S-memV: 大容量メモリを持つVMの分割マイグレーション

末竹 将人[1]　木津 巴都希[1]　光来 健一[1]

概要：仮想マシン（VM）をサービスとしてユーザに提供するIaaS型クラウドの普及にともない，一台のサーバで多くのVMを稼働させるだけでなく，大容量メモリを持つVMも提供されるようになってきた．一方で，VMのマイグレーションを行うには移送先のホストに十分な空きメモリ容量が必要となり，大容量メモリを持つVMはマイグレーションを行うのが困難になるという問題がある．マイグレーションのために大容量のメモリを備えたホストを確保しておくのはコストの面から難しいことが多いためである．本稿では，大容量メモリを持つVMを複数のホストに分割してマイグレーションすることを可能とするシステム *S-memV* を提案する．S-memVはVMの核となる情報とVMがアクセスすると予測されるメモリを移送先のメインホストに送り，メインホストに入りきらないメモリはサブホストに送る．VMがサブホストにあるメモリを必要とした時には，メインホストとサブホストの間で時間的，空間的局所性を利用したページングを行う．我々はS-memVをKVMに実装し，仮想メモリを用いた従来手法よりもマイグレーション時間とダウンタイムを短縮し，マイグレーション後の性能低下を抑えることができることを示した．

1. はじめに

クラウドコンピューティングのサービス形態の一つであるIaaS型クラウドでは，仮想マシン(VM)をサービスとしてユーザに提供する．IaaS型クラウドの普及にともない，1台のサーバに多くのVMを統合するだけでなく，大容量メモリを持つVMも提供されるようになってきた．例えば，Amazon EC2のX1インスタンスでは2TBのメモリが提供されている．このような大容量メモリを持つVMはApache Spark [1]やFacebook Presto [2]などを用いたビッグデータ解析に用いられる．できるだけメモリ上にデータを保持することで，ビックデータをより高速に解析することができる．また，メモリ上に大量のデータを保持するSAP HANA [3]やMicrosoft SQL Server [4]などの高速なインメモリ・データベースを用いることもできる．

VMはホストをメンテナンスする際などにマイグレーションにより他のホストに移動されることがあるが，大容量メモリを持つVMのマイグレーションには二つの問題がある．一つ目の問題はマイグレーション時間である．マイグレーション時間は基本的にVMのメモリサイズに比例して長くなる．この問題は，VMのメモリを並列に送る手法[5]や，高速なネットワークを用いる[6]ことで解決することができる．もう一つの問題は，VMのマイグレーションには移送先ホストに十分な空きメモリが必要になるということである．マイグレーションのために大容量メモリを備えた空きホストを常に確保しておくのは，可能だとしてもコストの大幅な増大を招く．VMのマイグレーションが行えなければ，ホストのメンテナンス中はVMを停止させなければならなくなり，VMのメモリ上の大量のデータを失うことになる．

一方で，クラウド上には小さな空きメモリを持ったホストが多く存在している．これらの空きメモリの合計は大容量メモリを持つVMのメモリサイズを上回る場合が多い．このような断片的なメモリを一つに統合するためにリモートページング[7], [8], [9]を用いた仮想メモリが提案されてきた．従来の仮想メモリは物理メモリに入りきらないメモリをディスクにページアウトすることで仮想的に大容量メモリを利用することを可能にする．リモートページングはローカルディスクの代わりにネットワーク上のホストの空きメモリを利用する．しかし，仮想メモリはマイグレーションとの相性が悪く，マイグレーション中やマイグレーション後に大量のページングを発生させる場合がある．その結果，マイグレーション性能やVMの実行性能が大幅に低下し，最悪の場合，スラッシングのせいでVMのマイグレーションが終了しないことにもなる．

そこで，本稿では，上記の問題を解決するために大容量メモリを持つVMを複数のホストに分割してマイグレーションすることを可能とするシステム *S-memV* を提案する．S-memVでは，マイグレーション時のVMの移送先のホストは1台とは限らず，1台の**メインホスト**と0台以上

[1] 九州工業大学
Kyushu Institute of Technology

の**サブホスト**からなる．S-memV は CPU やデバイスの状態のような VM の核となる情報を移送先のメインホストに送る．また，VM がマイグレーション後にアクセスすると予測されるメモリもできる限りメインホストに送る．一方，メインホストのローカルメモリに入りきらないメモリはサブホストのいずれかに送る．S-memV では移送先のメインホストまたはサブホストに直接メモリを送るため，マイグレーション中にはページングが発生しない．マイグレーション後はメインホストで VM を動作させ，VM がサブホストにあるメモリを必要とした場合には，メインホストとサブホストの間でリモートページングを行う．S-memV では，VM のメモリを分割する際に参照の時間的局所性を考慮するため，マイグレーション後の VM の性能低下を抑えることができる．

我々は S-memV を KVM に実装し，VM の分割マイグレーションを実現した．サブホスト上ではメモリサーバが動作し，メインホストで動作する VM のメモリの一部を管理する．拡張ページテーブルを用いて VM のメモリ参照情報を取得する機構を開発し，参照の時間的局所性を利用する LRU 近似アルゴリズムを実装した．ページングには Linux 4.3 で導入された userfaultfd 機構を用い，ページアウトのための拡張を行った．実験結果より，S-memV は仮想メモリを用いた従来手法よりもマイグレーション時間とダウンタイムを大幅に短縮できることが分かった．また，分割マイグレーション後の VM の性能低下を抑えられることも分かった．

以下，2 章で大容量メモリを持つ VM をマイグレーションする際の問題点について述べ，3 章で VM の分割マイグレーションを行う S-memV を提案する．4 章で S-memV の実装について説明し，5 章で S-memV を用いて行った実験について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. 大容量メモリを持つ VM のマイグレーション

VM マイグレーションは稼働している VM を停止させることなく別のホストへ移動させる技術である．マイグレーションを活用することで，VM が提供しているサービスを停止させることなくホストのメンテナンスを行うことができる．マイグレーションを行う際には，移送先ホストに新しい VM を作成し，移送元ホストで動いている VM のメモリの内容をネットワーク経由で移送先ホストの VM のメモリにコピーする．この間，移送元ホスト上で VM のメモリの内容は更新され続けているため，再度，更新されたメモリを転送する．これを繰り返して転送するメモリ量が十分小さくなったら移送元ホストの VM を停止し，VM の CPU やデバイスの状態および更新されたメモリを転送してマイグレーションを完了する．

