Vol.2009-OS-112 No.2
2009/8/5

仮想マシン技術を用いた OS 再起動のダウンタイム削減手法

山田浩史[†1,†2] 河野健二[†1,†2]

オペレーティングシステム (OS) の複雑化や多様化に伴い，バグ修正や新たな機能の組込みといった OS のアップデートが頻繁に起きている．OS のアップデートを適用するときには，一般的には OS 全体を再起動する必要がある．そのため，OS をアップデートするには，ユーザは再起動が招くダウンタイムを被らねばならない．このダウンタイムの長さはユーザの積極的なアップデートの妨げとなり得るため，可能な限り短いことが望ましい．本論文では，仮想マシン技術を用いた OS 再起動のダウンタイム削減手法，*BackGroundReboot* を提案する．提案手法では，OS 再起動の操作を，OS 再起動直後のスナップショットへの状態復元にすり替えることで，再起動に近い効果を得つつダウンタイムを削減する．提案手法を VirtualBox 1.6.0_OSE 上に実装したところ，予備実験において，再起動に要するダウンタイムを最悪でも約 78 %削減した．

A VMM-level Approach to Shortening the downtime of Operating Systems' Reboots

HIROSHI YAMADA[†1,†2] and KENJI KONO[†1,†2]

Operating System (OS) reboots are an essential procedure for patches and updates to commodity OSes in order to add new features, enhance performance, and fix bugs and security holes. Such patches and updates are announced at a higher frequency, and applying them requires rebooting the OS, resulting in downtime that is becoming increasingly costly. Surprisingly, an OS reboot is also demanded even by the update of recent applications such as Internet Explore and iTunes. This paper describes *BackGroundReboot*, a virtual machine monitor (VMM-) approach to shortening the downtime of OSes' reboot. The key idea is to make better use of snapshot technology. BackGroundReboot replaces an OS reboot procedure with restoring a VM state to the snapshot state where the newer OS is running. By doing so, we obtain an effect similar to the reboots of an OS with shorter downtime. Our prototype implemented on VirtualBox 1.6.0_OSE reduces the downtime of OS reboots by at least nearly 78% in our preliminary experiment.

1. はじめに

オペレーティングシステム (OS) の多様化や複雑化に伴い，OS カーネルのアップデートの頻度がますます高まっている．OS のアップデートには，新たな機能のインストールやパフォーマンスの向上，バグやセキュリティホールの修正などが含まれる．一般的に，OS をアップデートする場合，OS 全体を再起動することが必要不可欠である．古いバージョンの OS を停止し，新しいバージョンの OS を起動することでユーザはアップデートされた OS を使用できる．たとえば，Windows XP では，Windows Update という Web サイトからアップデートを取得し，OS のアップデートを実行する．そして，現在稼働している Windows XP に再起動を指示することで，新しいバージョンの Windows XP を使用できる．

OS のアップデートを適用する際には，OS 全体の再起動によってダウンタイムが生じてしまう．OS が再起動している間，ユーザは PC を利用することができない．この PC を利用できない期間をダウンタイムを呼ぶ．OS の再起動には物理デバイスや各種デーモンの停止や初期化，起動を行うため，再起動によって生じるダウンタイムは短くない．OS のアップデートの流れを図 1 に示す．OS にアップデートを適用した後，新しいバージョンの OS を起動するためにシステム全体の再起動を行う．この再起動には，物理デバイスやデーモンの停止や起動も含まれる．そして，ログイン画面が表示されて初めてユーザは PC を操作できるようになる．ユーザは再起動を指示した後，ログイン画面が表示されるまで PC を使用することができない．通常の再起動において，この期間がユーザの被るダウンタイムとなる．OS のアップデートの頻度も高まっていることから，結果として，ユーザが被るダウンタイムは長くなりつつある．

OS の再起動によるダウンタイムが長いと，ユーザの積極的なアップデートの妨げとなり得る．最悪の場合，ユーザはセキュリティホールが修正されていない OS を利用しつづけてしまう．また，近年では Web ブラウザや音楽プレーヤといったマルチメディアアプリケーションをアップデートする際にも，OS 全体の再起動を求められることが少なくない．たとえば，Windows XP 上で Internet Explore や iTunes といったアプリケーションをアップデートすると，レジストリファイルを書き換えるため，Windows XP 全体の再起動が求められる．

†1 慶應義塾大学
Keio University
†2 科学技術振興機構 CREST
CREST, Japan Science and Technology Agency

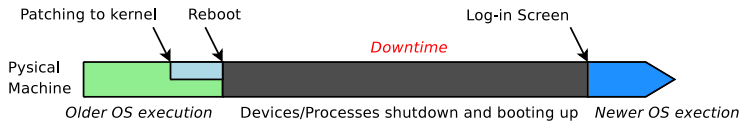

図 1 OS のアップデート．図は OS のアップデートの流れを示す．OS カーネルにパッチを適用したあと，アップデートした OS を起動するためにシステム全体を再起動する．再起動をすることで，古いバージョンの OS を停止し，新しいバージョンの OS を起動する．その間には物理デバイスやデーモンの停止や初期化，起動が行われる．この期間がユーザの被るダウンタイムとなる．

