Vol.2010-OS-115 No.1
2010/8/3

# オペレーティングシステムのメモリ情報を考慮した仮想マシンスナップショット高速化手法

山木田　和哉[†1]　山田浩史[†1,†2]
吉田哲也[†1]　河野健二[†1,†2]

仮想マシン (VM) スナップショット機能は，任意の時点での VM の状態を保存・復元する機能であり，ディザスタリカバリやアップグレードのためのバックアップ取得手法として広く用いられている．しかし，既存のスナップショット機能には保存・復元に要する時間が長いという問題点が存在する．また，既存のスナップショット機能は VM に割り当てるメモリ量が大きいほどディスク I/O が頻発し，保存・復元に要する時間が長くなる．従って，ユーザは作業を長く中断されてしまうために定期的なスナップショットの取得を怠る可能性がある．そこで本研究では，オペレーティングシステム (OS) のメモリ使用状況を考慮したスナップショット高速化手法 *SonicShot* を提案する．SonicShot では，OS の持つメモリ情報に着目して保存する内容を取捨選択し，スナップショットのサイズを縮小させる．具体的には，ソフトステートなメモリ領域を破棄し，システムが使用していないメモリ領域を保存しないようにする．これにより，スナップショットとして保存するデータが少なくなり，保存・復元に要する時間を短縮することができる．実験では既存のスナップショット機能に対し，保存・復元に要する時間を最大 95 % 削減することを確認した．

# Accelerating Saving/Restoring process of VM Snapshots using Kernel Memory Information

Kazuya Yamakita,[†1] Hiroshi Yamada,[†1,†2]
Tetsuya Yoshida[†1] and Kenji Kono [†1,†2]

Virtual machine (VM) snapshot is useful for disaster recovery and safe software update. Unfortunately, it is time-consuming to save and restore VM snapshots because severe disk I/O is involved especially when used for huge memory VMs. As a result, the saving/restoring processes cause significant downtime of the system. This obtrusive behavior discourages the use of VM snapshot and hence the users may fail to recover from system crashes caused by disasters and unsafe updates. We present *SonicShot*, a technique for accelerating the saving/restoring processes of VM snapshots. SonicShot prunes VM memory images to shrink the memory checkpoints; it examines the VM image and avoids saving pages that are unnecessary for the system to work correctly after the restore operation. Shrunk memory checkpoints reduce the amount of disk I/O involved in the snapshot operations. The preliminary experiments suggest that SonicShot reduces the downtime for VM snapshot saving/restoring by up to 95 %.

## 1. はじめに

仮想化技術には様々なメリットや機能が存在するが，その中の 1 つであるスナップショット機能はバックアップ・リカバリのために広く用いられている便利な機能である．このスナップショット機能を用いると，任意の時点での仮想マシン（VM）の状態をバックアップとして保存することができる．また，保存したスナップショットを基に，過去の時点における VM の状態を復元することも可能である．ここでの VM の状態とは，メモリ，レジスタ，ディスク内容などシステム全体の状態を意味する．このスナップショット機能は，様々な用途で用いられている．例えば，最も一般的な使われ方として，ディザスタリカバリのための定期的なバックアップの取得が挙げられる．他にも，ソフトウェアアップデートなどシステムの実行に異常を来し得る操作を行う際に，スナップショット機能が用いられる．また，実際にスナップショット機能を利用した既存システムや商用サービスも存在する [1,2].

スナップショット機能は様々な用途で利用されつつあるが，既存のスナップショット機能には問題が存在する．それは，既存のスナップショットの保存・復元に要する時間が非常に長いということである．そのため，スナップショット取得によって作業が長く中断されてしまい，デスクトップ環境においてユーザがスナップショット機能を利用しなくなるということが考えられる．結果として，定期的なスナップショットの取得を疎かにし，実際に不具合があったときにはシステムのバックアップが存在しないという状況が起こりうる．また，スナップショットの保存・復元はディスク I/O を伴うため，定期的にスナップショットを保存・復元するサーバ環境であっても，ディスク I/O 競合による他 VM のパフォーマンスへの影響が著しくなってしまう．

†1 慶應義塾大学
Keio University
†2 科学技術振興機構 戦略的創造研究推進事業
JST CREST

また，既存のスナップショット機能は VM に割り当てるメモリ量が大きいほど，ディスク I/O が頻発し，保存・復元に要する時間が長くなる．近年，マシンの搭載メモリが急激に増えてきているため，このままではスナップショットの保存・復元に要する時間がさらに長くなると考えられる．

