Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

仮想マシンを用いたIDSオフロードにおけるCPU資源管理

新井昇鎬[†1]　光来健一[†2,†3]　千葉滋[†1]

侵入検知システム (IDS) を監視対象の仮想マシン (VM) から別の VM にオフロードすることにより、IDS 自体が攻撃されにくくなりセキュリティが向上する。しかし、オフロード元の VM の一部と考えるべき IDS が別の VM で動作するため、VM 間の性能分離が難しくなる。本論文では、IDS とオフロード元の VM の組を CPU 資源管理の単位とすることができる Resource Cage を提案する。Resource Cage に対して CPU 資源に関する制約を設定することで、IDS と VM をひとまとまりとした性能分離が可能になる。我々は、Resource Cage を用いて CPU 資源管理を行うシステム OffloadCage を開発した。OffloadCage を用いて、2 種類の IDS をオフロードして実験を行い、IDS と VM の CPU 使用率の合計を設定した値以下に制御できることを確認した。

Resource Management for Offloading IDS with VMs

Sungho Arai,[†1] Kenicni Kourai[†2,†3]
and Shigeru Chiba[†1]

IDS offloading pulls an intrusion detection system(IDS) outside of a monitored virtual machine(VM) and puts it into another to avoid getting it attacked. Under such IDS offloading, however, it becomes difficult to keep performance isolation across VMs because an IDS is a part of an offloading VM but runs on another VM. This paper proposes a resource cage, which is a new management unit of CPU resources for a pair of an offloaded IDS and an offloading VM. The CPU limits to resource cages enforce performance isolation across the groups. We have developed a system for scheduling CPU resources based on resource cages, named OffloadCage. From our experimental results, we confirmed that OffloadCage could control the sum of CPU utilization of an IDS and a VM under the limit.

1. はじめに

侵入検知システム (IDS) は攻撃者の侵入に備えるためのセキュリティソフトウェアであり、侵入の兆候を検知して管理者に通知する。侵入者は IDS に検出されてしまうのを防ぐために、侵入後にまず IDS の無効化を試みる。このような攻撃に対処するために、近年、仮想マシン (VM) を用いたオフロードが提案されている[1)2)]。IDS のオフロードとは、IDS を監視対象の VM で外で動作させ、別の VM などからオフロード元 VM を監視する手法である。オフロードを行うことにより監視対象の VM に侵入されても、IDS がその中で動作していないため、攻撃されにくくなり、IDS のセキュリティを向上させることができる。

しかし、IDS のオフロードを行うと VM 間の性能がうまく分離できなくなる。ある VM の CPU 資源の消費が同じ物理マシン上の他の VM に保証された性能に影響を与えない時、VM 間の性能は分離できていると言える。例えば、ある VM に物理マシンのもつ CPU 資源の 50 %を保証することができれば、この VM の性能は他の VM から分離できていることになる。もし他の VM の性能に影響を与えてしまうと、他の VM が十分な品質のサービスを提供できなくなる可能性がある。オフロードした IDS はオフロード元の VM のために監視を行っているので、IDS が消費した CPU 資源はオフロード元の VM が消費したと考えるのが望ましい。しかし、実際には、IDS をオフロードした先の VM で CPU 資源を消費したことになるため、オフロード元の VM の CPU 消費量に IDS の CPU 消費量を足すと、VM に保証された量以上の CPU 資源を消費してしまうことになる。これによって、他の VM の性能分離を保証できなくなる可能性がある。

本論文では、IDS とオフロード元の VM を CPU 資源管理の単位とすることができる Resource Cage を提案する。Resource Cage は既存の VM 単位での資源管理ではなく、VM と VM 内のプロセスの組み合わせを単位とした資源管理を可能にする。Resource Cage に対して CPU 資源に関する制約を設定することで IDS と VM をひとまとまりとした性能分離を実現することができる。