そのため，ユーザは新しいバージョンのアプリケーションを利用する際にも，長いダウンタイムを被らなければならない．

OS のアップデートにより生じるダウンタイムを削減するために，これまで OS カーネルを動的にアップデート可能にする手法が提案されているが，これらの手法を適用することは容易ではない．なぜなら，これらの手法を利用するには，OS カーネルのソースコードレベルでの深い知識が必要となるからである．既存手法は，OS カーネルの改変[1),2)] や OS カーネル内の振る舞いを熟知した上での専用パッチの記述[3),4)] を要する．そのため，動的アップデートの機能を既存の OS に組み込むことは容易ではない．OS カーネルは肥大化，複雑化し続けており，OS カーネルの解析や修正は非常にコストが高い．また，ソースコードの入手や改変が困難な商用 OS への適用は現実的に不可能である．実際に，これまで Windows や Mac OS といった商用 OS を扱った再起動に伴うダウンタイムを削減する機構は提案されていない．

上述の問題を解決するために，本論文では 2 つの目標を定める．1 つめはコモディティ OS の再起動が招くダウンタイムを削減することである．再起動によるダウンタイムを短くすることで，ユーザの積極的なアップデートを促す．2 つめは提案手法を適用する際に OS カーネルのソースコードレベルの深い知識を要さないことである．OS の改変や専用パッチの記述を要さないことで，提案手法の導入コストを低く保つ．

本論文では OS カーネルを修正することなく OS の再起動が招くダウンタイムを削減する手法，*BackGroundReboot* を提案する．提案手法では仮想マシン (VM) 技術によって提供されるスナップショット機能を応用する．具体的には，OS の再起動操作を，再起動直後のスナップショットからの状態復元にすり替える．OS 再起動直後のスナップショットを取得するために，使用している VM のクローンを生成し，クローン側で OS の再起動を行う．クローン側で OS を再起動している間，ユーザはクローン元でアプリケーションの使用が可能である．これにより，ユーザが VM を使用しながら，OS の再起動直後のスナップショットを取得可能にしている．

提案手法を VirtualBox 1.6.0_OSE 上に実装し実験を行ったところ，再起動に要するダウンタイムを最悪でも約 78%削減することができた．

本論文の構成は以下のとおりである．2 章で関連研究について述べ，本研究の動機について述べる．3 章でアプローチについて述べ，4 章で BackGroundReboot の設計を示す．5 章で実装について述べ，6 章で予備実験の結果について示す．そして 7 章で本論文をまとめる．

2. 関 連 研 究

これまで，OS やアプリケーションのアップデートに伴う OS 再起動のダウンタイムを削減するために様々な手法が提案されている．K42[1),2)] は OS カーネルのアップデートを再起動することなく動的に適用できる OS である．カーネルは C++ で記述され，カーネルの管理する資源をオブジェクトで表現する．各オブジェクトへはオブジェクト変換表を参照することで間接的にアクセスする．OS のあるコンポーネントをアップデートするときには，対応する新たなオブジェクトを生成し，オブジェクト変換表内の古いオブジェクトのエントリを新たなオブジェクトを指すよう書き換える．こうすることで，OS 全体を再起動することなく新たなコンポーネントを使用することができる．

OS の実行中のコードを動的に変更することで，再起動を要することなく OS をアップデートする手法がある．LUCOS[3)] は仮想マシン技術を利用した OS の動的アップデート機構である．OS をアップデートするときには，OS カーネル内に常駐している Update manager にユーザレベルから専用パッチを送信する．Update manager は新たなコードを OS カーネル内に生成した後，ハイパーコール (hypercall) を発行することで，仮想マシンモニタ (VMM) 内で動作する Update server にアップデート要求を通知する．ハイパーコール発行後，制御は VMM に移り，通知を受けた Update server は OS カーネルの実行中のコードを編集する．たとえば，関数をアップデートする場合，Update manager は新たな関数を OS カーネルのメモリ内に作成し，Update server はアップデート対象の関数の先頭 5 バイトを新たな関数へのジャンプ命令に書き換え，制御を OS へ戻す．

DynAMOS[4)] はスレッドから常に参照されているカーネル関数を動的にアップデートする機構である．DynAMOS では，OS をアップデートする際にアダプションハンドラと呼ばれるハンドラを登録する．アダプションハンドラは，ある関数のバージョンを複数保持し，システムの状況に応じて使用するバージョンを指定できる．これを利用することで，たとえ

システム内のスレッドがアップデート対象のカーネル関数を使用していたとしても，安全に新しい関数へ移行できる．たとえば，ある処理の無限ループを含むカーネル関数にアップデートが生じたとする．多くの場合，Linux 上において，このような無限ループ中には `interruptible_sleep_on()` が含まれている．この性質を利用し，新しい関数に移行するための `interruptible_sleep_on()` を用意する，その関数の中で，カーネルスレッドの状態を保存し，現在のカーネルスレッドを終了させ，新たなカーネルスレッドを生成する．そして，そのスレッドにアップデート後の関数を実行させる．

Ksplice[5] はオブジェクトファイルの内容変化に着目した OS の動的アップデートを実現する機構である．Linux を対象に実装しており，OS をアップデートをする際，パッチ適用前の OS カーネルのオブジェクトファイルと適用後のオブジェクトファイルを用意する．そして両者の内容を比べ，アップデートが適用された関数のオブジェクトコードをパッチ適用後のオブジェクトファイルから抽出する．そして，抽出したオブジェクトコードを現在稼働している Linux カーネルとリンクする．リンク後，システム内の全てのスレッドを停止する `stop_machine()` を発行し，アップデート対象の関数がシステム内のスレッドに使用されているか否かを検査する．検査の結果，全スレッドが対象の関数を使用していなければ，アップデート前の関数の先頭をアップデートが適用された関数へのジャンプ命令に書き換える．

