live Migration of Virtual Machines, *In Proceedings of the 2012 8th International Conference on Computing Technology and Information Management*, ICCM 2012, Vol. 1, pp. 29–34 (2012).
[12] Hu, B., Lei, Z., Lei, Y., Xu, D. and Li, J.: A Time-Series Based Precopy Approach for Live Migration of Virtual Machines, *In Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, ICPADS 2011, pp. 947–952 (2011).

[13] Shi, B. and Shen, H.: Memory/Disk Operation Aware Lightweight VM Live Migration Across Data-centers with Low Performance Impact, *In Proceedings of the 38th IEEE Conference on Computer Communications*, INFOCOM 2019, pp. 334–342 (2019).

[14] Hines, M. R. and Gopalan, K.: Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning, *In Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, p. 51–60 (online), DOI: 10.1145/1508293.1508301 (2009).

[15] Ma, Y., Wang, H., Dong, J., Li, Y. and Cheng, S.: ME2: Efficient Live Migration of Virtual Machine with Memory Exploration and Encoding, *In Proceedings of the 2012 IEEE International Conference on Cluster Computing*, CLUSTER 2012, pp. 610–613 (2012).

[16] Jin, H., Gao, W., Wu, S., Shi, X., Wu, X. and Zhou, F.: Optimizing the Live Migration of Virtual Machine by CPU Scheduling, *Journal of Network and Computer Applications*, Vol. 34, No. 4, pp. 1088–1096 (2011).

[17] Yiqiu, F., Chen, Z. and Junwei, G.: Improvement on Live Migration of Virtual Machine by Limiting the Activity of CPU, *In Proceedings of the 2017 International Conference on Computer Systems, Electronics and Control*, ICCSEC 2017, pp. 1420–1424 (2017).

[18] QEMU/KVM: Autoconverge Live Migration, (online), available from ⟨`https://wiki.qemu.org/Features/AutoconvergeLiveMigration`⟩ (accessed 2021-02-02).

[19] Intel: Intel 64 and IA-32 Architectures Software Developer′s Manual, (online), available from ⟨`https://software.intel.com/en-us/articles/intel-sdm`⟩ (accessed 2021-02-02).

[20] Bochs: Bochs, (online), available from ⟨`http://bochs.sourceforge.net/`⟩ (accessed 2021-02-02).

[21] Hines, M. R., Deshpande, U. and Gopalan, K.: Post-Copy Live Migration of Virtual Machines, *ACM SIGOPS Operating Systems Review*, Vol. 43, No. 3, p. 14–26 (online), DOI: 10.1145/1618525.1618528 (2009).

[22] Sahni, S. and Varma, V.: A Hybrid Approach to Live Migration of Virtual Machines, *In Proceedings of the 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pp. 1–5 (online), DOI: 10.1109/CCEM.2012.6354587 (2012).

[23] QEMU/KVM: SPP-Patch, (online), available from ⟨`https://lore.kernel.org/kvm/20200516125507.5277-1-weijiang.yang@intel.com/`⟩ (accessed 2021-02-02).

[24] Redis: Redis, (online), available from ⟨`https://redis.io/`⟩ (accessed 2021-02-02).

[25] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking Cloud Serving Systems with YCSB, *In Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pp. 143–154 (2010).

[26] Yahoo!: YCSB, (online), available from ⟨`https://github.com/brianfrankcooper/YCSB`⟩ (accessed 2021-02-02).

[27] Standard Performance Evaluation Corporation: SPEC CPU 2017, (online), available from ⟨`https://www.spec.org/cpu2017/`⟩ (accessed 2021-02-02).

[28] Birke, R., Podzimek, A., Chen, L. Y. and Smirni, E.: Virtualization in the Private Cloud: State of the Practice, *IEEE Transactions on Network and Service Management*, Vol. 13, No. 3, pp. 608–621 (2016).

[29] 真島大輝，山田浩史：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，情報処理学会研究報告，Vol. 2019-OS-147, No. 14, pp. 1–6 (2019).

[30] Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *In Proceedings of the 1988 International Conference on Parallel Processing*, ICPP 1988, pp. 94–101 (1988).

[31] Zhang, J., Ding, Z., Chen, Y., Jia, X., Yu, B., Qi, Z. and Guan, H.: GiantVM: A Type-II Hypervisor Implementing Many-to-One Virtualization, *In Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '20, p. 30–44 (2020).

[32] Zekauskas, M. J., Sawdon, W. A. and Bershad, B. N.: Software Write Detection for a Distributed Shared Memory, *In Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, p. 8–es (1994).

[33] Schoinas, I., Falsafi, B., Lebeck, A. R., Reinhardt, S. K., Larus, J. R. and Wood, D. A.: Fine-Grain Access Control for Distributed Shared Memory, *In Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, p. 297–306 (1994).