「Debugging operating systems with time-traveling virtual machines」の要約

2014/7/3
藤田将輝

1 はじめに

VM を使用し，ロギングとリプレイにより，OS のデバッグを支援する環境の構成を理解するため，山本凌平の参考文献の 1 つである「Debugging operating systems with time-traveling virtual machines」[1] を読解している．まだ全て読解が完了していないため，3 章の 3 節までの要約を示す．

2 Abstract

OS のデバッグは難しい．非決定性，長時間の実行，ハードウェアとの相互作用，デバッグの操作が OS の状態を不安定にさせる，という理由があるためである．本稿では time-travel virtual machine という，これまでのデバッグの問題を解決する方法を提案する．タイムトラベルとは実行履歴や，過去の状態の再現を通して，プログラマが任意に OS の状態を過去に戻したり，先に進めたりすることである．タイムトラベルは一般的なデバッガにコマンドを実装することで組み込む．実際に time-travel virtual machine を使用し，OS のバグを解決していくことで評価する．従来のデバッグツールでは見つけにくい以下のバグを解決する．

(1) 非決定性により不安定なバグ
(2) デバイスドライバーのバグ
(3) 特定動作をさせるために長時間を必要とするバグ
(4) スタックを壊すバグ
(5) 関連性があるスタックフレームがあらわれたときに見つけられるバグ

3 Introduction

ソフトウェアのデバッグは通常，プログラマがバグの箇所を見つけるために，ソフトウェアを走行させ，問題が起こる点を見つけ，その時点の状態を調べる，ということを繰り返す．しかし，この方法では OS のデバッグを行うことは難しい．その理由に，OS の動作の非決定性，長時間の実行，OS とハードウェアデバイスが直接関係している，デバッグ自体が OS の状態を不安定にさせる，ということがある．これらの問題を解決するために，本稿では OS のデバッグの問題に対処するために，どのように time-travel virtual machine を使うかについて示す．VM を使うことで，デバッグ対象 OS の外側でデバッガを使用することができるため，デバッガがデバッグ対象 OS の動作を不安定にさせること

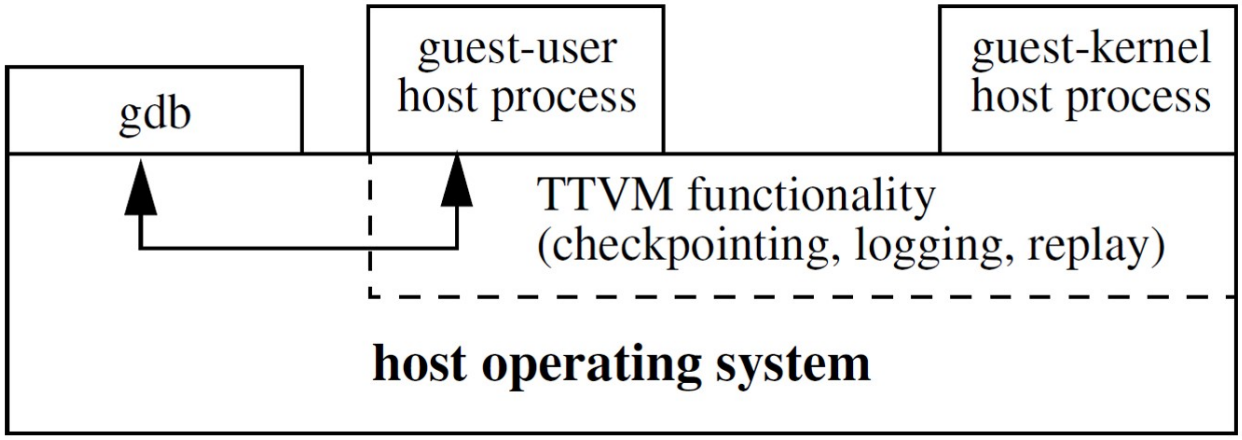

図 1　System structure

がない．ここでのタイムトラベルとは，実行履歴や，過去の状態の再現を通して，プログラマが任意に OS の状態を過去に戻したり，先に進めたりすることである．我々の方法では過去は不変なものとしている．これは実行履歴はただ 1 つのものだとするためである．従来の OS デバッグと違い，タイムトラベルを使うと，繰り返し実行しなくてもよいという利点がある．本稿では，time-travel virtual machine(TTVM) の実装を示す．我々はタイムトラベルを general-purpose debugger(gdb) に組み込み，reverse step(最後に実行された命令に戻る)，reverse breakpoint(命令が実行された最後の時点に戻る)，reverse watchpoint(変数が変更された最後の時点に戻る) の 3 つのコマンドを実装する．また，タイムトラベルを使用したデバッグの有用性を示す．

4　Virtual Machines

タイムトラベルを実装する VM は User-Mode Linux(UML) である．UML はホストデバイスドライバをサポートするように変更を加えられている．また，ホスト OS 上では以下の 2 つのユーザプロセスが走行している．

(1) ゲストカーネルのコードを走行させるプロセス (ゲストカーネルホストプロセス)
(2) ゲストユーザのコードを走行させるプロセス（ゲストユーザホストプロセス）

ゲストカーネルホストプロセスはゲストユーザホストプロセスで生成されたシステムコールや，信号を傍受する機能を用いている．ゲストユーザホストプロセスは別のユーザプロセスのアドレス空間に素早く切り替えるために VMM の拡張機能を使っている．本システムの概要図を図 1 に示し，以下で説明する．UML はホスト OS 上で，ゲストカーネルホストプロセスとゲストユーザホストプロセスを実行する．TTVM のプログラマが任意に OS の状態を過去に戻したり，先に進めたりする機能は変更を加えられたホスト OS に実装している．このタイムトラベル機能を使うために，gdb を拡張した．gdb はゲストユーザホストプロセスと遠隔シリアルプロトコルを通じてつながっている．

