仮想計算機を用いたサーバ統合における高速なソフトウェア若化手法

光来 健一　千葉 滋

仮想計算機（VM）を用いたサーバ統合が行われるようになるにつれて、仮想計算機モニタ（VMM）のソフトウェアエージングが問題になってきている。VMM の性能劣化はその上で動くすべての VM に影響するためである。このような問題を解決するために、ソフトウェア若化（software rejuvenation）と呼ばれる手法が提案されている。典型的な例は VMM の再起動であるが、単純な VMM の再起動はその上で動いているすべての OS の再起動を伴うため望ましくない。そこで、我々は Warm-VM Reboot と呼ぶ VMM の高速なソフトウェア若化手法を提案する。Warm-VM Reboot は VM のメモリイメージをディスクに保存することなく、高速にサスペンド・レジュームさせることにより、VMM だけを効率よく再起動する手法である。我々の実験によると、Warm-VM Reboot はサービスのダウンタイムを最大 83%減らすことができ、再起動直後のファイルキャッシュミスによる性能低下を防ぐことができた。

1　はじめに

ソフトウェアが時間とともに劣化する現象はソフトウェアエージング [6] と呼ばれている。その原因はシステムリソースの枯渇やデータの破壊などである。エージングはソフトウェアの性能劣化やクラッシュを引き起こす。近年、仮想計算機（VM）を用いたサーバ統合が広く行われるようになるにつれて、仮想計算機モニタ（VMM）のエージングが問題になってきている。このサーバでは多くの VM が VMM 上で動いているため、VMM のエージングはすべての VM に直接影響する。

このような問題を解決するために、ソフトウェア若化（software rejuvenation）と呼ばれる手法が提案されている [6]。ソフトウェア若化は時々VMM を停止させ、内部状態を正常に戻してから再開する。典型的な例は VMM の再起動であるが、VMM を再起動すると VM 上で動いているすべての OS を再起動することになる。このことは OS によって提供されているサービスのダウンタイムを増大させる。また、OS が保持していたファイルキャッシュは再起動によって失われるため、OS の再起動後にはキャッシュミスにより性能が低下する。このようなダウンタイムや性能低下はサーバにとって致命的である。

この論文では、Warm-VM Reboot と呼ぶ VMM の高速なソフトウェア若化手法を提案する。基本的なアイデアは、VMM の再起動中に VM のメモリイメージを保持しておき、再起動後に再利用することである。Warm-VM Reboot は VM のオンメモリ・サスペンド／レジューム機構および VMM のクイック・リロード機構を使うことで、VMM だけを効率よく再起動する。オンメモリ・サスペンド／レジューム機構を用いることで、VMM は再起動する前に VM を高速にサスペンドする。この際に、VM のメモリイメージはメインメモリ上に保持され、ディスク等の外部記憶には保存されない。VMM の再起動後には、保持されているメモリイメージを使って VM を高速にレジュームする。再起動中にメモリイメージを保

Kenichi KOURAI, Shigeru CHIBA, 東京工業大学, Tokyo Institute of Technology.

持するために、VMM はハードウェア・リセットを伴わないクイック・リロード機構により再起動される。Warm-VM Reboot は VM 上の OS の再起動を必要としないため、OS のダウンタイムを減らし、キャッシュミスによる性能低下を防ぐことができる。

Warm-VM Reboot を実現するために、我々は Xen [3] をベースにしたシステム *RootHammer* を開発した。我々の実験結果によると、Warm-VM Reboot はダウンタイムを最大 83%削減できた。一方、Xen 標準のサスペンド／レジューム機構を用いた場合は、逆にダウンタイムが 173%増加した。また、Warm-VM Reboot を行った後、ウェブサーバの性能は全く低下しなかった。しかし、通常の再起動を行った場合には、再起動直後のスループットが 69%低下した。

以下、2 章では、VMM の既存のソフトウェア若化手法の問題について述べる。3 章では提案する高速なソフトウェア若化手法および、ソフトウェア若化の利用モデルについて述べる。4 章では実験の結果を示す。5 章では関連研究について触れ、6 章で本論文をまとめる。

