2008－OS－109 (13)
2008／8／7

プロセスの実行状態を保持したままカーネルの更新ができるシステム高速再起動方式

小比賀亮仁† 菅原智義†

近年、セキュリティー対策としてカーネルにパッチを適用する機会が増えている。カーネルにパッチを適用するには、システムを再起動しなくてはならない。しかし、システムは24時間365日、ユーザに対してサービスを提供し続けることが求められている。よって、パッチ適用の際のシステム再起動時間は可能な限り短縮する必要がある。この問題に対して、我々は、システムの再起動時間のうち、プロセスの再起動が最も時間のかかる処理であることに着目し、プロセスの状態は保持したままカーネルのみを再起動するシステムの高速再起動方式を開発した。同方式は、メモリ上のプロセスデータは保持したままカーネルのみを入れ替え、その後、メモリ上に保持されているプロセスデータを用いて、再起動前に実行されていたプロセスを途中状態から再開することができる。本方式により、従来のシステム再起動時間を最大で約48%削減することができた。

A quick reboot mechanism with taking over process states for reducing down time

Akihito KOHIGA† Tomoyoshi SUGAWARA†

In this paper, we suggest a quick reboot mechanism which allows us to reduce down time on our system during updating kernel. There are non-stop systems which gives us some services during 24 hours, every day. However, in the past several years, number of cases that we have to apply some kernel patches for security updates has been increasing. In order to do it, we have to reboot our systems. Our mechanism reboot systems to updates the kernel with preserving some memory areas which was used by processes. After rebooting we resumes recent processes which were executed on our system before rebooting, using the memory areas which are preserved. Our mechanism can reduce reboot time of our system because our mechanism can get rid of disk access and reduce numbers of memory copy during rebooting. We have accomplished reducing 48% of system reboot time.

1. 背景

近年、セキュリティー対策としてカーネルにパッチを適用する機会が増えている。一方、システムは24時間365日、ユーザに対してサービスを提供し続けることが求められている。カーネルにパッチを適用するには、システムを再起動しなくてはならない。システムを再起動すると、当該システム上で実行されていたプログラム、すなわちサービスプロセスを停止することになる。

サービスプロセスには、状態を持つサービスプロセスと状態を持たないサービスプロセスがある。状態を持たないサービスプロセスの具体例としては、ATM、証券取引等のシステムが挙げられる。すなわちトランザクション処理を行うタイプのサービスプロセスである。状態を持つサービスプロセスの具体例としては、電話交換機や Video On Demand（VOD）サービス、電子会議等のシステムが挙げられる。すなわち、セッションを常に張り続けるタイプのサービスプロセスである。

本稿では、状態を持つサービスプロセスについて、当該サービスプロセスの停止時間を短縮する方法について論じる。

状態を持つサービスプロセスの停止時間を短縮する方法としては、冗長化された待機系システムへのプロセスマイグレーションや、ソフトウェアの動的更新技術などが挙げられる。

プロセスマイグレーション[11]とは、サービスプロセスを無停止で稼働状態のまま、別のサーバへ移動させることを指す。プロセスマイグレーションを利用することによって、システムの再起動をユーザから隠蔽することができる。しかし、プロセスマイグレーションを実行するには、冗長化された待機系システムが必要となりプロセスマイグレーションを行わないサーバと比較して、サーバの設置コストが二倍になる。よって、単体で運用されているようなサーバには利用できない。

一方、サービスプロセスの停止時間を短縮するのではなく、サービスプロセスの停止を回避する方法としては、ソフトウェアの動的更新技術を用いる方法がある。ソフトウェアの動的更新技術[8]とは、動作中のソフトウェアを停止させずに、当該ソフトウェアの一部を変更する技術である。ソフトウェアの動的更新技術を利用することによってシステムを再起動する必要がなくなる。しかし、ソフトウェアの動的更新技術を構成する機能そのものを更新するには、システムの再起動が必要になる。また、動的更新技術の適用後は、関数ポインタのリダイレクション等の処理を行うことになるので、ソフトウェアの実行速度が低下してしまうという問題がある。電話交換機や VOD サービスなど、リアルタイム性が要求されるサービスにおいて、ソフトウェアの実行速度低下は致命的である。

† NEC システムプラットフォーム研究所

以上の議論により、単体で運用されているサーバにおいて、サーバの処理性能を劣化させずに、カーネルにパッチを適用するには、サービスプロセスの停止時間を短縮する、すなわちシステムの再起動時間を短縮させることが妥当であるといえる。

2. 問題点

1 節で述べられた課題を整理すると以下のようになる。すなわち、単体で運用されるサーバの再起動時間を短縮させ、かつ、サービスプロセス（以下プロセス）の実行状態を再起動後も継続させることが要求される。

