

世代の概念を考慮したディスク書き込み追跡による高速な VM 転送機構の提案

高橋 一志[†] 笹田 耕一[†]

仮想マシン (VM) を他のモバイル機器にライブマイグレーションできるようにすれば、計算環境を容易に移動させて持ち運ぶことができ、便利である。近年では、VM のディスクイメージも含めて VM ライブマイグレーションを行う手法も開発されており、この方式はモバイル機器との親和性が高いと考えている。しかし、VM のディスクイメージは一般に巨大なファイルであり数十 GByte になることも珍しくなく、転送が完了するまでには大きな時間がかかりユーザの利便性が低下する。われわれは、以前、モバイル機器を挟むような VM ライブマイグレーションでは、転送した VM が、再びもとの環境へ転送される可能性が高いことに着目し、再転送時には変更されたディスクページのみを転送する差分ディスクイメージ転送を提案した。しかし、以前提案したアルゴリズムでは差分を正確に計算することができず、本来ならば転送しなくてもよいディスクブロックまで差分として転送してしまうという問題点があった。そのため、本論文ではその問題点を改良した新たな差分転送のアルゴリズムを提案する。

A fast VM transport mechanism that consider generations with disk dirty page tracking

KAZUSHI TAKAHASHI[†] and KOICHI SASADA

The mechanism that allows us to migrate easily Virtual Machine (VM) to other mobile devices is comfortable. Recently, new migration method has been developed. This method packs whole VM status that including VM storages. We think that this method has an opportunity in Mobile Internet Device VM industry. However, VM disk images generally have huge file size and the huge time is required until VM migration is completed. We think that it is not comfortable. Thus, We have focused on that there is a greater chance of returning a VM from the migrated host. Thus, we have developed an algorithm to reuse duplicate block pages when the VM re-migrate. However, our method could not extract exactly block pages that should transmit. In fact, even if there are blocks that were not necessary to transmit, our method transmits these blocks. Thus, we propose an improved algorithm in this paper.

1. はじめに

われわれは、個人用計算機環境における Virtual Machine (VM) の高速転送技術として VM ライブマイグレーションを利用することを考えている。個人用計算機環境として物理マシンを直接使用させる代わりに VM を使用させるソリューションには、セキュリティの高さ、計算機環境の可搬性の高さ、バックアップの容易さといった利点が存在する。そのため、今後受け入れられていくものと予想される。このとき、VM の実行コンテキストを維持したまま、VM を他のモバイル機器に移動させる技術を開発すれば VM を容易に移動させて持ち運ぶことができ便利であると考ええる。

しかし、従来のライブマイグレーションにはネットワーク透過なファイルシステムを使用することが前提

となっている。そのため、ライブマイグレーションを行う物理マシンの転送元と転送先との間で、NFS¹⁾ や cifs といったネットワークファイルシステムを用いて仮想マシン (VM) のディスクイメージの共有を行わなければならない。Hypervisor 自身が転送を行うのは VM の RAM データと VM を構成する仮想デバイスのステート情報だけである。

この制約を取り去る新たな VM ライブマイグレーション機構が Linux/KVM²⁾ と QEMU³⁾ の開発コミュニティより提案されている。この新たなライブマイグレーション機構は**ブロックマイグレーション**と呼ばれる。ブロックマイグレーションでは、その名が示す通り、RAM データや、マシンステートに加えて、VM のディスクイメージ (ブロックデバイス) もまとめて転送する。そのため、TCP/IP による通信環境さえ整っていれば、容易に VM ライブマイグレーションを行うことが可能となる。

しかし、この新たなブロックマイグレーションのメカニズムには転送に多くの時間が要求されるという問

[†] 東京大学大学院情報理工学系研究科

Graduate School of Information Science and Technology, The University of Tokyo

題点が存在する。VM のディスクイメージは一般的に巨大なファイルであり、数十 GByte になることも珍しくない。そのため、例えば、1Gbps の LAN 環境にて 20GB のディスクイメージを持つ VM をマイグレーションすると、約 10 分もの時間がかかる。マイグレーションが完了するまでに 10 分もの時間がかかるのはユーザの利便性を低下させると考える。

この問題点を解決するため、われわれは差分転送を利用したブロックマイグレーション高速化のメカニズムを考案した⁴⁾。例えば、A と B という 2 つの物理マシンが存在するとして、A から B に VM マイグレーションを行ったとする。その後、B から A へ再び VM ライブマイグレーションが行われたとき、転送元の A には、元の VM のディスクイメージが残されていることになる。そのため、A から B の初期転送にかかる時間は大きいものの、その後、B から A に VM ライブマイグレーションが行われたときには、転送先の B で汚された VM のディスクページのみを転送すればよく、従来のブロックマイグレーションに比べて大幅に転送時間を短縮することが可能になる。なお、このユーザシナリオについては 2 章にて詳しく議論する。

