

SMP を活用した Primary/Backup モデルによる カーネルデバッグ環境の提案

川 崎 仁^{†1} 追 川 修 一^{†1}

カーネルパニックが発生した際に、原因であるバグを短時間で特定する事は困難である。そこで、仮想マシンモニタを利用して OS の実行履歴を保存し、デバッグ時に OS 状態を再現する手法が提案されている。本稿では、SMP 環境を活用し、同一マシン上で実行履歴を用いて Primary/Backup モデルを実現するシステムを提案する。また、その基礎となる機能として、OS 状態の転送に必要である仮想マシン間の通信をリングバッファを用いて実装し、性能評価を行う。

Kernel Debugging System using SMP environment by Primary/Backup Model

JIN KAWASAKI^{†1} and SHUICHI OIKAWA^{†1}

When a kernel panic occurs, it is difficult to identify its responsible bug in a small amount of time. In consequence, there is a suggested method of saving the histories of previous runs and replaying the state on the OS utilizing its saved histories. This paper proposes the system of the Primary/Backup model by application of SMP environment. As a basis of the proposed system, implementing ring buffer for connections between virtual machines is required for transformation of the OS states. In addition, its performance evaluations are performed.

1. はじめに

ソフトウェアやハードウェアのバグを要因としてカーネルパニックが発生すると、そのバ

グを特定し修正するためにカーネルデバッグが行われる。しかし、パニックの原因を特定し修正するために利用できる情報は、クラッシュダンプデータなどパニック後のものに限られる。そうした限られた情報から、パニックに至る経路を特定する必要があるため、カーネルデバッグは困難を伴う作業である。特に、ダンプデータが破壊されている場合や、再現性の低いバグが発生する場合には、さらに困難なものとなり、デバッグ作業は経験や知識を積み上げた場合においても依然難しい作業となっている。特に、近年のカーネルの多機能化や複雑化は、バグ混入の確率を高めると共にデバッグをさらに困難にさせる方向にも影響を与えている。また、高度なデバイスが様々登場しており、それらのデバイスドライバ開発においても同様に、カーネルデバッグが困難なものとなっている。このように、カーネルデバッグの効率化の必要性はますます高まっている。

そこで、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はなくカーネルパニック前の状態も提供することで、効率的なデバッグ環境を提供しようという研究が行われている。既存の手法では、デバッグの際にチェックポイントとログを用いてロギング&リプレイを実現し、カーネルパニック前の OS の状態を再現している。本研究においては、現在一般的になっているマルチプロセッサの SMP 環境を活用し、さらに Primary/Backup モデル¹⁾を導入することで、2つの OS を時間差を設けて実行する方法を提案している。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 側ではカーネルパニックが起きる前の状況を作り出すことができる。そして、この状況から、デバッグを進めていくこととなる。

本稿においては、初めに提供するシステム全体の概要について述べる。次に、システムの中心となる仮想マシンモニタと実行履歴の扱いに関して設計と実装を示す。その後、提案手法の核となる機能であるプロセッサ間のリングバッファに関して性能測定を行う。そして、関連研究を示し、最後にまとめを述べる。

2. システム概要

本研究で提案するシステムの概要図を図 1 に示す。システムの実現には、仮想マシンモニタを利用し、2つの仮想マシンのそれぞれに SMP のコアを割り当てる。そして、それぞれの仮想マシンを Primary と Backup とに対応させることで、SMP を活用した Primary/Backup モデルを構築する。本稿では以後、デバッグに利用する側の OS を Backup、通常実行する側を Primary と呼ぶ。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 とし、対象とする OS は Linux を想定する。

^{†1} 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
Tsukuba University, Department of Computer Science

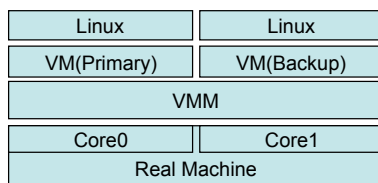


図 1 システムの概要図
Fig. 1 Figure of system.

