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

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．従来の VM ライブマイグレーションでは，VM のディスクイメージをネットワークファイルシステムで共有することを前提としており，ネットワーク接続性を保つことができないモバイル機器へ転送するのは困難だった．最近では，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．そこで，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する．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．さまざまなワークロードを持つ VM 上で実験を行った結果，20GB のディスクイメージを持つ VM にて数十分かかっていた VM のマイグレーションが数十秒に縮まり，本手法が有効であることが示された．

A fast VM migration mechanism using duplication disk pages

Kazushi Takahashi† and Koichi Sasada

The mechanism that allows us to migrate easily Virtual Machine (VM) to other mobile devices is confortable. Traditional VM live-migration mechanism requires the network file systems to share VM disk images and it is difficult to move VMs because the mobile device may loss network connections. Recently, the new migration mechanism without the Network file systems has been developed. This mechanism packs whole VM instance, including VM disk images and migrate to another machines. However, VM disk images generally have huge file size and the huge time is required until VM translation is completed. We think that it is not comfortable. Therefore, we think that there is a greater chance of returning the source VM from the destination VM in migration. When returning the VM, the source VM already has the original VM disk image and we can send different parts of the original VM disk image to speed up. In this paper, we design new VM virtual disk format to support the different translate and implement the prototype in Linux/KVM. We examine this mechanism in various VMs that have some workloads. And we show that our different translation mechanism is benefit.

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，NFS[1] や Samba 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/KVM[2] と QEMU[3] 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

† 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

れる．ブロックマイグレーションによる VM ライブマイグレーションでは NFS や Samba のように転送先と転送元との間で VM のディスクイメージを共有する必要がない．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，Samba や NFS といったネットワークファイルシステムを用意せずとも，TCP/IP による通信環境さえ整っていれば，VM ライブマイグレーションを行うことができる．

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．もちろん，ブロックマイグレーション行っている最中でも，転送元の VM 上では作業を続けることができる．しかし，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．2 章でさらに詳しく議論するが，我々は個人用計算機環境における VM のライブマイグレーションを想定している．このような環境下において，VM の転送が完了するまで 10 分もの時間がかかるというのはユーザの利便性を大きく低下させる．

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新たなブロックマイグレーション高速化のメカニズムを提案する．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．このユーザシナリオについては 3 章にて詳しく議論する．

本論文では上記のアイデアを 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．

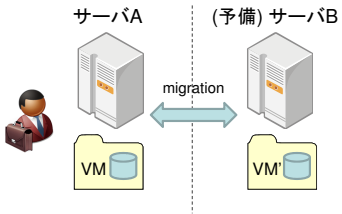

図 1 サーバ用途のマイグレーション
Fig. 1 The live-migration for server-side usage

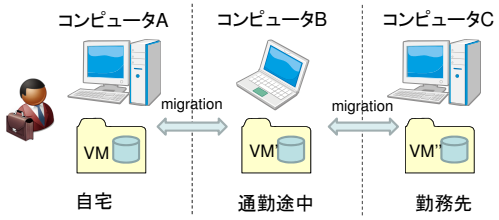

図 2 個人用途のマイグレーション
Fig. 2 The live-migration for personal usage

2. 問題分析

本章では，初めに，VM ライブマイグレーション (ブロックマイグレーション) の利用シーンについての現状を分析し，現状の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．

現在の VM ライブマイグレーション (ブロックマイグレーション) の利用シーンを**図 1** に示す．これは，VM 上で動作しているサービスを無停止で他の物理マシン上に移動させる利用法である．主に計算機に精通したサーバ管理者が利用するライブマイグレーションである．物理マシン A と B が存在していたとして，マシン A 上の VM でミッションクリティカルな Web サービスが駆動しているとする．この時，物理マシン A を停止してメンテナンスする必要性が生じたとき，VM ライブマイグレーションを使用して Web サービスを停止することなく，予備の物理マシン B 上に VM ごと移動させることができる．マシン A のメンテナンスが終わった後，再び VM マイグレーションを行い，B から A に Web サービスを移行させる．管理者は Web サービスを停止させることなく，物理マシンのメンテナンスを行うことができる．現在の VM ライブマイグレーションは，主にこの利用シーンを想定して設計が行われている．

