「SMP を利用した Primary/Backup モデルによるリプレイ環境の構築」の要約

2014/6/18

藤田将輝

1 はじめに

Primary/Backup を用いた OS のデバッグ方法を理解するため，情報処理学会研究報告である「SMP を利用した Primary/Backup モデルによるリプレイ環境の構築」[1] を読解した．本資料ではこの論文の要約を示す．

2 目的

OS の開発時やシステム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．この原因を特定するには過去の状態を推定する必要がある．デバッグ時に，利用できる情報はカーネルパニック後のものに限られる．このため，原因の特定が容易ではない．特に，再現性の低いバグに起因する場合や，ダンプデータが破壊されている場合には原因の特定がさらに困難なものとなる．そこで，本研究では，仮想マシンモニタを応用し，カーネルパニック後の状態だけでなく，カーネルパニック前の状態も再現することで，効率的なデバッグ環境を提供する．本研究ではマルチプロセッサの SMP 環境を活用し，Primary/Backup モデルにより，2 つの OS を時間差を設けて実行する方法を提案する．なお，SMP とは Symmetric Multiprocessing(対称型マルチプロセッシング) のことである．2 つの OS をそれぞれ Primary と Backup に割り当て，それらを時間差を設けて実行することで，Primary のカーネルパニック時に Backup 側でカーネルパニックが起きる前の状態を再現できる．

3 システム概要

3.1 概要

本研究のシステム概要図を図 1 に示し，以下で説明する．SMP 上で Primary と Backup の 2 つの VM を実行する．Core0 を Primary に割り当て，Core1 を Backup に割り当てている．そして，Primary/Backup 間でロギングとリプレイを実現することにより，Backup で Primary の過去の状態を再現する．ここで，ロギングとは，OS の実行を通常に行い，ログの取得と保存を行うことである．また，リプレイとはログの読出しと OS の再現実行をすることである．ここでのログとは OS の動作を再現するために必要な情報である．本稿ではこのログを実行履歴と呼ぶ．デバッグ時には再現した状態を利用する．

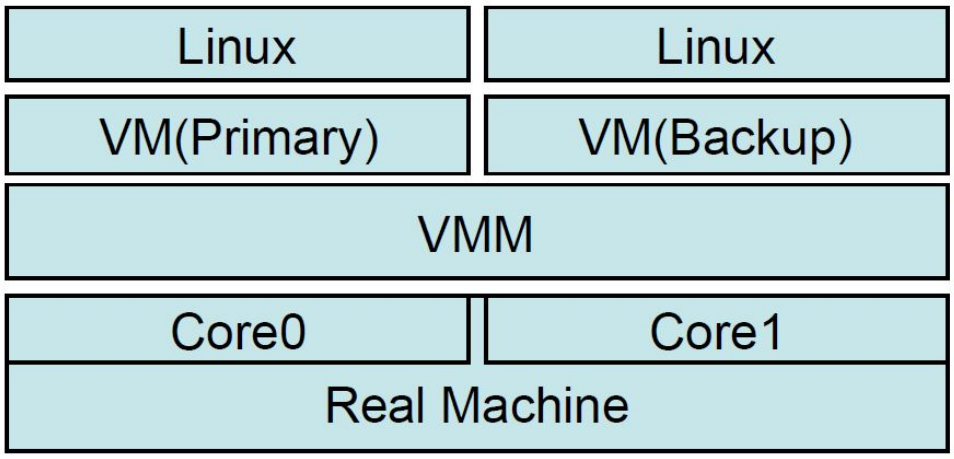

図 1　システム概要図

3.2　機能

本システムは，Primary でゲスト OS を起動し，Backup は Primary よりも遅れて実行を開始する．このとき，2 つの OS は完全に同一な挙動を示すようにする．その結果，Primary のカーネルが不具合を起こし，停止した時点において，Backup 側は不具合を発生する前の状態となる．デバッグではこの時点の Backup 側の OS を利用する．また，このリプレイ操作は何度でも繰り返して実行できる．

3.3　要求

3.2 で述べた機能を実現するためには，Primary と Backup 間で仮想マシンが同一の状態を保っていなければならない．2 つの仮想マシンを同一の状態に保つには以下の要件を満たす必要がある．

(1) イベントを同一のタイミングで処理し，同一の結果を得ること．
(2) 命令列を同一の順序で処理し，同一の結果を得ること．

2 つの仮想マシンを同一の状態に保つために，Primary から Backup に命令と割り込みのタイミング，実行結果を通知する．

3.4　実現方法

本研究では 3.3 の要求を踏まえて，仮想マシンモニタを利用する．Primary ではロギングを行い，Backup ではリプレイを行う．また，Primary と Backup 間で非同期的なイベントのタイミングと命令に関する情報を転送する必要がある．このため，Primary と Backup 間を共有メモリを用いて結合し，情報を Primary から Backup へ通知する．