近年，大容量メモリを持つ VM が使用されるようになってきた．例えば，Amazon EC2 では 244GB のメモリを持つ 8xlarge インスタンスが提供されている．最近では，新しく 2TB のメモリを持つ x1.32xlarge インスタンスも提供されるようになった．このような大容量メモリを持つ VM はビッグデータの解析 [1], [2] やインメモリ・データベース [3], [4] などのように，大量のデータを高速に扱う場合に用いられている．しかし，このような大容量メモリを持つ VM は，マイグレーション時に移送先を見つけることが困難になるという問題がある．移送先として大きな空きメモリ容量を持ったホストを確保し続けておくことは，コスト面からも難しいことが多い．十分なメモリ容量を持つホストが多数の小さな VM を動かすために使われていた場合，まず，それらの VM をマイグレーションして必要な空きメモリ容量を確保する必要がある．

2.1 仮想メモリを利用したマイグレーション

従来，移送先ホストに十分な空きメモリ容量がない場合には仮想メモリが用いられてきた．仮想メモリは物理メモリに入りきらないメモリをディスク上のスワップ領域に待避することで，物理メモリよりも大きなメモリを扱うことができる技術である．しかし，仮想メモリは大容量メモリを持つ VM のマイグレーションとは相性が悪い．マイグレーションの初期段階では VM のメモリが先頭から順番に転送されるため，移送先ホストの物理メモリに入りきらなくなった場合，VM のメモリはアクセスパターンに関わらず LRU に基づいて無条件にアドレスの小さいものから順にページアウトされてしまう．その結果，マイグレーション後に必要なメモリがページアウトされていると VM の性能低下を引き起こす．

メモリの再送時にもページングは頻繁に起こる．移送元ホストで書き換えられたために再送されたメモリが移送先ホストの物理メモリ上にない場合，そのメモリはディスクからページインされてから上書きされる．大容量メモリを持つ VM のマイグレーションでは最初のメモリ全体の転送に時間がかかるため，その間に書き換えられるメモリ量も増大する．マイグレーション完了時には，頻繁に書き換えられるページは物理メモリ上にあるが，読み込みしか行われないページは再送されないためページアウトされている可能性が高くなる．そのため，マイグレーション後にはディスクとの間で頻繁にページングが起こる．メモリに比べてディスクは非常に遅いため，ページングを繰り返すと性能が著しく低下する．HDD の代わりに SSD を用いることで性能低下を抑えられるが，SSD でも依然としてメモリよりも 1 〜2 桁低速である．

ディスクを用いたページングのオーバヘッドを削減するためにリモートページング [7], [8], [9] が提案されてきた．リモートページングはディスクの代わりにネットワーク上

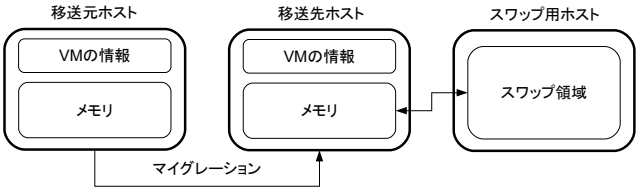

図 1 リモートページングを利用したマイグレーション

の別のホスト上のメモリとの間でページングを行う．ネットワークが十分高速であれば，リモートページングはディスクを用いたページングよりも高速である．しかし，リモートページングもマイグレーションとは相性が悪い．マイグレーションにより VM のメモリ全体が移送先ホストに転送されている間に，移送先ホストは図 1 のように，ページアウトされたメモリをスワップ用ホストに転送しなければならない．そのため，移送先ホストとスワップ用ホスト間のネットワーク帯域を消費してしまう．加えて，リモートページングによって移送先ホストのシステム負荷が増加し，マイグレーション性能にも影響を及ぼす．

2.2 ポストコピー・マイグレーション

上記のプレコピー・マイグレーションとは異なり，ポストコピー・マイグレーション [10] を用いるとページングの頻度を減らすことができる．ポストコピー・マイグレーションは，VM の実行に必要な核となる情報のみを最初に転送し，すぐに移送先ホストの VM に切り替える．VM のメモリを転送するのにはオンデマンド転送が用いられる．オンデマンド転送では，VM がメモリを必要とした時に移送元ホストからの転送を行う．移送先ホストが選択的にメモリをページアウトすることができるため，プレコピー・マイグレーションのような強制的なページングは起こらない．ただし，オンデマンド転送だけでは VM のメモリアクセスの遅延が大きくなるため，バックグラウンド転送と組み合わせて用いられる．バックグラウンド転送では，メモリアクセスがない時に裏で転送が行われる．プレコピー・マイグレーションと同様に，移送先ホストでアクセスされるメモリがページアウトされてしまう可能性が高くなる．

3. S-memV

本稿では，大容量メモリを持つ VM を分割して複数のホストにマイグレーションすることを可能とする S-memV を提案する．

3.1 分割マイグレーション

S-memV では，1 つのホストから複数のホストへのマイグレーションが可能である．移送先の複数のホストは図 2 のように 1 台のメインホストと 0 台以上のサブホストからなる．VM を動かす上で必要となる CPU やデバイスの状態などの核となる情報はメインホストに送る．また，VM がアクセスすると予測されるメモリはできる限りメインホストに送り，移送先ホストの VM がオーバヘッドなしでアクセスできるようにする．一方，メインホストに入りきらない VM のメモリはサブホストのいずれかに転送する．この際に，空きメモリを考慮して最適なメインホストとサブホスト群を選択できるようにするために，全ホストの空きメモリを管理するサーバを用意する．移送元ホストはこのサーバにアクセスすることにより，マイグレーション先のホスト群を決定する．また，マイグレーション時に複数のホストにメモリを並列に転送することでマイグレーションの高速化を図ることができる．

マイグレーション後はメインホスト上で VM を動作させる．アクセスされたメモリがメインホスト上にない場合は，サブホスト上にある要求されたメモリをメインホストに転送し，ページイン処理を行う．同時に，メインホスト上の今後アクセスされないことが予測されるメモリのページアウト処理を行い，ページインを行ったサブホストに転送する．この挙動はリモートページングと同様である．しかし，S-memV はマイグレーション中にページングを発生させない点が異なる．メインホストに入りきらないメモリはメインホスト経由でサブホストにページアウトされるのではなく，直接，サブホストに転送される．そのため，マイグレーションの最初のメモリ転送時も再送時も，メインホストとサブホスト間で無駄なネットワーク転送が行われることはない．また，VM がアクセスすることが予測されるメモリはメインホストに転送されているため，マイグレーション後のページング頻度は低くなることが期待できる．