MicroVisor[6] や AutoPod[7] はプロセス移送を利用することで，OS アップデートに伴うサービスのダウンタイムを削減する手法である．MicroVisor は，OS が物理マシン上で動作している環境に対して，VMM を動的に挿入する機構である．OS をメンテナンスするときには，VMM を挿入し，現在稼働している OS とは別に，アップデートをする OS を同一ホスト上に生成しアップデートを実行する．アップデートしている間，これまで稼働していた OS 上でサービスを提供する．アップデートが完了すると，これまで稼働していた OS 上のプロセスをアップデートが完了した OS に移送する．そして，これまで利用していた OS を停止させ，VMM を動的に削除する．AutoPod では OS のアップデートを行うとき，稼働しているプロセスを別のマシンへ移送させることでサービスの中断を最小限に抑える．OS のアップデートが完了すると，別マシンで稼働していたプロセスを元のマシンへ移送する．

上述手法を利用すれば OS 再起動に伴うダウンタイムを削減できる反面，利用するには OS カーネルのソースコードの入手やソースコードレベルの深い知識を要するといった問題点がある．そのため，提案手法を適用することが容易ではない．K42 で提供する動的アップデート機能を使用するには，現在利用している OS から K 42 へ変更しなければならず，ユーザは実行環境を大きく変更する必要がある．また，既存 OS へ K42 で提供されている同等の機能を組み込む場合，既存の OS の多くは C やアセンブリで記述されてるため，OS の構造を大幅に修正する必要がある．

LUCOS や DynAMOS では，専用パッチを作成する必要があり，その際には対象 OS カーネルのソースコードレベルでの深い知識が必要となる．Ksplice や AutoPod を利用するためには，OS に対してカーネルモジュールを挿入する必要がある．そのため，提案機構を他の OS へ移植する際には，改めて OS カーネルの解析が必要になる．OS カーネルの構造は OS の種類によって大きく異なるため，そのままカーネルモジュールを適用することは困難である．MicroVisor の恩恵を享受するには，プロセス移送の機能を OS が提供する必要がある．多くのコモディティ OS はプロセス移送の機能を備えていないため，やはり OS の拡張が必要になるため，導入や管理コストが高いと言える．

様々な手法が提案されているにもかかわらず，これまで Windows や MacOS X といった商用 OS を扱った OS 再起動の招くダウンタイムを削減する手法は提案されていない．そこで，本論文では OS カーネルに依存することなく，OS 再起動によって生じるダウンタイムを削減する手法の確立を目指す．

3. アプローチ

本論文ではカーネルを修正せずに OS 再起動が招くダウンタイムを削減する手法，*BackGroundReboot* を提案する．OS カーネルに依存しない機構を提供することで，様々な OS への容易な転用を可能にする．本論文では，提案機構を Linux に対してのみしか適用していないが，Windows といった商用 OS にも原理的に適用可能である．他 OS への適用は今後の課題である．

提案手法では仮想マシン技術を用いる．仮想化環境において，OS は仮想マシン (VM) と呼ばれる仮想的な物理マシン上で稼働し，VM は仮想マシンモニタ (VMM) というソフトウェアが管理をする．仮想マシンモニタ (VMM) 内に新たなサービスを実現すると，VMM 上で動作する OS 全てに対してその恩恵を与えることができる[8]．また，近年の VMM は OS に修正を施すことなく実行できる．

BackGroundReboot では仮想マシン技術によって提供されるスナップショット機能を応用することで，OS カーネルに修正を加えることなく再起動に伴うダウンタイムを削減する．具体的には，OS 再起動後のスナップショットを取得して，現在使用している VM の状態をそのスナップショットの状態にすることで，再起動に近い効果を得つつ，ダウンタイムを削

表 1 OS 再起動とスナップショットへの復元に要する時間の比較．OS の再起動とスナップショット機能による復元に必要な時間を比較した．スナップショットへの復元 (SN への復元) に要する時間の方が OS 再起動に要する時間より短い．ここで，再起動に要する時間は再起動の手続きを行ってからログイン画面が表示されるまでの時間を示す．

VM に割り当てたメモリ量	OS 再起動 (second)	SN への復元 (second)
128 MB	79.7	1.4
256 MB	80.3	1.3
768 MB	79.7	1.3

減する．

スナップショット機能とは VM の状態を過去の時点のものに任意のタイミングで復元できる機能である．ある時点の VM のメモリ内容ならびにディスクドライブ内容をスナップショットとして保存する．スナップショットを取得した後も VM は稼働し続けることができる．稼働し続けている VM に対してスナップショットを適用すると，VM はスナップショットからメモリ内容とディスクドライブ内容を復元し，スナップショットを取得した時点の状態へと戻る．過去にさかのぼって状態を復元するという点で，ある時点で VM を中断し，その後中断した時点の状態から VM を再開を行うサスペンド & リジューム機能とは異なる．スナップショット機能は，一般的にシステムの更新や設定変更など，深刻な結果を招く可能性のある操作に先立ってバックアップのための機能として用いられることが多い．本研究では OS 再起動に伴うダウンタイムを削減するための手法として応用する．