しかし、われわれが以前行った提案では、3 者以上の計算機をまたぐ転送の場合、変更された部分のみを正確に検出することができず、本来ならば転送しなくてもよいはずのディスクブロックまでもを差分として転送してしまうという問題点があった。そこで、本論文では、われわれの以前の研究をさらに発展させ、どのディスクブロックが汚れたかを世代とともに管理する。これにより、A, B, C といった 3 者間以上の転送でも、正確に、変更されたディスクブロックのみを検出し差分転送できることが可能となる。

上記のアイデアを KVM/Linux 上に実装して実験を行った。具体的には差分転送によるブロックマイグレーションをサポートするためのフォーマットを新たに設計し、KVM/Linux にプロトタイプを実装した。そして、さまざまなワークロードを持つ VM 上にて、われわれが提案する差分転送を利用した新たなブロックマイグレーションが従来のブロックマイグレーションと比べてどの程度高速化に寄与するのか評価を行い、差分転送によるブロックマイグレーションが十分な効力を発揮することを確認した。

2. 問題分析

本章では、現状の VM ライブマイグレーション（ブロックマイグレーション）に何が不足しているのかを分析した後、われわれの提案が何を改良することを目的としているのか明示する。この問題分析は、われわれが以前書いた論文⁵⁾に詳しく記述されているので、詳細はそちらをご覧いただきたい。ここでは、概要のみを記述する。

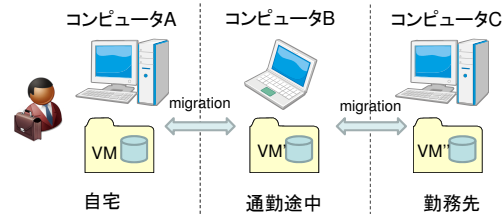


図 1 個人用途のマイグレーション

Fig. 1 The live-migration for personal usage

上述したように、われわれは、VM ライブマイグレーションを個人用計算機環境にて使用するユーザシナリオを考えている。現在、ユーザに対して物理マシンを割り当てる代わりに VM を割り当て、ユーザは物理マシンを直接使用せず VM を使用するというシステムが提案されている⁶⁾。

このような個人用計算機環境上で使用される VM に対して VM ライブマイグレーションが導入されることでより便利な個人用の仮想計算機環境が開発可能であると考える。図 1 に個人用計算機環境上で使用される VM ライブマイグレーションのイメージ図を示した。これは、Intel の Shivani Sud らのレポート⁷⁾に記載されていたシナリオのイメージ図である。彼らが挙げている個人用計算機環境上で使用される VM に対する VM ライブマイグレーションの導入シナリオを以下に引用する。

A さんは、今朝のミーティングで提出する必要があるプレゼンテーションを、自宅にあるパソコン (VM) を使ってレビューしている。出勤時間になったので、自宅のマシン上で使っていた VM 環境を丸ごと Netbook といった MID (Mobile Internet Device) に VM ライブマイグレーションを使ってシームレスに転送する。通勤途中の地下鉄の中でも、携帯端末上で自宅の PC 環境がそのまま使用できるため、プレゼンテーションの見直しを続けることができる。オフィスに到着すると、携帯端末上からオフィスにある計算機へと VM ライブマイグレーションを使用して VM を転送して、通勤途中の地下鉄の中で使っていた環境がオフィス上の計算機にシームレスに再現される。

しかし、既存のブロックマイグレーションには大きな問題点が存在する。それは、ブロックマイグレーションに要する全体の転送時間が大きくなってしまうことである。VM を構成するディスクイメージは一般的に巨大であり、数十 GByte の容量を持つものも少なくない。このレベルファイルサイズ (20Gbyte のディスクイメージ) をブロックマイグレーションで転送するには、現在普及している Gigabit の LAN 回線でも約 10 分もの時間を要する。

個人用計算機のライブマイグレーションの活用事例として、われわれが挙げた上記のシナリオではVMのライブマイグレーションに要する転送時間を低下させることが重要である。既存のVMライブマイグレーション(ブロックマイグレーション)では、極論ではあるが、VMのダウンタイムさえ短ければ転送の全体時間は長くても問題はなかった。事実、既存のPre-copy VMライブマイグレーションプロトコル^{8),9)}はそのような設計になっている。しかし、われわれが想定する、個人用途を想定したVMのライブマイグレーションでは転送の全体時間を縮小することは重要である。なぜなら、個人用のVMライブマイグレーションにとって重要なことは、ユーザが思い立った時に素早くVMの転送が行え、その後は一切の通信をすることなくVMが転送先が動作し、持ち運べることである。そのため、VMの転送に数十分もの時間を要するのは問題である。

3. 提案手法

本章では、2章で述べたブロックマイグレーションに要する全体転送時間をいかにして縮めるかという問題点をどのように解決するかについて議論する。

本研究のアイデアはVMライブマイグレーションによって移動したVMは再び移動元の計算機に戻ってくる可能性が高いというユーザシナリオに基づいている。上述の例の通り、個人用計算機におけるVMライブマイグレーションの場合、ユーザは特定の物理マシン上(上記の例では3者A, B, C)間にてVMを往来させる可能性が高い。この時、それぞれの計算機上でディスクページに加えらる変更点はわずかであると考えるため、差分転送によるディスクの転送が有効である。