そこで本研究では，OS のメモリ使用状況を考慮したスナップショット高速化手法である *SonicShot* を提案する．SonicShot は，OS の持つメモリ情報に着目して保存する内容を取捨選択し，スナップショットのサイズを縮小させる．これにより，スナップショットとして保存するデータが少なくなり，スナップショットの保存に要する時間を短縮することができる．具体的には，バッファキャッシュのようなソフトステートなメモリページを破棄し，システムが使用していないメモリ領域を保存しないようにする．これらのメモリページは破棄しても再度ディスクから読み込むことができるため，保存せずともスナップショット取得時と復元後でシステムの整合性は保たれる．また，SonicShot を用いると保存したスナップショットのサイズが縮小するため，スナップショットの保存だけでなく復元に要する時間も短縮することができる．従って，SonicShot を用いることで，既存の問題点であるスナップショットの保存・復元に要する時間を短縮することが可能である．

SonicShot を，オープンソースの仮想化ソフトウェア Xen 3.4.1 [3,4]，さらにその上で稼働する Linux 2.6.18 カーネル上に実装した．また，SonicShot の有用性を検証するために，ゲスト OS に割り当てるメモリ量やゲスト OS がユーザレベルで使用中のメモリ量を変更しながら，スナップショットの保存・復元に要する時間を計測した．この実験により，SonicShot を用いることで既存のスナップショットに対し，保存に要する時間を最大 94.2%，復元に要する時間を最大 95.3% 削減することができた．

本論文の構成は以下の通りである．2 章では既存のスナップショット機能や関連研究について述べる．3 章では提案機構 SonicShot について説明する．4 章では SonicShot の実装について説明する．5 章では SonicShot の有用性を確認した実験について述べる．6 章では今後の課題について説明する．最後に 7 章で本論文をまとめる．

## 2. 関 連 研 究

本章ではまず，スナップショット機能を実際に搭載している仮想化ソフトウェアを挙げ，それぞれ既存のスナップショット機能について説明するとともに，その問題点を指摘する．次に，本論文と同様にスナップショット機能の改善を試みた関連研究を紹介する．

オープンソースの仮想化ソフトウェアである Xen [3,4] にもスナップショット機能が搭載されており，VM の状態を保存・復元することが可能である．しかし，Xen のスナップショット機能は保存・復元に要する時間が長いという問題点が存在する．なぜなら，Xen のスナップショット機能は，メモリの使用状況に関わらずゲスト OS に割り当てたメモリ領域を全てスナップショットとして保存するためである．Xen では，ゲスト OS 起動時にゲスト OS へ割り当てる分のマシンメモリを全てマッピングし，そのマッピング表である P2M テーブル（Physical To Machine Table）を作成する．そして，Xen のスナップショット機能は P2M テーブルに存在する各エントリを全てスナップショットとして保存する．従って Xen のスナップショット機能は，メモリの使用状況に関わらずゲスト OS に割り当てたメモリ領域を全てスナップショットとして保存することになり，スナップショットの保存・復元に要する時間が長くなってしまう．

VirtualBox [5] のスナップショット機能は，ゲスト OS が一度でも使用したことのあるメモリ領域のみをスナップショットとして保存する．そのため，ゲスト OS の起動直後など，一度も使用していないメモリ領域が多く存在する場合には，スナップショットの保存・復元に要する時間を短縮することができる．しかし，ゲスト OS が一度でも多くのメモリを必要とすると，それ以降はスナップショットの取得に要する時間が長くなってしまう．なぜなら，一度でもゲスト OS に使用されたメモリ領域は，スナップショット取得時に使用されていなくとも，スナップショットとして保存されるためである．従って，スナップショットを取得する時点では使用されていないメモリ領域もスナップショットとして保存されることになり，保存・復元に要する時間が長くなってしまう．

VMware の製品にもスナップショット機能は搭載されており，デスクトップ環境でもサーバ環境でも使われている．ここで，VMware のスナップショット機能は有償の仮想化ソフトウェアのみにしか搭載されておらず，また VMware 社はスナップショット機能の具体的な性能について公表していないため，定量的な比較や詳細の説明は難しい．よって，VMware のスナップショット機能については，VMware 社の公表している文献 [6,7] を参考に推論していく．VMware のスナップショット機能は他の仮想化ソフトウェアにおけるスナップショット機能と同様，特定時点の VM 状態を丸ごと保存できるだけでなく，前回のスナップショットとの差分のみを保存することもできる．ここでいう差分というのは，メモリやディスク内容を指すと考えられる．従って，データセンタなど比較的頻繁にシステムのバックアップを作成するような環境では効果的であるといえる．なぜなら，スナップショットを取得する間隔が比較的短いため，前回からの差分が少ない状態でスナップショットを保存できるからである．しかし，差分のみの保存方式の場合，保存に要する時間は短縮できるが，復元に要す

る時間は短縮できない．なぜなら，差分のみの保存であっても VM の状態を復元する際には Xen と同様，VM に割り当てたメモリ領域を全て復元することになるからである．