S-memV はプレコピー・マイグレーションだけでなく，ポストコピー・マイグレーションにも適用することができる．オンデマンド転送の場合は，VM が要求したメモリを移送元ホストから移送先メインホストに転送する．メインホストの物理メモリに空きがない場合は，VM がアクセスしないと予測されるメモリをサブホストにページアウトする．バックグラウンド転送の場合は，プレコピー・マイグレーションと同様に，VM がアクセスすると予測されるメモリはメインホストに送り，メインホストに入りきらないメモリはサブホストに転送する．いずれの場合でも，移送

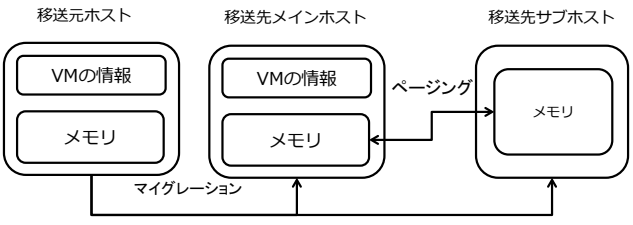

図 2 分割マイグレーション

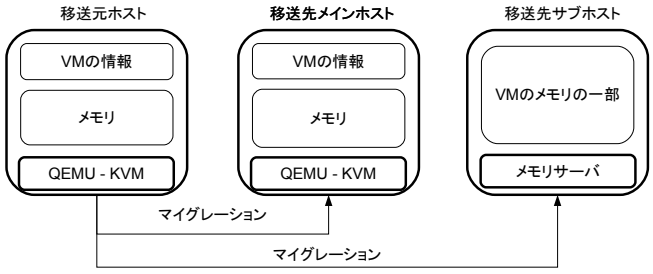

図 3 S-memV のシステム構成

先ホストに転送が完了したメモリについては必要に応じてメインホストとサブホストの間でページングを行う．

3.2 複数ホストにまたがる VM のマイグレーション

複数のホストに分割マイグレーションした VM は，ホストのメンテナンス終了後や十分な空きメモリ容量を持ったホストが用意できた時などに，再び 1 台のホストで動作するようにマイグレーションすることができる．その際に，移送元のメインホストとサブホスト上の VM のメモリを並列に転送することにより，高速にマイグレーションを行うことができる．マイグレーション中にページインまたはページアウトされたメモリについては，転送漏れや重複がないように転送する．

VM が動作している複数のホスト全体または一部をメンテナンスできるようにするために，複数のホストにまたがる VM の全体または一部を別のホスト群にマイグレーションすることもできる．メインホスト上にある VM の本体をマイグレーションする際には，移送先のメインホストに VM の核となる情報や VM がアクセスすると予測されるメモリを転送し，入りきらないメモリは既存のサブホストまたは新しく確保したサブホストに転送する．サブホスト上にある VM の一部をマイグレーションする際には，移送先のサブホストに VM のメモリの一部を転送する．マイグレーション中にページングが発生した場合には，必要に応じて追加で転送したり，転送済みのメモリを削除する．

4. 実装

S-memV を QEMU-KVM 2.4.1 および Linux 4.3 に実装した．現在のところ，プレコピー・マイグレーションを用いた分割マイグレーションをサポートしている．S-memV のシステム構成を図 3 に示す．移送元ホストと移送先メインホストでは S-memV を実装した QEMU-KVM が動作し，マイグレーション前後で VM を動かす．移送先サブホストでは VM のメモリの一部を管理するメモリサーバを動作させる．

4.1 分割マイグレーション

VM のメモリを分割してマイグレーションできるようにするために，QEMU-KVM のマイグレーション機構の拡張を行った．S-memV ではマイグレーションを開始した時に，移送先のメインホストで起動した QEMU-KVM だけでなく，サブホストで動作しているメモリサーバにもネットワーク接続を行う．次に，移送先メインホストとサブホストの空きメモリに応じて VM のメモリを分割する．メモリの分割は LRU に基づいて行い，マイグレーション後にアクセスされる可能性が高いメモリページから順にメインホストに，残りをサブホストに割り当てる．現在の実装ではマイグレーション開始時までのメモリ参照履歴に基づいて VM のメモリを分割するため，マイグレーション中にメモリアクセスは考慮していない．LRU に基づくメモリ分割については 4.4 節で詳しく述べる．

VM のメモリが分割できたら，それに応じてメモリページをメインホストまたはサブホストに転送する．メインホストには従来と同様にメモリブロックのオフセットやメモリページの内容を送るのに加え，移送元でのメモリ参照履歴の情報を送る．この付加情報により，移送先でも VM のメモリ参照履歴を引き継ぐことができる．一方，サブホストには VM の物理メモリアドレスとメモリページの内容を送信する．メモリブロックのオフセットではなく物理メモリアドレスを送るのは，サブホストではメモリブロックを管理していないためである．サブホストのメモリサーバでの処理については 4.2 節で述べる．サブホストに送ったページについては，メインホストにもサブホストの IP アドレスとメモリアドレスの情報を送る．メインホストでは各メモリページがどのホストにあるかを配列を用いて管理し，マイグレーション後のページングに利用する．

マイグレーションの終了時には，メインホストからサブホスト上のメモリサーバにネットワーク接続を行い，サブホストとの間でページングが行えるようにする．VM がメインホスト上にないページにアクセスした時にページングを行えるようにするために，Linux 4.3 で導入された userfaultfd 機構を用いる．リモートページングの実装については 4.5 節で詳しく述べる．

4.2 メモリサーバ

メモリサーバは VM のメモリの一部を管理するサーバであり，サブホスト上で動作する．メモリサーバはページ番号をインデックスとする配列（ページ配列）を用いて VM のメモリを 4KB のページ単位で管理する．メモリサーバが VM の物理メモリアドレスとメモリページの内容からなるページアウト要求を受信した時には，4KB のメモリを確保して送られてきたメモリページの内容をコピーし，メモリアドレスに基づいてページ配列に登録する．一方，VM の物理メモリアドレスからなるページイン要求を受信した時には，メモリアドレスを基にページ配列を探索し，VM のメモリページが見つかればそのデータを要求元に送信す