われわれの提案は以下の通りである。まず、Hypervisorに対してディスクページのダーティブロックトラッキング(DBT)を付加し、ディスクページの書き込みをすべてトラッキングしておく。次に、VMライブマイグレーションが起こったとき、もし、転送先にVMのディスクイメージが存在しなければ、通常のプロックマイグレーションを行う。そして、すでに、転送済みのディスクイメージが存在するときはDBTによって検出されたダーティブロックのみを転送する。これにより差分転送による高速なブロックマイグレーションを可能にする。

以前の我々が行った提案との違いは、DBTで記録されるビットの代わりに世代を記録する点にある。世代は、一つのVMディスクイメージに対して一つの数値が割り当てられ、VMが起動したときや、VMが他の物理マシンに移動したときにインクリメントされる数値である。DBTにより、VM上にてどのディスクブロックが汚れたかがすべて記録されるわけであるが、

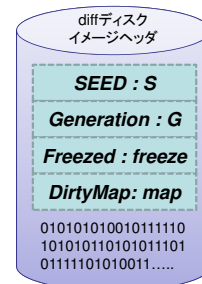


図 2 diff ディスクイメージのヘッダ
Fig. 2 The structure for diff diskimage

その時、どの世代によってそのブロックが汚されたのかも同時に記録する。DBTによる世代付きのディスクの書き込み追跡により、過去移動したことのあるディスクイメージに対してであれば、正確にその差分を割り出すことができ、本当に必要となる差分のみを抽出し、転送することが可能になる。

4. 設 計

差分転送による高速なブロックマイグレーションをサポートするためのdiff ディスクイメージと名付けた新たなVMのディスクイメージを設計した。本章では初めに、DBTの設計について述べ、次に、DBTを支援するためのdiff ディスクイメージがどのような構造を持つのかについて述べる。そして、diff ディスクイメージがどのようにして差分転送による高速なディスクイメージ転送を行うかについて論じる。

4.1 ダーティブロックトラッキング(DBT)

DBTは、ディスクの全書き込みを追跡しダーティマップにその記録を残すための機構である。DBTはHypervisor側から捕捉できるVMのディスク書き込みをフックすることで行う。ディスクの全体領域を特定の大きさを持つブロックに区切り、1ブロックにつき8ビットとしてダーティマップを構成することでDBTを実現する。8ビットの空間には世代番号が書き込まれる。世代番号とは上述したとおり、VMの一つのイメージごとに割り当てられる一つの数値であり、VMが他の物理マシンにマイグレーションで移動したときや、VMが起動したときにインクリメントされる数値である。8ビットの空間にはこの世代番号が書き込まれる。DBTとダーティマップによって、どの世代によって、どのディスクブロックが汚れたのかすべて記録される仕組みになっている。

4.2 diff イメージフォーマットを用いたマイグレーション

diff イメージは、DBTで得たダーティマップを構造化して管理することにより、差分転送をサポートするVMディスクイメージである。diff イメージの構造を図2に示す。

diff イメージヘッダには現在の世代番号 G , seed 番号 S , そのイメージが起動可能か否かを示す $freeze$, そして, どの世代によって, どのブロックが汚れたのかを記録するダーティページマップ Map によって構成されている. 4.3 節にてこれらのデータ構造がどのようにしてブロックマイグレーションをサポートするかについて述べる.

4.3 マイグレーションアルゴリズム

本節ではわれわれが提案するマイグレーションが $A \longleftrightarrow B \longleftrightarrow C$ という3つの仮想マシン間でどのように動作するかについて詳解する.

図3に $A \rightarrow B \rightarrow C$ 間の転送がどのように行われるのかについて図示する.

初めに A で作られた diff ディスクイメージはユニークな seed 番号 S_0 (UUID : UUID generation daemon によって生成されたユニークな番号) が付与される*. なお, 初代の世代番号は1から始まる.

まず, A にある VM イメージが起動される. 世代番号は OS が起動したときにインクリメントされ, 1から2になる (図中 A) この状態で, VM 上に作業が行われ, 世代番号2の汚れが DBT によって追跡され, ダーティマップに記録されてゆく. 図の例では, 2番目と4番目が世代2によって汚されたことがわかる.

次に, $A \rightarrow B$ へと転送が行われる. この時 B には差分転送に利用できる diff イメージはないので, 通常の低速なブロックマイグレーションが行われ, さらに B の世代番号がインクリメントされる (図中 B) $A \rightarrow B$ へのブロックマイグレーションが行われた後, A にある diff ディスクイメージは凍結される (図中 C) これはヘッダーの $freeze$ フラグを1にすることで行われる. $freeze$ フラグが1の diff ディスクイメージは一時的に Hypervisor から起動できないようになる. なぜ, この $freeze$ フラグが必要なのかは後述する. この時, A には $generation = 2, SEED = S_0, freeze = 1$ の diff ディスクイメージが残ることになる. B 上の VM イメージは, ライブマイグレーションにより VM が移譲され OS が起動するため, 再び世代がインクリメントされる (図中 D) 結果的に $generation = 4, SEED = S_0, freeze = 0$ の diff ディスクイメージが作られることになる. B 上で作業を行うにつれて diff ディスクイメージのディスクブロックはどんどん汚れてゆく. 図の場合では最終的に世代4によって3番目のディスクブロックが汚されていることが DBT により記録される (図中 D)

