Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

カーネルトレース技術を用いた仮想化クラスタ技術の研究

金成昊[†]　大島訓[†]　新井利明[†]

仮想化技術を利用したクラスタ制御技術を開発した．従来のクラスタ管理ソフトウェアでは，ハートビットによるノード間監視・制御を行ったため，障害が発生し検知までに時間がかかり結果としてフェイルオーバーが遅くなる，障害の詳細情報が取得できない等の問題があった．本研究では，VMM を利用し，ＯＳ及びアプリの改造することなく，カーネルトレースモジュールを仮想マシンのＯＳに挿入することで障害検知の高速化と詳細な障害情報の取得を実現した．提案手法の実効性を示すため，Linux の標準仮想化技術の KVM 上で仮想化クラスタを構築し実測を行った結果，障害検知から系切替までの時間を５０％短縮した．

Clustering Virtual Machines with Kernel Trace Technology

Sungho Kim[†], Satoshi Oshima[†] and Toshiaki Arai[†]

We developed the method to cluster virtual machines, utilizing the virtualization technology. In conventional cluster-wares, virtual machines in cluster are monitored by periodic heartbeat signal. That results in fail-over latency. In this paper, the new method to monitor virtual machines using kernel trace technology fully utilizing the benefit of the virtualization technology is introduced. The method shows 50% improvement on fail-over latency in our evaluation environment.

1. はじめに

近年，CPU を始めてするハードウェアの高機能化によって低コストのＰＣサーバは，企業におけるファイル共有サーバ，Web サーバだけでなく，金融機関，社会インフラシステム等のミッションクリティカルな大規模システムへと適用範囲が拡大している．このような基幹業務向けシステムでは信頼性を加え可用性が重要であり，多くの場合，複数のサーバを用いてクラスタシステムを構成することで可用性の向上を図っている．

一方，サーバ台数の増加による管理負荷の増大やサーバシステムのハードウェア性能の著しい向上より仮想化技術が注目されている．仮想化技術の導入により，１つの物理ハードウェアの上で複数のＯＳを動かすことができ，これにより既存サーバシステムの集約や計算機リソースの効率的な利用が可能である．

以上により，仮想化環境におけるクラスタシステムの構築・運用技術は，効率的なサーバ資源の活用及びシステムの可用性向上の観点から，非常に重要である．また，仮想化環境でクラスタシステムを構成する場合，仮想化技術が提供している機能を利用することで，障害の検知等を高速に行なうことが可能である．

仮想化クラスタ技術は，Heartbeat[5]に代表されるオープンソースプロジェクトで開発されているクラスタソフトウェアが他商用ソフトウェアにより先行して開発されている．しかし，これらのソフトウェアは仮想化環境においても，一定間隔のメッセージ交換による仮想マシンの生死監視を行なっている．この方法では，サービスを提供している仮想マシンのノードから一定時間応答が無い場合，クラスタ管理ソフトウェアは該当ノードに障害が発生したと判断し，サービス提供ノードを切り替える．このような方法では，実際に障害が発生してから，クラスタ管理ソフトウェアが障害を検知するまでに時間がかかり結果としてフェイルオーバが遅くなる，障害が発生したノードの詳細な情報が取得できない等の問題がある．このように現存の仮想化クラスタ技術は，仮想化技術が提供している機能を十分活用していない．この問題を解決するため，我々は，仮想化技術を最大に活用できる仮想化クラスタ技術の研究に取り組んだ．

本論文では，その一歩として，仮想化環境のクラスタシステムにおける，クラスタノードの障害検知の高速化と詳細な障害情報の取得を実現するための障害検知機構の検討，開発を行なった．

以下，本論文では，2 章で既存クラスタ管理ソフトウェアでの仮想マシンの異常検知手法の概要とその問題点について述べる．次に，3 章では仮想化環境においての仮

[†] 株式会社日立製作所システム開発研究所
System Development Research Laboratory, Hitachi Ltd.

