# SMPを活用したOSのロギング&リプレイ

川﨑　仁†　　追川　修一†
†筑波大学 コンピュータサイエンス専攻

## 1　はじめに

オペレーティングシステム（OS）の開発時やシステムの運用時にカーネルパニックによる障害が発生することがある．こうした場合，一般的に障害時点から遡ってデバッグを行うことになるが，これは過去の状態を推定することであり容易ではない．もし過去の状態を再現できるなら，デバッグの負担を軽減することが可能である．

本研究では，ロギング&リプレイにより過去の状態を再現することを目的としている．そして，新たにSMP環境上でロギング用とリプレイ用の2つのOSを起動し，リプレイ用OSの実行を遅らせることで常時過去の状態を保持する手法を提案する．

本手法では，SMP環境上でPrimaryとBackupの2つのOSを実行する．Primary側でOSの実行履歴の取得と保存（ロギング）を行い，Backup側ではログの読み出しとOSの再現実行（リプレイ）を行うことで，Backup側に常にPrimary側の過去の状態を保持させる．障害時には，Backup側のOSを利用して障害が発生するまでの処理を何度でも再現することができ，再現性の低いタイミングに依存するバグのデバッグも可能になる．

本手法は，近年一般的になったSMP環境を活用し，特に共有メモリを利用することで実行履歴を転送するオーバヘッドを抑えることが可能である．また，Primary側のゲストOSと同時にBackup側のゲストOSも動作させておくことで，実行履歴の削減も期待できる．

本研究は仮想マシンモニタにより実現する．2つの仮想マシンをそれぞれSMPのプロセッサの1つに割り当て，それぞれの仮想マシンをPrimaryとBackupとに対応させる．対象とするアーキテクチャは一般的に広く利用されているIA-32アーキテクチャとし，対象とするOSはLinuxを想定する．提案手法の核となるロギング&リプレイ機構の設計と実装を行い，その評価を行った．

## 2　設計と実装

ロギング&リプレイ機構は，本研究室でスクラッチから開発したSesta仮想マシンモニタ[1]に実現した．実装にはIntel製プロセッサが提供するH/W仮想化支援機構Intel VT-xを利用している．SestaをSMP環境に対応させるための拡張も行ったが，本稿ではロギング&リプレイの設計と実装に絞って述べる．

### 2.1　実行履歴

実行履歴とは，BackupとPrimaryを同一状態に保つために必要な情報である[2)-4)]．「非同期的なイベント」と「結果が外部要因に影響を受ける命令（非決定的な命令）」に対して，取得する必要がある．本研究では，以下の5つの情報を実行履歴とした．

- イベントの種類
- イベントの発生した命令アドレス
- 分岐回数
- ECXの値
- 命令の実行結果

命令アドレスだけではイベントの発生したタイミングを確定させることができないため，分岐回数と併せて確定する．分岐回数はPMC（Performance Monitering Counter）を用いて取得することができる．また，REP命令でイベントが発生した場合，上記2つだけではタイミングを確定させることができないため，ECXの値と併せて確定する．プロトタイプの開発を行うにあたり，タイマおよびシリアルラインの2つのデバイスを実行履歴取得の対象として設計を行った．具体的には，タイマとシリアルラインからの割り込みが「非同期的なイベント」であり，シリアルラインに対するin命令が「非決定的な命令」である．

### 2.2　実行履歴の取得と転送

Primaryでは，ゲストOSの起動時に，PMCによる分岐回数のカウントを開始する．非同期的なイベントが発生した際には，分岐回数を読み出し，他の実行履歴と共に保存する．そして，PMCを初期化し，実行を再開する．非決定的な命令が実行された際には，命令の実行結果を他の実行履歴と共に保存し，実行を再開する．

実行履歴は，SMP環境であることを活用し，共有メモリを通して行う．また，アクセスする仕組みとしては，リングバッファを実装して実行履歴の転送に利用した．リングバッファの処理を高速化するために，個々のエントリは固定長として定義した．今回は，IA-32のキャシュラインサイズとの整合性を考慮し，1つのエントリは32 byteに決定し実装を行った．

### 2.3　実行履歴の再現

Backupにおいても，ゲストOSの起動時に，PMCによる分岐回数のカウントを開始する．また，共有メモリから実行履歴を読み出す．

非同期的なイベントの場合，Debug Registerを使用してイベントの発生した命令アドレスに対してBreakpointを設定する．そして，停止時に分岐回数を比較し，一致

Logging & Replay using a SMP environment
†Jin KAWASAKI　†Shuichi OIKAWA
†Department of Computer Science, Tsukuba University

表 1　ゲスト OS の起動時間

| | 起動時間 [s] |
|---|---|
| ロギング＆リプレイなしの Primary | 2.284 |
| ロギング＆リプレイありの Primary | 2.286 |
| ロギング＆リプレイありの Backup | 2.319 |