2 VMM のソフトウェア若化

VMM は長時間動作するソフトウェアであるため、他のコンポーネントよりソフトウェアエージングの影響を受けやすい。例えば、Xen には VM を再起動したり、エラー処理を行う際に利用可能なヒープが減少するバグがあった。Xen のヒープサイズは物理メモリサイズによらずデフォルトで 16MB であるため、メモリリークが起きると容易にメモリ不足になる。VMM がメモリ不足になると性能が劣化し、直接すべての VM に影響する。

VMM のエージングに加えて、特権 VM も他の VM に影響を及ぼす可能性がある。特権 VM は図 1 のように Xen や VMware ESX サーバなどで用いられ、VM の管理や VM の I/O 処理を行う。特権 VM の中では修正を加えた通常の OS が動いており、カーネルメモリやスワップスペースのようなシステムリソースが時間とともに枯渇することが OS のエージングとして知られている [5]。特権 VM では巨大なサーバは動

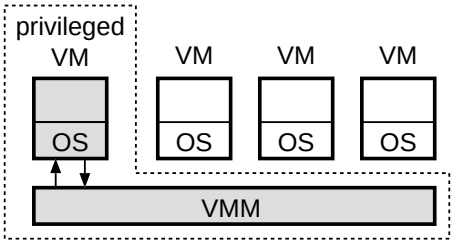


図 1 想定する VM アーキテクチャ

かさないため、割り当てられるメモリサイズは一般に大きくなく、メモリの枯渇が起こりやすい。例えば、Xen には `xenstored` という再起動できないデーモンにメモリリークを起こすバグがあった。エージングにより特権 VM での I/O 処理性能が低下すると、他の VM の性能も低下する。さらに、特権 VM は VMM に強く依存しているため、その再起動は VMM の再起動を伴う。そのため、我々は特権 VM も VMM の一部と考える。

このようなソフトウェアエージングに対処するために、ソフトウェア若化と呼ばれる手法が提案されている [6]。ソフトウェア若化は時々VMM を停止させ、内部状態を正常にしてから再開する。典型的な例は VMM の再起動である。VMM のような長時間動作するソフトウェアの状態は時間とともに劣化するので、ソフトウェア若化による予防保守がエージングの問題を減少させると考えられる。しかし、VMM を再起動する時には、VMM がその上で動いている VM を終了させる前に VM 上で動いている OS をシャットダウンしなければならない。そして、VMM を再起動した後で、新たに作られた VM 上で OS を起動してすべてのサービスを再開しなければならない。

このことは OS が提供しているサービスのダウンタイムを増大させる。第一に、VMM を再起動する時には、多くの OS がシャットダウンされ、再起動される。複数の OS を同時にシャットダウンおよび起動するとリソース競合を起こすため、それぞれの OS を再起動する時間は VM の数に比例する。同時に動く VM の数は Intel VT や AMD Virtualization などの CPU サポートや CPU のマルチコア化により増加する傾向にある。さらに、近年のサーバでは JBoss ア

プリケーションサーバのような重いサービスを動かすことが多いため、サービスの終了および開始にかかる時間も増大している。第二に、OS のシャットダウン、VMM の再起動、OS の起動は順番に行われるため、VMM の再起動は OS のダウンタイムを増大させる。VMM の再起動は VMM のシャットダウン、ハードウェア・リセット、VMM の起動を含む。特に、ハードウェア・リセットは時間のかかるメインメモリのチェックや SCSI のチェックなどを伴なう。

加えて、VM 上で動く OS の性能は OS の再起動後に低下する。その主原因はファイルキャッシュを失うことである。OS はファイルの内容をキャッシュとしてメモリ上に保持することでファイルアクセスの性能を向上させている。OS を再起動するとメモリが初期化され、OS が保持していたファイルキャッシュは失われる。そのため、OS を再起動した直後は頻繁なキャッシュミスによりサーバの性能が低下する。近年の OS は空きメモリの多くをファイルキャッシュとして利用するため、ファイルキャッシュを再起動前の状態に戻すのには長い時間がかかる。64 ビット CPU や安価なメモリにより、1 台のマシンに搭載されるメインメモリは増えており、VM に割り当てることができるメモリも増えている。