最後に，本論文と同様にスナップショット機能の改善を試みた関連研究を紹介する．VNsnap [8] は，仮想ネットワーク環境下においてスナップショット機能がもたらすダウンタイムを最小限に抑えるシステムである．VNsnap ではスナップショット機能を高速化するために，Live Migration [9] をベースアイデアとした Pre-Copy 方式のスナップショット機能を採用している．スナップショット取得命令が発行されると，VM を稼働させた状態でメモリ領域をスナップショットとして書き出す．そして，書き出し中に更新されたメモリ領域を再度スナップショットに上書きする．このループを何回か繰り返した後に，VM を停止させて残りの更新されたメモリ領域をスナップショットに書き出して処理が完了する．しかし，VNsnap はユーザがスナップショットの取得命令を発行後，しばらくしてからのメモリ領域をスナップショットとして保存することになるため，ユーザの求める時点におけるスナップショットを取得できない可能性がある．なぜなら，最終的にスナップショットとして保存されるのは，VM 稼働中に更新されたメモリを書き出すループが完了した後のメモリ領域となるためである．つまり，ユーザがスナップショットを取得したい時点から少し経過した時点でのスナップショットが保存されることになる．よって，ユーザの保存したい VM の状態をスナップショットとして保存できないことになる．また，VNsnap はスナップショットを取得する際のダウンタイムを削減するというのが目的であるため，復元に要する時間は短縮できない．

また，Zhang らはリストア操作に伴うダウンタイムを削減するために，リストア時にはワーキングセットをプリフェッチしてからシステムをリスタートし，残りのメモリをオンデマンドに復元していくという手法を提案している [10]．リスタート後にアクセスされるであろうメモリ領域を予測して，そのメモリ領域から優先的に復元していくことで，パフォーマンスの低下を抑えている．具体的には，スナップショットを取得した時点前後のメモリアクセスを記録しておいて，システムリスタート後にはそのメモリ領域から復元していく．これは，スナップショットから状態を復元した後も，スナップショットを取得したときと同様の動作をするという想定に基づいており，これが当てはまる場合はパフォーマンスの低下を大きく抑えることができる．しかし，この手法はリストア操作に伴うダウンタイムを削減することが目的であり，リストア操作に伴うディスク I/O 量を削減するわけではないので，稼働中のシステムや他 VM のパフォーマンスを低下させることになる．また，実際は全てのメモリ領域を復元することになるので，リストア操作全体に要する時間も削減できない．

## 3. 提案手法 SonicShot

本研究では，OS のメモリ使用状況を考慮したスナップショット高速化手法である *SonicShot* を提案する．既存のスナップショット機能と SonicShot の概要図を図 1 に示す．SonicShot は，OS の持つメモリ情報に着目して保存する内容を取捨選択し，スナップショットのサイズを縮小させる．スナップショットのサイズを縮小させることにより，スナップショットの保存に要する時間の短縮することができる．具体的には，スナップショットを取得する前にバッファキャッシュのようなソフトステートなメモリ領域を破棄する．これらのメモリ領域を破棄した後，システムが使用していないメモリ領域のマッピングを P2M テーブルから削除することで，これらのメモリ領域を保存しないようにした．なお，本論文ではソフトステートなメモリ領域やシステムが使用していないメモリ領域を**ソフトページ**と称する．ソフトページは破棄しても再度復元することができるため，保存せずともスナップショット取得時と復元後でシステムの整合性は保たれる．ソフトページを破棄して保存するメモリ領域を減らすことで，スナップショットの保存に伴うディスク I/O 量を減らすことができるので，保存に要する時間を短縮することができる．また，SonicShot を用いると保存したスナップショットのサイズが縮小するので，スナップショットの保存だけでなく復元も速くなる．

SonicShot において破棄するソフトページとして，バッファキャッシュが挙げられる．スナップショット取得前にバッファキャッシュを解放することで，これをスナップショットとして保存しないようにした．バッファキャッシュを解放してもファイルデータ本体はディスク上に存在するため，ディスクから再度読み込むことで，スナップショット取得時と復元後のシステムの整合性は保たれる．多くの OS では，空きメモリ領域があるとそのほとんどを

