B_031

アプリケーションの継続利用を考慮したリブート高速化手法
A quick reboot mechanism with continuously using application programs.

小比賀　亮仁† 高橋　雅彦† 菅原　智義†
Akihito Kohiga Masahiko Takahashi Tomoyoshi Sugawara

はじめに

OS へパッチを適用した場合や原因不明な障害への対処方法として OS のリブートが一般的に行われている。この際、実行中のアプリケーションはすべて終了され、アプリケーションが使用していたメモリイメージはすべて破棄されてしまう。また、リブート中はユーザの作業が数分程度中断されてしまう。

そこで我々は上記の問題点を解決するために、OS 起動後、RAM 上のアプリケーションデータをそのまま再利用するリブート高速化手法を提案する。

提案手法は、リブートを実行する際に BIOS を経由せず、リブート前の OS（1st OS）がリブート後の OS(2nd OS)を直接起動する。そして、アプリケーションの実行状態のうち、OS 内のプロセス管理情報を退避させるための予約領域を RAM 上に設け、この領域を経由してプロセス管理情報を引き継ぐことによって、リブート前に実行していたアプリケーションを復元する。提案手法は、リブート前にアプリケーションが使用していたメモリイメージを再利用することによって、高速なリブート手法を実現し、ユーザの作業中断時間を短縮することが可能となる。

従来のリブート高速化手法における問題点

バージョンアップやセキュリティパッチの適用など、稼働中の OS に変更を加える場合、特に、Linux カーネルなどのような OS の最も基本的な部分に変更を加える場合は、OS のリブートが必要となる。リブートはユーザに作業の中断や作業内容の保存・復元という負担を強いることになる。そこで、中断時間の影響を最小限に留めるためには、リブートの高速化と作業状態の復元、すなわち、実行中のアプリケーションデータの退避・復元を高速化する必要がある。

OS のリブートと作業状態の復元において最も時間のかかる処理は、デーモン、ウィンドウシステム、ブラウザなど、各種アプリケーションの起動と作業状態の復元である。よって、これらの処理を高速化することで、リブート時間を大幅に短縮し、リブートによる影響を最小限に留めることができる。

高速化や継続利用のための従来手法として、チェックポインティングおよび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では、アプリケーションの実行状態（カーネル内のプロセス管理情報およびメモリイメージ）を保存し、次回実行時にこれを復元することで継続利用を行う。ただし、OS のリブート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安定になるので、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ことになる。

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 OS やアプリケーションの起動処理を省略できるため、リブートと作業状態の復元を高速化することができる。しかし、OS 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 OS のメモリイメージを直接書き換える必要があり困難である。

アプリケーションの継続利用を考慮したリブート高速化手法

そこで我々は上記の問題点を解決するために、OS 起動後、RAM 上のアプリケーションデータをそのまま再利用するリブート高速化手法を提案する。我々の手法の特徴は以下の 2 点である。①BIOS を経由せず、1st OS から 2nd OS を直接、ブートさせることで RAM 内のデータを残す。②リブート時に RAM 内のデータを利用して、アプリケーションの実行状態を復元する。①により、ハードの初期化時間を短縮でき、②によりアプリケーションの初期化時間と作業状態の復旧時間を短縮できる。チェックポインティングでは、OS のリブート時に、メモリ初期化の影響を避けるため、カーネル内のプロセス管理情報に加えてプロセスのメモリイメージをハードディスクに保存する必要があった。本手法では、RAM 以外の記憶装置を使用しないので、高速なリブートが可能である。

以下に、図 1を用いて本方式の概要を説明する。図中の矢印は物理メモリ中におけるデータの遷移を表している。1st OS はリブート前に起動していた OS である。AP1 使用領域、AP2 使用領域は、それぞれアプリケーションが使用していたメモリ領域、2nd OS はリブート後に起動する OS である。プロセス状態退避領域は、1st OS が保持していたアプリケーションの管理情報（プロセス構造体やメモリ管理情報など）を一時的に退避させておくために 1st OS によって予約された固定領域である。以下にて図中の動作順序について説明する。

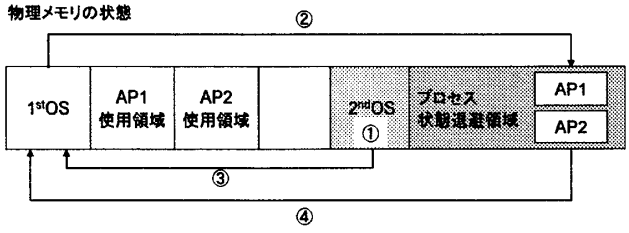

図 1 提案方式の概要説明図

† 日本電気株式会社 システムプラットフォーム研究所
† NEC Laboratories America, Inc.