本システムが実現する機能は次の通りである。まず、Primary でゲスト OS を起動し、Backup はそれから遅れて実行される。このとき、Primary と Backup は完全に同一の挙動を示すようにする。このようにすることで、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 側は不具合が発生する前の状態（つまり、この時点の Primary にとって、過去にあたる状態）となっている。そこで、この段階の Backup 側の OS を利用して順次デバッグを行っていくことによりカーネルパニックが発生するより前の情報を用いた効率的なデバッグを行うことができる。

デバッグでは命令をステップ実行により進めていくことでカーネルパニックが発生するまで処理を継続していく。このようにして、カーネルパニックが発生する前の情報を利用したカーネルデバッグの方法を提供することが可能となる。

しかし、以上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に保たなければならない。なぜなら、同一の状態でなければ、命令実行に対する結果がこととなることになり、正確なデバッグが不可能だからである。そのためには、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、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²⁾⁻⁴⁾。非同期的なイベントについては、それが発生したタイミングと内容を Primary から Backup へと通知し、Backup 側で非同期的なイベントを同一のタイミングと同一の内容で OS に通知することで対応可能である。一方、命令列の同一順序での処理は、命令列だけを考えれば決定的な処理であり、一度実行を開始すると同一の順序で実行される。しかし、前述の非同期的なイベントが命令列の実行中に発生するために、環境によって命令実行順が変わってしまう可能性がある。この問題も、前述の非同期的なイベントに対する処理で解決可能である。また、同一の結果という点に関しては、タイマ値の取得命令などは実行環境で結果が変わってしまう可能性があるため、命令の実行結果を Primary から Backup へと通知し、Backup 側が

その命令の実行時に同一の結果を返すことで対応することが求められる。

仮想マシン間で非同期的なイベントと命令に関する情報を転送することが必要となるため、本研究では Primary と Backup 間で共有メモリを用いて結合し、情報を Primary から Backup へと通知する方法を採用する。転送において、Primary と Backup 間で同期は取らないものとする。既存の手法においては、別マシン上で OS 状態を再現するためにネットワークを利用して転送を行う方法や、同一マシン上であっても同時に OS 状態を再現するのではなく後にログを元に再現を行う方法が採られてきた。ネットワークを利用する手法では、転送速度やエラー処理を検討し対応することが必要となるが、提案手法では SMP を活用し共有メモリを利用するため転送速度は高速でありエラー処理を検討する必要も無い。また、後に再現を行う手法では、再現できる長さはログの最大量に限定されてしまうが、本手法においては再現側の OS も同時に動作しているため再現を行う長さはログの最大量に影響を受けない。

以下の節では、まず本研究で提案するシステムの基礎となる仮想マシンモニタについて述べ、続いて Primary から Backup へのイベントの通知を実現する仕組みについて述べる。

3. システムの設計と実装

本研究では、仮想マシンモニタ Sesta⁵⁾ をベースとして、それを SMP 対応に拡張しコア間での Primary/Backup モデルを実現した仮想マシンモニタを開発する。Sesta は、筆者の所属する研究室においてスクラッチから実装が行われた仮想マシンモニタである。OS のマイグレーションを目的として開発され、ホスト OS を必要とせず直接実機上で動作する Type1 型の仮想マシンモニタである。また、CPU、メモリ、一部のデバイスドライバの仮想化を実現しており、仮想化の実現のためにハードウェア仮想化支援機構 Intel VT-x⁶⁾ を利用している。以下、Sesta の概略を述べ、続いて本研究のために加えた実装について述べる。