システムの再起動時間のうち、最も時間のかかる部分は、ユーザプロセスの起動時間である。ユーザプロセスは、デーモンやユーザが実行中のプログラムなどから構成される。システムの再起動は大きく分けて以下の 4 つの部分から構成される。

1. プロセスのシャットダウン処理（約 26 秒）
2. BIOS の起動処理（約 3 秒）
3. カーネルの起動処理（約 16 秒）
4. プロセスの起動処理（約 40 秒）

表 1 再起動時間の評価環境

CPU	Core2 Duo 2.6Ghz
メモリ	2GB
VMM	VMware workstation6
ホスト OS	Windows Vista
ゲスト OS	Fedora core5(Linux2.6.18)
ゲスト OS の使用メモリ	512MB
起動デーモン数	38

それぞれの処理時間について簡単な計測を行った。計測結果は、各項目の最後のカッコ内の数値である。計測対象となる OS として Fedoracore5 を選択した。Fedoracore5 のインストール環境として、インストールパッケージ「サーバ」と「GNOME デスクトップ環境」を選択した。そして、テキストログインを選択し、デフォルト設定のまま再起動時間を計測した。評価環境は表 1 の通りである。

計測により、システムの再起動時間約 85 秒のうち、プロセスの再起動時間は約 66 秒であり、プロセスの再起動時間がカーネルの再起動時間に比べてはるかに長いことがわかる。よって、プロセスの再起動時間をできるだけ短縮することがシステムの再起動時間を短縮するためには効果的であると言える。

一方、プロセスの実行を継続させる既存技術としては、チェックポインティングやハイバネーションが挙げられる。

チェックポインティングとは、プロセスの実行途中状態の保存と復元を指す。プロセスの実行途中状態をディスクに保存しておき、予期せぬシステムの障害により同プロセスが計算途中で終了させられた場合に、ディスクに保存しておいた実行状態を使ってプロセスを再開させることによって、計算時間の損失を軽減させることができる。システムの再起動においてチェックポインティングを利用する際は、BIOS のハードウェアリセットによりメモリ状態が不定になるので、チェックポイントイメージはメモリ上ではなくハードディスクなどに退避しなくてはならない。通常、チェックポイントイメージはプロセスが使用するメモリ内容をすべてディスクに保存するため、チェックポイントの性能はハードディスクの転送速度に依存する。また、プロセスが利用するメモリ使用量に比例して、プロセスの復元時間が長くなるという欠点がある。

ハイバネーションは電源 OFF 時に実行中のアプリケーションを含めた OS 全体のメモリイメージをハードディスクに保存しておき、電源 ON 時にこれをロードすることで作業を再開する手法である。メモリイメージを直接読み込むことでカーネルやアプリケーションの起動処理を省略できるため、システムの再起動時間を短縮することができる。しかし、カーネル にセキュリティパッチなどの変更を加えたい場合には、ハイバネーションイメージ中の カーネル のメモリイメージを直接書き換える必要があり、このような場合にハイバネーションは利用できない。

以上の論点をまとめると、プロセスの再起動時間をできるだけ短縮することがシステムの再起動時間を短縮するためには効果的であり、プロセスの状態復元にチェックポインティングを利用する場合、チェックポインティングの性能は、ハードディスクの性能に依存し、プロセスのメモリ使用量に比例して、プロセスの再起動時間が長くなるという欠点がある。

3. カーネルのみを入れ替えるシステムの高速再起動方式

プロセスの再起動時間が最も時間のかかる処理であることに着目し、かつチェックポインティングの欠点を克服するため、我々は、プロセスの状態は保持したままカーネルのみを再起動するシステムの高速再起動方式を開発した。我々の開発した高速再起動方式は、メモリ上のプロセスデータは保持したままカーネルのみを入れ替え、その後、メモリ上に保持されているプロセスデータを用いて、再起動前のプロセスを復元する。本方式は、

- 再起動前のカーネルから、プロセスの管理情報をメモリ上に設置した退避領域へ退避する
- プロセスの利用していたメモリ空間は保持しつつ、パッチを適用したカーネルをロードする
- プロセスの管理情報を退避領域から取り出して、プロセスを復元する

本方式は、再起動前にプロセスが利用していた

メモリ領域をそのまま再利用することにより、ディスクアクセスを無くし、また、メモリコピーは、プロセスの管理情報のみなので、プロセスのメモリ使用量に比例して、プロセスの再起動時間が長くなるということはない。