る．同時にそのデータをページ配列から削除する．

4.3 メモリ参照履歴の管理

S-memV では VM のメモリ参照履歴を管理するために，VM の拡張ページテーブル（EPT）をたどって各ページのメモリ参照に関する情報を取得する．そのために，QEMU-KVM は定期的に Linux カーネル内の KVM に対して ioctl システムコールを発行する．現在の実装では，1 秒おきに KVM を呼び出している．その際に VM のメモリサイズに応じたビットマップを確保し，ioctl の引数として KVM に渡す．KVM は VM のすべてのメモリページについて EPT をたどり，ページテーブルエントリ（PTE）を取得する．PTE のアクセスビットは対応するページにアクセスした時に 1 にセットされるため，アクセスビットの値を渡されたビットマップに記録する．その後，次の期間のアクセスを記録できるようにするために，アクセスビットをクリアする．

S-memV は LRU 近似アルゴリズムとして，クロックアルゴリズムとエージングアルゴリズムをサポートしている．クロックアルゴリズムでは各ページについて 1 ビットのメモリ参照履歴が必要になるため，定期的に EPT をたどって取得したアクセスビットの値をメモリ参照ビットマップに蓄積していく．具体的には，図 5 のように取得したアクセスビットが 1 の場合にはメモリ参照ビットマップのビットを 1 にし，0 の場合にはメモリ参照ビットマップの更新を行わない．このように定期的にメモリ参照ビットマップを更新するのは，ページングの際にメモリ参照情報を取得するオーバヘッドを減らしつつ，できるだけ最新のメモリ参照情報を利用できるようにするためである．メモリ参照ビットマップはクロックアルゴリズムを実行することによっても更新される．クロックアルゴリズムの実装については 4.5 節で述べる．

一方，エージングアルゴリズムでは各ページに対してある程度の過去までのメモリ参照履歴が必要になる．現在の実装では各ページに 8 ビットを割り当てて参照履歴を管理している．定期的に，EPT をたどって取得したアクセスビットの値はしばらくの間，この 8 ビットの最上位ビットに蓄積していく．これは 1 秒間のメモリ参照を 1 ビットに

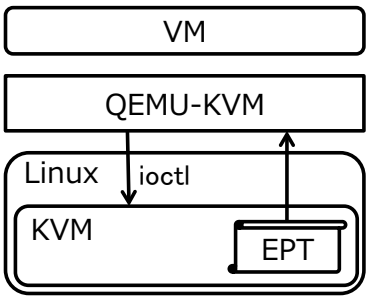

図 4 メモリ参照履歴の取得

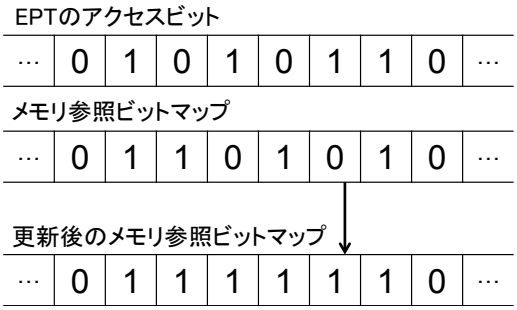

図 5 クロックアルゴリズム用のメモリ参照履歴の更新

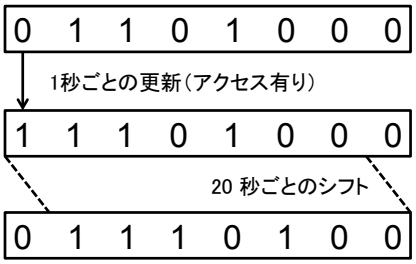

図 6 エージングアルゴリズム用のメモリ参照履歴の更新

対応させると 8 秒間の履歴しか利用できないためである．この参照履歴は定期的に 1 ビット右にシフトし，再び最上位ビットにアクセスビットを蓄積していく．このようにすることで，より最近にアクセスされたページほどメモリ参照履歴の値が大きくなる．エージングアルゴリズムの詳細についても 4.5 節で述べる．

4.4 LRU に基づくメモリ分割

S-memV では，マイグレーションを行う際にメモリ参照履歴に基づいて VM のメモリを分割する．このメモリ分割は連続するメモリページをチャンクとよばれるかたまりに分け，チャンク単位で行う．これはリモートページングがチャンク単位で行われるためである．チャンクのサイズは 1 以上の 2 のべき乗とする．メモリ分割の方法は用いる LRU 近似アルゴリズムによって異なる．

クロックアルゴリズムを用いる場合，チャンクごとにメモリ参照ビットマップの対応するビットの値の合計を計算し，その値が大きいチャンクに含まれるメモリページから順にメインホストに割り当てる．ビットの合計値を用いるのは，アクセスされたページ数が多いチャンクを優先するためである．チャンクサイズが 1 の場合には，メモリ参照ビットマップのビットが 1 のページをメインホストに割り当てる．ビットが 1 のすべてのページを割り当ててもメインホストに空きがあれば，ビットが 0 のページもメインホストに割り当てる．メインホストの空きがなくなったら残りのページはサブホストに割り当てる．

エージングアルゴリズムを用いる場合，それぞれのチャンクのページの中でメモリ参照履歴の 8 ビット値が最大のものを見つけ，その値がより大きいチャンクに含まれる

ページから順にメインホストに割り当てる．メモリ参照履歴の最大値を用いるのは，チャンクの中の最も最近アクセスされたページを重視するためである．

4.5 リモートページング

マイグレーション後のリモートページングは Linux の userfaultfd 機構を用いて実装した．リモートページングのシステム構成を図 7 に示す．S-memV はまず，userfaultfd 機構に VM のメモリを登録する．VM のメモリは匿名メモリマッピングを用いて確保されており，マイグレーション時に受信したデータを書き込んだページだけに実メモリが割り当てられる．それ以外のページについては実メモリは割り当てられていない．Transparent Huge Pages が有効になっていると 2MB 単位で実メモリが割り当てられてしまい，チャンク単位でのページングが行えないため，この機能は無効にした．VM がまだ実メモリが割り当てられていないページにアクセスすると，ページフォールトが発生し，userfaultfd 機構によって QEMU-KVM にイベントが通知される．