3.1 仮想マシンモニタ：Sesta

Sesta は、単一の仮想 CPU をゲスト OS に対して提供する仮想マシンモニタであり、1つの仮想マシンを提供する。また、仮想化にハードウェア仮想化支援機構 Intel VT-x を利用しており、これにより特権命令実行時のゲスト OS から仮想マシンモニタへの遷移やイベント発生時のハンドリングを比較的容易に実装することが可能である。これは、あらかじめ設定を行っておくことで、特権命令や例外により、もしくは明示的に指定することにより、自動的にゲストから仮想マシンモニタの環境へと遷移し、仮想マシンモニタ側で処理を行うことができるためである。Intel VT-x の、MMU の仮想化に対するハードウェアによる支

援はまだ一部のプロセッサに限られているため、特権命令と例外による仮想マシンモニタへの遷移と仮想メモリを利用した仕組みによりメモリの仮想化を実現し、ゲスト OS と仮想マシンモニタのアドレス空間の切り分けを実現している。

仮想メモリ機構では、CPU にアドレス変換表であるページテーブルを登録し、メモリアクセスの際にそのページテーブルをたどることで物理アドレスを求める。さらに、Intel VT-x で、仮想マシンモニタが利用するページテーブルとゲスト OS が使用するページテーブル（シャドウページテーブル）の 2 つを準備し、仮想マシンモニタとゲスト OS の環境を遷移する際にそれらも切り替えることが可能である。これらの仕組みを用いて、ゲスト OS と仮想マシンモニタ間のアドレス空間の切り分けを次のように実現している。

まず、ゲスト OS がページテーブルを変更する命令を実行すると、Intel VT-x の機能により仮想マシンモニタに遷移が起きる。仮想マシンモニタ側では、ゲスト OS が CPU に登録しようとしたページテーブルを走査し、シャドウページテーブルに設定する。そして、ゲスト OS に遷移する際に、このシャドウページテーブルを登録することで、仮想マシンモニタによるゲスト OS のメモリアクセスを管理を実現している。Sesta ではゲスト OS の物理アドレス空間を定められたサイズに制限しており、それらの空間を仮想メモリ空間と 1 対 1 で対応づけている。また、仮想マシンモニタをゲスト OS の物理アドレス空間より上位の領域に配置することで、仮想マシンモニタとゲスト OS の切り分けを実現している。

また、Sesta は、デバイスドライバについても仮想化を実現しており、一部のデバイスについてはデバイス状態を含め仮想化している。しかし、その他のデバイスについては単に I/O 命令を直接デバイスに通知するのみとなっている。

3.2 Sesta の拡張

本研究のシステムでは、2 つのプロセッサのそれぞれで OS を実行しなければならないが、ベースとする Sesta はマルチプロセッサをサポートしていないため機能の追加が必要である。また、2 つの OS を同一の物理メモリ空間上で実行するために、アドレス空間の仮想化もそれらに対応させなければならない。これら 2 点とデバイスドライバに関連する点に関する拡張について、以下の節で順に述べる。

3.2.1 複数コアの利用

まず、IA-32 では、始めに立ち上がるプロセッサを BSP と呼び、それ以外のプロセッサを AP と呼んでいる。BSP は電源投入時に自動的に立ち上がるが、AP は自動的に立ち上がりず BSP から起こす必要がある。

また、レガシーな IA-32 アーキテクチャでは、割り込みコントローラとして i8259a が利

用されてきた。アーキテクチャのマルチプロセッサ化により、割り込みコントローラもマルチプロセッサに対応したものが、Advanced Programmable Interrupt Controller (APIC) である。APIC には、外部からのイベントを始めに受け取る I/O APIC と、各プロセッサごとにイベントを受け取る LOCAL APIC とがある。イベント伝達の流れとしては、外部からのイベントはまず I/O APIC に伝わり、事前に設定しておいた内容に従い、各プロセッサの LOCAL APIC に伝わる。そして、LOCAL APIC が各プロセッサにイベントを通知することになる。電源投入時には、互換性維持のためにレガシーな i8259a が用いられるようになっているため、APIC を利用するための準備を行い、それを有効にしなければならない。