3 高速なソフトウェア若化手法

我々は VMM の再起動はその上で動作している OS の再起動とは独立しているべきであると考える。OS も VMM と同様にソフトウェア若化が必要になるが、そのタイミングは VMM とは異なる。もし、VMM を再起動する際に OS がソフトウェア若化を必要としていなければ、OS の再起動は無駄になる。

3.1 Warm-VM Reboot

VMM のソフトウェア若化の影響を最小化するために、我々は Warm-VM Reboot と呼ぶ高速なソフトウェア若化手法を提案する。基本的なアイデアは VMM の再起動を通して VM のメモリイメージを保持し、再起動後にそのメモリイメージを再利用することである。Warm-VM Reboot は VM のオンメモリ・サスペンド／レジューム機構および VMM のクイック・リロード機構を用いて、VMM だけを効率よく再起動することを可能にする。VMM は再起動する前にオンメモリ・サスペンドを用いてすべての VM を高速にサスペンドし、クリック・リロードを用いて自分自身を高速に再起動する。VMM が再起動したら、オンメモリ・レジュームを用いてすべての VM を高速にレジュームする。

オンメモリ・サスペンドは VM が使用しているメモリイメージをそのまま凍結し、VM を一時停止させる。そのメモリイメージは VM が再開されるまで VMM の再起動を通してメモリ上に保持される。この機構はメモリイメージをディスク等の外部記憶に保存する必要がなく、フラッシュメモリ等の不揮発性メモリにコピーする必要もない。オンメモリ・サスペンドにかかる時間は VM に割り当てられているメモリサイズにほとんど依存しないため、非常に効率がよい。すべての VM に割り当てられるメモリサイズの合計がさらに大きくなってもこの機構はスケールすると考えられる。メモリイメージ以外の VM の実行状態については、VMM の再起動を通して保持されるメモリ領域にコピーする。

オンメモリ・レジュームは凍結されたメモリイメージを解凍し、そのメモリイメージを再利用して VM の実行を再開する。この機構もメモリイメージを外部記憶から読み込んだり、不揮発性メモリからコピーしたりする操作を必要としない。VM のメモリイメージは完全に復元されるので、再起動直後であってもキャッシュミスによる性能低下を防ぐことができる。同時に、VM の実行状態も復元される。オンメモリ・サスペンド／レジューム機構は、メインメモリ上のメモリイメージにアクセスすることなくサスペンド／レジュームできるという点で、ACPI S3 状態（Suspend To RAM）に似ている。

クイック・リロードは VM のメモリイメージを保持したまま高速に VMM を再起動させる。通常、VMM の再起動は VMM の実行ファイルを読み込むためにハードウェア・リセットを必要とするが、ハードウェア・リセットはメモリの内容が保持されることを保証しない。また、2 章で述べたように、ハードウェア・リセットには時間がかかる。クイック・リロードはソ

フトウェア的に新しい VMM の実行ファイルをメモリ上に読み込み、そのエントリ・ポイントにジャンプして実行を開始することでハードウェア・リセットを回避する。このようなソフトウェア機構が VMM の再起動中のメモリを管理するので、メモリの内容が保持されることを保証できる。さらに、クイック・リロードは VMM の初期化の際に凍結された VM のメモリイメージが破壊されるのを防ぐ。

多くの VMM が VM のサスペンド／レジューム機構を提供しているが、メモリイメージの保存先としてディスクを用いるため、VMM のソフトウェア若化時に利用するのには適していない。従来のサスペンド／レジューム機構は ACPI S4 状態（Suspend To Disk）であり、一般にハイバネーションと呼ばれる。この機構は多くのディスクアクセスを必要とし、非常に時間がかかる。一方、オンメモリ・サスペンド／レジュームはメモリイメージをディスクに保存する必要がない。クイック・リロードがメモリ上に保持された VM のメモリイメージを VMM の再起動後に再利用することを可能にしているためである。