スナップショット機能に要する時間を測定するために予備実験を行った．予備実験で用いたマシンは Pentium D 3.0 GHz の CPU，1 GB のメモリ，SATA で接続されたハードディスクを備え，Linux 2.6.18 が稼働している．Linux 上では，VirtualBox 1.6.0_OSE が稼働しており，VM として Gentoo Linux 2007.0 (`gentoo`) が稼働している．起動するデーモンは通常のインストール手順を踏んだ後，変更していない．VM は 20 GB の仮想ディスクドライブを備えている．VM に与えるメモリ量を変化させながら，2 種類の時間を計測した．まず VM 上で稼働している OS の再起動に要する時間を計測した．ここでは，再起動に要する時間は，再起動を指示してからログイン画面が表示されるまでの時間としている．次にログイン画面が表示した時点でのスナップショットを予め保存しておき，そのスナップショットに復元する時間を計測した．

結果を表 1 に示す．`gentoo` を VM 上で再起動すると，与えたメモリ量とは関係なく，約 80 秒を要した．一方，スナップショットへの復元は，こちらも与えたメモリ量とは関係なく，高々約 1.4 秒にとどまった．これはログインプロンプトが現れるまでに使用されるメモリ量が搭載しているメモリ量に関係なくほぼ一定であるためである．予備実験より，スナップショット機能による状態復元を応用することで，再起動に伴うダウンタイムの削減を期待することができる．

4. BackGroundReboot

4.1 基 本 設 計

BackGroundReboot では，VM のクローンを生成し，クローン側で OS の再起動を行う．そして，クローン側で再起動後のスナップショットを取得し，クローン元の VM をスナップショットの状態に復元する．具体的には，BackGroundReboot は次の 4 つのフェーズからなる．

(1) OS カーネルにパッチを適用した後，現在使用している VM のクローンを生成する．以下，クローン元の VM を親 VM，クローン側の VM を子 VM と呼ぶ．子 VM は親 VM のメモリ内容や仮想ディスクドライブ内容といった実行状態を引き継ぐ．ここでの親子関係は Unix 上における `fork()` を呼び出した際にできるプロセスの親子関係と同等である．

(2) 子 VM 内の OS を再起動する．子 VM で再起動が行われている間，ユーザは親 VM を操作し続けることができる．子 VM 内では古いバージョンの OS が停止し，新しいバージョンの OS が起動する．

(3) 子 VM 内で稼働している OS の起動が完了した時点で 子 VM のスナップショットを取得する．スナップショット取得後，子 VM を破棄する．

(4) 親 VM の状態を第 3 フェーズで取得したスナップショットに復元する．スナップショットへ復元することで，親 VM は新たなバージョンの OS が稼働している状態となる．

BackGroundReboot の動作の概略を図 2 に示す．OS の再起動操作の代わりに，子 VM を生成し，そちら側で OS を再起動する．子 VM 上で OS が再起動している間，親 VM はアプリケーションを実行し続けることができる．子 VM の再起動が完了すると，子 VM のスナップショットを取得する．そして，親 VM を取得したスナップショットの状態にする．提案手法は，従来の再起動操作と異なり，デバイスやデーモンのシャットダウンや初期化のフェーズ中においても，ユーザは親 VM を利用してアプリケーションを実行できる．BackGroundReboot で被るダウンタイムは，第 1 フェーズでの VM のクローン生成時に生じるダウンタイムと第 4 フェーズの OS 再起動後のスナップショットへの復元時に生じるダウンタイムである．

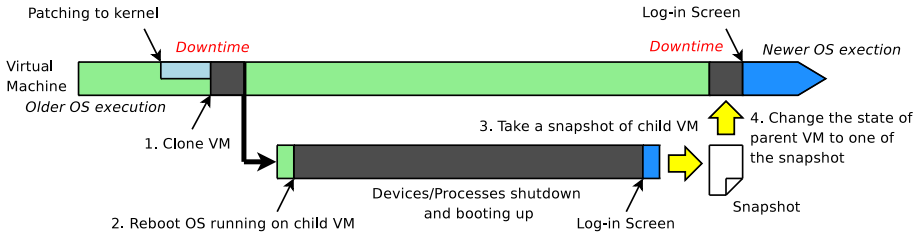

図 2 **BackGroundReboot**. 図は BackGroundReboot の流れを示す．OS カーネルにパッチを適用した後，VM のクローンを生成して，クローン側 (子 VM) で OS の再起動を行う．子 VM で再起動している間，ユーザはクローン元の VM (親 VM) を使用することができる．子 VM の再起動が終了し，新しい OS が起動してログイン画面が表示されたところでスナップショットを取得する．そして，親 VM をスナップショットの状態に復元する．提案手法でユーザが被るダウンタイムは，VM のクローンとスナップショットの状態への復元に要する時間である．

ここで，BackGroundReboot は OS 再起動に伴うダウンタイムを削減することが目的であることに注意されたい．BackGroundReboot では，OS を再起動する代わりに，OS 再起動直後のスナップショットを取得してその状態に VM を復元する．具体的には上述の 4 つの手続きを踏む．結果として，BackGroundReboot を用いると，新たなバージョンの OS を使用できるまでに要する時間は，従来の OS 再起動を用いたときに要する時間よりも長いと考えられる．しかし，ユーザが被るダウンタイムは BackGroundReboot の方が短い．そのため，ダウンタイムの長さよりも新たなバージョンの OS をいち早く使用したい場合は従来手法を，できるだけダウンタイムを短くして新たなバージョンの OS を使用したい場合は BackGroundReboot を使用するというように，用途に応じて両者を使い分けることが望ましい．また，ダウンタイムを削減しながら再起動に近い効果を得ることが目的であるため，第 4 フェーズでスナップショットの状態に戻した段階で，親 VM で稼働しているプロセスの状態は全て破棄される．これは通常の再起動を行ったときと同等である．