一方で，われわれが想定してる利用シーンを**図 2** に示す．これは，上述した VM ライブマイグレーションとは少し異なるユーザシナリオであり，計算機に詳しくない個人用の計算機環境を対象とした VM ライブマイグレーションである．個人用の VM 環境に対して

VM ライブマイグレーションを活用する研究例はすでに存在している．例えば，Intel の Shivani Sud らは 2009 年のレポート[4] にて個人用計算機環境における VM ライブマイグレーションの活用方法について模索している．彼らは性能の低い Atom プロセッサを搭載したネットブックと，性能の高い Core Duo プロセッサを搭載したラップトップ型計算機を使用し，両者を無線 LAN で接続し，その上で Linux/KVM を使った VM ライブマイグレーションを行い評価を行っている．

個人用の計算機環境を対象とした VM ライブマイグレーションについて詳しく述べる．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[5]．このような VM を中心としたコンピューティング環境にはいくつかの利点が存在する．計算機のバックアップも VM のディスクイメージをコピーするだけで，容易にバックアップすることが可能である．また，ウイルスの感染によるシステムの破壊が起こったとしても，バックアップを行った VM ディスクイメージから簡単に作業環境を復帰することができる．さらに，VM ディスクイメージを持ち運ぶことで，Hypervisor がインストールされている環境さえあれば，いつでもどこでも自分の作業環境を再現することができる．このように，VM を中心とした個人用のコンピューティング環境は今後普及していくことが考えられる．

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．これは，上述した Intel の Shivani Sud らのレポート[4] も同様の指摘を行っている．図 2 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．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の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，通期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．

このような個人用の VM ライブマイグレーションでは，個人が容易に使用できるマイグレーションのメカニズムを考えることが重要である．そのため，ネットワークファイルシステムによる VM のディスクイメージ共有が前提となっている VM ライブマイグレーションは好ましくない．しかし，この問題は VM のディスクイメージごと転送するブロックマイグレーションによってすでに解決されている．ブロックマイグレーションでは TCP/IP による通信環境さえ整っていればディスクの転送が可能であるため，煩雑なネットワーク環境を構築することなく，極めて容易に VM を転送することが可能である．

しかし，われわれが注目する利用シーンにおいて，既存のブロックマイグレーションにも依然として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[6),7)] 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．初期転送時にはすべてのディスクイメージを転送しなければならず，それには従来のブロックマイグ

レーションと同様の時間がかかるが，それ以降の転送を考えたとき，転送元には転送先の VM ディスクイメージが残されていることになる．そのため，転送先から転送元へと，再び VM を移動させるときは，転送先で汚されたディスクページのみを転送すればよい．さらに，転送先でディスクページが大量に汚れることは考えにくく，ごくごく一部のページが汚されるにとどまるのではないかと考えられる．そのため，Hypervisor 上ですべてのディスク書き込みをトラッキングしておき，汚された一部のページのみを転送すれば高速な転送が可能である．

われわれが主眼におく個人用計算機における VM ライブマイグレーションの例をもとにこのユーザシナリオについて議論する．図 2 に示すように，A さんは自宅の計算機と通勤途中の携帯端末，勤務先の計算機との間でそれぞれ独立した計算機 A，B，C を所持している．A さんが自宅の計算機から通勤途中で使用する B に VM を転送する．その後，オフィスについた後，通勤途中で使用していた計算機 B から，オフィスで使用している計算機 C に転送する．その後，仕事が終わり，帰宅時間になったとき，オフィスで使っている計算機 C から，通勤途中で使用している計算機 B に再び VM を転送する．帰宅途中は計算機 B を使って作業を行い，帰宅した後は，計算機 B から，自宅で使用している計算機 C に VM を転送する．

このように，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 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れ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 DirtyPage トラッキング機能を付加し，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，通常のブ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 DirtyPage トラッキングによって検出された Dirty ページのみを転送する．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．

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，ディスクの DirtyPage トラッキング機構の設計について述べ，次に，DirtyPage トラッキングを支援するための 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．