3.2 ソフトウェア若化の利用モデル

Warm-VM Reboot によって減少するダウンタイムを見積もるために、ソフトウェア若化の利用モデルを考える。通常、VMM のソフトウェア若化は OS のソフトウェア若化とともに用いられる。OS の方が VMM より複雑なソフトウェアであるため、一般に、OS のソフトウェア若化は VMM のそれより頻繁に行われる。単純化のために、ソフトウェア若化は定期的に行われる [4] と仮定する。Warm-VM Reboot を用いる場合、図 2 (a) のように、OS のソフトウェア若化のタイミングは VMM のそれに影響されない。これは Warm-VM Reboot が OS のソフトウェア若化を伴わないためである。一方、VMM が通常の再起動によってソフトウェア若化を行う場合（Cold-VM Reboot）、VMM のソフトウェア若化は OS のそれを含むため、図 2 (b) のように OS のソフトウェア若化のタイミングに影響を与える。OS のソフトウェア若化の回数を減らすために、VMM のソフトウェア若化の時点から OS のソフトウェア若化を再スケジュールする。

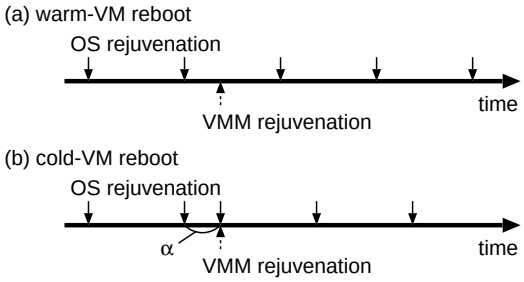


図 2 VMM と OS のソフトウェア若化のタイミング

Warm-VM Reboot を用いる場合、VMM のソフトウェア若化によるダウンタイムはすべての VM のサスペンド、VMM の再起動、すべての VM のレジュームによって生じる。ダウンタイムの増加は、

$$d_w(n) = reboot_{vmm}(n) + resume(n)$$

である。n は VM の数、$reboot_{vmm}(n)$ は n 個の VM をサスペンド・レジュームする時に VMM の再起動に要する時間、$resume(n)$ は n 個の VM を同時にオンメモリ・サスペンド／レジュームするのに要する時間である。

一方、Cold-VM Reboot を用いる場合、VMM のソフトウェア若化によるダウンタイムはすべての OS のシャットダウン、ハードウェア・リセット、VMM の再起動、すべての OS の起動によって生じる。ダウンタイムの増加は、

$$d_c(n) = reset_{hw} + reboot_{vmm}(0) + reboot_{os}(n) - reboot_{os}(1) \times \alpha$$

である。$reset_{hw}$ はハードウェア・リセットに要する時間、$reboot_{os}(n)$ は n 個の OS を同時にシャットダウンおよび起動するのに要する時間、α（$0 < \alpha \leq 1$）は前回の OS のソフトウェア若化から VMM のそれまでに経過した時間が OS のソフトウェア若化の間隔に占める割合である。OS のソフトウェア若化は VMM のそれの後、再スケジュールされるため、VMM のソフトウェア若化の際に同時に行われる OS のソフトウェア若化を除くと、OS のソフトウェア若化の回数は α だけ減る。

Warm-VM Reboot によって減少するダウンタイム

は $d_c(n) - d_w(n)$ により計算でき、

$$r(n) = reset_{hw} + reboot_{vmm}(0) - reboot_{vmm}(n) + reboot_{os}(n) - reboot_{os}(1) \times \alpha - resume(n)$$

となる。

3.3 実装

我々は Xen 3.0.0 をベースにして、Warm-VM Reboot を実現するシステム *RootHammer* を開発した。Xen 同様、VM はドメインと呼ばれ、VM の管理および I/O 処理を行う特権 VM はドメイン 0、それ以外の VM はドメイン U と呼ばれる。Xen では物理的なメモリをマシンメモリと呼び、ドメインに見せる仮想的な物理メモリを疑似物理メモリと呼ぶ。RootHammer では VMM の再起動後にメモリを再利用できるようにするために、P2M マッピング・テーブルを用意している。このテーブルは各ドメインについて、疑似物理メモリに対して割り当てられたマシンメモリを管理する。