さらに, $B \rightarrow C$ 間の転送がどのように行われるのかについて解説する. $A \rightarrow B$ のときと同様に C にも差分転送に利用できる diff イメージはないので通常のブロックマイグレーションが行われ, さらに C の世

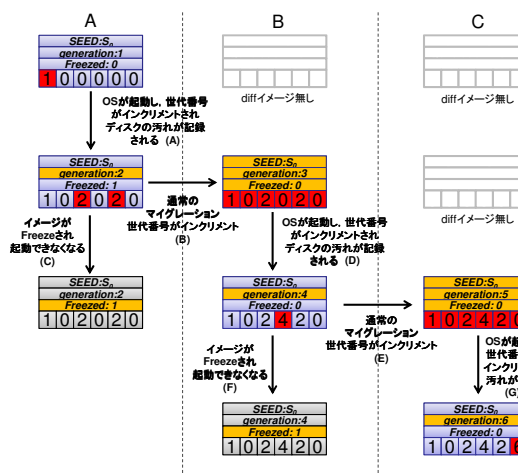


図3 $A \rightarrow B \rightarrow C$ 間のマイグレーション

Fig. 3 The migration between $A \rightarrow B \rightarrow C$

代番号がインクリメントされる (図中 E) このマイグレーションが終わった後, A にあるイメージが凍結され, Hypervisor から一時的に起動することができなくなる (図中 F) C 上の VM イメージは, ライブマイグレーションにより VM が移譲され, OS が起動することによって再び世代がインクリメントされる (図中 G) 結果的に, C 上には $generation = 6, SEED = S_0, freeze = 0$ の diff ディスクイメージが作られることになる. C 上で作業が進むにつれ, ディスクが汚れてゆく. 図の例では, 世代6によって5番目のディスクブロックが DBT によって記録されていることがわかる (図中 G).

このように, 計算機間でブロックマイグレーションが行われたり, OS の起動が起こるたびに世代番号 G がインクリメントされていき, それらの diff イメージファイルは共通 SEED 番号: S_0 を持っていることになる. また, DBT によって, どの世代でどのディスクブロックが汚されたかが記録されることになる.

さらに, 図4に $C \rightarrow B \rightarrow A$ 間のブロックマイグレーションについて解説する.

初めに, $C \rightarrow B$ 間のブロックマイグレーションについて解説する. すでに, B 上には差分転送できる $generation = 4$ の diff ディスクイメージがあるので差分転送をつかった高速なブロックマイグレーションが可能である. まず初めに, C 上と B 上の diff ディスクイメージに記録されている SEED 番号: S_0 を見て, これが同一の SEED 番号である S_0 であることを確認する. 仮に, S が異なる場合, C, B 間にはまったく異なるディスクイメージで, 差分転送は不可能であると判定する. 次に, ダーティマップを順に見てゆき, 相手の世代番号4より大きい世代のブロックのみを転送する. そのうち世代番号がインクリメントする (A). 今回の場合, 5番目の6ブロック目のみが世代番号が

* qemu-img コマンドで diff ディスクイメージを作成したり, 他のフォーマットから変換したときに付与される.

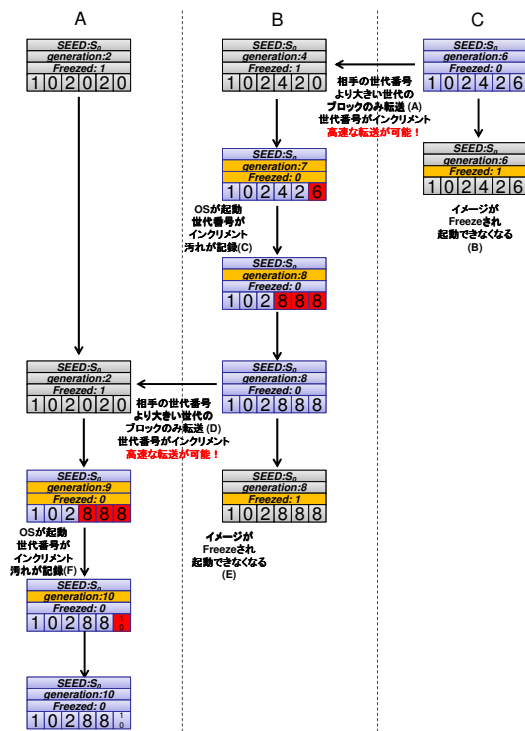


図 4 C → B → A 間のマイグレーション

Fig. 4 The migration between C → B → A

6 となっており、相手の世代番号よりも大きくなっている。そのため、このブロックのみを転送する。これで、高速な差分転送が可能である。転送元は同じように $freeze = 1$ となり、このイメージから起動することが不可能となる (B)。次に転送された先では、OS が再び起動することで世代番号がインクリメントされ、世代番号は 8 となる。この状態でディスクの汚れが記録される。図の例では、3, 4, 5 番目のブロックが世代番号 8 によって汚されていることがわかる (C)。