5 Time-traveling vitrual machines

5.1 TTVM-introduction

TTVM は 2 つの機能を持っている．1 つは，VM が起動してから最後の命令が実行された時までの任意の時点の完全な状態を再現できる機能である．2 つ目は，任意の時点から実行を開始し，オリジナルの実行と同じ命令列を実行する機能である．この章では，ロギングとリプレイ，チェックポインティングを通して，2 つの機能を実装する方法を示す．

5.2 Logging and replaying a VM

TTVM における基礎的な機能は，オリジナルの実行と一致するように，与えられた時点から実行を再生する機能である．リプレイは VM の状態とオリジナルの実行の状態を同じ状態にする．この機能を実現するために TTVM では ReVirt というロギング/リプレイの機能を使っている．VMM は非決定的な割り込みなどの操作を再現する．オリジナルの実行からデバイスを読む呼び出しをロギングすることと，リプレイ実行から同じデータを再現することで割り込み操作を再現できる．ロギングした点は，実行が始まってからの分岐回数と，割り込み命令のアドレスによって一意に決められる．

5.3 Host device drivers in the guest OS

一般的には，VMM は限られた仮想デバイスのセットを提供している．いくつかの VMM はハードウェアに存在するデバイスを提供しており，それ以外の VMM はハードウェアに存在しないものも提供している．限られたデバイスのセットを提供することはたいてい VM の利点とみなせる．なぜなら無数のホスト OS のデバイスドライバへ対応しなくてもよいためである．しかし，OS のデバッグの際にはこの限定されたセットはデバイスのデバッグやデバイスを使う際の障害となる．提供されたデバイスドライバしか使用できないためである．この問題を解決するためにソフトウェアエミュレータが対象のデバイスに存在しない場合でもゲスト OS で実ドライバを走行させる方法を実現する．

5.4 Checkpointing for faster time travel

VMM 起動時点からのロギングとリプレイは，実行中のほかの時点から実行中の任意の時点での状態を再作成するのに十分である．しかし，ロギングまたはリプレイの単独での実行はすぐにこの状態を再生するのに十分ではない．仮想マシンが必要なポイントに最初から各命令を実行する必要があり，この期間は何日にまたがる可能性があるためである．長期間にわたるタイムトラベルを素早く実行するため，TTVM は定期的にチェックポイントを取る．一番簡単な方法は VM の状態を完全にコピーすることである．完全なコピーを取ることは簡単であるが，効率的ではない．オーバヘッドを削減するために copy-on-write と versioning という機能を使う．前のチェックポイントから変更されたメモリページのみを保存している．前のチェックポイントの状態の復元はアンドゥログを用いて行う．チェックポイ

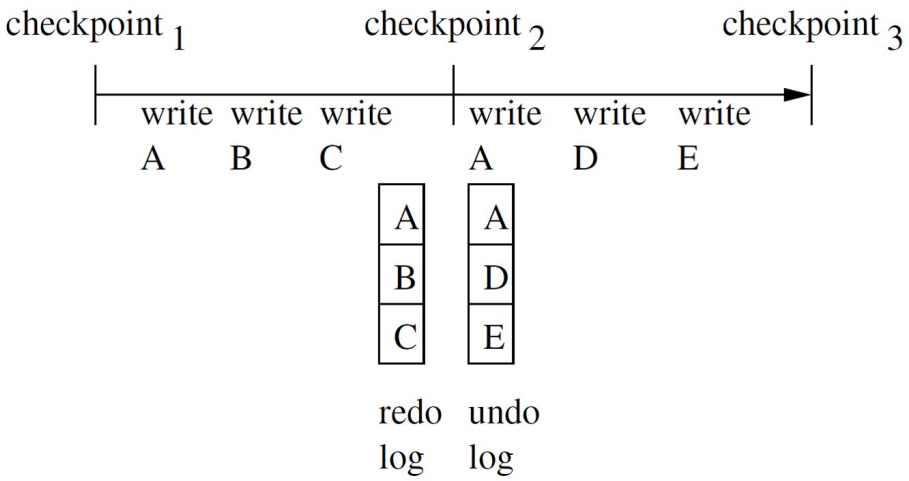

図 2　Checkpoint

ント n のアンドゥログにはチェックポイント n とチェックポイント n+1 での変更点のメモリセットが含まれている．また，TTVM には次のチェックポイントへ進むためのリドゥログも実装されている．チェックポイント n のリドゥログにはチェックポイント n-1 とチェックポイント n の変更点の情報を含んでいる．もし連続する 2 つのチェックポイントのインターバルでメモリマップが書き換えられたら，2 つのインターバルの間のチェックポイントのアンドゥログとリドゥログは同じ値を持つことがある．TTVM ではこの状態を見つけだし，アンドゥログとリドゥログでその値を共有させる．図 2 ではリドゥログの A とアンドゥログの A が同じ値を持っているため，この値を共有している．

6　おわりに

設計の途中までを読解した．従来の OS デバッグの問題点と，TTVM のタイムトラベル機能の概要を理解した．引き続き読解を進め，機能の詳しい部分の理解を深める．

参考文献

[1] Samuel, T.K., George, W.D. and Peter M.C.: Debugging operating systems with time- travelling virtual machines, Proceedings of The USENIX Annual Technical Conference, pp.1-15(2005).