VMM を再起動する時にはドメイン 0 をシャットダウンし、その後ですべてのドメイン U にサスペンドイベントを送る。Xen ではドメイン 0 がイベントを送るが、RootHammer では VMM が送る。これは、ドメイン U をサスペンドするタイミングをドメイン 0 のシャットダウン終了まで遅延し、ダウンタイムを減らすためである。ドメイン U がサスペンドイベントを受け取るとオンメモリ・サスペンドを行うが、Xen の従来のサスペンドとは違い、VM に割り当てられていたマシンメモリを解放しない。そして、VMM の再起動後に P2M マッピングテーブルに登録されているマシンメモリを再度 VM に割り当てて、オンメモリ・レジュームを行う。

VMM の再起動中にドメイン U のメモリイメージを保持するために、Linux の kexec 機能をベースにしてクイック・リロード機構を実装した。kexec 機構は Linux カーネルをソフトウェア的に再起動する機構である。Xen 3.0.4 から、クイック・リロードと同様に、kexec 機構の技術を用いてハードウェア・リセットなしに VMM を再起動する機構が組み込まれたが、ドメイン U のメモリイメージを保持する機能はない。

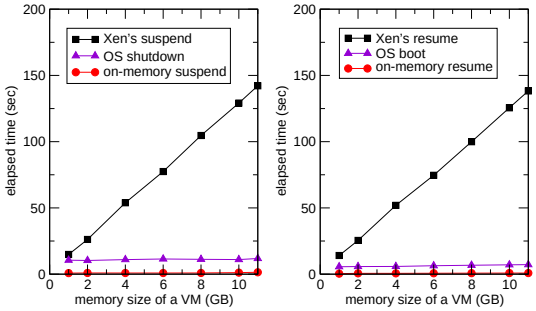
4 実験

我々は Warm-VM Reboot の有効性を示す実験を行った。サーバマシンには、デュアルコア Opteron Model 280 を 2 基、PC3200 DDR SDRAM を 12 GB、15,000 rpm の SCSI ディスク、ギガビットイーサネットを搭載した PC を用いた。VMM としては、RootHammer の VMM および、比較のために、Xen のオリジナルの VMM を用いた。VMM の上で動く OS は Xen 用に修正された Linux 2.6.12 であった。ディスクの 1 つのパーティションを 1 つの VM の仮想ディスクとした。ドメイン 0 に割り当てたメモリは 512MB であった。クライアントマシンには、Xeon 3.06 GHz を 2 基、2 GB のメモリ、ギガビットイーサネットを搭載した PC を用いた。OS は Linux 2.6.8 であった。

4.1 オンメモリ・サスペンド／レジュームの性能

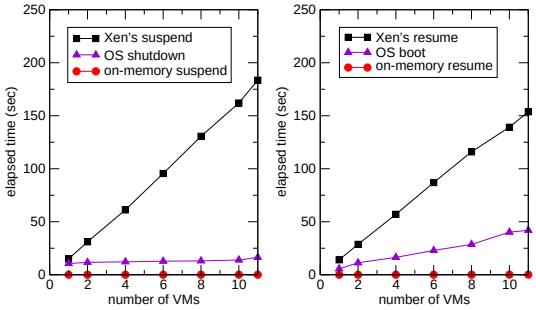
我々は VMM の再起動の前後の処理にかかる時間を測定した。この実験は (1) オンメモリ・サスペンド／レジューム、(2)Xen のサスペンド／レジューム、(3) シャットダウンと起動の 3 つの手法について行った。まず、1 つの VM だけを動かし、その VM に割り当てるメモリサイズを 1 GB から 11 GB まで変化させて、再起動の前後の処理に要した時間を測定した（図 3）。Xen のサスペンド／レジュームは VM のメモリイメージをディスクに書き出すので、VM のメモリサイズに依存している。一方、オンメモリ・サスペンド／レジュームは VM のメモリイメージにアクセスしないので、VM のメモリサイズに依存していない。メモリサイズが 11 GB の時、オンメモリ・サスペンド／レジュームに要した時間は Xen のサスペンド／レジュームのそれぞれ 0.06%および 0.7%であった。