次に、B → A の転送を考える。この場合も、データマップを順に見てゆき、相手の世代番号よりも大きな世代番号を持つブロックのみを転送し、世代番号がインクリメントされる (D) 図の例では、世代番号 8 によって汚された、3, 4, 5 番目のブロックのみを転送することになる。同様に、転送元であった世代番号 8 のイメージは Freeze され、起動することができなくなる (E)。一方で、転送された VM は、OS が起動することで再び世代番号がインクリメントされたうえで、汚れが記録されることになる (F) 図の例では世代番号 10 によって、5 番目のブロックが汚されていることがわかる。

このようにして、今回提案するアルゴリズムでは過去にマイグレーションされた場所であれば、そこに存在する diff イメージとの差分を正確に割出し、差分のみを転送し、高速なブロックマイグレーションが可能と

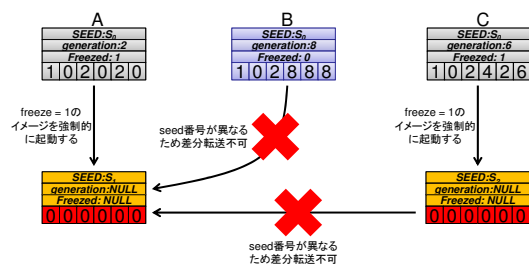


図 5 ディスクイメージの freed を解除した場合の各仮想マシンの挙動

Fig. 5 The behavior of VMs, if users unfreeze vm disk images

なる。

最後に、 $freed$ の役割について詳細に述べる。 $freed = 1$ となった diff ディスクイメージは基本的に起動不可能である。 $freed = 1$ のディスクイメージは基本的に、差分転送によるブロックマイグレーションを待機している状態であり、ブロックマイグレーションによる VM の移譲による起動以外の方法で起動することはできない。

もし、ユーザが強制的に $freed = 1$ のディスクイメージを起動したときにはどのような現象が起こるのかを図 5 に示す。A, B, C の仮想マシン上で B のマシンが起動中であり、A, C の仮想マシン上のディスクイメージは $freed = 1$ である。ユーザが、 $freed = 1$ のディスクイメージ上にある A を強制的に起動したとする。この時、Hypervisor は A 上の仮想マシン上にあるディスクイメージに新たな SEED 番号である S_1 を割り振る。次に、 $generation$ と Map も両方 NULL にクリアする。つまり、A 上の仮想マシン上にあるディスクイメージはまったく新しい別のディスクイメージとして生まれ変わることになる。双方のディスクイメージは SEED 番号が根本的に異なるため、起動中の B からの高速な差分転送は不可能となる。

4.4 議論

DBT による書き込み追跡の代わりに、転送時に $rsync^{10)}$ と同様の手法である $Compare-by-hash^{11)}$ の手法を導入することが考えられる。つまり、転送先と転送元で、すべてのディスクブロックのハッシュ値を突合せ、ハッシュ値の異なるブロックをすべて転送するという手法をとれば DBT による書き込み追跡を行わずとも、差分転送が可能である。しかし、予備評価では 4GByte のディスクを 512KB 単位のブロックに区切って SHA-1 ハッシュをかけただけで 2 分近い時間が消費されてしまうことが判明したためこの手法は採用しない。

5. 実装

われわれは、4 章で述べた diff フォーマットを

Linux/KVM,QEMU に対して実装した。実装内容は以下に述べる 2 つの機能である。

1 つ目はダーティマップを作成するための DBT である。上記の二つのダーティマップは、仮想ディスク全体をある一定の大きさのブロック (セクタ) に区切り、各ブロック (セクタ) 1 つにつき 8 ビット*としてビットマップを構成している。ちなみに、現在の実装では 2048 セクタを 1 ブロックとしている。なお、1 セクタあたりの容量は 512 バイトでありこれは QEMU のデフォルト値である。4Gbyte 程度の仮想ディスク容量であれば、4 キロバイト程度のビットマップに収まり、20Gbyte 程度の仮想ディスク容量であっても 20 キロバイト程度のビットマップに収まる。そのため、ビットマップが物理ハードディスク容量を圧迫することはない。

2 つ目が上記のダーティマップを QEMU のディスクイメージ階層とライブマイグレーション階層との間でやり取りするための API と、ライブマイグレーションが完了して、世代番号をインクリメントすることをディスクイメージに伝えるための 2 つの API が必要である。QEMU の実装はうまく構造化されている。例を挙げると、vmdk, qcow, qcow2 といったディスクイメージを操るディスクイメージ固有の処理を行う層は QEMU 内のディスクドライバとして書かれており、これらのドライバの開発者は QEMU 側から指定された API 群を記述していくことで新しい QEMU のファイルフォーマットを開発することができる。しかし、記録したダーティマップをディスクイメージ階層とライブマイグレーション階層とでやり取りする API や、世代番号をインクリメントするための API は存在しないため、これら二つの API の追加を行った。