① セキュリティパッチが適用された新たな OS（2nd OS）を予め、物理メモリの任意の位置にロードしておく。
② リブートを開始する際には、OS が管理しているプロセス状態をプロセス状態退避領域に退避する。
③ 1st OS の使用領域に 2nd OS を上書きし、2nd OS の起動を始める。
④ 2nd OS 起動の処理では、AP 使用領域はクリアせずに、退避したプロセス状態を OS 使用領域に書き戻す。

③の動作は、通常の OS のリブート処理とは異なる。通常のリブートでは、シャットダウン処理が終了した後にハードウェアリセットによりメモリ状態が不定になる。シャットダウン処理によりアプリケーションが終了されてしまうと、アプリケーションの継続利用が困難となる。また、ハードウェアリセットが実行されると、メモリに残っている情報が消されてしまう恐れがある。よって、1st OS のシャットダウン処理は、2nd OS の上書きと、2nd OS のブートシーケンスを呼び出す処理のみである。

④の処理では、まずプロセス状態退避領域を認識し、プロセス状態退避領域に退避されているメモリ管理情報を使って、アプリケーションが使用しているメモリ領域を認識した上でメモリの初期化を行う必要がある。メモリの初期化後、プロセス構造体や付随するデータを使って、リブート前に実行していたアプリケーションを復元していく。

以上が本方式によるリブート高速化手法である。本方式は、ディスク入出力を伴わないことと、アプリケーションの復元が、OS の管理するプロセス情報の復元だけである、すなわち、アプリケーションが使用していたデータをメモリ上に展開する必要が無いことから、チェックポインティングと比較し、さらにリブートを高速化することができる。

提案方式の実装

提案手法を実現するための第一段階として、Linux カーネルに改良を施し、RAM ディスクとチェックポインティングライブラリを用いた実装を行った。本実装では、チェックポインティングを使用するので、アプリケーションの使用しているメモリデータをコピーする必要があるが、RAM ディスクを使用するのでハードディスクへの入出力が無い。よって、本方式は、通常のチェックポインティングを用いたリブートよりも高速化される。

2nd OS の起動には、kexec[1] および kdump[2] を用いた。kexec は、シャットダウン処理を省略し BIOS を経由せずに新しいカーネルを起動する仕組みである。kdump は、リブート前のカーネルとアプリケーションのメモリダンプを取ることのできる機能である。kdump では、1stOS が使用していた領域は、2nd OS が起動したあともそのままの状態で残されるので、アプリケーションデータを再利用することができる。そして、チェックポインティングには、ユーザレベルのチェックポイントツール、libckpt[3] を使用した。

本実装では、Linux kernel 2.6.15 を改造し、kdump によって 2nd OS がロードされる領域のすぐ後ろのメモリ領域をプロセス情報退避領域として予約した。予約した領域にアクセスするために、専用のカーネルモジュールを作成し、カーネルモジュールには、プロセス情報退避領域を RAM ディスクとして認識させる。RAM ディスクを認識させた後の動作は以下の通りである。

① RAM ディスクにファイルシステムを作成する。
② 各プロセスのチェックポインティングを始める。
③ チェックポイントイメージを退避領域に退避する。
④ 2nd OS を起動する。
⑤ RAM ディスクを再マウントする。
⑥ 退避領域に格納されているチェックポイントイメージを用いてアプリケーションを復元する。

評価

表 1の評価環境を用いて、実装したリブート高速化手法を評価した。なお、アプリケーションも含めた通常のリブート時間は 83 秒である。

表 1 評価環境

OS	Linux kernel 2.6.15 (fedoracore4)
CPU	Pentium M 1.0GHz
メモリ	768MB

本評価では、kexec とチェックポインティングを用いたリブートと、我々の提案する方式における実装を比較した。kexec とチェックポインティングを用いたリブートとは、リブート前にチェックポイントイメージをハードディスクに保存し、kexec にて OS を起動後、アプリケーションを再開させる手法である。ユーザアプリケーションに相当するプログラムとして、30MB のメモリにランダム数を書き込むプロセスを 10 個程度生成するプログラムを使用した。

表 2 評価結果

kexec＋チェックポインティング手法	51 秒
提案手法	46 秒

表 2に、リブートに要した時間を示す。表 2の両方式の結果が通常と比較し、30 秒以上短縮されている理由は、両方式共に kexec を使用しているためである。また、kexec＋チェックポインティング方式と本方式を比較すると、リブート時間を 5 秒短縮していることがわかる。これは、先にも述べたとおり、ハードディスクへの入出力を伴わないため、リブートが高速化されているものと考えられる。

おわりに

本稿では、アプリケーションの継続利用を考慮したリブート高速化手法を提案し、提案方式の設計と、予備実装を行って評価した。評価の結果、予備実装は kexec とチェックポインティングを用いたリブート高速化と比較し、リブート時間を 5 秒短縮することができることを示した。今後は、チェックポインティングとは異なり、アプリケーションの使用しているメモリデータの退避・復元を行わず、プロセスの管理情報のみをリブート後に引き継ぐ手法を実装することにより、本提案の実現を目指す。

参考文献

[1] A. Pfiffer. Reducing System Reboot Time With kexec. Open Source Development Labs, http://www.osdl.org/docs/reducing_system_reboot_time_with_kexec.pdf, April 2003.
[2] V.Goyal, E.W.Biederman, H.Nellitheertha. Kdump, A Kexec-based Kernel Crash Dumping Mechanism. OLS, http://lse.sourceforge.net/kdump/documentation/ols2oo5-kdump-paper.pdf, 2005.
[3] J.S.Plank, M.Beck, G.Kingsley, and Kai Li. Libckpt: Transparent Checkpointing under Unix. Technical Report CS-94-242, University of Tennessee, August 1994.