次に、複数の VM を同時に動かした時の VMM の再起動の前後の処理に要した時間を測定した。それぞれの VM に割り当てるメモリサイズは 1 GB に固定し、VM の数を 1 から 11 まで変化させた。図 4 はその結果を示しており、どの手法も VM の数に依存している。VM の数が 11 の時、オンメモリ・サスペン



(a) pre-reboot task (b) post-reboot task

図 3 VM のメモリサイズを変化させた時の VMM の再起動前後の処理に要した時間



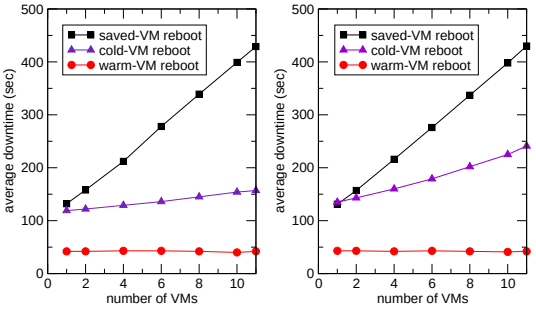
(a) pre-reboot task (b) post-reboot task

図 4 VM の数を変化させた時の VMM の再起動前後の処理に要した時間

ド／レジュームに要した時間は Xen のサスペンド／レジュームのそれぞれ 0.02%および 2.7%であった。また、この実験結果から、OS の起動にかかる時間は VM の数が増えると大幅に増えていることが分かる。

4.2 クイック・リロードの効果

クイック・リロードにより VMM がどの程度高速に再起動されるか調べるために、VMM を再起動するのに要する時間を測定した。クイック・リロードを用いた場合、再起動にかかる時間は 11 秒であった。それに対して、ハードウェア・リセットを行った場合は 59 秒要した。これらの結果から、我々の実験環境ではクイック・リロードが VMM の再起動を 48 秒高速化していることが分かる。



(a) ssh (b) JBoss

図 5 ssh サーバと JBoss のダウンタイム

4.3 サービスのダウンタイム

VMM を再起動した時に OS によって提供されているサービスのダウンタイムを測定した。クライアントホストからサーバホストの VM にパケットを送りながら VMM を再起動した。この実験では、各 VM のサービスが停止してから VMM の再起動後に再開するまでの時間を測定した。我々はこの実験を (1)Warm-VM Reboot、(2)Xen のサスペンド／レジュームを用いた VMM の再起動（Saved-VM Reboot）、(3)OS のシャットダウン／起動を伴なう VMM の再起動（Cold-VM Reboot）の 3 つの手法について行った。それぞれの VM に割り当てるメモリは 1 GB に固定し、VM の数を 1 から 11 まで変化させた。

まず、それぞれの VM で ssh サーバを動かし、VMM の再起動時のダウンタイムを測定した。図 5 (a) から、Saved-VM Reboot によるダウンタイムは VM の数に大きく依存していることが分かる。VM の数が 11 の時、ダウンタイムは平均 429 秒であった。Warm-VM Reboot によるダウンタイムは 42 秒であり、Saved-VM Reboot の 9.8%である。さらに、Warm-VM Reboot によるダウンタイムは VM の数にほとんど依存していない。一方、Cold-VM Reboot によるダウンタイムは 157 秒であり、Warm-VM Reboot の 3.7 倍である。

次に、それぞれの VM で JBoss アプリケーションサーバを動かし、VMM の再起動時のダウンタイムを測定した。JBoss は巨大なサーバであり、ssh サーバより開始に時間がかかる。図 5 (b) から、Warm-VM Reboot と Saved-VM Reboot によるダウンタイムは ssh サーバの場合とほぼ同じであった。これは JBoss