Sesta では、各仮想マシンで提供する仮想 CPU は 1 つであり、本研究の拡張においてもこの点是不変である。そのため、実機においては割り込みコントローラとして APIC を利用するが、ゲスト OS に提供する仮想マシンでは仮想 i8259a を提供することとした。また、Backup では Primary の実行結果を元に動作を再現するため外部イベントを直接受け取り処理する必要が無い。そこで、I/O APIC からは Primary の LOCAL APIC のみに外部イベントを通知する。Primary から Backup への外部イベントの情報転送については、後述する。

3.2.2 メモリ管理

本研究で提案するシステムでは、同一の物理メモリ上にゲスト OS のための物理アドレス空間を複数持つため、別のゲスト OS の物理アドレス空間を書き換えてしまわないように、それらを保護しなければならない。加えて、Primary の動作を再現するために、Backup 用の物理アドレス空間上でゲスト OS が動作したとしても、Primary の物理アドレス空間上で動作しているものと変わらない透過的なアクセスを実現しなければならない。これらの課題を、以下のように、前述のシャドウページングの仕組みを応用することで実現する。

まず、物理アドレス空間上の下位 OFFSET 分を Primary に、続く OFFSET 分を Backup に割り当てる。したがって、Primary の物理アドレスと対応する Backup の物理アドレスは、Primary のアドレスに OFFSET を加えたものとなるのが分かる。そこで、Backup 側においては、シャドウページングの設定の際に、登録されているアドレスに OFFSET を加えたものを変換後の物理アドレスとして登録する。また、Backup 側でゲスト OS の登録しようとするページテーブルを走査し辿っていく際にも、OFFSET を加えたアドレスを用いて辿っていく。このようにすることで、Backup 側のゲスト OS は自身が Primary と異なる物理アドレス空間で動作していることを認識すること無く動作することができる。Backup におけるメモリ管理の概要図を図 2 に示す。

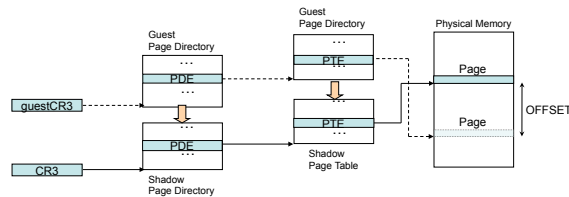


図 2 Backup におけるメモリ管理の概要図
Fig.2 Figure of managing memory on Backup.

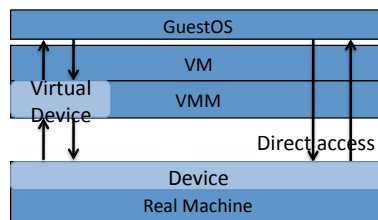


図 3 デバイスドライバの処理
Fig.3 Processing of device drivers.

3.2.3 デバイスドライバ

本研究においては、Backup で Primary のゲスト OS の動作を再現するために、Primary でのゲスト OS のデバイスアクセスとその結果を Backup に転送し再現する必要がある。再現する際に、Backup のゲスト OS がデバイスドライバにアクセスしようとする場合には、ゲスト OS から仮想マシンモニタに遷移し、実デバイスにアクセスするのではなく Primary から転送されたデバイスアクセスの結果をゲスト OS に通知することになる (図 3)。これを実現するためには、デバイスの仮想化が必要であり、本研究では Sesta の仮想デバイスの仕組みを活用することにする。従って、本システムで利用できるデバイスは、Sesta が仮想デバイスを提供しているデバイスとなる。

耐故障性を念頭に置いて Primary/Backup モデルを実現する場合には、Primary と Backup のそれぞれが実デバイスにアクセスする必要がある。本研究で提案するシステムにおいて Primary と Backup の OS は、前述の通り仮想デバイスにはアクセスするが実デバイスにはアクセスする必要が無い。

3.2.4 プロセッサ間通信

ここまで述べてきたように、Primary と Backup の OS を同一の状態を保って実行するには、Primary から Backup への情報転送が必要となる。そこで、Sesta を拡張しプロセッサ間通信を実現する。プロセッサ間通信の実現には、SMP の特徴を活かし共有メモリを用いることにした。共有メモリの詳細に関しては、4 で述べる。