システムの再起動時間を可能な限り短縮することが我々の目的である。本方式の特徴は、プロセスの再起動時間を短縮することができることである。2 節で計測した結果を考慮し、本方式適用によって、システムの再起動時間 77%削減を目標として設定する。

4. 設計

本節では我々の提案する高速再起動方式の実装方法について説明する。本実装では以下の手順でカーネルを再起動させる（図 1）。

1 パッチ適用済みのカーネル(New カーネル）をメモリ上にロードする
2 パッチ適用前のカーネル（Old カーネル）からプロセスの管理情報を一時退避領域にコピーする
3 パッチ適用前のカーネル（Old カーネル）からパッチ適用済みのカーネル（New カーネル）を呼び出す
4 パッチ適用済みのカーネル(New カーネル）は、一時退避領域から、手順 2 においてコピーしておいたプロセスの管理情報を取り出して、プロセスを復元する。

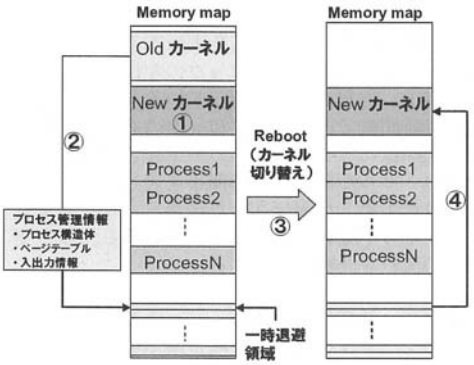

図 1 高速再起動方式

プロセスの管理情報とは、ユーザプログラムの実行に必要な情報であり、カーネルが管理している情報である。プロセス管理情報は、プロセス構造体、ページテーブル、入出力バッファ、その他から構成される。

一時退避領域とは、プロセスの管理情報を保存しておく領域である。一時退避領域は、メモリ上の任意の位置に設置される。パッチ適用済みのカーネルとパッチ適用前のカーネルは、一時退避領域に退避されたプロセス管理情報を引き継ぐために、この領域の位置情報を共有しておく必要がある。

5. 実装

実装は 2 段階に分けて行った。最初にカーネルチェックポインティングを用いた高速再起動方式を実装し、次に、プロセスの管理情報だけを一時退避領域に退避するために、プロセスイメージのゼロコピー機能を実装した。2 段階に実装を分けた理由は、ディスクアクセスを無くすという再起動時間の削減効果が明確な部分を最初に実現すべきであると考えたためである。

5.1 カーネルチェックポインティングを用いた高速再起動方式の実装

カーネルチェックポインティングを用いた高速再起動方式を実現するために、我々は要素技術として以下の既存機能を用いた。

- kexec
- kdump
- TICK（カーネルチェックポインティング）

kexec[4]は、BIOS を経由せずにシステムを起動するための仕組みである。kexec は、起動したいカーネルを予めメモリにロードしておく。再起動の際に、kexec はロードしておいたカーネルを所定の場所（物理メモリの 1MB 付近）にロードしてシステムを起動する。kexec における起動処理高速化のポイントは 2 つある。1 つ目は BIOS によるハードウェアの初期化処理を省略できる点である。2 つ目は、システムのアイドル時間にカーネルをロードすることで、再起動の際のカーネル展開時間が無くなる点である。

kdump[5]は、kexec をベースとしたカーネルのダンプ機能である。kdump は、ロードしたカーネルをそのままの場所で起動することができる。すなわち、再起動後のカーネルは任意の場所から起動することができる。また、再起動後のカーネルは、再起動前に使われていたメモリ領域を使わないので、再起動前の状態をダンプすることができる。kdump は再起動の際に kexec の機能の一部を使って、BIOS の経由なしに再起動後のカーネルを呼び出す。kexec を使うことで BIOS によるハードウェアの初期化処理が無くなり、メモリに以前の状態が保存されていることが保証される。つまり、再起動後のカーネルを使って、再起動前に利用されていたメモリ領域のダンプを取得することができる。

TICK[9]は、Roberto らによって作られたカーネルレベルチェックポインティングツールである。カーネルレベルチェックポインティングは、カーネル本体、もしくはカーネルモジュールによって実行されるチェックポイント・リスタートを指す。カーネルレベルチェックポインティングを用いることによって、ウェブサーバなどのように実用的なプログラムのチェックポインティングが可能となる。

我々は再起動前にプロセスが利用していたメモリ領域を保持するために kexec と kdump を用いた。また、再起動後のカーネルにプロセスを引き継ぐために TICK を用いている。幾つかのカーネルレベルチェックポインティングツールの中から TICK を採用した理由は、他のカーネルレベルチェックポインティングと比較して実装規模が小さく、改造に適していると考えたためである。