ⓒ2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

想マシンの障害検知の高速化機構について述べ，4 章では提案した障害検知機構の評価について述べる．最後に 5 章で本論文の内容を纏める．

2. 仮想マシンの異常検知手法の概要

仮想マシン異常には，仮想マシン上で動作するアプリケーション等のソフトウェア異常（以下，仮想マシン内部異常）と，仮想マシン上のＯＳを含むその下位からの異常（仮想マシン自体の異常）がある．現在，仮想化クラスタに対応しているクラスタ管理ソフトウェアの異常検知手法は，仮想化環境におけるクラスタシステムの構成方法に大きく依存している．その構成方法は大きく２つに分類できる．一つは，仮想マシン上でのＯＳにクラスタ管理ソフトウェアをインストールし，仮想マシン同士でクラスタ構成をする方法（図 1-(a)）と，もう一つは，仮想化環境を提供するホストマシン上でのＯＳにクラスタ管理ソフトウェアをインストールし，ホストマシン同士でのクラスタ構成をする方法（図 2-(b)）である．後者の場合，仮想化環境ではない，つまり物理サーバで構成されるクラスタ構成と同様なクラスタ構成である．各クラスタ構成方法での仮想マイン異常検知方法について詳細に述べる．

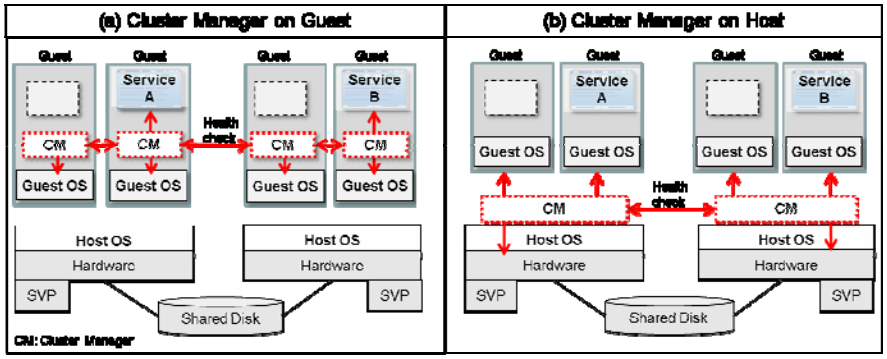

図１　クラスタ構成方法

2.1 仮想マシン同士でのクラスタ構成時の異常検知方法

2.1.1 概要

本構成の場合，仮想マシン内部状態の異常検知は通常の物理サーバのクラスタ構成時の検知方法と同様である．ただし，ハードウェア関連異常の場合は，仮想化環境を提供するホストＯＳでのエミュレーションが必要となる．本方式のメリットは仮想化内部の詳細異常が検知可能であることである．一方、問題点としては以下のようなものが挙げられる．

2.1.2 仮想マシン自体の異常検知の問題点

本構成では，クラスタ管理ソフトウェアが仮想マシン上のＯＳにインストールされているため，仮想マシン自体の異常検知はできない．そのため，仮想化クラスタ上の仮想マシンのノード間での定期的なメッセージ交換（ハートビット信号）による生死監視が必要となる．生死監視の周期によってはその異常検知の時間は相当大きいものとなる．異常検知の高速化のため，生死監視周期を短くする場合，ネットワークのオーバヘッドとしてシステム全体の性能に影響を及ぶこととなる．また，ハードウェア異常時の障害情報がクラスタ管理ソフトウェアで取得できない．

2.2 ホストマシン同士でのクラスタ構成時の異常検知方法

2.2.1 概要

本構成の場合，クラスタ管理ソフトウェアが動作しているホスト OS で，仮想マシンの内部異常及び仮想マシン自体異常の情報を取得しなければならない．前述した方式に比べ，クラスタ管理ソフトウェアがホストＯＳで動作しているため，ＯＳ下位レベルのハードウェア関連の異常検知は得意である．また，仮想マシンをサービス用ＩＰアドレスやサービスデーモンプロセスのように一つのクラスタリソースとして扱っているため，仮想マシンの管理が容易である．したがって，ＯＳから下位レベルの異常時に仮想マシンのライブマイグレーションがクラスタ管理ソフトウェアレベルで可能である．

2.2.2 仮想マシンの異常検知の問題点

本構成では，仮想マシンの管理上で優れている構成であるが，仮想マシンの異常検知に関しては以下のような問題がある．

（１）仮想マシン自体異常の検知

現に，本構成で運用可能な Heartbeat 等のクラスタ管理ソフトウェアでは，クラスタ管理ソフトウェアで定期的に仮想マシンの生死状態を監視することで，異常検知を行っている．そのため，仮想マシン同士でのクラスタ構成時の異常検知と同様な問題に抱えている．つまり，異常検知時間が生死監視周期に依存してしまう．