サーバの再起動を必要としなかったためである。一方、Cold-VM Reboot によるダウンタイムは ssh サーバより長くなっている。VM の数が 11 の時、ダウンタイムは 241 秒であり、ssh サーバの場合の 1.5 倍である。つまり、Cold-VM Reboot は動かすサービスによってダウンタイムが増大するということである。

さらに、VM の数が 11 の時の JBoss サーバの可用性について調べてみた。OS のソフトウェア若化を週に 1 回行い、VMM のソフトウェア若化を 4 週に 1 回行うことを想定する。我々の実験によると、OS のソフトウェア若化によるダウンタイムは 33.6 秒であった。Cold-VM Reboot については、3.2 節の α の期待値を 0.5 とする。このような仮定の下で、Warm-VM Reboot、Saved-VM Reboot、Cold-VM Reboot を用いる場合の JBoss サーバの稼働率はそれぞれ 99.993%、99.985%、99.977%となる。Warm-VM Reboot を用いる場合は 4 つの 9 を達成できるのに対し、他の手法では 3 つの 9 しか達成できない。

4.4 再起動後の性能低下

VMM の再起動直後のファイルキャッシュミスによる性能低下を調べるために、VMM の再起動の前後でファイルアクセスのスループットを測定した。ファイルキャッシュの効果を調べるために、それぞれ 2 回測定を行った。この実験ではすべてのファイルブロックがキャッシュに載るように、1 つの VM に 11 GB のメモリを割り当てた。まず、Warm-VM Reboot を行った場合と Cold-VM Reboot を行った場合について、512 MB のファイルを読んだ時のスループットを測定した（図 6 (a)）。Cold-VM Reboot を用いた場合は再起動直後のスループットは 91%低下したが、Warm-VM Reboot を用いた場合はスループットは再起動前後で変わらなかった。これは再起動直後にファイルアクセスを行ってもキャッシュミスが発生していないことを示している。

次に、VMM の再起動の前後でウェブサーバのスループットを測定した。この実験のワークロードでは、Apache ウェブサーバが 512 KB のファイル 10,000 個にアクセスするようにした。クライアントホストで httperf のプロセスを 10 個動かし、同時にリクエストを送信させた。図 6 (b) はその実験結果であり、Warm-VM Reboot を用いた時は再起動直後のスループットは低下しなかったが、Cold-VM Reboot を用いた時はスループットが 69%低下した。

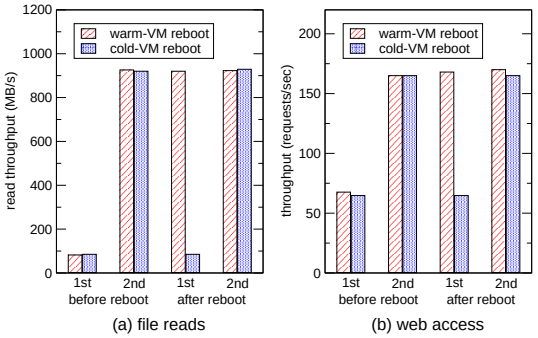


図 6 VMM の再起動前後におけるスループット

4.5 利用モデルへの適用

VM を 11 個動かした時の我々の実験結果から、3.2 節のモデルで使われている関数は以下のようになる。

$$reboot_{vmm}(n) = -0.55n + 43$$
$$resume(n) = 0.43n - 0.07$$
$$reboot_{os}(n) = 3.8n + 13$$
$$boot(n) = 3.4n + 2.8$$
$$reset_{hw} = 47$$

これらの関数を用いると、Warm-VM Reboot によって減少するダウンタイムの関数は以下のようになる。

$$r(n) = 3.9n + 60 - 17\alpha$$

$r(n)$ は $\alpha \leq 1$ の下では常に正の値なので、我々の実験環境では Warm-VM Reboot は常にダウンタイムを減らすことができることが分かる。

5 関連研究

マイクロリブート [2] はソフトウェアの障害から回復するためにアプリケーションのコンポーネントだけを再起動する。小さいコンポーネントの再起動で回復できなければ、そのコンポーネントを含むもう少し大きなコンポーネントを再帰的に再起動する。より小さいコンポーネントの再起動だけで回復できれば、アプリケーションのダウンタイムを減らすことができる。同様に、マイクロカーネル OS [1] ではプロセスとし