4. 実行履歴

本節では、Primary の動作を再現するために Backup 側に転送される情報と、情報の転送方法について述べる。

4.1 実行履歴の取得

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。そのためには、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²⁾⁻⁴⁾。そして、これらを実現するためには、Primary から Backup に対して必要な実行履歴を送る必要がある。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、その内容
 - 結果が外部要因に影響を受ける命令 (例: I/O ポートの read) と、その結果
- といった情報を実行履歴として転送することが求められる。

本稿では、プロトタイプの開発を行うため最小限の構成として、タイマおよびシリアル of 2 つを実行履歴取得の対象として設計を行う。このうちシリアルは、ゲスト OS の操作とその実行結果を表示するために、コンソールの入出力として利用する。

通常のシステムにおいては、最小限の構成としては上記に加えてディスクデバイスが存在する。本研究においては、initramfs を利用することでディスクデバイスの利用することなくゲスト OS を実行している。initramfs は、linux カーネルの起動時に利用される小さなルートファイルシステムのことであり、起動時に物理メモリ空間上に展開される。本来はその後の起動シーケンスにおいて、ディスクデバイスのファイルシステムがマウントされることになるが、initramfs をそのまま利用する。ゲスト OS に対して十分大きな物理メモリ空間を割り当てることで、実験環境として実用上の問題は無い。UNIX コマンドの利用は、Busybox を用いることで実行ファイルについても削減している。

Primary 側では、非同期的なイベントが発生した際に、それを契機として仮想マシンモニタが実行履歴を作成し、共有メモリにそれを格納する。Backup 側では、それを共有メモ

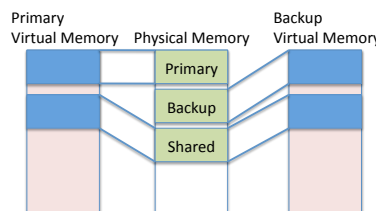


図 4 共有メモリを利用した転送
Fig. 4 Transfer using shared memory.

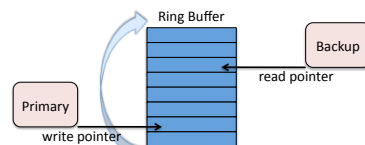


図 5 リングバッファの概要図
Fig. 5 Figure of the ring buffer.

りから読み出し、ゲスト OS 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 で起きたのと同じタイミングで同一の内容を通知する。これを実現する方法としては、イベントを決められたタイミングでまとめて適応する方法¹⁾や命令数をカウントすることによりタイミングを求める方法が知られている^{2),3)}。

本研究においては、後者の手法を採用し、IA-32 アーキテクチャの Performance Monitoring Counter (PMC) を利用し分岐命令数をカウントすることによりこれを実現する。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。これらのうち、分岐命令数をカウントすることで、イベント発生のタイミングを確定させることにする。結果が外部要因に影響を受ける命令についても同様にイベントとして扱う。イベントの内容として、命令種別とその結果を保存する。本手法における実行の流れは以下に示すようになる。

- (1) PMC を設定し、分岐命令数のカウントを開始する
- (2) イベント発生時に分岐命令数を取得し、発生したイベントの内容と分岐命令数、現在の命令実行アドレスを共有メモリに格納する
- (3) 1 に戻って処理を継続する

Backup 側でのイベントの再現については、4.4 で取り上げる。

4.2 実行履歴の転送

Primary 側で実行履歴を取得すると、それを Backup 側へと転送することになるが、前述の通り共有メモリを利用してこれを実現する。そのため、物理メモリ上に Primary と Backup の両方からアクセスできる同一の物理アドレス空間を確保する。これにより、SMP である特徴を活かし、同一マシン上のメモリを共有した高速な情報転送が可能となる。共有