イベントを受け取った QEMU-KVM はページフォールトが発生した QEMU-KVM 上のメモリアドレスを VM の物理メモリアドレスに変換し，サブホストのメモリサーバにページイン要求を送る．その際に，まずそのメモリアドレスを含むページについてページイン要求を送り，続けてそのページを含むチャンクの残りのすべてのページについてページイン要求を送る．このような順番で要求を送ることで，ページフォールトが発生したページを最初に処理して VM の実行を再開できるようにしている．メモリサーバがその要求を受け取ると，渡されたメモリアドレスに対応するページのデータを QEMU-KVM に返送する．ページのデータを受け取った QEMU-KVM は userfaultfd 機構を用いてデータを該当メモリに書き込むことでページイン処理が完了する．同時に，ページの所在を管理するページ配列を更新して，ページインしたメモリをメインホストで管理する．

ページインにともなってメモリ量のバランスを取るために，メインホストからサブホストにメモリをページアウトする．まず，S-memV は 4.3 節で述べた VM のメモリ参照履歴に基づいて今後アクセスされないことが予測されるメモリチャンクを選択する．次に，ページアウトするページの内容を取得し，そのページへの実メモリの割り当てを QEMU-KVM から削除する．この操作をアトミックに行うために userfaultfd 機構を拡張し，対応するページテーブルエントリをクリアしてそのページの内容を返せるようにした．次に，選択されたチャンクに含まれるページに対してメモリサーバにページアウト要求を送る．それに加えて，ページ配列を更新してページアウトしたページをメインホストで管理しないようにする．

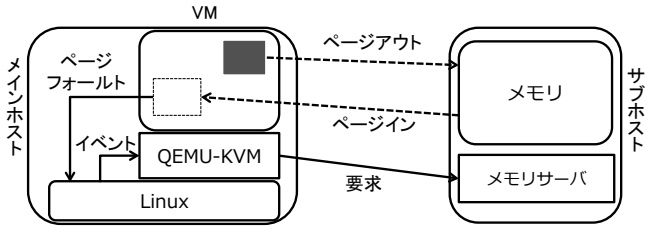

図 7 リモートページングのシステム構成

ページアウトするチャンクは LRU に基づいて決定する．クロックアルゴリズムを用いる場合には，チャンクごとにメモリ参照ビットマップの対応するビットの合計値を求めて，その合計値が最小のチャンクを選ぶ．これにより，アクセスされていないページが多いチャンクを優先して選ぶことができる．エージングアルゴリズムを用いる場合には，チャンクに含まれるページのメモリ参照履歴の中から最大値を見つけ，その値が最小のチャンクを選ぶ．このようにすることで，チャンクの中で最も最近アクセスされたページを重視することができる．

5. 実験

S-memV の有効性を示すために，マイグレーション性能やマイグレーション後の VM の性能などを調べる実験を行った．比較のために，移送先ホストに十分なメモリがある場合と仮想メモリを用いた場合についても調べた．実験には，Intel Xeon E3-1270v3 3.5GHz の CPU，16GB のメモリ，230GB の HDD を搭載したマシン 2 台をそれぞれ移送元ホストおよび移送先メインホストとした．S-memV を用いる場合は移送先メインホストの空きメモリを 6GB と仮定し，仮想メモリを用いる場合には移送先ホストの空きメモリが 6GB になるように OS の起動時にメモリ量を制限した．移送先サブホストには，Intel Xeon E3-1270v2 3.5GHz の CPU，12GB のメモリ，345GB の HDD を搭載したマシンを用いた．仮想メモリを用いる場合には，移送先ホストで Crucial MX300 SSD にスワップ領域を設定した．これらのマシンは 10 ギガビットイーサネットで接続した．ホスト OS には Linux 4.3.0 を用い，仮想化ソフトウェアには QEMU-KVM 2.4.1 を使用した．VM には仮想 CPU を 1 つ，メモリを 12GB 割り当てた．VM のメモリには起動時にランダムな値を書き込むことで，実メモリが割り当てられるようにした．また，ページ内のデータがすべて 0 の時の最適化は無効にした．

5.1 マイグレーション性能

S-memV のマイグレーション性能を調べるために，マイグレーション時間とダウンタイムを測定した．マイグレーション時間は図 8 のようになり，メモリが十分にある場合と比べて，メモリ不足によりスワップ領域が使われた場合には 2.2 倍の時間がかかった．それに対して，S-memV で

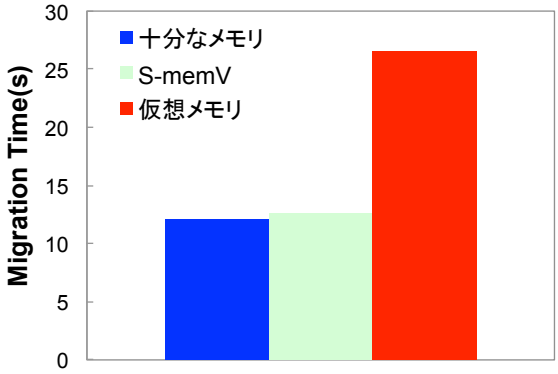

図 8 マイグレーション時間

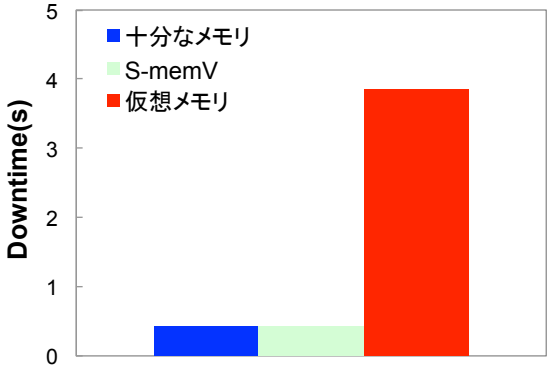

図 9 ダウンタイム

は 5 % の性能低下であった．ダウンタイムは図 9 のようになり，仮想メモリを用いた場合にはメモリが十分にある場合より 3.4 秒増加し，9.3 倍になった．この時間のほとんどは仮想デバイスの復元に費やされており，復元時にアクセスするメモリがページアウトされているためだと考えられる．それに対して，S-memV におけるダウンタイムの増加は 7 ミリ秒であった．この結果より，S-memV は仮想メモリを用いる場合よりもマイグレーション性能の低下を大幅に抑えることができることがわかった．