6. 評価

評価は 2 つの観点から行った。1 つ目の評価は DBT の書き込み追跡が VM の致命的なディスクの書き込み性能の低下を引き起こさないことを確認するためのものである。2 つ目の評価は、さまざまなワークロードを VM を起動させ、ディスクブロック (1 ブロック 2048 セクタ) がどの程度汚されるのかを確認し、どの程度差分転送による高速化が有益に使用できるのかどうかを確認する。

6.1 DirtyPage トラッキングの性能評価

本節では、DBT によるダーティマップを更新する処理が致命的なディスク書き込み性能の低下を引き起こさないことを確認する。評価環境には、Lenovo ThinkPad X220 ラップトップコンピュータを使用した。CPU は Intel Core i5-2430M@2.40GHz で、物理メモリは 4GB である。HDD は Seagate 製の 7200rpm

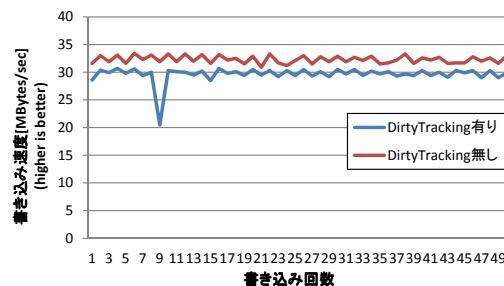


図 6 ダーティマップ更新時における dd に書き込み速度
Fig. 6 Writing cost by dd to update bitmaps

HDD で 16MB のキャッシュメモリが搭載されている。ダーティマップの更新は書き込み処理時のみに行われるので、書き込み性能のみを評価した。dd コマンドを使い 1MB のブロックを 100 回書き込む評価を 50 回行った。評価結果を図 6 に示す。縦軸が書き込み速度 MBytes/sec で、横軸が書き込み回数である。トラッキング無しのときで平均 32.304MBytes/sec で、トラッキング有りの時が平均 29.656MBytes/sec であり、DBT による 8% 程度のディスク書き込み性能低下があることがわかる。

6.2 異なるワークロードによるブロックマイグレーションの速度

次に、実際に差分転送による効果の測定を行った。現実的なワークロードを考えて、以下に示す 4 つの作業を行った後で差分転送によるブロックマイグレーションを行った。パワーポイントファイルをダウンロードして Open Office による閲覧と編集と保存、夏目漱石の「こころ」(約 400KB のテキストファイル) を青空文庫よりダウンロードして閲覧、Firefox を使用した YouTube でのビデオ再生、約 3MB の PDF ファイルをダウンロードして閲覧。上記 4 つのテストをユーザに 5 分間行ってもらった後、ライブマイグレーションを行った後で差分転送によるブロックマイグレーションを行い、通常のブロックマイグレーションと比べて、差分転送がどの程度有効であるかを確かめる測定を行った。

評価環境は、転送先が Lenovo の ThinkPad X60 ラップトップコンピュータで、CPU は Intel Core Duo CPU T2400 @ 1.83GHz で 2GBytes の物理メモリが搭載されている。転送元は DELL LATITUDE D630 ラップトップコンピュータで、CPU は Intel Core2Duo T7300 2.0GHz で、2GBytes の物理メモリが搭載されている。回線は 1Gbps の LAN 環境を使用した。

表 1 に Ubuntu 11.04 を使用したブロックマイグレーション結果を示す。一番時間がかかっているのはプレゼンテーションの 30.141 秒である。一方で、時間がもっともかかっていないのは「こころ」の 25.900 秒である。全体的に見て約 10 分間かかっていた転送が、差分転送の効果で数十秒に低下していることが

* これは自由に増減することができる

表 1 Ubuntu 11.04 desktop (x86) 上でのブロックマイグレーションの測定結果
Table 1 The Block migration measurement result on Ubuntu 11.04 desktop (x86)

	差分転送なし	PDF	プレゼンテーション	YouTube	こちら
マイグレーション全体時間 (sec)	919.358	29.139	30.141	28.720	25.900
差分転送によるディスク転送量 (MBytes)	—	101	104	111	90

表 2 Windows 7 Professional (x86) によるブロックマイグレーションの測定結果
Table 2 The Block migration measurement result on Windows 7 Professional (x86)

	差分転送なし	PDF	プレゼンテーション	YouTube	こちら
マイグレーション全体時間 (sec)	991.044	89.028	68.892	78.450	86.703
差分転送によるディスク転送量 (MBytes)	—	933	613	927	907

わかる。差分転送によって転送されるディスクブロックも 20GB から数百 MB 内に縮小していることがわかる。

次に、表 2 に Windows 7 Professional を使用したブロックマイグレーション結果を示す。Ubuntu の時と比べると、Windows の場合、全体的に書き込まれているディスクブロックも多く、転送に比較的時間がかかっていることがわかる。一番時間がかかっているのは PDF 閲覧の 89.028 秒である。一方で、時間がもっともかかっているのは、プレゼンテーションの 68.892 秒である。Windows (NTFS) の場合、Ubuntu (ex4) よりも多くのディスクブロックを汚すようである。また、Ubuntu のときはプレゼンテーションが一番時間がかかっていたのに対して、Windows の場合、PDF 閲覧が最も時間がかかる結果となった。また、Ubuntu の場合、もっとも時間がかかっているのは、YouTube の 28.720 秒であったのに対して、Windows の場合、もっとも時間がかかっているのは PDF 閲覧であった。Windows の場合、数十秒に低下しているとは言えないものの、おおむね一分前後の転送時間であることがわかる。