我々が開発した OffloadCage は、Resource Cage を用いてオフロードした IDS とオフロー

†1 東京工業大学
Tokyo Institute of Technology
†2 九州工業大学
Kyushu Insitute of Technology
†3 科学技術振興機構, CREST
Japan Science and Technology Agency, CREST

Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

ド元 VM の CPU 使用率を一括で管理し、Resource Cage に設定された CPU 制約に応じてスケジューリングを行うシステムである。OffloadCage は、OC-Monitor と OC-Scheduler と OC-Limiter の 3 つから構成される。OC-Monitor はオフロードした IDS による CPU 資源の消費を VMM で CR3 レジスタの値の変化を調べることで計測する。OC-Scheduler は Xen[3] の既存のスケジューラであるクレジットスケジューラ[4] をベースとして、オフロードした IDS の CPU 消費量を考慮してオフロード元 VM のスケジューリングを行う。OC-Limiter はオフロードした IDS による CPU 資源の消費が Resource Cage に設定された制限を越えないように IDS の CPU 資源の消費を調整する。

OffloadCage により VM 間の性能が分離できるかどうかを確かめるために、2 種類の IDS をオフロードして性能を測定する実験を行った。パケット受信のイベントごとにチェックを行う Snort[5] と一定時間ごとにファイルシステムのチェックを行う Tripwire[6] を用いた。この実験により、オフロードした IDS とオフロード元 VM を単位として CPU 資源の制限ができていることが確認できた。また、オフロードした IDS の CPU 使用率を VMM で測定した場合とゲスト OS で測定した場合とで比較し、VMM で測定するほうが正確な CPU 使用率が得られることを示した。

以下、2 章では IDS のオフロードの利点と問題点について述べ、3 章では Resource Cage を提案する。4 章では OffloadCage の設計と実装について述べ、5 章では OffloadCage による性能分離を確かめる実験を述べ、6 章ではマルチコア環境に Resource Cage がうまく適用できるかどうかについて議論する。7 章で関連研究について述べ、8 章で本論文をまとめる。

2. IDS オフロードの問題

VM を利用した IDS のオフロードは、**図 1** のように IDS を監視対象の VM から IDS-VM と呼ばれる VM に移動させて動作させる手法である。IDS-VM 内で動く IDS は VMM の機能を使うことにより、オフロード元の VM を監視することができる。IDS-VM 自体は攻撃を受けないように、外部に対するサービスの提供を制限する。例えば、ネットワーク型の IDS である Snort のオフロードを行う場合を考える。Snort を IDS-VM 内で動作させると、Snort 自体のプロセスを停止されてしまうことを防ぐことができる。また、Snort が使用するルールやポリシも IDS-VM 内に置かれるために、これらの改竄も防ぐことができる。

IDS のオフロードを行わなければ VM 間の性能を分離することができる。VM 間の性能分離とは、ある VM の CPU 資源の消費が他の VM に保証された性能に影響を与えないことである。たとえば、VM が二つ存在しそれぞれの VM に最大 CPU 使用率 50 %を設定した場合は、各 VM がその使用率を守り、CPU 資源の 50 %を使用できることが保証される。VM の中で IDS が動いていたとしても、IDS が使用する CPU 資源は VM に割り当てられたものである。IDS がどれだけ CPU 資源を消費したとしても、割り当てられた 50 %の範囲内であり、他の VM に影響を与えることはない。

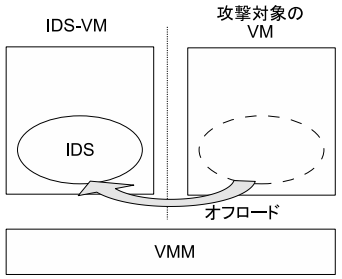

図 1 VM を利用した IDS のオフロード
Fig. 1 Offloading IDS with VMs

しかし、IDS を VM の外へオフロードすると VM 間の性能分離がうまく行えなくなる。オフロードした IDS はオフロード元 VM のために動作しているので、IDS が消費した CPU 資源はオフロード元 VM が消費したと見なすのが望ましい。実際には、IDS は IDS-VM 内で動作しているため、IDS-VM の CPU 資源を消費することになる。例えば、オフロード元 VM が最大 CPU 使用率の 50 %使用し、オフロードされた IDS が CPU 資源を 30 %使用したとすると、オフロード元 VM のために 80 %の CPU 資源が使用されたことになる。もし、CPU 資源を 50 %割り当てられた別の VM が存在していた場合、IDS のオフロードの影響によりその VM は 50 %の CPU 資源を利用することができない。