（２）仮想マシン内部異常の検知

本構成で運用可能な Heartbeat 等のクラスタ管理ソフトウェアでは，仮想マシン自体異常のみ管理しているため，仮想マシン内部異常検知ができない．つまり内部のＷＥＢデモンプロセス異常等の監視は不可能である．仮想マシン内部異常を検知するためには，仮想マシン上で動作する別途の監視ソフトウェアの運用が必要である．また，仮想マシン上の監視ソフトウェアと，ホスト OS 上のクラスタ管理ソフトウェアの連

ⓒ2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

携が必要となる．

3. 仮想マシンの異常検知高速化機構

従来の仮想化クラスタ上での仮想マシンの異常検知には，前章で述べたような問題がある．本論文では，ホストマシン同士でのクラスタ構成においての異常検知手法を提案する．基幹業務システムのクラスタ構成では，ハードウェア異常の処理が非常に重要であることと，大規模システムのため仮想マシンの管理が容易である必要があるため，ホストマシン同士でのクラスタ構成による仮想化クラスタ構築が大規模な基幹業務システムには適切であると判断したためである．前章で述べた問題点を踏まえ仮想マシンの異常検知高速化機構を提案する．

3.1 仮想マシンの異常検知高速化の検討

従来のメッセージ交換による生死監視は，ポーリング方式であるため障害検知に遅延が生じていた．これを解決する方法の一つとして，イベントドリブン方式の障害検知がある．これは，障害と判断するイベントをあらかじめ登録しておき，運用中に該当イベントが発生した場合，即座に障害と判断しフェイルオーバを実行する方式である．この方式では，イベントの登録と検出を行なう必要がある．以下，仮想マシン内部異常検知用及び仮想マシン自体異常検知用のイベント登録と検出について述べる．

3.1.1 仮想マシンの内部異常検知用のイベントの登録と検知

仮想マシンの内部異常検知用のイベントは，仮想マシンが提供している各サービスデモンプロセス（例えば，Web 用 apache デモン）の特性に合わせた異常発生実行パスを予測し，策定する必要がある．また，それを実行する際にクラスタ管理ソフトウェアに通知する手段も考える必要がある．簡単なイベントの例として，各デモンプロセスの異常終了時にＯＳから送られるシグナル（SIGTERM 等）が考えられる．そのシグナルハンドラを異常検知用イベントとして登録し，ハンドラ処理の延長でクラスタ管理ソフトウェアへの異常通知をすることで，イベントドリブン方式の障害検知が可能である．監視対象のプロセスからイベントで通知することで，最大にクラスタ管理ソフトウェアからのポーリング方式時の周期分の高速化が可能である．ただし，イベントの策定とイベント処理を各監視対象のプロセス毎に追加する必要がある問題がある．処理追加の場合，ソフトウェアのソースが公開されている場合には問題ないが，大概の商用ソフトウェアは公開されていないため，容易ではない．

3.1.2 仮想マシン自体の異常検知用のイベントの登録と検知

ＯＳから下位レベルの異常検知においては，アプリケーションレベルの仮想マシンの内部異常検知に比べて，イベント登録と検知は容易である．Linux をはじめとする UNIX 系ＯＳは無論，大概の商用ＯＳではＯＳから下位レベルのハードウェア異常の通知手段（割り込み等）が用意されている．ただし，ＯＳ上で動作しているクラスタ管理ソフトウェアとの連携するためには，ＯＳレベルでの連携機能を実装が必要とされる．通常，ＯＳとアプリケーションレイヤでの連携は仮想デバイスドライバを通じて行われる．Linux を例とすると，ＯＳまたは下位レイヤでのイベントをクラスタ管理ソフトウェアへ通知するために relayfs[8]や Netlink[9]といった，カーネル空間とユーザ空間の連携用の Linux のサブシステムが利用できる．連携用デバイスドライバでこれらを通じて下位レイヤのイベントをクラスタ管理ソフトウェアへ通知するのである．ただし，ＯＳからリソース枯渇や各プロセスの内部のデッドロック状態等の外部へ提供しない異常に対する検知パスが必要な場合、ＯＳの改造が必要となる。したがって、仮想マシンの内部異常と同様，ＯＳとクラスタ管理ソフトウェアでイベント処理追加が必要となる．

以上のように，仮想マシンの内部異常及び仮想マシン自体の異常検知用イベントの登録と検知には，アプリケーションやＯＳの別途処理追加等の改造が必要となる問題が存在する．本論文では，その問題を解決するため，アプリケーションやＯＳの変更が不要な汎用的なイベントドリブン方式を提案する．