て実装されている OS のサブシステムだけを再起動することができ、Nooks [8] では OS のデバイスドライバだけを再起動することができる。これらはサブコンポーネントを高速に再起動する手法であるのに対し、Warm-VM Reboot はサブコンポーネントの状態を保持しながら親コンポーネントを高速に再起動する手法である。

Warm-VM Reboot は VMM のソフトウェア若化を高速に行う手法であるが、チェックポインティング [7] は OS の高速なソフトウェア若化を可能にする。この機構は OS の再起動前にプロセスの状態をディスクに保存し、再起動後に状態をディスクから復元する。これは VM のサスペンド／レジュームに似ているが、VM のサスペンド／レジュームはより多くのメモリを扱わなければならないため、高速化が難しい。

ディスクを用いた VM のサスペンド／レジュームを高速化するために、いくつかの手法が用いられている。VMware はサスペンド時にメモリイメージの変更された部分だけを保存する。この手法によりサスペンド時のディスクアクセスを減らすことができる。Windows XP は OS のハイバネーションの際にメモリイメージを圧縮して保存する。この手法はサスペンドとレジュームの両方でディスクアクセスを減らすことができる。

ドメイン 0 のソフトウェアエージングを緩和するために、Xen はデバイスドライバ専用のドライバドメインを提供している。Xen のデフォルトの設定ではデバイスドライバはドメイン 0 で動作するため、デバイスドライバのソフトウェア若化はドメイン 0 および VMM の再起動を必要とする。ドライバドメインの再起動は VMM の再起動を必要としない。しかし、ドライバドメインをサスペンド／レジュームすることはできないため、VMM を再起動する時にはドライバドメインをシャットダウンおよび起動しなければならず、ダウンタイムが増加する。

6 まとめ

本論文では、Warm-VM Reboot と呼ぶ VMM の高速なソフトウェア若化手法を提案した。この手法はオンメモリ・サスペンド／レジューム機構とクイック・リロード機構を用いて、VMM だけを再起動することを可能にする。オンメモリ・サスペンド／レジュームはメモリイメージにアクセスせずに VM を高速にサスペンド・レジュームし、クイック・リロードは VMM の再起動中に VM のメモリイメージを保持する。Warm-VM Reboot はダウンタイムを減少させ、再起動直後の性能低下を防ぐ。我々はこの手法を Xen をベースに実装し、有効性を示す実験を行った。その結果、ダウンタイムを最大 83%減少させることができ、再起動直後にもスループットを保つことができた。

参考文献

[1] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M.: Mach: A New Kernel Foundation for UNIX Development, *Proceedings of the USENIX 1986 Summer Conference*, 1986, pp. 93–112.

[2] Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., and Fox, A.: Microreboot – A Technique for Cheap Recovery, *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004, pp. 31–44.

[3] Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., and Neugebauer, R.: Xen and the Art of Virtualization, *Proceedings of the Symposium on Operating Systems Principles*, 2003, pp. 164–177.

[4] Garg, S., Huang, Y., Kintala, C., and Trivedi, K.: Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality, *Proceedings of the 1st Fault Tolerance Symposium*, 1995, pp. 22–25.

[5] Garg, S., Moorsel, A., Vaidyanathan, K., and Trivedi, K.: A Methodology for Detection and Estimation of Software Aging, *Proceedings of the 9th International Symposium on Software Reliability Engineering*, 1998, pp. 283–292.

[6] Huang, Y., Kintala, C., Kolettis, N., and Fulton, N.: Software Rejuvenation: Analysis, Module and Applications, *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, 1995, pp. 381–391.

[7] Randell, B.: System Structure for Software Fault Tolerance, *IEEE Transactions on Software Engineering*, Vol. SE-1,No. 2(1975), pp. 220–232.

[8] Swift, M., Bershad, B., and Levy, H.: Improving the Reliability of Commodity Operating Systems, *Proceedings of the 19th Symposium on Operating Systems Principles*, 2003, pp. 207–222.