仮想メモリを用いる場合について，HDD と SSD でのマイグレーション性能の差を調べた．移送先ホストのスワップ領域として HDD, Intel X25-E SSD（読み込み 250MB/s，書き込み 170MB/s），Crucial MX300 SSD（読み込み 530MB/s，書き込み 500MB/s）を用いた．実験結果を図 12 と図 13 に示す．X25-E を用いた場合のマイグレーション時間は，MX300 を用いた場合の 2.9 倍になった．HDD を用いた場合には 5.3 倍となった．メモリが十分にある場合と比べると，それぞれ 6.5 倍，11.7 倍の時間がかかった．一方，X25-E を用いた場合のダウンタイムは，MX300 を用いた場合より 2.3 秒増加し，HDD を用いた場合には 26.6 秒増加した．

次に，VM 内で 6GB のメモリを書き換え続けるプログラムを動作させ，大量のメモリが再送される状態でマイグレーションを行った．実験結果を図 10 と図 11 に示す．メモリが十分にある場合でも，上の実験と比べてマイグレー

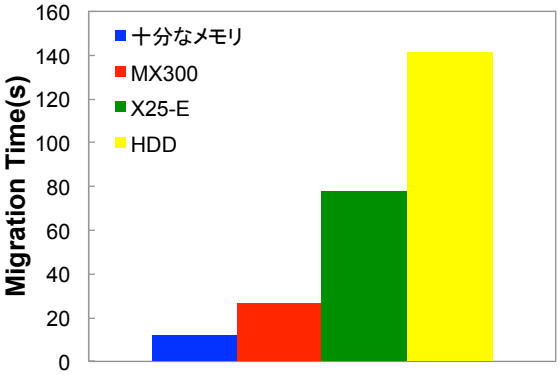

図 10 ディスクによるマイグレーション時間の違い

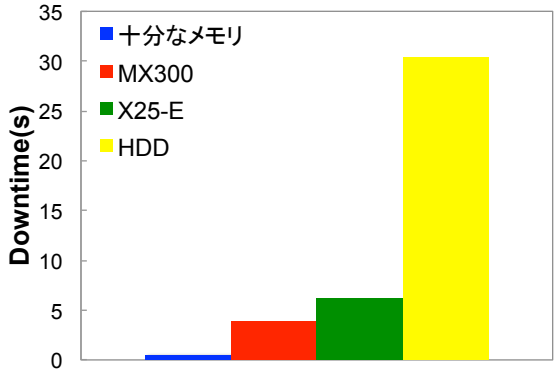

図 11 ディスクによるダウンタイムの違い

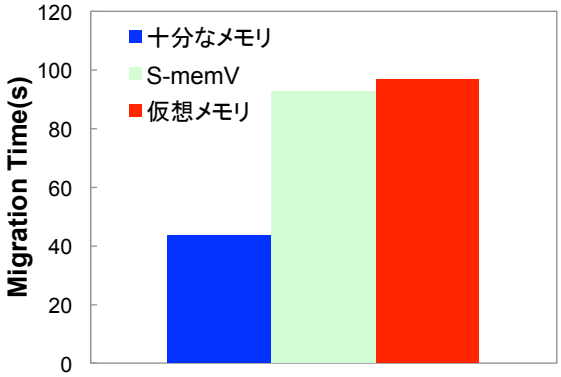

図 12 マイグレーション時間 (メモリ負荷時)

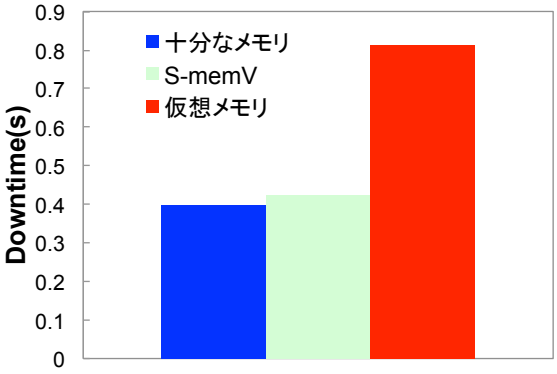

図 13 ダウンタイム (メモリ負荷時)

ション時間が 3.6 倍に増加した．それと比べて，仮想メモリを用いた場合にはマイグレーション時間がさらに 2.2 倍になった．S-memV では 2.1 倍となり，仮想メモリを用いた場合とほぼ同じマイグレーション時間がかかった．一方，ダウンタイムは，メモリが十分にある場合と比べて，仮想メモリを用いた場合には 414 ミリ秒増加した．それに対して，S-memV での増加は 27 ミリ秒であった．

5.2 マイグレーション後の VM の性能

LRU に基づいたメモリ分割およびリモートページングの効果を調べるために，マイグレーション後の VM の性能を測定した．まず，VM 内で 2GB のデータに対して GNU sort を 60 秒間実行して一時停止させた．次に，VM のマイグレーションを行って GNU sort を再開し，終了するまでにかかる時間を測定した．S-memV では上の実験と同様に，クロックアルゴリズムとエージングアルゴリズムを用いた．実験結果を図 14 に示す．

メモリが十分にある場合と比べて，仮想メモリを使用した場合の性能低下は 35 %であった．これに対して，S-memV でクロックアルゴリズムとエージングアルゴリズムを用いた場合，性能低下をそれぞれ 15 % と 16 % に抑えることができた．アルゴリズムによる差が小さいのは，sort のメモリ参照の局所性を考慮するのに，1 ビット分の参照履歴があれば十分であったためと考えられる．

sort を実行中に発生したページインの回数を図 15 に示す．仮想メモリを用いた場合には約 4,000 回のページインが発生したのに対し，S-memV では約半分となった．これは LRU に基づくメモリ分割によって，sort がアクセスするメモリがより多くメインホストに転送されたためと考えられる．また，Linux の仮想メモリでは 8 ページをまとめてページインするのに対し，S-memV では 256 ページをまとめてページインしたことも，ページイン回数が減った理由と考えられる．一方，エージングアルゴリズムを用いた方がページング回数を 200 回減らせており，より精度の高いメモリ参照履歴が利用できていると考えられる．