メモリ領域を確保した場合のメモリアーキテクチャを図 4 に示す。

4.3 リングバッファ

実行履歴の転送は共有メモリを通して行うが、共有メモリへアクセスする仕組みとして、リングバッファを実装する(図 5)。リングバッファとは、バッファ領域をリング状に見立て、切れ目の無いアクセスを可能にした仕組みである。通信の分野で広く用いられており、仮想マシンモニタ Xen においては仮想マシン間の通信にも採用されている⁷⁾。

リングバッファの処理を高速化するために、個々のエントリを固定長として定義する。したがって、実行履歴もその種類とは無関係に固定長で保存を行うこととし、開いた領域はパディングをする。また、Backup の最大遅延時間はリングバッファ全体のサイズに影響を受けるため、遅延時間を長くするためにはリングバッファ全体のサイズを大きくすることになる。一方、Primary と Backup は時間差はあるものの同時にゲスト OS を実行するため、実行を開始してからカーネルパニックが起きるまでの時間がリングバッファ全体のサイズに影響することは無い。

注意すべき点として、Primary と Backup とで非同期にゲスト OS を実行することを想定しているため、リングバッファにおいて一方が他方に追いついてしまう可能性がある。特に、Backup が Primary に時間的に過多に接近した場合、時間的に遅れさせて実行させることによりパニック前の状況を再現するという本来の目的を達成し得なくなるかもしれない。これを避けるために、Backup 側が一定の時間間隔を維持して実行できるように調整することも必要となる。

4.4 実行履歴の反映

Backup では、共有メモリから実行履歴を取得すると、それをゲスト OS に通知することにより仮想マシン状態を Primary のものと同一の状態に保つ。まず、共有メモリから取得した実行履歴中の分岐命令数を利用し、分岐命令数とアドレスを元に割り込みを設定する。その際に、割り込みの設定場所は該当アドレスより十分前にしておき、その地点から該当アドレスまでステップ実行により進める。該当アドレスまで進むと、そこでイベントをゲスト OS に通知する。本手法における実行履歴反映の流れをまとめると、以下に示すようになる。

- (1) 実行履歴を共有メモリから取得する
- (2) 分岐命令数と命令実行番地を元に、該当アドレス付近まで実行する
- (3) 該当アドレスまでステップ実行を行い、ゲスト OS に実行履歴を通知する
- (4) 1 に戻って処理を継続する

表 1 実験環境

Table 1 Experimental environment.

項目名	内容
実機	Dell Precision 490
CPU	Xeon 5130 2.00GHz
Memory	1GByte
入出力	シリアル× 2, ディスプレイ
ゲスト OS	Linux 2.6.23

表 2 スループットの測定結果 (MB/s)

Table 2 Throughput evaluation results(MB/s).

	4 byte	8 byte	16 byte	32 byte
write(Primary)	406.30	646.97	812.23	1202.38
read(Backup)	439.57	688.81	913.37	1216.66

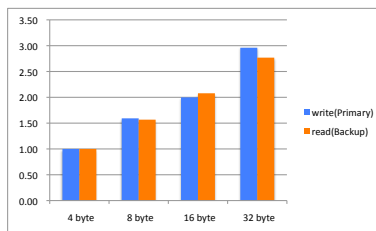


図 6 スループットの測定結果

Fig.6 Throughput evaluation results.

5. 実験と考察

本研究で提案する手法を実現するために必要なリングバッファについて実装を行い、その性能を測定した。

性能測定に用いた実験環境を表 1 に示す。ゲスト OS への入出力は、実験環境の実機をシリアルケーブルによりホストマシンに接続して行った。また、仮想マシンモニタの実行を観測するためにディスプレイを接続し、その上にクロック数などの情報を表示した。

まず、リングバッファのエントリサイズの違いによる実行時間の変化を調べた。Primary 側では、リングバッファへの 1000 回の書き込みにかかるクロック数を測定した。同様に、Backup 側ではリングバッファからの 1000 回の読み込みをについて測定した。これを、リ