これらの既存機能を使った高速再起動方式の概要は以下の通りである。

1. TICK を用いてプロセスのチェックポイントイメージを作成し、RAM ディスクに格納する
2. kexec と kdump を用いてシステムを再起動する。
3. 元の RAM ディスクを再認識し、TICK を用いて、RAM ディスクに保存したチェックポイントイメージを使ってプロセスを復元する

我々は TICK, kexec, kdump の既存機能を利用すると共に、領域確保機能、一時退避領域アクセス用カーネルモジュールを作成した。

図 2 と図 3 は本方式を実現するために必要な機能と、簡単な処理の流れを表している。図 2 は、パッチ適用前のカーネルに必要な機能と、kexec 実行前に行われる処理の流れを表している。図 3 はパッチ適用後のカーネルを実行中に、本方式を実行するための機能と処理の流れを表している。

領域確保機能は、カーネル起動時に特定のメモリ領域を予約するための機能である。予約された領域はプロセス管理情報や、プロセスが利用していたメモリ領域のアドレス情報を保存するために用いる。カーネルチェックポインティングを用いた高速再起動方式の実装においては、チェックポイントイメージを保存するために用いられる。

チェックポイント作成機能は、プロセスが利用していたメモリ領域とプロセスの管理情報をファイルに格納する。つまり、カーネルレベルチェックポインティングツール TICK である。

一時退避領域アクセス用カーネルモジュールは、領域確保機能によって予約されているメモリ領域を識別し、同領域を RAM ディスクとしてユーザに見せるためにデバイスファイルを作成する。RAM ディスクのマウントとフォーマットはユーザプロセスにて行う。

一時退避領域は、一時退避領域アクセス用カーネルモジュールによって作られる RAM ディスクである。一時退避領域を RAM ディスクとして扱う理由は、TICK で作成されるチェックポイントイメージがファイルとして扱われるである。

レジューム機能は、一時退避領域に保存されているチェックポイントイメージを使って再起動前に実行されていたプロセスを復元する。

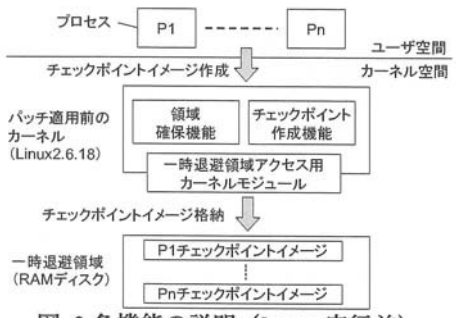

図 2 各機能の説明（kexec 実行前）

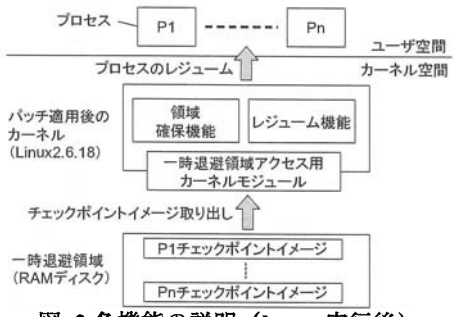

図 3 各機能の説明（kexec 実行後）

図 4 は高速再起動方式を実行する際の処理の流れを示している。各ブロックの左に記されている数字は、処理の順番を表している。数字の左の矢印は時間軸である。各ブロックの右に記されている矢印はパッチ適用前のカーネルとパッチ適用後のカーネルが動作する時間帯を表している。図では処理手順⑤が実行された後からパッチ適用後のカーネルが動作していることがわかる。以下で処理の流れを詳細に説明する。

最初に、領域確保機能は、パッチ適用前のカーネルをブートする際に、一時退避領域を確保する（①）。一時退避領域の位置と容量はカーネルのブートオプションとして追加する。追加後のブートオプションは次のようになる。

kernel /boot/vmlinuz-2.6.18first ro root=/dev/hda1 crashkernel=128M@16M shelter=150M

crashkernel オプションは kdump によって利用されるメモリ領域の容量と位置を示している。領域確保機能は、reserve_bootmem カーネル関数を使ってメモリ領域を予約する。取得したメモリ領域へのポインタは、カーネル内のグローバル変数に格納する。reserve_bootmem カーネル関数は、カーネルのブート時にカーネル内で数ページ（1 ペー

ジは4KB）以上のメモリ領域を確保するための関数である。アットマークの前が容量、後ろが位置を表している。shelter 変数は一時退避領域の容量を表している。一時退避領域は、kdump によって利用されるメモリ領域の直後に配置した。図 5 はカーネル起動後のシステムマップである。図 5 を見ると、一時退避領域が kdump によって利用される領域の直後に配置されていることがわかる（図中の"Shelter area"）。