上記の 4 つの手順を踏むことで OS 再起動に伴うダウンタイムの削減が期待できるが，次の 2 つの問題点を考慮しなければならない．まず 1 つめは，第 4 フェーズにおいて親 VM の状態を子 VM から取得したスナップショットに復元するので，第 2，3 フェーズにおいて親 VM で生じたディスク書き込みが破棄されてしまうという点である．第 4 フェーズでは，OS のブートが終了した時点での子 VM のメモリ内容，およびディスク内容を親 VM 上で復元する．そのため，全フェーズ終了後，親 VM では新しいバージョンの OS は稼働しているものの，第 2，および第 3 フェーズで行ったファイルの更新や生成，削除は反映されていない状態となる．

2 つめは子 VM の使用する資源使用量についてである．VM を物理ホスト上に 2 台稼働させるため，親 VM と子 VM 両者が物理ホストの計算機資源を共有することになる．そのため，過剰な資源競合が生じてしまい，第 2，第 3 フェーズにおいて親 VM 上で動作しているアプリケーションのパフォーマンスや応答性が著しく劣化する可能性がある．結果として，親 VM が稼働しているにもかかわらず，実質的にダウンタイムとほぼ同等となってしまう可能性がある．

以上の 2 点の問題点を解決するために，これらの問題に対して次の対策を講じる．提案手法では *Persistent Virtual Disk* という特殊な仮想ディスクドライブを用意することで 1 つめの問題を解決している．2 つめの問題に関しては，子 VM への資源の割り当て方を既存研究を組み合わせることで解決する．以降，本章ではこれらについて述べる．

4.2 Persistent Virtual Disk

子 VM が再起動している間に親 VM 上で生じたディスク書き込みを保存するために，*Persistent Virtual Disk* という仮想ディスクドライブを用意する．Persistent Virtual Disk はスナップショットへの復元による影響を受けない仮想ディスクドライブである．たとえスナップショット機能を用いて VM の状態を過去のものにしても，Persistent Virtual Disk は過去の状態に戻らず現時点の状態のままである．たとえば，ある VM が Persistent Virtual Disk を搭載しており，ある時点で取得したスナップショットに状態を復元する場合を考える．スナップショット機能を用いて過去の時点に VM の状態を戻すと，メモリ内容や通常の仮想ディスクドライブの内容はスナップショットを取得した時点のものになるが，Persistent Virtual Disk の状態は過去のものにならない．結果として，VM はスナップショットを取得した時点以降に行われた更新が反映された状態の Persistent Virtual Disk を使用することができる．

Persistent Virtual Disk を用いることで，子 VM のスナップショットに親 VM の状態を復元した後も，子 VM が再起動している間に親 VM 上で行った仮想ディスクドライブへの更新を保持することができる．4.1 節で述べた第 2 および第 3 フェーズにおいて，親 VM を使用するときには，Persistent Virtual Disk 上に構築したファイルシステムをマウントしてそこにデータを記録するようにする．そして，親 VM を取得したスナップショットの状態に復元した後，改めて Persistent Virtual Disk をマウントする．こうすることで，第 2，3 フェーズで親 VM 内で保存したファイルデータを，第 4 フェーズ完了以降も使用することができる．

ここで，OS のブートシーケンスにおいて，Persistent Virtual Disk をマウントしないよう設定する必要がある．なぜなら，親 VM をスナップショットの状態に復元した後，親 VM が Persistent Virtual Disk 上に保存したファイルデータにアクセスできない可能性があるためで

ある．多くのファイルシステムはスーパーブロックといったファイルシステムのメタ情報をディスクパーティションをマウントするときに一度だけ読み込み，以降はそれらの情報をメモリ内に保持する．ファイルシステムのメタ情報に更新があると，適宜エントリを更新し，ディスクの該当箇所に書き込む．そのため，子 VM 上で OS が起動途中に Persistent Virtual Disk をマウントすると，その時点以降で 親 VM 上で生じた Persistent Virtual Disk に対する更新が破棄される可能性がある．そのため，Persistent Virtual Disk のマウントは親 VM の実行状態をスナップショットから復元した後に行う必要がある．

たとえば Linux において，ホームディレクトリ (/home) を Persistent Virtual Disk に割り当て，他のディレクトリを通常の仮想ディスクドライブに割り当てる設定が考えられる．Linux カーネルのアップデートを行うときには，通常，ホームディレクトリとは別のディレクトリ以下にある Linux のカーネルソースにパッチを適用して，Linux カーネルのコンパイルを行う．そして，カーネルイメージをインストールした後に再起動を行い，そのカーネルイメージを起動する．また各種デーモンは多くの場合，ホームディレクトリ以下とは異なる箇所にある設定ファイルを読み込み起動する．実際，Gentoo Linux 2007.0，Fedora Core10，Ubuntu 9.04，CentOS 5.3 の 3 種類の Linux ディストリビューションをデフォルトでインストールしたときの OS 起動中のディスクアクセスを調査したところ，ホームディレクト以下へのアクセスは見られなかった．そのため，Persistent Virtual Disk をホームディレクトに設定してもシステムの起動には支障がない．このように Persistent Virtual Disk を設定すると，子 VM が再起動している間に行ったホームディレクトリ以外への変更は破棄されてしまう．しかし，実質の作業ディレクトリへの変更は可能であるため，実際に多くの作業が OS 再起動中に行えることが期待でき，ユーザは BackGroundReboot の恩恵を享受できる．