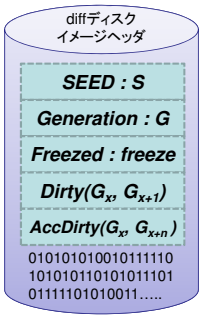

図 3　diff ディスクイメージのヘッダ
Fig. 3　The structure of diff diskimage

4.1 Dirtypage トラッキング

DirtyPage トラッキングは，ディスクの全書き込みを追跡し DirtyPageBitmap にその記録を残すための機構である．ディスクの DirtyPage トラッキングは 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 1 ビットとして DirtyPageBitmap を構成することで DirtyPage トラッキングを実現する．

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，DirtyPage トラッキングで得た DirtyPageBitmap 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 3** に示す．

diff イメージはヘッダーに世代番号 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，1 世代前の差分用 DirtyBitmap である $Dirty(G_x, G_{x+1})$ と，n 世代前からの差分を記録した DirtyBitmap である $AccDirty(G_x, G_{x+n})$ がヘッダーに格納されている．

$A \longleftrightarrow B \longleftrightarrow C$ という 3 つの仮想マシン間で我々の diff イメージを用いた差分転送メカニズムがどのように動作するかを解説する．

図 4 に $A \longrightarrow B \longrightarrow C$ 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 (UUIDD : 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☆．世代 G は一代目であるので G_0 となる．まず初めに，$A \longrightarrow B$ 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

☆ 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．

のブロックマイグレーションが行われる．$A \longrightarrow B$ 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．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 からは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 G_0,S_0, $freeze = 1$ の diff ディスクイメージが残ることになる．一方で B には G_1, S_0,$freeze = 0$ の diff ディスクイメージが作られる．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．最終的に B 上には G_0 と G_1 の差分を記録した DirtyPageBitmap である $Dirty(G_0, G_1)$ が作られ B 上の diff ディスクイメージに記録される．

次に，$B \longrightarrow C$ 間の転送がどのように行われるのかについて解説する．$A \longrightarrow B$ 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る．このマイグレーションが終わった後，C 上には G_2，S_0, $freeze = 0$ の diff イメージファイルが作られる．C 上で作業が進むにつれ，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いく．最終的に C 上には G_1 と G_2 の差分を記録した DirtyPageBitmap である $Dirty(G_1, G_2)$ が作られ，$Dirty(G_0, G_1)$ に上書きされる．さらに，世代番号 G_0 と G_2 の差分を記録した DirtyPageBitmap である $AccDirty(G_0, G_2)$ も作られ記録される．このとき，$AccDirty(G_0, G_2) = Dirty(G_1, G_2) \cup Dirty(G_0, G_1)$ である．B 上の diff イメージファイルは $freeze = 1$ となり，これも Hypervisor から一時的に起動することができなくなる．

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るたび，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，diff ディスクイメージは常に 1 世代前との差分を記録した $Dirty(G_x, G_{x+1})$ と，G_0 から n 世代までの差分を記録した $AccDirty(G_x, G_{x+n})$ の 2 つの DirtyPageBigtmap を記録している．

さらに，**図 5** に $C \longrightarrow B \longrightarrow A$ 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，$C \longrightarrow B$ 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 G_1diff 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 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配置転送は転送は差分転送は不可能であると判定する．次に，G_1 と G_2 の差分を記録した $Dirty(G_1, G_2)$ を見て，汚れているページのみを転送

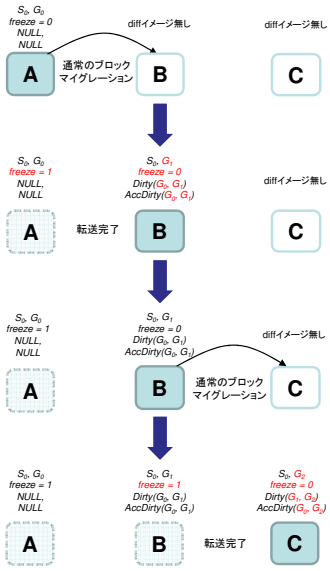

図 4 $A \longrightarrow B \longrightarrow C$ 間のマイグレーション
Fig. 4 The migration between $A \longrightarrow B \longrightarrow C$