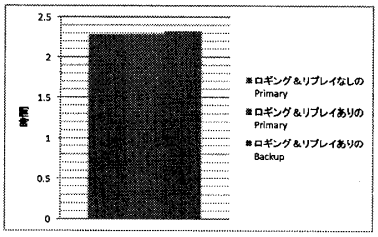


図 1　ゲスト OS の起動時間

表 2　実行履歴の内訳

| | 個数 | 割合 [%] |
|---|---|---|
| タイマ割込み | 207.5 | 7.998 |
| シリアル割込み | 2 | 0.077 |
| in 命令 | 2385 | 91.93 |

するまで繰り返す．分岐回数が一致するなら，さらに ECX と比較する．ECX が一致しないなら，REP 命令であると判断し，ステップ実行機能を利用して ECX が一致するまで実行する．ECX が一致するなら，そこでイベントをゲスト OS に通知して，実行の再現を行う．

非決定的な命令については，該当命令の実行時に実際に実行させることなく，実行履歴から得た結果をゲスト OS に返すことで再現する．

## 3　評価

現在，ゲスト OS をロギング＆リプレイを利用して起動可能である．評価として，ゲスト OS の起動完了までのロギング＆リプレイのオーバヘッドと実行履歴の量を測定した．測定は，Linux カーネルが命令アドレスの 0x100000 以降に展開された状態から，init スクリプトを実行する nash が sh を起動する直前までの間で行った．実験は，CPU：Xeon 5130 2.00GHz，メモリ：1GByte の環境で行い，ゲスト OS には Linux 2.6.23 を使用した．

### 3.1　ロギング＆リプレイのオーバヘッド

ロギング＆リプレイのオーバヘッドを測定するために，起動時間を比較した（表 1）．時間は，Time Stamp Counter（TSC）の値を元に算出した[*1]．この結果では，Primary のオーバヘッドは数ミリ秒と非常に小さく，システムに与える影響は小さいと考えられる．Backup の場合は数十ミリ秒とやや大きいが，2 %以内に収まっている．

### 3.2　実行履歴の総量と内訳

Primary が取得した実行履歴の総量と内訳を表 2 に示す．起動時間が約 2.3 秒，総量は約 81.08 KiB となっている．なお，値は複数回繰り返した結果の平均値である．

[*1] TSC はロギング&リプレイの対象としていない．

起動中と同程度の量のイベントが起動完了後も発生するとすれば，Backup 側を 1 分遅れて実行するためには 2 MiB 以上の領域が必要となる．しかし，1 つの実行履歴を 32 byte の固定長で保存しているため未使用の領域もあり，削減することは可能である．

また，シリアルラインを利用しているにも関わらず内訳を見るとシリアルラインの割り込みが非常に少ない．これはシリアルの初期化処理が起動シーケンスの終わり近くで行われているためだと考えられる．

## 4　関連研究

ロギング＆リプレイの研究として，ReVirt[3]，軽量仮想計算機モニタ[4]の研究がある．本研究は，ロギング＆リプレイの実現方法としてこれらの手法を参考にしているが，SMP 環境を活用してゲスト OS の実行と再現を同時に行う点と，実装方法の詳細が異なる．

Bressoud ら[2]は，仮想マシンモニタにより Primary/Backup モデルに基づいたゲスト OS のレプリケーションを実現した．この研究では，命令列を epoch と呼ばれる間隔に区切り，epoch の終了時点で非同期的なイベントを適用する手法が示されている．本研究も Primary/Backup モデルを採用したが，実行履歴を随時取得する点が異る．

## 5　まとめ

本稿では，カーネルデバッグの支援を目的として，ロギング＆リプレイを SMP を活用して実現する手法を提案した．これは，各プロセッサを Primary/Backup モデルの Primary と Backup に割り当てる方法であり，仮想マシンモニタにより実現する．Backup を Primary に対して時間差を設けて実行することで，Primary の過去の状態を Backup に作り出すことができる．現在，機能の一部であるロギング＆リプレイを実装している．今後はロギング＆リプレイ機能の実装を完了させ，本機能を利用した効率的なカーネルデバッグ環境の提供を目指す．

## 参考文献

[1] 青柳信吾：組込みシステムのための VMM による OS マイグレーションの実現, Master's thesis, 筑波大学システム情報工学研究科コンピュータサイエンス専攻 (2009).

[2] Bressoud, T. and Schneider, F.: Hypervisor-based fault tolerance, *ACM Transactions on Computer Systems (TOCS)*, Vol. 14, No. 1, pp. 80–107 (1996).

[3] Dunlap, G., King, S., S., C., Basrai, M. and Chen, P.: ReVirt: Enabling intrusion analysis through virtual-machine logging and replay, *ACM SIGOPS Operating Systems Review*, Vol. 36, pp. 211–224 (2002).

[4] 竹内理, 坂村健：軽量仮想計算機モニタを利用した OS デバッガのロギング＆リプレイ機能の提案, 情報処理学会論文誌, Vol. 50, No. 1, pp. 394–408 (2009).