7. 関連研究

Linux/KVM 上のブロックマイグレーション以外にも、VM のディスクを含めてマイグレーションするための既存研究がいくつか存在する。Luo¹²⁾ や Bradford¹³⁾ らは、Xen 上で特殊なバックエンドデバイスドライバを実装することで、Linux/KVM のブロックマイグレーションとはほぼ同等のことを実現している。つまり、共有ファイルシステムを使わずに VM ライブマイグレーションを達成している。しかし、転送先に過去のディスクイメージが存在していた場合には差分転送のみを行うといったアプローチについては言及されておらず、本研究とは異なる。また、広瀬¹⁴⁾ の提案では、Linux 上に独自のブロックデバイスドライバ /dev/nbd0 を実装することで、同様に共有ファイルシステムを使わない VM ライブマイグレーションを達成している。VM を起動するときローカルに存在する通

常の VM イメージファイルの代わりに /dev/nbd0 を使うことで、ライブマイグレーション時には /dev/nbd0 が自動的に VM のディスクコピーも行う。われわれの研究との違いは、VM のライブマイグレーション先に過去のディスクブロックが存在していたとき、それらを転送しないことで高速化を図るというメカニズムを提案した点にある。

また、Sapuntzakis¹⁵⁾ らは、さまざまなユーザシナリオを想定した上で、仮想マシンのマイグレーションに関して、転送高速化に関するいくつかの提案を行っている。その中で、彼らはツリー構造を持つディスクイメージの管理手法を提案している。まず、ルートイメージと呼ばれる全ディスクイメージの元ファイルを作成する。ユーザは、このルートイメージをチェックアウトして、ルートイメージから見て子となるディスクイメージを作ることができる。さらに、ユーザが作成したディスクイメージから見てさらに子となる VM イメージも作ることができる。このようにして、ルートイメージ以外の、すべての VM イメージがそれぞれ親を持つというツリー構造が出来上がる。子は、ディスクに変更が加えられない限りは親へのポインタのみを持っており、実データは持たない構造になっている。マイグレーションのときは、親へのポインタのみを持つ子ノードファイルを転送することで転送量を減らす。転送後、必要なディスクブロックがあった場合、ネットワーク経由にて、オンデマンドで親をたどって必要なディスクブロックを見つけるといった手法である。われわれの研究との違いは、彼らが、ネットワーク経由にて子ファイルを元にディスクブロックを後から要求できる状況を考えたうえで、ツリー構造を持つディスクイメージの管理手法を提案したのに対して、われわれは、初めからすべてのディスクイメージがマイグレーション元と先とで完全にコピーされる状況を考えた上で、ツリー構造よりもよりシンプルな差分記録による高速な転送手法を提案した点が異なる。

また、Intel Research から Internet Suspend/Resume (ISR) と呼ばれる手法に関するテクニカルレポートがいくつか出ている。ISR とは、VM を Suspend してから転送して、転送元で Resume することで、VM の

コールドマイグレーションを実現する手法のことである。Kozuch¹⁶⁾とTolia¹⁷⁾らは、Coda¹⁸⁾と呼ばれる分散ファイルシステムを使って、ディスクイメージを含んだすべての VM のステートをすべて Coda 上にいったん格納し、転送先からネットワーク経由にてオンデマンドで VM のステートを要求することで ISR を実現している。Coda はファイルの先読み機能とキャッシング機構を備えているため、オンデマンドの転送以外にもバックグラウンドで VM のステータ転送が行われ、やがてネットワークに接続せずとも Coda 経由で転送先から VM のステータを読みだすことができるようになる。われわれの研究との違いは、彼らが、ネットワーク経由にてディスクブロックを後から要求できる状況を考えたうえで VM のコールドマイグレーション手法を提案したのに対して、われわれは、初めからすべてのディスクイメージがマイグレーション元と先とで完全にコピーされる状況を考えた上で、よりシンプルな差分記録による高速な転送手法を提案した点が異なる。また、ISR であるため、ライブマイグレーションでないという点も異なる。

さらに、VMware 社の製品である VMware Storage VMotion¹⁹⁾が挙げられる。これは、ストレージを含む VM ライブマイグレーションを行うための機構である。われわれとの研究の違いは、配置先にすでに再利用可能なディスクイメージがあった場合、それを利用して高速な VM 転送を行う機構について提案している点である。

最後に、穂山ら²⁰⁾の提案を挙げる。これは、われわれの仮定と同様に、いったん転送 (VM ライブマイグレーション) した VM は再び転送元へと戻ってくる可能性が高い。というユースケースの基で、VM ステータの RAM 情報に着目し、VM の RAM 情報を転送元で保存しておいて、再び転送元へと VM が戻ってくるときは、汚れた DirtyPage のみを転送することで VM の転送速度の高速化を図っている。本研究との違いは、われわれはこれを VM のディスクストレージで行った点が異なっている。