3.2 仮想マシンの異常検知高速化機構の提案

前節の検討から，異常通知高速化のために必要なイベントドリブン方式を実現するための要件として，ソースレベルでのアプリケーションやＯＳの別途処理追加が不要であることがわかった．つまり，監視対象に関わらないイベント登録と検知が必要である．この問題を解決するには，稼働中の仮想マシンに対して，動的にイベントを挿入できる手法が必要である．この問題を解決するために本研究では，Linux の Kprobes[1][6]に代表される動的カーネルトレース機能（以下，動的プローブ）を適用した仮想マシンの異常検知高速化機構を提案する．このようなトレース技術を適用することで,アプリケーションやＯＳを変更せず,イベントの登録と検知が可能である．動的プローブ機能により，下記項目が可能になる．

- システム停止が不要な動的なプローブの挿入
- 任意のアドレスへのプローブの挿入
- プローブしたイベントが発生した際の迅速な検知

仮想化環境上で動作しているクラスタシステムにおいて，上記の特徴を持った動的プローブ機能を利用し，イベントドリブン方式の生死監視を行なうことで，フェイルオーバの高速化，効率化が可能である．

理解の助けのため、具体的に Linux の Kprobes の例を挙げて説明する．Linux では動

ⓒ2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

的プローブ機能として Kprobes というインフラが提供されている．Kprobes を含むカーネルモジュールを作成，ロードすることで，Linux カーネルに対し，動的にプローブを挿入することができる．カーネルがプローブされた命令を実行すると，あらかじめユーザが定義した関数が呼ばれるため，詳細な情報の取得も可能である．そのユーザの定義関数部分をイベント通知用に活用できる．つまり，監視対象によって監視箇所のカーネル実行パスを自由に変更でき，さらにカーネルレベルでの異常検知のため，監視対象のアプリケーションに寄らず共通な異常検知用イベントが登録可能である．ただし，動的プローブから取得した詳細情報をホストＯＳ上で動作しているクラスタ管理ソフトウェアに通知する手段が必要である．また，上述した Kprobes とする動的プローブ機能は，他マシンのＯＳへのプローブ挿入機能が無いため，Kprobes を利用した動的プローブ機能を本構成で利用する場合には，次の機能が必要となる．

- ホスト OS から仮想マシンの OS へのプローブ挿入機能
- 収集した情報のホスト OS への転送機能

本課題を解決するために，Xenprobes[2][3]が提案されている．Xenprobes は Kprobe の概念を仮想マシン間プローブに拡張したもので，ホスト OS から仮想マシンの OS にプローブを挿入することが可能である．しかし，Xenprobes は VMM が提供するデバッグ機能を必要であり，プローブ挿入時には仮想マシンの OS を停止しなければならない．また，プローブされた命令を実行すると，必ずホスト OS に処理が遷移するため，非常にオーバヘッドが大きく，サービスの品質を確保することが難しい．

そこで，本研究では，動的プローブの挿入及びプローブの収集情報を転送のため，VMM が提供している仮想マシン間通信機能を用いてこととした．仮想マシン間通信機能により，仮想マシンの OS とホスト OS ではメモリを共有できるため，実際にプローブの挿入や取得情報の転送が低オーバヘッドで可能となる．

3.3 仮想マシンの異常検知高速化機構の設計

以下の 3 点を仕様設計の方針とした．

- 異なる VMM 環境への対応が容易なこと
- クラスタ管理ソフトウェアはホスト OS で動作すること
- 仮想マシンの OS のユーザ空間の障害の影響を受けにくいこと

これに基づき，機能要件を次のように定義した．

1. ホスト OS および仮想マシンの OS のカーネルの変更が不要であること．カーネルモジュールとして実装することで，移植性が向上すると共に，システムへの機能の追加，削除が容易に可能になる．
2. 特定の VMM に依存した構造にしないこと．仮想マシン間通信部等 VMM に依存してしまう部分はあるが，最小限にすることで，移植性を高める．
3. ホスト OS のみが仮想マシンの OS へプローブモジュールを転送できること（プローブを挿入できること）．前章で述べたように，仮想化環境においてはホスト OS 上でクラスタ管理ソフトウェアが動作する．そのため，プローブの挿入や削除といった作業もホスト OS で一括管理するべきである．
4. 仮想増しの OS で収集した情報は即座にホスト OS へ転送すること．収集した情報を即座にホスト OS へ渡すことで，ホスト OS 上のクラスタ管理ソフトウェアへの迅速な障害発生の通知が可能になる．
5. 仮想マシンの OS ではプローブモジュールをロードするときにカーネル空間だけ利用すること．ホスト OS から転送されたモジュールを仮想マシンの OS がロードする際に，ユーザ空間を使用しないことで，ユーザ空間のみに障害が生じている場合でも，障害の調査が可能となる．