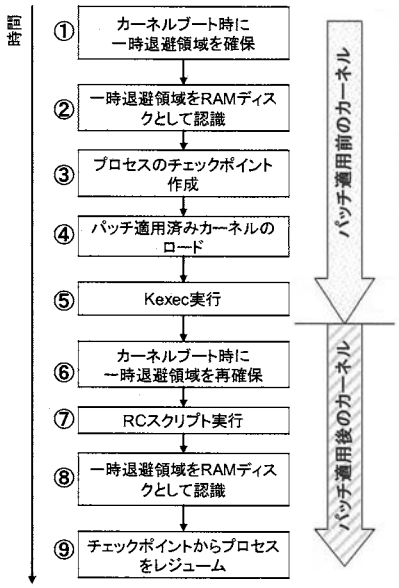

図 4 処理の流れ

```
00100000-1feeffff : System RAM
      00100000-002ee89f : Kernel code
      002ee8a0-003a888b : Kernel data
      01000000-08ffffff : Crash kernel
      09000000-125fffff : Shelter area
1fef0000-1fefefff : ACPI Tables
...
```

図 5 システムマップ(/proc/iomem)

次に、領域確保機能は一時退避領域を RAM ディスクとして認識させる（②）。RAM ディスクとして認識させるために、一時退避領域アクセス用カーネルモジュールを用いる。一時退避領域アクセス用カーネルモジュールは、図 5 に示したシステムマップファイルから一時退避領域の位置と大きさを認識し、領域確保機能から一時退避領域へのポインタを引き継ぐ。そしてデバイスファイルを作成する。デバイスファイルの作成後は、ユーザプロセスによって、RAM ディスクのフォーマットとデバイスのマウントを行う。

次に、チェックポイント作成機能はプロセスのチェックポイントイメージを作成する（③）。チェックポイントイメージの格納先には RAM ディスクを指定する。

次に、パッチ適用済みカーネル(New カーネル)をロードする（④）。カーネルのロードは、kexec ツールと呼ばれるカーネルロード用のユーザプログラムを用いる。ロードするカーネルファイルが ELF フォーマットである場合、カーネルファイルから各セクションと ELF ヘッダが抜き出され、カーネル内の kexec_info 構造体に格納される。kexec ツールは、kexec_info 構造体の情報を基にカーネルをロードする。格納の際に一時退避領域を追加セクションとして kexec_info 構造体に登録する。そして、一時退避領域の容量と位置を New カーネルのブートオプションに追加する。追加後のブートオプションは次のようになる。

```
root=/dev/hda1 4 irqpoll memmap=exactmap
memmap=640K@0K memmap=3624K@16384K
memmap=126807K@20649K
memmap=153600K@147456K
shelter=153600K@147456K
```

memmap=exactmap は BIOS から提供されるメモリマップに設定を追加するためのオプションである。memmap=exactmap の後の memmap オプションでパッチ適用後のカーネルが利用できるメモリ領域を指定している。shelter オプションは、一時退避領域の容量と位置を表している。アットマークの前が一時退避領域の容量、後ろが位置を表す。

次に、kexec を実行する（⑤）。kexec は ALT+SysRq+c の 3 つのキーを同時に押すか、英小文字の「c」を/proc/sysrq-trigger ファイルに書き込むことによって実行される。ALT+SysRq+c を利用する場合は、カーネルのコンパイル時に SysRq キーを有効にしておく必要がある。kexec を実行する前に、チェックポイントイメージファイルがファイルキャッシュのまま、Old カーネルの中に残らないように、sync コマンドを発行する。

次に、領域確保機能は New カーネルのブート時に一時退避領域を再確保する（⑥）。ブートオプションとして指定されている shelter オプションから一時退避領域の容量と位置を確認する。そして、reserve_bootmem カーネル関数を使ってメモリ領域を予約する。これでカーネルのブート処理が終了し、次のステップからは RC スクリプトの実行に移る（⑦）。RC スクリプトの中には、RAM ディスクの認識とチェックポイントファイルからのプロセスの復元コマンドが記述されている。

次に、領域確保機能は一時退避領域を RAM ディスクとして認識させる（⑧）。これは基本的に手順②と同じである。しかし、一時退避領域アクセス用カーネルモジュールによって RAM ディスクデバイスを作成したら、ユーザプロセスによる RAM ディスクのフォーマットは行わずマウントする。RAM ディスクのフォーマットを必要としない理由は、kexec 実行前の時点で同領域がすでにフォーマットされているからである。