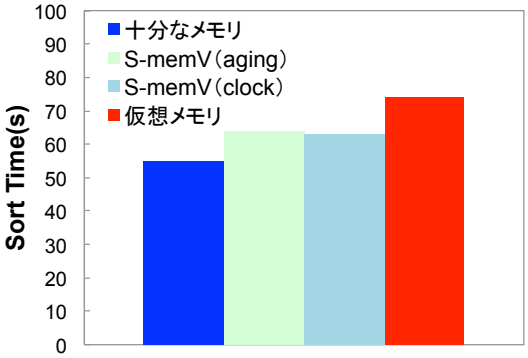

図 14 sort の実行時間

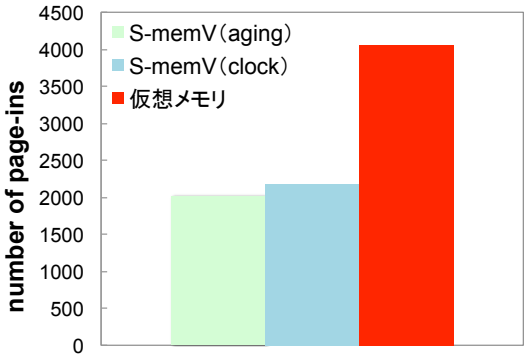

図 15 sort 実行中のページイン回数

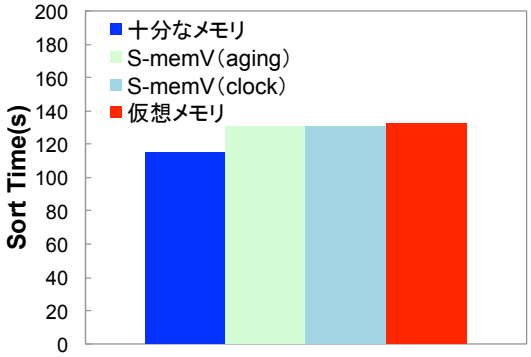

図 16 マイグレーション後に開始した sort の実行時間

ここで，メモリ分割とリモートページングのどちらに LRU を用いた効果があったのかを調べるために，マイグレーション後に sort の実行を開始して性能を測定した．この場合，sort のメモリ参照を考慮せずにメモリ分割が行われるため，LRU に基づくリモートページングの効果だけを確認することができる．図 16 に示すように，S-memV における性能低下は仮想メモリを用いた場合と同程度であった．つまり，LRU に基づくリモートページングの性能は SSD を用いたページング性能と同等であるということである．これより，S-memV がマイグレーション時に行っている LRU に基づくメモリ分割のほうが効果があることが示された．

5.3 チャンクサイズの性能への影響

リモートページングを行う際にまとめてページングを行うページ数であるチャンクサイズが，マイグレーション後の性能にどのような影響を及ぼすかを調べた．この実験では，チャンクサイズを 1 〜512 まで 2 のべき乗になるように変化させ，マイグレーション後に GNU sort を実行した．VM にはメモリを 2GB 割り当て，移送先メインホストの空きメモリは 1GB とした．2GB のデータに対してソートを行った時の実行時間を図 17 に示す．

実験結果より，チャンクサイズが大きくなるにつれて実行時間が短くなっていることがわかる．これは図 18 に示すように，チャンクサイズが大きくなるとページングが発生しにくくなるためである．しかし，チャンクサイズが

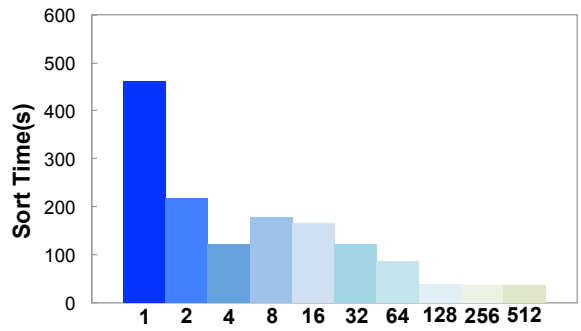

図 17 チャンクサイズごとの sort の実行時間

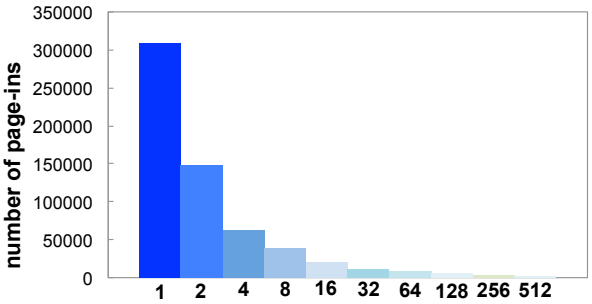

図 18 チャンクサイズごとのページイン回数

256 の時に実行時間が最短となり，チャンクサイズを 512 にすると実行時間が長くなった．チャンクサイズを大きくしすぎるとページインしたページの中にアクセスされないページが増える一方で，アクセスされるページがページアウトされやすくなるためと考えられる．

5.4 メモリ参照情報の取得時間

定期的に EPT からメモリ参照情報を取得するオーバヘッドを測定した．この実験では VM のメモリサイズは 2GB とした．実験結果を図 19 に示す．VM を起動した後のアイドル状態では，メモリ参照情報を取得するのにかかる時間は 1 ミリ秒であった．VM 内で 500MB を割り当てた memcached [11] を動作させ，memaslap ベンチマーク [12] を用いてメモリに負荷をかけた状態では，取得時間が 3 ミリ秒に増加した．これはメモリに負荷をかけることにより多くのメモリページがアクセスされ，EPT が肥大化したためである．この場合でも，S-memV が 1 秒ごとにメモリ参照情報を取得する時のオーバーヘッドは 0.3 % である．大容量のメモリを持つ VM ではオーバーヘッドはより大きくなると考えられる．しかし，EPT のエントリはアクセスされないページについては解放されるため，オーバヘッドはある程度以上は大きくならないことが予想される．