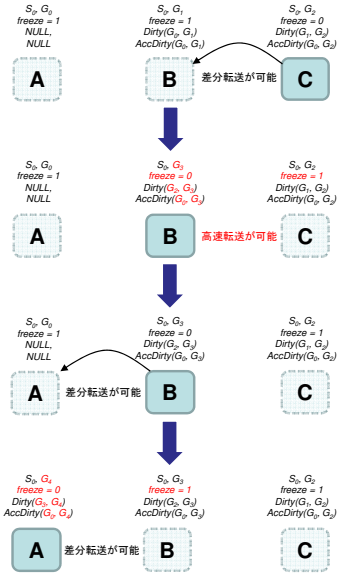

図 5 $C \longrightarrow B \longrightarrow A$ 間のマイグレーション
Fig. 5 The migration between $C \longrightarrow B \longrightarrow A$

する．これで高速な転送が可能である．

次に，$B \longrightarrow A$ の転送を考える．この場合，C を経由したことで世代を 1 つ飛び越した転送となるため，$Dirty(G_2, G_3)$ を使うことができない．そのため G_2 と G_3 の差分を記録した $Dirty(G_2, G_3)$ を見るのではなく，累積差分を記録した $AccDirty(G_0, G_3)$ を使用して転送を行う．

最後に，freezed の役割について詳細に述べる．$freezed = 1$ となった diff ディスクイメージは基本

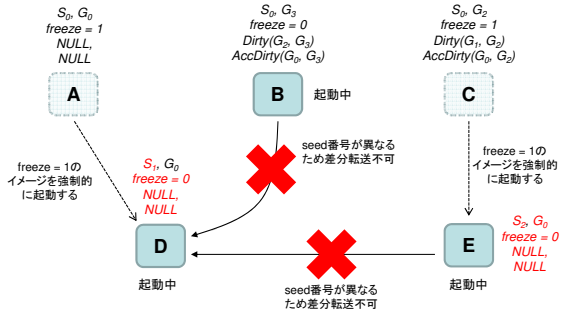

図 6 ディスクイメージの freezed を解除した場合の各仮想マシンの挙動

Fig. 6 The behavior of VMs, if users unfreezed vm disk images

的に起動不可能である．$freezed = 1$ 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 $freezed = 1$ 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 6** 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 $freezed = 1$ は不可能となる．ユーザが，$freezed = 1$ 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は新たな SEED 番号である S_1 を割り振る．次に，$Dirty(G_x, G_{x+1})$ と $AccDirty(G_x, G_{x+n})$ も両方 NULL にクリアされる．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．このように，仮想マシン C 上にあるディスクを強制的に起動したとすると，同様に，新たな SEED 番号である S_2 が割り振られ，$Dirty(G_x, G_{x+1})$ と $AccDirty(G_x, G_{x+n})$ も両方 NULL にクリアされる．これも，SEED 番号が変わったため，まったく新しいディスクイメージとみなされ，D への高速なブロックマイグレーションは不可能になる．

4.3 議　　論

4.2 節で議論した手法には問題点がある．それは．A, B, C の間で VM が移動を繰り返してゆくと，n 世代前からの差分を記録した DirtyBitmap である $AccDirty(G_x, G_{x+n})$ が徐々に汚れてゆき，最終的にはすべて汚れてしまいことで，ディスクをすべて転送するのと時間的に見てあまり変わらなくなってしまうという問題点がある．これを解決するためには 2 つの手法が存在する．1 つ目の手法は DirtyBitmap による転送の代わりに `rsync`[8] と同様の手法である *Compare-by-hash*[9] の手法を導入することである．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この問題は解決する．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．2 つ目の手法としては，ある程度 *AccDirty* が汚れてしまったら，*Dirty*, *AccDirty* ともどもクリアし，Seed 番号 S を新しい値に変更してしまうという手法が考えられる．これにより，再び，ディスクをすべて転送しなければならなくなるが，*AccDirty* は NULL になるので，汚れはすべて消え去る．ディスクのデフラグと同様に，ユーザに対して，VM の転送が遅くなってきたと感じられたらこの操作を行ってもらうという方法をとることでこの問題を解決できる．われわれは転送速度の高速化を優先するため今回は後者の手法を選んだ．