最後に、レジューム機能はチェックポイントイメージからプロセスを再開させる（⑨）。

5.2 プロセスイメージのゼロコピー機能の実装

プロセスイメージのゼロコピー機能は、再起動前のプロセスが利用していたメモリ領域をプロセス復元の際に再利用するための機能である。カーネルチェックポインティングを用いた高速再起動方式では、プロセスが利用していたメモリ領域を一時退避領域へコピーしていたが、本機能の実現によって、一時退避領域へのコピーはプロセスの管理情報のみとなる。プロセスイメージのゼロコピー機能は、プロセスが利用していたメモリ領域の保存機能と認識機能、プロセスの復元機能から構成される。

メモリ領域の保存機能は、チェックポイントイメージ作成の際に（図 4 の手順③）実行される。保存機能は２つの部分機能から構成される。１つ目はプロセスが利用していたメモリ領域のコピーを行う代わりに、プロセスが利用していたページフレームへのポインタをチェックポイントイメージの一部として記憶しておく機能である。2 つ目は、プロセスが利用していたメモリ領域を、カーネルや他のプロセスに利用させずに保存しておく機能である。2 つ目の機能は、当該プロセスのエントリをカーネルのランキューから取り除き、タスクの状態を「TASK_UNINTERRUPTIBLE」にしたまま、kexec を実行するまで停止させておくことで実現した。

メモリ領域の認識機能は、プロセスが利用していたメモリ領域を、kexec 実行後のカーネル（パッチ適用済みカーネル）に認識させる機能である。メモリ領域の認識機能は、パッチ適用後のカーネルをブートする際（図 4 の手順⑥）に実行される。

プロセスの復元機能は、チェックポイントイメージからプロセスをレジュームする際（図 4 の手順⑧）に実行される。レジューム機能によってプロセスのメモリマップを復元する際に、プロセスの復元機能は、保存しておいたページフレームのページフレーム番号をページテーブルに登録していく。プロセスの復元機能を実現するために、do_anonymous_page カーネル関数と同じ処理を実行する関数をカーネルレベルチェックポインティングモジュールに実装した。do_anonymous_page 関数は、ページフォルト発生の際に、オンデマンドページングを行うための関数である。do_anonymous_page 関数内の__alloc_page 関数の代わりに再起動前に利用していたページをページテーブルエントリに割り当てる。

6. 評価

我々が提案する高速再起動方式の有効性を検証するために、既存の手法と我々の提案方式の再起動時間を比較した。結果、本方式により、既存の手法と比較して、システム再起動時間を最大で約 48％削減することができた。

6.1 評価環境と評価手順

評価環境は表 1 と基本的には同じである。評価対象をゲスト OS の再起動時間とした。ゲスト OS 上では、10～100MB までのメモリを取得し、そこにランダム数を書き込む評価用ユーザプロセスを用意した。これは、プロセスのメモリ使用量に比例してチェックポインティング時間が増大することを確認するためである。また、本評価では、カーネルチェックポインティングの機能が不足しているため、デーモンプロセスのチェックポインティングは行わず、syslogd, irqbalance デーモン等、カーネル内の処理に密接に関係する最小限のデーモンが稼働するランレベルを評価用に用意した。デーモン数は 10 である。シャットダウンコマンドを実行する際には、シャットダウンコマンドが実行される前に、評価用ユーザプロセスのチェックポインティングが行われるようにラッパースクリプトを作成した。評価手順は下記の通りである。これらはシェルスクリプトによって自動的に実行される。

1. 評価用ユーザプロセスを実行する
2. シャットダウンコマンドを実行する
3. 評価用プログラムのチェックポインティングが行われる
4. デーモンのシャットダウン処理が実行される
5. カーネルとデーモンが起動され、ルートユーザのオートログインが実行される
6. 評価用ユーザプロセスのレジュームが実行される

評価時間は、手順 1 が実行される前と、手順 6 が実行された後の時刻を取得し、それらの差分をシステムの再起動時間とした。

6.2 評価結果

プロセスの状態継続を実現する下記の４方式について、評価用プロセスの使用するメモリ量に比例して、システムの再起動時間がどのように変化するか評価した。

1. 通常の再起動＋チェックポインティング＋ディスク使用
2. kexec＋チェックポインティング＋ディスク使用
3. 高速再起動方式（RAM ディスク版）
4. 高速再起動方式（メモリゼロコピー版）

評価結果を図 6 に示す。以下では、１、２、３の方式と４を比較した結果である。

１と４の方式を比較した場合、通常の再起動+チェックポインティング+ディスク使用の場合の再起動時間は約 98 秒。そして、高速再起動方式（メモリゼロコピー版）の再起動時間は約 51 秒であった。よって、既存手法と比較して、システム再起動時間を最大で約 48%削減できたことになる。