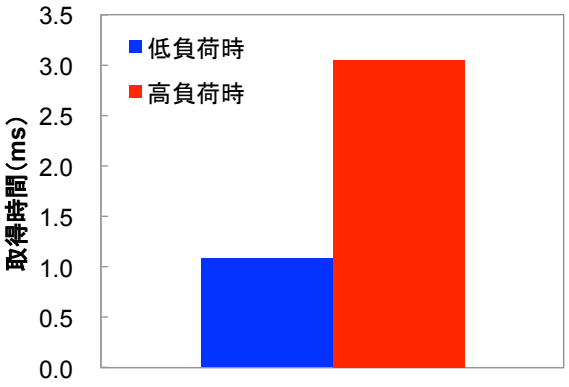

図 19 メモリ参照履歴の取得時間

6. 関連研究

ポストコピー・マイグレーション [10] は，VM を実行する上で最低限，必要な情報だけを先に転送して移送先ホストの VM に切り替える手法である．移送先ホストに存在しないメモリは移送元ホストからオンデマンドまたはバックグラウンドで転送する．このように，ポストコピー・マイグレーションは 2 台のホストを使って VM を動かすことができ，分割マイグレーションの特殊な場合とも考えられる．しかし，この状況は一時的なものであり，マイグレーションが完了すると VM は 1 台のホストで動作する．そのため，ポストコピー・マイグレーションは大容量メモリを持ったホストを 2 台必要とする．

Scatter-Gather マイグレーション [13] は，ポストコピー・マイグレーションを行う際に，移送元ホストと移送先ホストの間で複数の中間ホストを用いる．VM のメモリをできるだけ速く中間ホストに転送してしまうことにより，移送元ホストで VM が停止するまでの時間を短縮することができる．移送先ホストでは，ポストコピー・マイグレーションのオンデマンド転送やバックグラウンド転送を用いて中間ホストから VM のメモリを取得する．複数のホストに VM のメモリを転送する点では S-memV に似ているが，Scatter-Gather マイグレーションでは最終的に VM のすべてのメモリが 1 台の移送先ホストに送られる．S-memV では移送先ホストに VM のメモリが入りきらないことを想定している点が異なる．

MemX [8] は S-memV と異なり，VM の起動時から複数のホストのメモリを利用することを可能にする．MemX-VM モードでは VM 内のゲスト OS がブロックデバイスを提供し，そのデバイス経由で他のホストのメモリへのアクセスを可能とする．このモードでは，VM のマイグレーション時に他のホスト上のメモリの転送は行わない．MemX-DD モードでは Xen のドメイン 0 でブロックデバイスを提供するが，マイグレーション時にこのデバイスの状態を転送しないため，マイグレーションには対応していない．一方，MemX-VMM モードでは，VM の拡張メモリを通して透過的に他のホストのメモリへのアクセスを可能とする．VM がそのホストに存在しない物理メモリ領域にアクセスすると，他のホストにあるメモリを取得する．このモードは S-memV の分割マイグレーション後の動作と同じである．

Virtual Multiprocessor [14] や vNUMA [15] では，複数

のホストの CPU やメモリを用いて一つの VM を動かすことを可能とする．これにより，1 台のホストではリソースが不足していても複数のホストを用いることで巨大な VM を動かすことができる．分散共有メモリを用いることでメモリが存在するホストを意識することなく他のホストのメモリにアクセスができる．S-memV とは複数ホストのリソースを使用する点で類似しているが．これらのシステムでは複数のホストで 1 台の VM を動かすのに対し，S-memV では VM を動かすホストは 1 台で，残りのホストはメモリを提供するだけである．また，これらのシステムは VM のマイグレーションには対応していない．

7. まとめ

本稿では，大容量メモリを持つ VM を複数のホストに分割してマイグレーションすることを可能とするシステム S-memV を提案した．S-memV は VM の核となる情報と VM がアクセスすると予測されるメモリを移送先のメインホストに転送し，メインホストに入りきらないメモリをサブホストに転送する．マイグレーション後，VM はメインホストで動作し，必要に応じてサブホストとの間でリモートページングを行う．S-memV を KVM に実装し，分割マイグレーションを実現した．実験結果より，マイグレーション性能とマイグレーション後の VM の性能の低下を抑えることができることを確認した．

今後の課題は，様々なアプリケーションに対してより多くの実験条件で性能評価を行うことである．また，分割マイグレーションにより複数のホストにまたがって動作する VM をマイグレーションできるようにすることも計画している．

謝辞

本研究の一部は電気通信普及財団の助成を受けたものである．

参考文献

[1] Apache Software Foundation. Apache Spark – Lightning-Fast Cluster Computing http://spark.apache.org/.
[2] Facebook, Inc. Presto: Distributed SQL Query Engine for Big Data https://prestodb.io/.
[3] SAP SE. SAP HANA https://hana.sap.com/.
[4] Microsoft Corporation. SQL Server 2014 https://www.microsoft.com/en/server-cloud/products/sql-server/.
[5] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen. Parallelizing Live Migration of Virtual Machines. In Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Ecvironments. page 85-96 (2013).
[6] Mellanox Technologies. Accelerating Virtual Machine Migration over vSphere vMotion and Mellanox End-to-End 40GbE Interconnect Solutions. http://www.melloanox.com/related-docs/solutions/SB_Accelerating_Virtual_Machine_Migration.pdf (2016).
[7] D. Comer and J. Griffioen. A New Design for Distributed System: The Remote Memory Model. In Proceedings of the summer 1990 USENIX Conference, pages 127-135 (1990).
[8] U. Deshpande, B. Wang, S. Haque, M. Hined, and K. Gopalan. MemX: Virtualization of Cluster-Wide Memory. In Proceedings of International Conference on Parallel Processing (2010).
[9] M.J Feeley, W.E.=Morgan, E.P. Pighin, A.R. Karlin, H.M. Levy, and C.A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In Proceedings of the 15th ACM Symposium on Operationg Systems Principles, pages 201-212 (1995).
[10] M.R. Hines, and K. Gopalan. Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In Proceedings of International Conference on Virtual Execution Environments (2009).
[11] B. Fitzpatrick. memcached – A Distributed Memory Object Caching System http://memcached.org/.
[12] B. Aker. memaslap – Load Testing and Benchmarking a Server http://docs.libmemcached.org/bin/memaslap.html.
[13] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan. Fast Server Deprovisioning through Scatter-Gather Live Migration of Virtual Machines. 7th IEEE International Conference on Cloud Computing (2014).
[14] 金田憲二, 大山恵弘, 米澤明憲. 単一システムイメージを提供するための仮想マシンモニタ. 情報処理学会 ACS 論文誌, Vol.47, No.SIG 3, pp.27-39 (2006).
[15] M. Chapman and G. Heiser. vNUMA: A Virtual Shared-Memory-Multi Processor. In Proceedings of Conference USENIX Annual Technical Conference (2009).