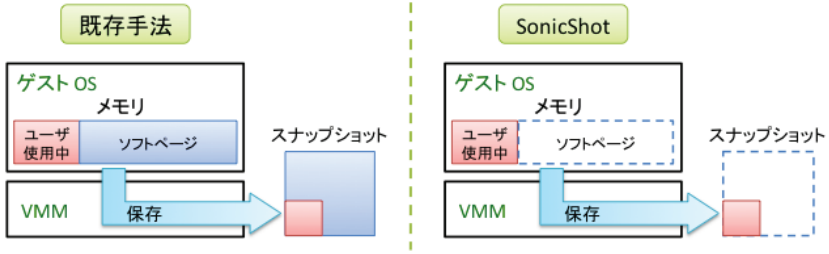


**図 1** 既存のスナップショット機能と SonicShot の概要図

バッファキャッシュとして利用する．これはファイルデータへのアクセスを高速化し，システム全体のパフォーマンスを向上させるが，スナップショット取得時にはネックとなる．なぜなら，メモリの使用領域が少ない場合でも，ほとんどの空きメモリ領域をバッファキャッシュとして利用するため，システムに使われていないメモリ領域がほとんど存在しなくなるからである．よって，VirtualBox [5] のようにシステムが一度使用したことのあるメモリ領域のみをスナップショットとして保存する場合でも，ほとんど全てのメモリ領域を保存することになり，保存・復元に要する時間が長くなってしまう．

また，リソースマネージャが自身で保持しているキャッシュ領域もソフトページであり，具体的にはスラブキャッシュが挙げられる．スラブキャッシュはカーネルのメモリオブジェクト用キャッシュとして使用されているメモリ領域なので，破棄したところでシステムの整合性に問題はなく，バッファキャッシュと同じように解放可能である．

さらに，システムが使用していないフリーなメモリ領域もソフトページとして挙げられる．なぜなら，スナップショット取得時にフリーなメモリ領域に該当するマッピングを削除しても，リストア後に再度マッピングを作成してフリーなメモリ領域を再割り当てすることができるからである．

ただし，これらのソフトページを破棄してからスナップショットを保存すると，スナップショットからリストアした後しばらくは，データアクセスのパフォーマンスが多少低下すると考えられる．なぜなら，スナップショット取得前にバッファキャッシュやスラブキャッシュを破棄するため，スナップショットからリストアした直後はこれらのキャッシュが存在しないからである．従ってこれは，スナップショット機能の高速化と，一時的なパフォーマンス低下のトレードオフであるといえる．

また，OS デバッグのような，ある時点での VM の状態を完全保存する必要がある場合には，通常のスナップショット機能の代わりに SonicShot を用いることはできない．なぜなら，SonicShot ではキャッシュなどのソフトページを破棄した状態でスナップショットを保存するので，ソフトページがバグの原因であった場合にデバッグが不可能になってしまうためである．従って，キャッシュなどのソフトページも含めて VM の状態を保存する必要がある場合には SonicShot を用いることはできない．

## 4. 実　　装

SonicShot は，オープンソースの仮想化ソフトウェア Xen 3.4.1 に搭載されているスナップショット機能を用いた．そこで本研究では，ゲスト OS のカーネル（Linux カーネル 2.6.18）にモジュールを追加し，Xen 用のゲスト OS を改変することで提案手法を実現した．

また，本実装では破棄するソフトページとして，バッファキャッシュとスラブキャッシュを対象とした．なぜなら，Linux カーネルは空きメモリ領域をほとんどキャッシュとして利用するため，バッファキャッシュとスラブキャッシュの破棄は効果的に働くと考えられるからである．

### 4.1 SonicShot の全体像

SonicShot における動作の流れ図を図 2 に示す．ドメイン 0 がドメイン U に動作開始を通知し，ドメイン U 内で必要な処理が完了した後にドメイン 0 からスナップショットの取得命令を発行するというのが大まかな流れである．なお，ドメイン間の通信方式については，実装上の手間を省くためにドメイン U 上の Linux にサーバを稼働させ，ドメイン 0 と通信を行うという方法を採用した．

ドメイン 0 からドメイン U へ動作開始の通知が来ると，スナップショット取得前とリストア後でデータ内容の一貫性を保つために，ドメイン U 内でゲスト OS のダーティバッファをディスクへ書き出す．ダーティバッファのディスクへの書き出しは，`sys_sync()` システムコールによって実行する．これは，SonicShot を用いるとリストア後にスナップショット取得前のキャッシュ内容が存在しなくなるためである．

ダーティバッファをディスクへ書き出した後，ソフトページの破棄を行う．具体的に，SonicShot ではバッファキャッシュとスラブキャッシュを解放する．これらのキャッシュは解放後に空きメモリ領域となる．バッファキャッシュを解放するために Linux カーネル内に存