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 $Dirty(G_x, G_{x+1})$ と $AccDirty(G_x, G_{x+n})$ を実装するための DirtyPage トラッキング機能である．上記の二つの DirtyPageBitmap 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 つにつきを 1 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，500 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 2Kbyte 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 DirtyPageBitmap 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 DirtyPageBitmap 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

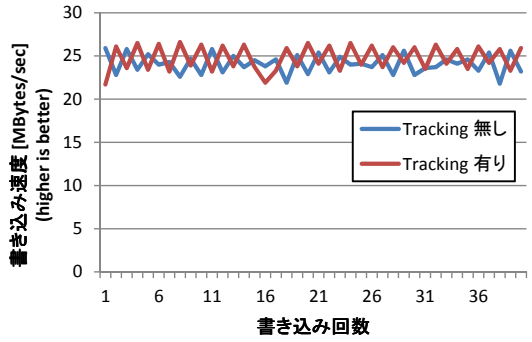

図 7 ビットマップ更新時における dd に書き込み速度
Fig. 7 Writing cost by dd to update bitmaps

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 つ目の評価はダーティページのビットマップである $Dirty(G_x, G_{x+1})$ と $AccDirty(G_x, G_{x+n})$ を更新する処理が 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持つ 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，ダーティページのビットマップである $Dirty(G_x, G_{x+1})$ と $AccDirty(G_x, G_{x+n})$ 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，DELL LATITUDE D630 ラップトップコンピュータを使用した．CPU は Intel Core2Duo T7300 2.0GHz で，物理メモリは 2GB である．HDD は WDC 製の 7200rpm HDD で 8MB のキャッシュメモリが搭載されている．ビット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 40 回行った．評価結果を**図 7** に示す．縦軸が書き込み速度 MBytes/sec で，横が書き込み回数である．グラフをみると，トラッキング有りとトラッキング無しで，書き込み速度にそれぞれ周期があるのがみられる．トラッキング有りのときの書き込みがトラッキング無しのときよりも高速な時があるが，これはキャッシュの関係だと考えられる．それぞれの平均速度は，トラッキング無しのときで 24.08MBytes/sec で，トラッキング有りの時が 24.77MBytes/sec である．そのため，トラッキングが致命的な書き込み速度低下を起こすことはないということが確認できた．

6.2 差分転送によるブロックマイグレーションの性能評価

本評価の目的は，現実的な VM のワークロードをいくつか想定した上で，どの程度ディスクが汚されるのか．そして，差分転送による転送がどの程度有効なのかを確かめることが目的である．

評価環境は，転送元が Lenovo の ThinkPad X200 ラップトップコンピュータで，CPU は Intel Core 2 Duo CPU P8400 @ 2.26GHz で 4GBytes の物理メモリが搭載されている．転送先は 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．

VM は 2 つ用意した．1 つ目は Windows 7 Professional (32bit) をインストールした VM である．2 つ目は ubuntu 11.04 desktop edition (32bit) をインストールした VM である．それぞれの VM には 2GB のメモリを割り当て，ディスク容量は 20Gbyte の容量を割り当てた．この 2 つの VM は，それぞれ，NTFS, ext4 といった異なるファイルシステムを持っている．ファイルシステムの違いによって同一のオペレーションを行ってもディスクが汚されるパターンが異なることが考えられる．そのため，NTFS, ext4 といったよく使用される代表的な 2 つのファイルフォーマットを用意してそれぞれ評価を行った．

6.2.1 ブートによるディスクブロックの変化

VM のディスクイメージは単に起動しただけでも書き込みが行われる．差分転送による速度測定の効果を確かめる前に，われわれは，ブートしただけで，Windows 7 と ubuntu 11.04 が両 OS がどの程度ディスクを汚すのかについて調べた．起動を行った後ログインし，仮想マシンを 10 分間放置した後でどの程度ブロックが汚れるのかについての調査を行った．

図 8 に Ubuntu 11.04 のブートからシャットダウンまでの間，どの程度のディスク書き込みが行われたかのグラフを示す．ブートが完了してログインを行った瞬間，ディスクは 10 個のディスクブロック (10MB 分) に対するディスク書き込みが行われた．その後，デスクトップ環境が立ち上がり，しばらく放置したところ，シャットダウンを行う前までに，累計で 43 個分のディスクブロックに対しての書き込みが行われたことを確認した．