IDS のオフロードによる性能分離の問題は VMM が VM 単位で CPU 資源の分配を行っていることが原因である。例えば、Xen では各仮想マシンに CPU 使用率の上限と分配比率を設定するが、このような設定だけでは IDS-VM にオフロードした IDS を考慮して、オフロード元 VM に CPU 資源を割り当てることができない。単に IDS とオフロード元 VM に対して別々に CPU を制限しても不十分である。オフロードした IDS に 20 %、オフロード

Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

元 VM に 30 %という固定割り当てをすれば、合計を 50 %に抑えることができるが、IDS が CPU を使わないときに余った CPU 資源をオフロード元 VM に使わせることはできない。

3. Resource Cage

本論文では、IDS とオフロード元の VM を CPU 資源管理の単位とできるようにするための Resource Cage を提案する。Resource Cage はこれまで VMM 内において VM 単位で行われていた資源管理を拡張し、VM という実行単位から資源管理を切り離す。これにより、監視対象の VM とその外側で実行される IDS プロセスをひとまとまりとして扱うことができるようになる。**図 2** の例では、VM1 からオフロードした IDS1 とオフロード元の VM1 が一つの Resouce Cage を構成し、同様にして IDS2 と VM2 が別の Resource Cage を構成している。オフロードした IDS とオフロード元の VM の CPU 使用率は対応する Resource Cage に記録され、その合計が Resource Cage の CPU 使用率となる。

Resource Cage に対して CPU 使用率の上限等を設定することで、IDS と VM をひとまとまりとした性能分離を実現することができる。たとえば、オフロードした IDS とオフロード元 VM からなる Resource Cage に 50 %の最大 CPU 使用率を設定し、さらに Resource Cage の中の IDS に CPU を最大 30 % CPU を使用させるといった制御ができる。

Resource Cage は VM と IDS プロセスの組み合わせ以外で構成することもできる。IDS ごとに IDS-VM を用意するならば、Resource Cage ではオフロード元 VM と IDS-VM の 2 つの VM から成る。一方、IDS をオフロードしない VM に対しては、VM を一つの Resource Cage として CPU 資源を割り当てる。

4. OffloadCage の設計と実装

IDS のオフロードに対して Resource Cage を用いて CPU 資源管理を行うシステム OffloadCage を開発した。

4.1 OffloadCage の構成

OffloadCage は OC-Monitor と OC-Scheduler と OC-Limiter の 3 つから構成される。OC-Monitor と OC-Scheduler は**図 3** のように VMM の中で動作している。OC-Monitor はオフロードした IDS とオフロード元 VM の CPU 使用率を監視する。OC-Scheduler は IDS の CPU 使用率を考慮してオフロード元 VM の割り当て CPU 使用量を調整する。OC-Limiter は IDS-VM の中で動作し、オフロードした IDS の CPU 使用量を制限する。