4 ロギング&リプレイ

4.1 実行履歴の種類

Primary での動作を Backup で再現するためには，Primary と Backup 間で同一の状態を保つことが求められる．このためには，2 つの仮想マシンにおいて割り込みなどの非同期的なイベントを同一のタイミングで処理し同一の結果を得ること，また命令列を同一順序で処理し同一の結果を得ることが必要である．これらを実現するには，Primary から Backup に実行履歴を通知する必要がある．具体的には非同期的なイベントの種類と起きたタイミングとその内容，また，結果が外部要因に影響を受ける命令と，その結果を実行履歴をして転送することが求められる．実行履歴として，本研究では以下の情報のうちイベントごとに必要なものを実行履歴として保存する．

(1) イベントの種類
イベントの種類を識別するために必要である．具体的には，割り込みなのか命令なのか，割り込みであればどの種類の割り込みなのか，命令であればどの種類の命令なのか，といった点を識別する値を保存する．
(2) イベントの発生した命令アドレス
Backup に，Primary と同一の命令アドレスでイベントの通知や結果の適用を行うために必要である．
(3) 命令の実行結果
Primary が非決定的な命令を実行した際に得られた結果を保存し，Backup で同一の命令が実行されたときに保存された結果の値を返すために必要である．
(4) ECX
ECX をカウンタとして利用する命令を使用する際に，Primary と同一のタイミングで Backup にイベントを通知するために必要である．ECX とは x86 が提供するカウンタレジスタである．
(5) 分岐回数
命令のタイミングを特定するために必要である．分岐命令を経由すると同一アドレスを複数回通過することになり，命令アドレスだけでは同一のタイミングを特定することができない．そこで，分岐命令の回数を保存しておき，分岐命令の回数と命令アドレスにより同一のタイミングを特定する．

4.2 実行履歴の取得の流れ

実行履歴の取得の流れを以下に示す．

(1) Primary は，非同期的にイベントが発生した際に，これを契機としてゲスト OS から仮想マシンモニタに実行を遷移する．
(2) 仮想マシンモニタが実行履歴を作成し，共有メモリにこれを格納する．

(3) Backup の仮想マシンモニタでは，実行履歴を共有メモリから読み出し，ゲスト OS に適切に通知する．このとき，Primary と Backup とでイベント処理後の仮想マシン状態を同一のものにするため，イベントは Primary で起きたものと同一のタイミングで同一の内容を通知する．

4.3 実現方法

4.2 で述べた処理の流れをを実現する方法として，分岐回数をカウントすることによりタイミングを求める方法をとる．これには，IA-32 アーキテクチャの Performance Monitoring Counter(PMC) を利用する．PMC は様々なハードウェアイベントを計測するために利用できるカウンタ群であり，複数のハードウェアイベントを同時に計測することができる．本手法における実行の流れは以下のようになる．

(1) PMC を設定し，分岐回数のカウントを開始する．
(2) イベント発生時に分岐回数を含む実行履歴を取得し，共有メモリに格納する．
(3) (1) に戻って処理を継続する．

4.4 実行履歴の転送

Primary 側で実行履歴を取得し，Backup 側へと転送するために，前述のとおり共有メモリを利用する．このため，物理メモリ上に Primary と Backup の両方からアクセスできる物理アドレス空間を確保する．

4.5 実行履歴の再現

Backup では，共有メモリから実行履歴を読出し，ゲスト OS に通知することにより仮想マシンの状態を Primary と同一の状態に保つ．非同期的なイベントについては以下のように処理する．なお，以下の操作はすべて Backup が行う．

(1) 実行履歴を共有メモリから取得する．
(2) 実行履歴の命令アドレスをもとに，その命令アドレスまで実行する．
(3) 分岐回数と ECX を比較する．
　（A）一致すれば次に進む．
　（B）一致しなければ (2) に戻り繰り返す．
(4) ゲスト OS にイベントを通知する．
(5) (1) に戻って処理を継続する．

また，非決定的な命令については，当該命令を実行時に実際に命令を実行して結果を得るのではなく，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．

5 まとめ

本稿ではカーネルデバッグの支援方法として，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，SMP の各プロセッサを 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 で再現できる．

6 おわりに

本資料では「SMP を利用した Primary/Backup モデルによるリプレイ環境の構築」の要約を示した．Primary と Backup を同一の状態にするために，Primary で発生した非同期処理のタイミングや，非決定的な命令結果を共有メモリに保存し，Backup に通知する仕組みを理解した．山本凌平の特別研究報告にある，NIC ドライバの割り込み処理を対象としたデバッグ支援環境にも共有メモリが使用されていたため，共有メモリを使った情報の受け渡しが大変参考になった．

参考文献

[1] 川崎仁，追川修一：SMP を利用した Primary/Backup モデルによるリプレイ環境の構築，情報処理学会研究報告，Vol.2010-OS-113，No.12，pp.1-8(2010)．