次に，**図 9** に Windows 7 Professional のブートからシャットダウンまでの間に，どの程度のディスク書き込みが行われたかのグラフを示す．Ubuntu 11.04 と比べると，Windows 7 はブートシーケンスの処理だけでも 42 個のディスクブロックに対して書き込みが行われており，Ubuntu 11.04 と比べると多いことがわかる．最終的にシャットダウンが行われるまでには 208 個のディスクブロックが汚されている．Windows

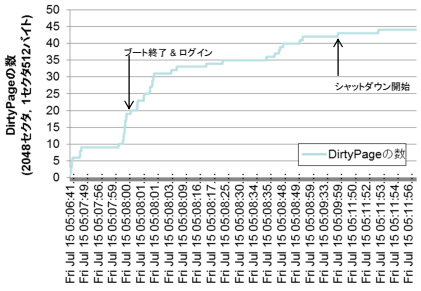

図 8 Ubuntu 11.04 ブートシークエンスによるディスクの更新
Fig. 8 Ubuntu 11.04 disk updating status by the boot sequence

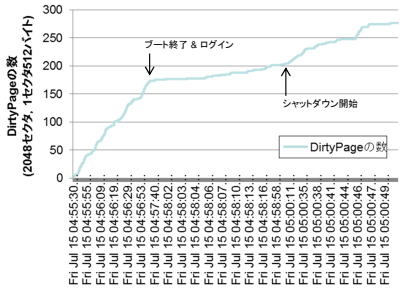

図 9 Winows 7 ブートシークエンスによるディスクの更新
Fig. 9 Windows 7 disk updating status by the boot sequence

7 は Ubuntu 11.04 よりも多くのディスクブロックに対して書き込みを行うことがわかる．

まとめると，ただブートしただけでも，Windows 7 は Ubuntu と比べて多くのディスクブロック書き込みが行われ，差分転送による転送データ量も Windows 7 のほうが比較的大きくなることが判明した．

6.2.2 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．

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 YouTube の 41.660 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 26.979 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プレゼンテーションの 59.486 秒である．一方で，時間がもっともかかっていないのは，Ubuntu の時と同様「こころ」の 39.197 秒である．しかし，全体的に見ても約 10 分間かかっていた VM の転送が，差分転送の効果で数十秒に低下していることは同様である．

まとめると，差分転送の効果により，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わかった．

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．Luo[10] や Bradford[11] 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，広渕ら[12] 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/ndb0` 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，Sapuntzakis[13] 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

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	こころ
マイグレーション全体時間 (sec)	643.001	32.017	35.558	41.660	26.979
差分転送によるディスク転送量 (MBytes)	—	100	127	164	105

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	こころ
マイグレーション全体時間 (sec)	662.326	42.001	59.486	48.185	39.197
差分転送によるディスク転送量 (MBytes)	—	234	323	334	225

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う手法である．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．Kozuch[14] と Tolia[15] らは，Coda[16] 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 のステート転送が行われ，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 のステート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 VMotion[17] 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，穐山ら[18] 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．というユーザシナリオの基で，VM ステートの 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 上にて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) Sud, S., Want, R., Pering, T., Lyons, K., Rosario, B. and Gong, M. X.: Dynamic Migration of Computation through Virtualization of the Mobile Platform, *MobiCASE*, pp. 59–71 (2009).
5) MokaFive: MokaFive Player, `http://www.moka5.com/`.
6) 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).
7) 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).
8) 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.
9) Black, J.: Compare-by-hash: a reasoned analysis, *Proc 2006 USENIX Annual Technical Conference*, pp. 85–90 (2006).
10) 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).
11) 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).
12) 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).
13) 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).
14) Kozuch, M., Satyanarayanan, M., Bressoud, T. and Ke, Y.: Efficient State Transfer for Internet Suspend/Resume, *Intellectual Property*, No. May (2002).
15) 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).
16) 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).
17) VMware, Inc.: VMware Storage VMotion: Non-disruptive, live migration of virtual machine storage, `http://www.vmware.com/products/storage-vmotion/`.
18) 穐山空道, 広渕崇宏, 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.