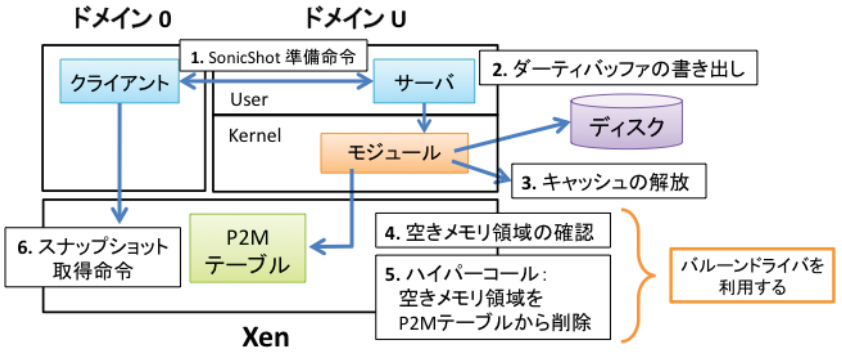


**図 2**　SonicShot の流れ図

在する drop_pagecache() 関数を用い，スラブキャッシュを解放するために drop_slab() 関数を用いた．

キャッシュ解放後，空きメモリ領域のマッピングを P2M テーブルから削除する．Xen のスナップショット機能は P2M テーブルに存在する全てのマッピングに対してメモリ領域を保存するため，空きメモリ領域のマッピングを P2M テーブルから削除することで，使用中のメモリ領域のみをスナップショットとして保存する．空きメモリ領域のマッピングを削除するために，空きメモリ領域を特定し，空きメモリ領域に該当する P2M テーブルエントリを削除するハイパーコールを発行する．

ここまで完了したら，スナップショット取得の準備が完了した旨をドメイン U からドメイン 0 に通知する．なお，これらドメイン U 内での処理は lock_kernel() 関数にてジャイアントカーネルロックをかけた状態で行う．これには，2つの理由がある．まず 1 つ目の理由は，ダーティバッファをディスクへ書き出してからバッファキャッシュを解放するまでに，他のプロセスによって新たにダーティバッファが作られるのを防ぐためである．2つ目の理由は，バッファキャッシュを解放してから空きメモリ領域のマッピングを削除するまでに，他のプロセスによって新たにバッファキャッシュが作られるのを防ぐためである．

そして，スナップショット取得の準備が完了した段階で，ドメイン 0 から VMM へスナップショットを取得するよう命令し，VMM が対象 VM のスナップショットを保存する．

### 4.2 空きメモリ領域の削除

Xen のスナップショット機能は P2M テーブルに存在する全てのマッピングに対してメモリ領域を保存するため，SonicShot ではキャッシュの解放を行った後に空きメモリ領域のマッピングを P2M テーブルから削除する．そして，ゲスト Linux が使用中のメモリ領域のみをスナップショットとして保存する．これにより，スナップショットの保存に要する時間の短縮を図る．

そこで P2M テーブル上のソフトページのエントリを削除するために，空きメモリ領域を特定した後，空きメモリ領域に該当する P2M テーブルエントリを削除するハイパーコールを用いた．どこが空きメモリ領域かというのは OS が把握しているため，実装上は削除すべき空きメモリページの数を調べてハイパーコールを発行すればよい．なお，DMA 領域のための空きページは確保しておかないとシステムに異常を来すため，DMA 用の空きページはマッピング削除の対象から除くことにした．

### 4.3 削除したメモリ領域の再割り当て

SonicShot では，リストア後のゲスト OS に対して空きメモリを再度割り当てる必要がある．なぜなら，SonicShot は空きメモリ領域のマッピングを削除してスナップショットを保存するため，リストア後には空きメモリ領域のマッピングが存在せず，ゲスト OS の総メモリ量が減ってしまう．例えば，ゲスト OS にメモリを 1GB 割り当てて起動した場合に，SonicShot によって 800MB の空きメモリ領域を削除してスナップショットとして保存したとする．すると，そのスナップショットを基にリストアしたとき，ゲスト OS には 200MB しか割り当てられていない状態で復元されてしまう．そこで SonicShot では，この場合リストア直後に 800MB の空きメモリを割り当てる必要がある．

リストア時における削除したメモリ領域の再割り当てを行うために，バルーンドライバを用いて実現した．バルーンプロセスを利用することで，復元するゲスト OS のメモリ量が起動時のメモリ量よりも減っていた場合に，ゲスト OS の起動時割り当てメモリ量まで空きメモリを割り当てることが可能となる．つまり，前述した例を用いると，リストア直後に 800MB の空きメモリを割り当てることで，復元後はゲスト OS の割り当てメモリ量が 1GB となる．よって，SonicShot を用いてスナップショットを保存しても，リストア後にキャッシュがなくなるだけで，システムの整合性に問題はない．

## 5. 実　　験

本研究では，SonicShot の有用性を示すために 2 つの実験を行った．なお，本実験は 16GB のメモリ，Quad-Core Xeon 3GHz プロセッサ，73GB SAS NHS 10,000 rpm のディスクを搭載した物理マシン上で行った．物理マシン上には Linux 2.6.18 カーネルを導入し，OS は x86_64 アーキテクチャ用の Fedora9 を使用した．スナップショットの保存対象となる VM 上には SonicShot を実装した Linux 2.6.18 カーネルを導入し，OS は x86_64 アーキテクチャ用の Fedora8 を使用した．