4.3 VM Cloning のバックグラウンドタスク化

子 VM の資源使用による親 VM のパフォーマンス劣化を抑制するために，親 VM をフォアグラウンドタスク (以下，FG タスク) として，子 VM をバックグラウンドタスク (以下，BG タスク) として扱って計算機資源を割り当てる．BG タスクとして特徴づけられた 子 VM には，FG タスクである親 VM と資源競合が生じないように資源を割り当てる．こうすることで，親 VM 上で動作しているアプリケーションのスループットや応答性の劣化を防ぐ．

これまでに FG，BG タスクのスケジューリングはアプリケーションに対して行われており，様々なスケジューリング手法[9)–14)] が報告されている．これらの成果を利用することで，子 VM に親 VM の動作を妨げることなく資源を割り当てることができると考えられる．たとえば Idletime スケジューリング[14)] を利用することで，ディスク読み込みリクエストによって生じるディスク帯域の競合を避けることができる．ディスク帯域が起こす競合はアプリケーションのパフォーマンスや応答性に及ぼす影響が大きいことが知られており，Idletime スケジューラによって BG タスクを制御することで，FG タスクのパフォーマンス劣化を抑制することができる．こうしたスケジューラを VMM に組み込み，親 VM と子 VM とを管理することで，子 VM による親 VM への干渉を防ぐ．

5. 実　　装

提案手法のプロトタイプを VirtualBox 1.6.0_OSE[15)] 上に実装した．VirtualBox は x86 アーキテクチャ上で動作する VMM であり，既存 OS に修正を加えることなく実行できる．VirtualBox は VMware Workstation と同様のアーキテクチャを採用している．VirtualBox を実行するときにはホスト OS にカーネルモジュールを組み込む．カーネルモジュールはワールドスイッチを行うことで，ホスト環境と仮想化環境を切り替え，VM を動作させる．本論文では VM Cloning と Persistent Virtual Disk の実装について述べる．子 VM の BG タスク化に関しては現在実装を行っているところである．

5.1 VM Cloning

VM Cloning を実現するために，プロトタイプでは VirtualBox が提供するスナップショット取得機能を利用した．VirtualBox では，仮想ディスクドライブのある時点の状態を保存するために，差分ディスクファイルを生成する．保存したい時点以降に生じた仮想ディスクドライブへの更新は差分ディスクファイルに保存する．VirtualBox 上で，ある VM のスナップショットを取得するとき，まず VM のメモリ内容をファイルとして保存する．そして，VirtualBox は仮想ディスクドライブの差分ディスクファイルを新たに生成する．VirtualBox では，これまで利用していた仮想ディスクドライブと保存したメモリ内容を対にして VM のスナップショットとしている．取得したスナップショットの状態を復元するには，VM のメモリ内容を保存したファイルから復元し，ディスクドライブは差分ディスクファイルを生成する前に使用していた仮想ディスクドライブに設定する．これによりスナップショットを取得した時点に VM の状態を戻している．

子 VM を生成するために，プロトタイプでは VM Cloning を指示した時点での親 VM のスナップショットを活用する．VM Cloning を指示すると，まず子 VM を VirtualBox に登録する．次に，メモリサイズや仮想ディスクドライブといった親 VM のハードウェア情報を取得して同様の構成に子 VM を設定する．子 VM を現時点での親 VM と同様の状態で起動するために，親 VM のスナップショットを取得する．子 VM を起動するとき，子 VM のメ

モリ内容を先ほど取得した親 VM のメモリ内容が保持されているファイルから復元し，子 VM に接続している仮想ディスクドライブすべてに対して差分ディスクファイルを生成する．以上が完了した時点で子 VM を起動する．このようにすることで，別 VM 上に親 VM と同様の状態を作り出し，子 VM としている．

現在の実装では，スナップショット機構を利用しているため，OS のメモリ使用量に従ってダウンタイムが長くなると考えられる．高速な VM Cloning を実現するために，親 VM とメモリ領域を Copy-on-Write 技術で共有する手法が考えられる．また，高速に VM のクローンを生成する既存手法[16),17)] を適用することで，さらにダウンタイムを削減することができる．

5.2 Persistent Virtual Disk

Persistent Virtual Disk を実現するために，プロトタイプでは VirtualBox で提供されているディスク形式を拡張した．VirtualBox では 3 種類の仮想ディスクの形式を提供している．Normal ディスクと Immutable ディスク，そして Write-through ディスクである．Normal ディスクは読み書き可能なディスクであり，スナップショットも取得可能である．Immutable ディスクはディスク書き込みを一時的に保存するディスクである．VM が稼働している間は，Immutable ディスクは Normal ディスクのように振る舞い，ディスクを読み書きすることができる．しかし，VM が停止すると，Immutable ディスクに生じたディスク書き込みを破棄し，VM に接続される前の状態に戻る．Write-through ディスクは差分ディスクファイルを生成することができないディスクである．そのため，Write-through ディスクに接続された VM がディスク書き込みを発行した際には，必ずそれらが Write-through ディスクに反映されることを保証している．差分ディスクファイルを作成することができないため，VirtualBox では Write-through ディスクを接続している VM のスナップショットを取得することができない．