２と４の方式を比較した場合、kexec+チェックポインティング+ディスク使用の場合の再起動時間は約 80 秒。そして、高速再起動方式（メモリゼロコピー版）の再起動時間は約 51 秒であった。よ

って、この場合は、システム再起動時間を約36%削減できたことになる。この場合は、kexecを利用することによる再起動時間削減効果が大きい。また、従来技術を組み合わせて実現できる最良の方式といえる。

3と4の方式を比較した場合、高速再起動方式（RAMディスク版）使用の場合の再起動時間は約52秒。そして、高速再起動方式（メモリゼロコピー版）の再起動時間は約51秒であった。よって、この場合は、システム再起動時間を約2%削減できたことになる。100MB程度のメモリを利用するプロセスを実行するシステムの再起動については、評価結果の誤差を数秒程度と考慮すると、メモリのゼロコピーに対する効果がほとんど現れなかったということになる。

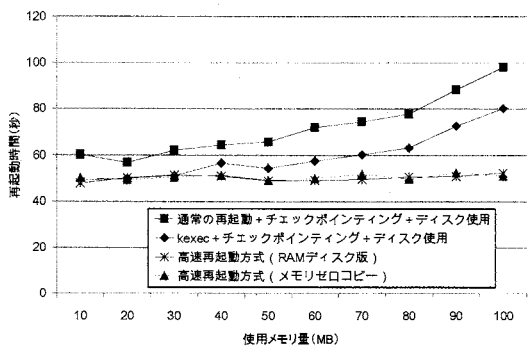

図 6 評価結果

7. 考察

本方式により、既存の手法と比較して、システム再起動時間を最大で約48%削減することができたが、目標としていた77%削減は達成できていない。これには2つの原因があると考えている。

一つ目は、デーモンのチェックポインティングを取ることができなかったためである。我々の使用したカーネルチェックポインティングツールはネットワークリソースや、パイプ、共有メモリ等のソフトウェアリソースを利用するプロセスのチェックポインティングを実行することができない。よって、評価の際に残したデーモンの再起動時間を短縮することができなかった。

二つ目は、評価用プロセスが使用しているメモリ量が少ないためである。本方式の効果は、サービスプロセスの利用するメモリ量が大きいほど効果が高い。評価結果を基に計算すると、約400MBを使用するサービスプロセスが存在すると、システム再起動時間の削減率は77%に達する。

8. 関連研究

システムの停止時間を短縮するという目的において、システムを高速に再起動させるということと、ソフトウェアの動的更新技術は同じ目的を持つといえる。本節では、8.1においてシステムの起動・再起動を高速化する手法を紹介し、8.2において、ソフトウェアの動的更新技術手法を紹介する。

8.1 起動、または再起動の高速化

serial[13]は、SMPマシンにおいて、システムの起動時にデーモンの起動を複数のCPUで同時に実行するための機能をserialデーモンと呼ばれるスーパーデーモンにて実現する。serialはシステムの起動時間を最大で38%削減できるとしている。

8.2 ソフトウェアの動的更新技術

AYA[1]は、Portalと呼ばれる状態監視機能をもつOSである。AYAは状態監視機能によって、プログラムの実行状態を監視する。アップデートは、アップデート対象となるプログラムが実行されていない時点に行われる。対象プログラムが実行されていないことはPortalによって保障される。さらに、AYAはアダプテーションシステム[2]と呼ばれる機能を備えている。アダプテーションシステムは、関数のインターフェイスの整合を取るための機能である。例えば、関数の引数が変更された場合、当該関数を利用するすべてのプログラムを変更する必要がある。アダプテーションシステムは、このような場合に、引数の整合性を維持することができる。具体的には、新旧関数のインターフェイスに関する情報（引数の数、型、意味）を登録しておくことによって、呼び出し側が旧式の呼び出し方法を利用していたとしても、アダプテーションシステムが新方式の呼び出し方法に変換して、新しい関数を呼び出す。

同じように、DynAMOS[3]は、状態監視のための機能をカーネル内に持ち、データ構造の変換を動的に行う。そして、新しいコードと古いコードを平行に実行し、これらのコードの切り替えは「トランポリンコード」と呼ばれるリダイレクションコードを変更対象となる関数の入り口と出口に挿入することによって切り替える。

AYAもDynAMOSも、状態の監視、データ変換、関数のリダイレクションによって、オンラインバージョンアップを実現しているが、これらの手法は、ユーザプログラム実行のオーバーヘッドを高くするという欠点を持つ。