さらに，本実験時にはゲスト OS である Fedora8 のデーモンを一部停止させた．具体的には，ConsoleKit，acpid，auditd，bluetooth，cpuspeed，cupsd，gpm，mdmonitor，pcscd，restorecond，sendmail のデーモンを停止させた．これらのデーモンは起動直後から常にメモリを使用して稼働しており，システムの運用上では特段必要がない．従って，本実験時で起動直後におけるゲスト OS の使用メモリ量を厳密に定めるため，これらのデーモンを停止させた．

### 5.1 VM に割り当てるメモリ量と保存・復元に要する時間の比較

#### 5.1.1 目的と方法

本実験の目的は，提案手法である SonicShot の保存や復元に要する時間を測定し，既存

のスナップショット機能と比較することである．既存のスナップショット機能では，ゲスト Linux に割り当てるメモリ量が大きいほど，スナップショットの保存・復元に要する時間が長くなる．それに対し，SonicShot はゲスト Linux へ割り当てるメモリ量に依らず，スナップショットの保存・復元に要する時間が短縮できることを本実験で検証する．つまり，ゲスト Linux に割り当てるメモリ量を増やした場合でも，スナップショットの保存・復元に要する時間は増加せずに一定であることを示す．

ゲスト Linux がユーザレベルでメモリを 200MB 使用している状況で，ゲスト Linux に割り当てるメモリ量を変えながら，SonicShot の保存・復元に要する時間，既存のスナップショットの保存・復元に要する時間をそれぞれ 30 回ずつ測定し，その平均値を計算した．ゲスト Linux に割り当てるメモリ量は，VM を起動する前に設定ファイルを書き換えることで，256MB から 8GB まで変化させた．また，ゲスト Linux がユーザレベルで使用している 200MB の中にはキャッシュメモリは含まれない．ちなみに，これらの保存に要する時間というのは，実際にドメイン 0 が対象ドメインの保存を命令し，全ての実行が完了するのに要した時間である．復元に要する時間というのも同様で，ドメイン 0 がリストア命令を発行し，リストアした後に空きメモリを割り当てて全て動作が完了するのに要した時間である．

#### 5.1.2 実験結果

測定結果を図 3 に示す． 図 3 は，ゲスト Linux がユーザレベルでメモリを 200MB 使用している状況で，ゲスト Linux のスナップショット保存・復元に要する時間を，ゲスト Linux の割り当てメモリ量毎に比較したものである．図 3 より，SonicShot はゲスト Linux へ割り当てるメモリ量に依らず，スナップショットの保存や復元に要する時間を短縮できていることが分かる．それに対して既存のスナップショット機能は，割り当てメモリ量が増加すると保存・復元に要する時間も増加している．ゲスト Linux にメモリを 8GB 割り当てた状態で既存のスナップショット機能を使用すると，保存・復元に合わせて約 242 秒もの時間が必要であった．一方 SonicShot は，割り当てメモリ量を増加させても保存・復元に要する時間はほぼ増加せずに一定であり，総じて保存・復元に要する時間を削減することができた．具体的には，スナップショットの保存に要する時間を最大 94.1%，復元に要する時間を最大 95.1% 削減することができた．

### 5.2 VM が使用中のメモリ量と保存・復元に要する時間の比較

#### 5.2.1 目的と方法

本実験の目的は，前実験と同じく提案手法である SonicShot の保存や復元に要する時間

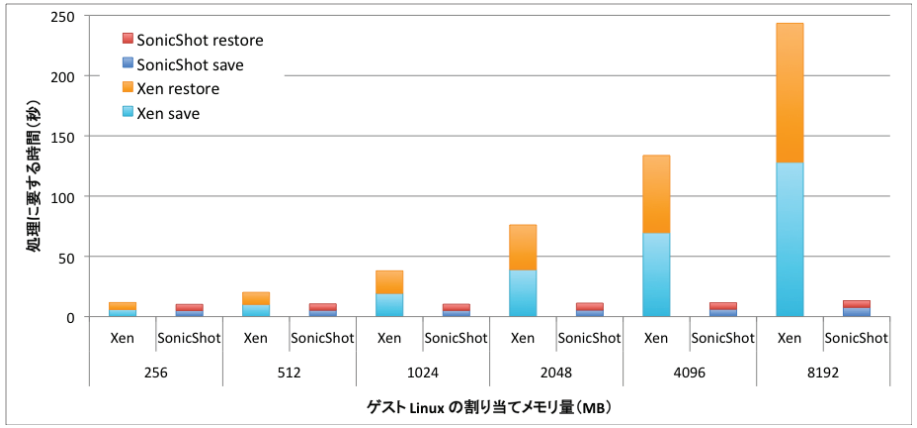


**図 3** ゲスト Linux の使用中メモリが 200MB の場合の保存・復元に要する時間の比較

を測定し，既存のスナップショット機能と比較することである．既存のスナップショット機能では，ゲスト Linux に割り当てたメモリ領域を全てスナップショットとして保存するため，ゲスト Linux のメモリ使用量に依らずスナップショットの保存・復元に要する時間が長くなる．それに対し，SonicShot はゲスト Linux のメモリ使用量に応じてスナップショットの保存・復元に要する時間を短縮できることを本実験で検証する．つまり，ゲスト Linux がユーザレベルで使用しているメモリ領域が少ない場合には，スナップショットの保存・復元に要する時間が短縮するということを示す．