プロトタイプでは Persistent Virtual Disk をスナップショット取得可能な Write-through ディスクとして実装した．プロトタイプ上でスナップショットを取得すると，Persistent Virtual Disk の差分ディスクファイルを作成することなく，常に Persistent Virtual Disk そのものがスナップショットとして保存される．そのため，Persistent Virtual Disk を備えた VM のスナップショットは，常に現在使用している Persistent Virtual Disk を指す．結果として，予め取得したスナップショットの状態に VM を戻したとき，VM は Persistent Virtual Disk をそのまま使用するので，スナップショット取得後に Persistent Virtual Disk へ生じた更新内容をその VM 上で利用できる．

6. 予 備 実 験

BackGroundReboot の有用性を示すために予備実験を行った．実験で用いたマシンは 3 章で用いたものと同じである．プロトタイプを実装した VirtualBox を稼働させ，VM には仮想ディスクドライブとして，20 GB の Normal ディスクと 10 GB の Write-through ディスクを割り当てた．

BackGroundReboot が OS 再起動のダウンタイムを削減することを示すために，BackGroundReboot によって生じる親 VM の停止時間を計測した．提案手法で OS 再起動に近い効果を得るための 4 つのフェーズにおいて，次の 2 つのタイミングで親 VM が停止する．VM Cloning を行うときと，子 VM から取得したスナップショットに親 VM の状態を戻すときである．本実験では，VM Cloning と子 VM から取得したスナップショットへの復元による親 VM の停止時間をそれぞれ計測した．VM に与えるメモリ量を 256MB，512 MB，768 MB と変更していき，4 種類の OS を用いて実験を行った．稼働させた OS は Gentoo Linux 2007.0 (`gentoo`)，centOS 5.3 (`cent`)，Fedora 10 (`fedora`)，そして Ubuntu 9.04 (`ubuntu`) である．すべて標準インストールを行い，稼働させるデーモンは変更していない．まず，それぞれでログインを行い，新たなバージョンの Linux カーネルを起動するよう設定した後，VM Cloning を行い，子 VM を再起動する．再起動終了後，子 VM のスナップショットを取得して，親 VM の状態を変更する．この手順で生じた VM Cloning，ならびにスナップショットへの復元に伴う親 VM のダウンタイムを測定した．比較として，通常の再起動が招くダウンタイムも計測した．再起動の要するダウンタイムとして，再起動を指示してからログイン画面が表示されるまでの時間を計測した．

結果を表 2，3 に示す．表 2 は BackGroundReboot によって生じるダウンタイムを示している．VM Cloning と VM から取得したスナップショットへの復元に要するダウンタイム，ならびにその合計を示す．表 3 は 通常の OS 再起動を行ったときのダウンタイムを示している．表より提案手法が OS 再起動に伴うダウンタイムを削減していることがわかる．`gentoo` の場合，再起動に生じるダウンタイムが最大で約 80 秒であるのに対し，提案手法では 5 秒以内にダウンタイムを抑えている．768 MB のメモリ量を搭載した `fedora` 場合でも，再起動によるダウンタイムは約 64 秒であるに対し，提案手法では約 15 秒以内にとどまっており，約 78 % のダウンタイム削減を行っている．

また，表 2 より，OS によって VM Cloning やスナップショットへの復元に要する時間が異なることが判る．これは OS が利用するメモリ量に起因する．`gentoo` 以外の OS では，ロ

表 2 BackGroundReboot に伴うダウンタイム．BackGroundReboot によって生じるダウンタイムを計測した．具体的には，VM Cloning と子 VM 上で取得したスナップショットへの復元に伴う親 VM の停止時間を計測した．与えるメモリ量を変更しながら 4 種類の OS を稼働させた．

	gentoo (second)			cent (second)			fedora (second)			ubuntu (second)		
VM に割り当てたメモリ量	VM Cloning	SN への復元	合計	VM Cloning	SN への復元	合計	VM Cloning	SN への復元	合計	VM Cloning	SN への復元	合計
128 MB	3.3	1.3	4.6	8.5	3.7	12.2	8.4	3.4	11.8	8.3	3.4	11.7
256 MB	3.0	1.3	4.3	13.6	4.3	17.9	9.4	4.3	13.7	8.4	3.5	11.9
768 MB	3.3	1.3	4.6	12.9	4.9	17.8	9.3	4.9	14.2	8.9	3.2	12.1

表 3 OS 再起動に伴うダウンタイム．OS の再起動によって生じるダウンタイムを計測した．OS に再起動を指示してからログイン画面が表示されるまでに要した時間をダウンタイムとした．与えるメモリ量を変更しながら 4 種類の OS を稼働させた．

VM に割り当てたメモリ量	gentoo (second)	cent (second)	fedora (second)	ubuntu (second)
128 MB	76.7	130.6	64.3	52.8
256 MB	80.3	134.9	64.6	53.8
768 MB	79.8	146.9	64.6	52.9