上記 1，2 により移植性を確保し，3，4 によりホスト OS 上のクラスタ管理ソフトウェアへ対応する．また，5 により仮想マシンの OS のユーザ空間の障害の影響を低減できる． 1 を満たすために仮想デバイスドライバとして実装する．2 の実現するために，階層構造を持たせ，VMM に依存する部分を最小限にする．3，4 の要件を満たすためにはホスト OS と仮想マシンの OS の間で，何らかの通信をする必要があるが，Linux といったオープンソースでは，virtio[7]という準標準のインタフェース定義されている．virtio では，フロントエンドドライバとバックエンドドライバからなる分割ドライバ方式の仮想マシン間通信インフラが用意されている．また，5 では，本来仮想マシンの OS のユーザ空間で実施する作業を，ホスト OS のユーザ空間で実施する．階層構造や実装の詳細については次章以降で説明を行なう．

4. 実装と評価

4.1 仮想マシンの異常検知高速化機構の実装

本論文では，Linux をホストＯＳとし，仮想マシンのＯＳも同様に Linux とし，提案手法の実装を行った．また，VMM は Linux 標準仮想化基盤の KVM[4]とした．仮想マシンお OS の情報を収集するために，プローブの挿入には Kprobes を利用し，プローブハンドラにおける情報の記録及びクラスタ管理ソフトウェアとの連携には relayfs を利用する．しかし，Kprobes は前述したように，ローカルシステムへのプローブ挿

ⓒ2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

入機能であるため，ホスト OS から仮想マシンの OS といった，外部の OS に対してプローブを直接挿入することはできない．また，Kprobes を利用してプローブを挿入する場合，カーネルモジュールとして実装するのが普通である．そのため，Kprobes を利用して仮想マシンの OS にプローブを挿入するためには, Kprobes を利用したプローブモジュールをホスト OS から仮想マシンの OS にロードする必要がある．このモジュールロード機能を Probe Loader とし，分割ドライバ方式で実装する．また，プローブモジュールは，プローブ時の情報を記録するために relayfs を利用する．プローブモジュールは仮想マシンの OS 内で動作しているため, 記録した情報を仮想マシンの OS からホスト OS へ転送する必要がある．この記録したデータを転送する機能を Probe Listener とし，Probe Loader と同様に分割ドライバ方式で実装する（図 3）．上述したように，本機構は 2 組の分割ドライバから構成されるが，これらのドライバは下層にある VMM に強く依存しているため, VMM 毎に実装する必要がある. そのため, VMM 依存部分を最小限にするため，UI 層，Action 層，Communication 層からなる 3 層構造で設計した（図 4）．これにより，本機構は VMM に依存する Communication 層を入れ替えるだけで，各種 VMM 上で動作することが可能になる．ここで，UI 層はクラスタ管理ソフトウェア等のアプリケーションとのインタフェースとなる層である．Action 層は UI 層，Communication 層からの要求を処理する中心的な層である．具体的には，仮想マシンの OS へのモジュールのロードやアンロード，仮想マシンとホスト OS 間での情報共有の実質的な処理を行なう．