ゲスト Linux にメモリを固定的に割り当てた状況で，ゲスト Linux がユーザレベルで使用中のメモリ量を変えながら，SonicShot の保存・復元に要する時間，既存のスナップショットの保存・復元に要する時間をそれぞれ 30 回ずつ測定し，その平均値を計算した．ゲスト Linux に割り当てるメモリ量は，VM を起動する前に設定ファイルを書き換えることで，256MB から 8GB まで変化させた．ゲスト Linux がユーザレベルで使用中のメモリの中にはキャッシュメモリは含まれない．なお，起動後のゲスト Linux 上で `malloc()` 関数を用いて動的にメモリを確保して，そのままスリープするプロセスを実行することで，ゲスト Linux がユーザレベルで使用中のメモリ量を変化させた．また，これらの保存・復元に要する時間という定義は前実験と同様である．

#### 5.2.2 実験結果

測定結果の一部を図 4 に示す．図 4 は，ゲスト Linux にメモリを 8GB 割り当てた場合

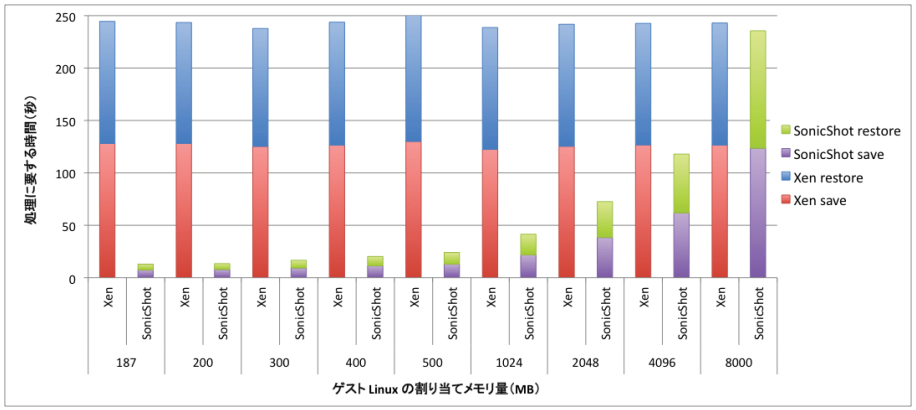


**図 4** メモリを 8GB 割り当てた場合の保存・復元に要する時間の比較

の，既存のスナップショット機能と SonicShot で，保存と復元に要する時間を比較したグラフである．また，図 4 の横軸で最左列のメモリ量は，ゲスト Linux 起動直後の使用メモリ量である．

図 4 より，SonicShot は既存のスナップショット機能よりも高速に保存・復元を実行できていることが分かる．それに対して既存のスナップショット機能は，ゲスト Linux がユーザレベルで使用中のメモリ量に関わらず，保存・復元には常に一定の時間を要する．具体的には，ゲスト Linux にメモリを 8GB 割り当てた状態で，ゲスト Linux 使用中のメモリ量が少ない場合でも，保存・復元に合わせて約 243 秒もの時間を必要とした．これは，既存のスナップショット機能ではゲスト Linux に割り当てた全てのメモリ領域をスナップショットとして保存するためである．これに対して SonicShot では，ゲスト Linux がユーザレベルで使用中のメモリ量に応じてスナップショットの保存・復元に要する時間を短縮させることができた．具体的には，SonicShot を用いることで，スナップショットの保存に要する時間を最大 94.2%，復元に要する時間も最大 95.3% 削減することができた．これは，SonicShot がシステムの使用しているメモリ領域のみを保存したことを示している．

ただし，図 4 の最右列，つまりゲスト Linux に割り当てたメモリをほとんど全てユーザレベルで使用している状況では，SonicShot を用いてもスナップショットの保存・復元に要する時間はあまり削減できなかった．これらの状況では，ゲスト Linux に割り当てたメモリ領域をほとんどシステムで使用しているため，既存のスナップショット機能と比べて，保存するメモリ量がほぼ変わらないことになる．そのため，スナップショットの保存・復元に要する時間をあまり削減できなかった．しかしこれは，SonicShot を拡張することで改善可能であると考えられる．現在の SonicShot は OS のみを修正し，アプリケーションを改変するということは行っていない．アプリケーションの中にはアプリケーション内でキャッシュを持つものも多く存在し，スナップショットを取得する前にこれらのキャッシュも破棄するように修正をすれば，アプリケーションが多くのメモリを使用している状況でもスナップショットの保存・復元に要する時間を短縮できると考えられる．