グイン後にアップデート通知やユーザが利用するファイルのキャッシュといったサービスが起動するため，ログインしただけでメモリ使用量が増える．プロトタイプの VM Cloning の実装は VM のメモリ内容を一旦ファイルに書き出すため，OS のメモリ使用量が増えると，その分時間を要する．また，gentoo 以外の OS は様々なデーモンが起動するため，その分メモリ使用量が増える．結果として，スナップショットの状態に復元するためのデータ量が増えるために，要する時間も増える．

さらに，表 3 より，cent はメモリ量に従って再起動に要する時間が長くなっていることが判る．これは cent がブート中に，システム内のファイルを予めキャッシュしようとする動作に起因する．readahead_early と呼ばれるデーモンが，搭載されているメモリ量に合わせてシステムが利用するファイルのキャッシュを行う．VirtualBox において I/O 仮想化のオーバヘッドは依然として高く，また，OS のキャッシュ生成に合わせて VMM 内でシャドーページテーブル (shadow page table) を作成するため，起動に要する時間が長くなる．

7. まとめと今後の課題

本論文では，OS のアップデートに伴う再起動が招くダウンタイムを削減する手法，BackGroundReboot を提案した．提案手法は仮想マシン技術を応用することで OS カーネルを修正することなく OS 再起動に伴うダウンタイムを削減する．具体的には，OS 再起動後のスナップショットを取得して，現在利用している VM の状態をそのスナップショットの状態に復元することで，ダウンタイムを削減しつつ OS 再起動に近い効果を得る．

今後の課題として，親 VM と子 VM をそれぞれ FG タスク，BG タスクとしてスケジューリングする機構を実装することが挙げられる．実装後には，そのスケジューリングの効果を確認するために，子 VM の再起動による親 VM への干渉を調査する必要がある．また，Linux だけでなく，Windows といった商用 OS を用いて，実際のアップデート事例を用いた実験を行う予定である．

参 考 文 献

1) Baumann, A., Appavoo, J., Wisniewski, R.W., Silva, D.D., Krieger, O. and Heiser, G.: Reboots are for Hardware: Challenges and Solutions to Updating an Operating System on the Fly, *Proceedings of the USENIX Annual Technical Conference (USENIX '07)*, pp.337–350 (2007).
2) Baumann, A., Heiser, G., Appavoo, J., Silva, D.D., Krieger, O. and Wisniewski, R.W.: Providing Dynamic Update in an Operating System, *Proceedings of the USENIX Annual Technical Conference (USENIX '05)*, pp.279–291 (2005).
3) Chen, H., Chen, R., Zhang, F., Zang, B. and Yew, P.-C.: Live Updating Operating Systems Using Virtualization, *Proceedings of the 2nd ACM International Conference on Virtual Execution Environments (VEE '06)*, pp.35–44 (2006).
4) Makris, K. and Ryu, K.D.: Dynamic and Adaptive Updates of Non-Quiescent Subsystems in Commodity Operating System Kernels, *Proceedings of the 2nd ACM European Conference on Computer Systems (EuroSys '07)*, pp.327–340 (2007).
5) Arnold, J. and Kaashoek, M.F.: Ksplice: Automatic rebootless kernel updates, *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys '09)*, pp.187–198 (2009).
6) Lowell, D.E., Saito, Y. and Samberg, E.J.: Devirtualizable Virtual Machines Enabling General, Single-Node, Online Maintenance, *Proceedings of the 11th ACM International Confer-*

ence on Architectural Support for Programming Languages and Operating Systems (ASPLOS '04), No.211–233 (2004).
7) Potter, S. and Nieh, J.: Reducing Downtime Due to System Maintenance and Upgrades, *Proceedings of the 19th USENIX Large Installation System Administration Conference (LISA '05)*, pp.47–62 (2005).
8) Chen, P.M. and Noble, B.D.: When Virtual is Better than Real., *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pp.133–138 (2001).
9) Ryu, K.D., Hollingsworth, J.K. and Keleher, P.J.: Exploiting Fine Grained Idle Periods in Networks of Workstations, *IEEE Transactions on Parallel and Distributed Systems*, Vol.11, pp.683–698 (2000).
10) Lumb, C.R., Schindler, J., Ganger, G.R. and Nagle, D.F.: Towards Higher Disk Head Utilization: Extracting Free Bandwidth From Busy Disk Drives, *Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation (OSDI '00)*, pp.87–102 (2000).
11) Lumb, C. R., Schindler, J. and Ganger, G. R.: Freeblock Scheduling Outside of Disk Firmware, *Proceedings of the 1st USENIX Symposium on File and Storage Technologies (FAST '02)*, pp.10–22 (2002).
12) Venkataramani, A., Kokku, R. and Dahlin, M.: TCP Nice: A Mechanism for Background Transfers, *Proceedings of the 5th USENIX Symposium on Operating System Design and Implementation (OSDI '02)*, pp.329–344 (2002).
13) Gupta, A., Lin, B. and Dinda, P.A.: Measuring and Understanding User Comfort With Resource Borrowing, *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC '04)*, pp.214–224 (2004).
14) Eggert, L. and Touch, J.D.: Idletime Scheduling with Preemption Intervals, *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pp.249–262 (2005).
15) Sun Microsystems: VirtualBox (2008). `http://www.virualbox.org`.
16) Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G. and Savage, S.: Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm, *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pp.148–162 (2005).
17) Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., de Lara, E., Brudno, M. and Satyanarayanan, M.: SnowFlock: Rapid Virtual Machine Cloning for Could Computing, *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys '09)*, pp.1–12 (2009).