4.2 評価

本節では KVM 上の仮想マシンで Web サービスを提供するクラスタシステムを構成し，本論文の提案手法の評価を行なう．

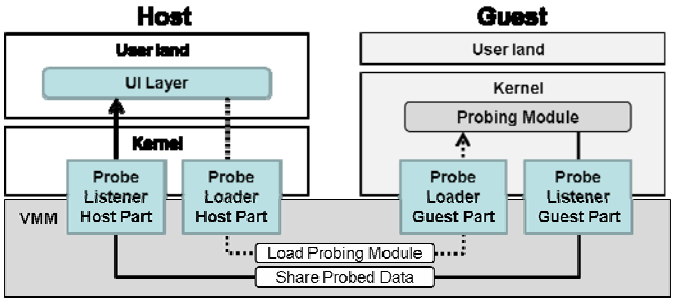

図 2　提案手法の構造

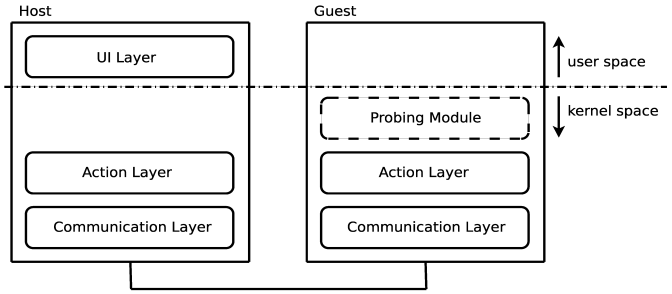

図 3　提案手法の階層構造

4.2.1 評価環境

評価環境としては，物理サーバを 2 台用意し，各サーバで仮想マシンを 2 つ起動する．物理サーバ 1 で動作している仮想マシンを VM1-1，VM1-2 とし，物理サーバ 2 で動作している仮想マシンを VM2-1，VM2-2 としたとき，VM1-1 と VM2-1 でクラスタ 1 を構成し，VM1-2 と VM2-2 でクラスタ 2 を構成する（図 5）．クラスタ管理ソフトウェアとしては Linux-HA が提供している Heartbeat を利用する．ここで，クラスタ 1 は通常の Heartbeat より 10 秒のポーリング周期で監視し，クラスタ 2 は Heartbeat を拡張し本論文の提案手法で監視を行なう．クラスタ 1 では Heartbeat で監視するが，監視方法としては，仮想マシンを一つの資源として監視する．そのため，内部の apache デモンプロセスの監視は対応していないため行なわない．監視対象の仮想マシンに異常が発生した場合には，Heartbeat によってサービス提供ノードの切り替えを行なう．

4.2.1 提案手法の仮想マシンの異常監視方法

本論文の提案手法を用いて仮想マシンのＯＳ情報をホストＯＳで取得し，正常／異常終了の監視を行った．今回は異常判断のイベントとしてシグナルと OOM killer の監視を用いた．シグナルの監視では仮想マシン内のカーネル関数の send_signal をプローブし，apache デモンプロセスに送信されたシグナルより，仮想マシンの内部障害と判断した．一方，OOM killer の監視では仮想マシン内のカーネル関数の out_of_memory をプローブし，本関数が呼び出される場合には仮想マシン自体障害と判断した．

ⓒ2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

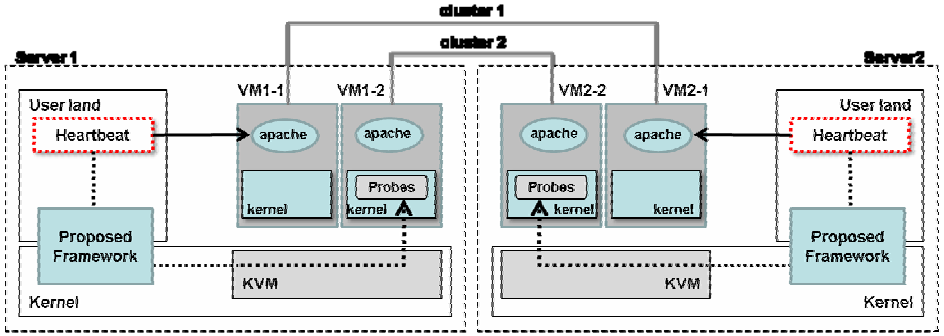

図４　評価環境

4.2.2 結果と考察

（１）apache デモンプロセスの障害

apache デモンプロセスの障害では，何らかの原因で apache デモンプロセスにシグナルが送られ，apache デモンプロセスが終了してしまう障害を想定した．実際にシグナルが送信されてから，Heartbeat が障害を検知するまでの時間を表 1 に示す．

表１　apache デモンプロセス障害検出時間[100 回の平均]

監視方法	障害検出までの時間[sec]
Heartbeat	5.4
Heartbeat＋提案手法	2.1

Heartbeat のみで監視した場合，障害検出までに 5.4 秒かかっている．この値は Heartbeat のポーリング周期に比例して前後する．通常運用では，ポーリング周期が実験時よりも長く設定されることが多いため，障害検出までの時間はより大きくなると考えられる．一方，提案手法を利用した場合には，約 2 秒で障害を検出でき，Heartbeat のみでの監視に比べ，障害検出時間が改善できていることが分かる．