IBMの開発したオブジェクト指向型OSであるK42は、オブジェクトのホットスワップ機構[7]を持つ。K42のホットスワップ機構は、デザインパターンのうちの一つであるファクトリパターンを用いて、オブジェクトのホットスワップを実現している。ファクトリパターンとは、デザインパターンの一つである。オブジェクトとオブジェクトの結びつきを小さくすることで、プログラムの再利用性を高めるためのデザインパターンである。K42は、ファクトリパターンを用いて、データ型やインターフェイスに関してオブジェクト間の依存性を無くすことによって、AYAやDynAMOSと比較して、動的更新可能なソフトウェアの範囲を広げている。

9. 今後の課題

9.1 パッチ適用により、プロセスの継続実行が困難な場合に対する対応

カーネルパッチの内容がプロセスの実行に影響を及ぼすものは、我々の手法を利用することができない。すなわち、システムコールのインターフェイスが変更される場合や、カーネル内のデータ構造が変更される場合である。このような場合にはソフトウェアの動的更新技術との組み合わせが考えられる。

9.2 更なる再起動高速化の余地

本方式は、プロセスの再起動処理を省略し、カーネルのみを再起動するシステムの高速再起動方式である。カーネルの再起動処理には、ハードウェアの初期化も含まれる。しかし、ハードウェアは再起動前に初期化済みであり、これらを再度初期化するのは、ハードウェア自身が実行状態を保持しない限り無意味と言える。ハードウェアの初期化は、再起動前の情報を用いることによって、初期化処理を省略することが可能であると考えられる。

10. まとめ

本稿では、システムの再起動を行う際に、プロセスの再起動が最も時間のかかる処理であることに着目し、カーネルのみを再起動するシステムの高速再起動方式を開発した。同方式は、メモリ上のプロセスデータは保持したままカーネルのみを入れ替え、その後、メモリ上に保持されているプロセスデータを用いて、再起動前に実行されていたプロセスを途中状態から再開することができる。本方式により、従来のシステム再起動時間を最大で約 48%削減することができた。

参考文献

[1] 小林良岳，佐藤友隆，唐野雅樹、結城理憲、前川守: 彩：コンパイル時に自動生成される Portal を基に動的再構成可能なオペレーティングシステム，電子情報通信学会論文誌. D-I 情報・システム I-情報処理 Vol.J84-D-1, pp. 605-616(2001)

[2] 古久澤学，小林良岳，中山健，前川守: 動的更新におけるプログラム部品間の差分を吸収するアダプテーションシステム, The 8th JSSST SIGSYS Workshop on Systems for Programming and Applications (SPA2005), pp.93-100 (2005).

[3] Kristis Makiris. and Kyung Dong Ryu.: Dynamic and Adaptive Updates of Non-Quiescent Subsystems in Commodity Operating System Kernels, Proceedings of the 2nd ACM SIGOPS/Eurosys European Conference on Computer Systems 2007, pp.327-340 (2007)

[4] Andy Pfiffer: Reducing System Reboot Time With kexec, OSDL whitepaper, July 2005, http://devresources.linux-foundation.org/andyp/kexec/whitepaper/kexec.pdf

[5] Vivek Goyal, Eric W. Biederman, Hariprasad Nellitheertha: Kdump, A Kexec-based Kernel Crash Dumping Mechanism, OLS2005,July 2005, http://lse.sourceforge.net/kdump/documentation/ols2oo5-kdump-paper.pdf

[6] Tim R.Bird: Methods to Improve Bootup Time in Linux, Proceedings of the Linux Symposium, OLS2004, July 2004

[7] Andrew Baumann, Gernot Heiser, Jonathan Appavoo, Dilma Da Silva, Orran Krieger, Robert W. Wisniewski, Jeremy Kerr: Providing Dynamic Update in an Operating System, In Proceedings of the annual conference on USENIX Annual Technical Conference, pp279-291,2005

[8] Segal M E, Frieder O: On-the-fly program modification: systems for dynamic updating, Software IEEE Volume: 10 Issue: 2, pp53-65, Mar 1993

[9] Roberto Gioiosa, Jose Carlos Sanchosong Jiang, Febrizio Petrini, Kei Davis: Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers, IEEE/ACM Supercomputing 2005, Nov 2005.

[10] Jason Duell: The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart, Berkeley Lab Technical Report (publication LBNL-54941), http://ftg.lbl.gov/CheckpointRestart/blcr.pdf

[11] Dejan S.Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, Songnian Zhou: Process Migration, ACM Computing Surveys (CSUR), Volume 32, Issue 3, pp.241-299, Sep 2000.

[12] Xen source project page, Located at http://www.citrixxenserver.com/Pages/default.aspx

[13] serel – fast boot software project page, Located at http://www.fastboot.org/