ングバッファのエントリサイズが 4 byte, 8 byte, 16 byte, 32 byte のものについて、それぞれ 10 回実施しその平均値を結果とした。なお、リングバッファのエントリ数は全て 4096 個である。結果を、表 2 に示す。

書き込みと読み込みのスループットを比較すると、読み込みが書き込みに比べて高いスループットを示していることが分かる。そのため、読み書きだけが続けていけば、Backup が Primary に追いついてしまう。また、今後実行履歴の再現処理を含めた計測を行った場合、再現のためのコストがにより逆に Primary が Backup に追いつく可能性もある。Backup が Primary に接近することは、Primary のパニック時にパニック前の状況を作り出すという目標を達成し得なくなるため認められない。Primary が Backup に接近する場合この問題は無いが、対応を検討する必要がある。

表 2 の結果について、エントリサイズが 4 byte ときの結果を 1 として正規化したものをグラフにし、図 6 に示す。左から順に、4, 8, 16, 32 byte とならんでおり、各組の左側が Primary によるリングバッファへの書き込み、右側が Backup によるリングバッファからの読み込みである。このグラフから、エントリサイズを大きくすればより高いスループットを得られることが分かる。一方で、サイズが大きくなれば読み書きに要する時間は伸びるというトレードオフもあるため、転送する必要のある情報はどの程度のサイズになるかを含めて検討が必要である。

また、今回実装した環境は L2 キャッシュを 2 つのコアが共有していた。L2 キャッシュを共有していないモデルにおいて、どのような影響が見られるかについても今後の課題とする。

6. 関連研究

OS の実行状態の再現に関しては、耐故障性の観点から研究を行った例も見られる。1) では、仮想マシンモニタを用いることで、Primary/Backup モデルに基づいた OS のレプリケーション実現可能であることが明らかにされた。この手法では、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応することによりレプリケーションを実現する手法を提案した。本研究においても Primary/Backup モデルを採用したが、命令列を区切ることなく Primary で発生したタイミングでイベントを適応する。

また、Remus⁸⁾ は、ネットワークで接続された 2 つのマシン間でスナップショットを転送することでレプリケーションを実現し、高可用性を実現する手法である。本研究においては、同一マシン上の SMP 環境を活用することで、高速化を目指している。

ReVirt^{2),3)} や軽量仮想計算機モニタ⁴⁾ は、デバッグを目的として仮想マシンモニタの活

用し、ゲスト OS のロギング&リプレイを実現している。本研究は、ロギング&リプレイの実現方法として、ReVirt の手法を採用しているが、OS の再現方法の点で上記 2 つの研究とは異なる。本研究においては、SMP 環境を活用し、ゲスト OS の実行と再現を同時に行う。

7. おわりに

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/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。Primary から Backup への実行履歴の転送には、共有メモリをリングバッファによりアクセスする機構を利用する。

また、本システムを実現する上で必要なプロセッサ間通信機能をリングバッファにより実装し、その性能測定を行った。

今後は、実行履歴の取得と再現に関して実装を進め、システム全体の評価を目指す。

参 考 文 献

- 1) Bressoud, T. and Schneider, F.: Hypervisor-based fault tolerance, *ACM Transactions on Computer Systems (TOCS)*, Vol.14, No.1, pp.80–107 (1996).
- 2) 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).
- 3) King, S., Dunlap, G. and P.M., C.: Debugging operating systems with time-traveling virtual machines, *In Proceedings of the 2005 Annual USENIX Technical Conference* (2005).
- 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).
- 5) 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, 修士論文, 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).
- 6) *Intel 64 and IA-32 Architectures Software Developer's Manual*.
- 7) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *ACM SIGOPS Operating Systems Review*, Vol.37, No.5, pp.164–177 (2003).
- 8) Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N. and Warfield, A.: Remus: High availability via asynchronous virtual machine replication, *In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)* (2008).