（２）OOM 障害による仮想マシン自体の障害

仮想マシン自体の障害では，システムで利用するメモリが不足し，OOM killer が動作する場合を想定した．なお，本実験では OOM killer の発生を障害と判断している．これは OOM が動作している環境では，何らかのプロセスが強制終了されられており，正しいサービスを提供できない可能性があるためである．OOM killer が動作してから，Heartbeat が障害を検出するまでの時間を表 2 に示す．

表２　OOM killer 障害検出時間

監視方法	障害検出までの時間[sec]
Heartbeat	－
Heartbeat＋提案手法	2.0

表２からわかるように，Heartbeat のみによる監視では今回の障害を検知することができなかった．これは，OOM killer が多くの場合，実験用のメモリを大量に消費するプロセスを終了させており，apache デモンプロセスには影響を与えなかったためである．一方，提案手法を利用して監視した場合には，表 1 の場合とほぼ同程度の約 2 秒で障害を検出し，これまで検出不可能であった障害の検出も可能になったことが分かる．

（３）考察

上記で示した結果より提案手法の有効性を示した．ただし，提案手法は仮想マシンの障害情報を即座にホストＯＳに転送し，ホストＯＳでアクセス可能となっているため，理想的には障害検出時間は 0 に近い値となるはずである．ホストＯＳでは仮想マシンでの障害発生直後からアクセス可能なことから，上記結果で示した 2 秒のオーバヘッドは，提案手法と協調動作する Heartbeat の障害検出機能が，提案手法に最適化されていないためであると考えられる．今後，Heartbeat を最適化していくことで，障害検出時間をより短縮することが可能である．

一方で，提案手法での監視を行う際に，2 つの問題が生じた．1 つは，仮想マシンの監視場所の決定である．提案手法では，提供しているサービスに応じて監視場所を変更することが可能だが，逆に監視場所を決定することが非常に難しく，熟練者による知識が必要となってしまう．2 つ目は，プローブのオーバヘッドである．Xenprobes に比べ，オーバヘッドは格段に小さくなったが，それでも，多数のプローブを挿入すると，本来のサービスに影響を与える可能性がある．1 つ目の問題と関係するが，やはり，適切な場所だけにプローブを挿入する必要がある．

ⓒ2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.5
Vol.2011-OS-117 No.5
2011/4/13

5. おわりに

本論文では，仮想化環境においてホスト OS 上で仮想マシンの異常検知の高速化手法を提案し，その設計や実装方法について説明した．そして，クラスタシステムにおける障害検知遅延を改善したことを示し，その有効性を示すとともに，課題を述べた．

今後，今回の実験においての課題を解決するために，イベントドリブン方式に最適なクラスタ監視ソフトウェアの最適化手法及び，異常監視用イベントとしての監視場所の策定手法について研究していく．

参考文献

1) Ananth N. Mavimalayanahalli et al., Probing the Guts of Kprobes, In Proceedings of Ottawa Linux Symposium, 2006.
2) Nguyen Anh Quynh, Xenprobes, A Lightweight User-space Probing Framework for Xen Virtual Machine, In USENIX Annual Technical Conference Proceedings, 2007
3) Nitin A. Kamble et al., Evolusion in Kernel Debugging using Hardware Virtualization With Xen, In Proceedings of Ottawa Linux Symposium, 2006
4) Avi Kivity et al, kvm: the Linux Virtual Machine Monitor, In Proceedings of Ottawa Linux Symposium, 2007
5) Alan Robertson, Linux-HA Heartbeat Design, In Proceedings of the 4th International Linux Showcase and Conference, 2000
6) M.Hiramatsu et al., Djprobe-Kernel probing with the smallest overhead, In Proceedings of Ottawa Linux Symposium, 2007
7) Rusty Russell, virtio: towards a de-facto standard for virtual I/O devices, ACM SIGOPS Operation Systems Review, 2008
8) Karim Yaghmour, relayfs: An efficient unified approach for transmitting data from kernel to user space, In Proceedings of Ottawa Linux Symposium, 2003
9) Pablo Neira Ayuso et al, Communicating between the kernel and user-space in Linux using Netlink sockets, SOFTWARE-PRACTICE AND EXPERIENCE 2010; 00:1-7

ⓒ2011 Information Processing Society of Japan