## 6. 今後の課題

今後の課題として，SonicShot から状態を復元した際のパフォーマンス変化を検証するということが挙げられる．SonicShot では，スナップショットを取得する前にソフトステートなメモリ領域を破棄する．現状ではバッファキャッシュとスラブキャッシュをスナップショット取得前に破棄している．従って SonicShot からリストアすると，これらのキャッシュが存在しなくなる．これにより，リストア直後はシステムのパフォーマンスが低下すると考えられる．具体的には，キャッシュが存在しなくなるためにディスクアクセスが頻繁に起こり，システムのスループットが低下すると考えられる．ただし，一度ディスクにアクセスしたファイルデータは再びメモリ上にキャッシュされるため，リストア後しばらくするとシステムのパフォーマンスは回復する．

また，既存のスナップショット機能を利用するシステムへ SonicShot を導入するということも課題として挙げられる．なぜなら，SonicShot は既存のスナップショット機能を改変して実装したためである．従って，SonicShot を導入することにより，既存のスナップショット機能を利用するシステムのパフォーマンスを上げることが可能であると考えられる．例えば，通常の OS 再起動の代わりにスナップショット機能を用いて再起動を行う FlashReboot [1] では，ゲスト OS 起動直後の状態をスナップショットとして保存し，リストア機能で再起動と同等の効果を得る．実験により FlashReboot が従来の再起動よりも高速に処理を完了させることを証明しているが，ここで SonicShot を既存のスナップショット機能の代わりに用いることで，さらに高速なシステムの再起動が実現できると考えられる．

## 7. おわりに

本論文では，仮想化環境におけるバックアップ取得機能であるスナップショット機能を高速化する手法 SonicShot を提案した．SonicShot は，OS の持つメモリ情報に着目して保存

する内容を取捨選択し，スナップショットのサイズを縮小させることで，従来長時間必要であった保存に要する時間を短縮する．具体的には，スナップショットを取得する前に，バッファキャッシュのようなソフトステートなメモリ領域を破棄し，システムが使用していないメモリ領域のマッピングを P2M テーブルから削除することで，これらのメモリ領域を保存しないようにした．これにより，スナップショットとして保存するメモリ領域が縮小できるので，スナップショットの保存に要する時間を短縮することができた．キャッシュのようなソフトステートなメモリ領域は，破棄しても再度ディスクから読み込むことができるため，保存せずともシステム稼働上の問題はない．また，SonicShot を用いると保存したスナップショットのサイズが縮小するので，スナップショットの保存だけでなく復元に要する時間も短縮することができる．

SonicShot をオープンソースの仮想化ソフトウェア Xen 3.4.1，さらにその上で稼働する Linux 2.6.18 上に実装した．また，SonicShot の有用性を検証するために，ゲスト OS に割り当てるメモリ量とゲスト OS が使用中のメモリ量を変更しながら，スナップショットの保存・復元に要する時間やスナップショットのサイズを測定した．この実験により，SonicShot を用いることでスナップショットの保存に要する時間を最大 94.2%，復元に要する時間を最大 95.3%，スナップショットのサイズを最大 96.9% 削減することができた．

## 参 考 文 献

1) Kobayashi, T., Yamada, H. and Kono, K.: Quick Reboot-based Recovery for Commodity Operating Systems in Virtualized Server Consolidation, *Proceedings of EuroSys 2010 Workshop on Isolation and Integration for Dependable Systems (IIDS'10)* (2010).
2) MokaFive: MokaFive. http://www.mokafive.com.
3) XenSource: Xen. http://www.xen.org.
4) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and Art of Virtualization, *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp.164–177 (2003).
5) Sun Microsystems: VirtualBox (2008). http://www.virtualbox.org.
6) VMware: VMware Technical Resource Center. http://www.vmware.com/technical-resources/.
7) VMware: VMware Data Recovery. http://www.vmware.com/files/jp/pdf/data_recovery_datasheet.pdf.
8) Kangarlou, A., Eugster, P. and (contact author), D.X.: VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime, *Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)* (2009).
9) 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 and Implementation (NSDI '05)*, pp. 273–286 (2005).
10) Zhang, I., Barr, K., Garthwaite, A., Baskakov, Y., Pool, J. and Christopher, K.: Fast Restore of Checkpointed Memory using Working Set Estimation, *Presented at the 22th ACM Symposium on Operating Systems Principles (SOSP 2009) Work-In-Progress Poster Session* (2009).