8. おわりに

本研究では、われわれはモバイル機器を挟むような VM ライブマイグレーションでは、転送した VM が、再びもとの環境へ転送される可能性が高いことに着目し、再転送時には変更されたディスクページのみを転送する差分ディスクイメージ転送を提案した。本論文では、この転送手法をサポートするための新たな VM のディスクイメージフォーマットを設計し、Linux/KVM 上にプロトタイプを実装した。Ubuntu 11.04 (32bit) と Windows 7 Professional (32bit) の 2 つの VM 上で実験を行った結果、1Gbps の LAN 環境上にて、それぞれ 20GB のディスクイメージを持

つ VM にて約 10 分かかっていた VM のマイグレーションが数十秒から 1 分前後に縮まり、本手法が有効であることが判明した。今後の課題としてはより長期的に駆動させた VM 上にどれくらいのディスクページが汚されるのかを調査することが挙げられる。

参 考 文 献

- 1) Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M. and Noveck, D.: Network File System (NFS) version 4 Protocol (2003).
- 2) Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: the Linux Virtual Machine Monitor, *Proceedings of the Linux Symposium*, pp. 225–230 (2007).
- 3) Bellard, F.: QEMU, a fast and portable dynamic translator, *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*, Berkeley, CA, USA, USENIX Association, p. 41 (2005).
- 4) 高橋一志, 笹田耕一: 差分転送を用いた高速な VM ディスクイメージ転送手法の提案, コンピュータシステム・シンポジウム (ComSys 2011) 発表予定 (2011).
- 5) 高橋一志, 笹田耕一: WinKVM : 異なるホスト OS 間の VM ライブマイグレーション実現に向けて, 情報処理学会コンピュータシステム・シンポジウム論文集 (ComSys 2009), Vol. 2009, No. 13, 日本ソフトウェア科学会, pp. 59–66 (2009).
- 6) MokaFive: MokaFive Player, <http://www.moka5.com/>.
- 7) Sud, S., Want, R., Perring, T., Lyons, K., Rosario, B. and Gong, M. X.: Dynamic Migration of Computation through Virtualization of the Mobile Platform, *MobiCASE*, pp. 59–71 (2009).
- 8) Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live migration of virtual machines, *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, Berkeley, CA, USA, USENIX Association, pp. 273–286 (2005).
- 9) Nelson, M., Lim, B.-H. and Hutchins, G.: Fast transparent migration for virtual machines, *Proceedings of the USENIX Annual Technical Conference, ATEC '05*, Berkeley, CA, USA, USENIX Association, pp. 25–25 (2005).
- 10) Tridgell, A. and Mackerras, P.: The rsync algorithm, Technical Report TR-CS-96-05, Australian National University, Department of Computer Science (1996). <http://rsync.samba.org>.

- 11) Black, J.: Compare-by-hash: a reasoned analysis, *Proc 2006 USENIX Annual Technical Conference*, pp. 85–90 (2006).
- 12) Luo, Y., Zhang, B., Wang, X., Wang, Z., Sun, Y. and Chen, H.: Live and incremental whole-system migration of virtual machines using block-bitmap., *CLUSTER'08*, pp. 99–106 (2008).
- 13) Bradford, R., Kotsovinos, E., Feldmann, A. and Schiöberg, H.: Live wide-area migration of virtual machines including local persistent state, *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, New York, NY, USA, ACM, pp. 169–179 (2007).
- 14) Hirofuchi, T., Ogawa, H., Nakada, H., Itoh, S. and Sekiguchi, S.: A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds, *Cluster Computing and the Grid, IEEE International Symposium on*, Vol. 0, pp. 460–465 (2009).
- 15) Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S. and Rosenblum, M.: Optimizing the migration of virtual computers, *SIGOPS Oper. Syst. Rev.*, Vol. 36, pp. 377–390 (2002).
- 16) Kozuch, M., Satyanarayanan, M., Bressoud, T. and Ke, Y.: Efficient State Transfer for Internet Suspend/Resume, *Intellectual Property*, No. May (2002).
- 17) Tolia, N., Tolia, N., Bressoud, T., Bressoud, T., Kozuch, M., Kozuch, M. and Satyanarayanan, M.: Using Content Addressing to Transfer Virtual Machine State, Technical report (2002).
- 18) Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H., David and Steere, C.: Coda: A Highly available File System for a Distributed Workstation Environment, *IEEE Transactions on Computers*, Vol. 39, pp. 447–459 (1990).
- 19) VMware, Inc.: VMware Storage VMotion: Non-disruptive, live migration of virtual machine storage, <http://www.vmware.com/products/storage-vmotion/>.
- 20) 穂山空道, 広瀬崇宏, 高野了成, 本位田真一: メモリの再利用により移動後の性能低下を抑えたライブマイグレーション, 先進的計算基盤システムシンポジウム SACSIS 2011 (2011). ポスターセッション.