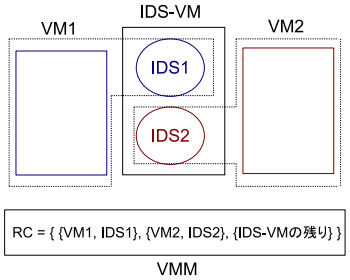

図 2 Resource Cage
Fig. 2 Resource Cage

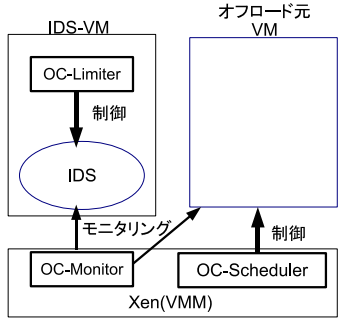

図 3 アーキテクチャ
Fig. 3 Architecture

Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

4.2 OC-Monitor

OC-Monitor は、IDS プロセスとオフロード元 VM の CPU 使用率を監視する。VMM で計測した IDS と VM の CPU 使用率をそれらが属する Resource Cage に記録する。VMM から VM 内のプロセスの CPU 使用率を取得するために、CR3 レジスタの変化に基づいてプロセスの実行時間を測定する[7]。CR3 レジスタにはプロセスのページディレクトリの物理アドレスが格納されている。プロセスが切り替わる時には VMM が呼び出され、CR3 レジスタの値を次のプロセスのページディレクトリの物理アドレスの値に変える。IDS プロセスの実行時間を記録するために、そのプロセスに対応するページディレクトリのアドレスが CR3 レジスタに格納されている時間を測定する。OC-Monitor が追跡すべき CR3 レジスタの値は、IDS-VM からハイパーコールを使って通知させる。

IDS プロセスの CPU 使用量はゲスト OS で測るほうが一般的だと考えられるが、以下の二つの理由から OffloadCage では VMM での計測方法を利用している。一つ目は、ゲスト OS では VM の切り替えが考慮されていない場合があることである。例えば、Linux の O(1) スケジューラの場合、VM がスケジューリングされていないときに溜まっていたタイマ割り込みがまとめて送られ、切り替え直前、又は直後に動作していたプロセスに次々と課金されてしまう。これは、O(1) スケジューラがタイマ割り込みの時に動いていたプロセスに課金する設計になっているためである。一方、Completely Fair Scheduler(CFS) を使った最近の準仮想化 Linux カーネルの場合、VMM からの情報を利用することで VM の切り替えを考慮するようになっている[8]。ただし、完全仮想化の場合は VM の切り替えを考慮するのは難しい。現在の実装では IDS-VM はドメイン 0 であり、ドメイン 0 の Linux は O(1) スケジューラを使っている。

もう一つの理由は、正しく CPU を使用した時間が課金されていない場合があることである。O(1) スケジューラではタイマ割り込みの時にだけプロセスの実行時間に課金されるので、タイマ割り込みよりも前にプロセス切り替えが起こった場合には課金されない。そのため、I/O バウンドなプロセスはタイマ割り込みよりも短い間だけ動作していることが多いので、CPU を利用しているのに課金されないことがある。一方、CFS の場合は、プロセス切り替え時に課金されるので上記のような問題は起こらない。

4.3 OC-Scheduler

OC-Scheduler は Resource Cage 内の IDS プロセスの CPU 使用率を考慮して VM のスケジューリングを行う。このスケジューラは Xen の仮想マシンスケジューラであるクレジットスケジューラを改良して作成した。クレジットスケジューラは Xen の現在のデフォルトの仮想マシンスケジューラである。Xen の仮想マシンスケジューラは各 VM に一つ以上の VCPU を割り当てる。VCPU とは VM が持つ仮想的な CPU である。VCPU を物理 CPU のランキューに入れてスケジューリングを行う。OS はプロセスをランキューに入れてスケジューリングを行うが、Xen では VCPU をランキューに入れてスケジューリングを行う。このスケジューラでは各 VM の VCPU に配布する CPU 時間をクレジットとして表現する。10ms ごとに現在実行状態にある VCPU のクレジットが減らされる。30ms ごとに各 VM に設定された cap と weight を基にクレジットが計算、配布される。cap は VM が使用できる CPU 利用率を表す絶対的な値である。weight は VM 間の CPU 資源の分配の比率を示す相対的な値である。

OffloadCage では VM ではなく、Resource Cage に対して cap と weight を設定する。cap が設定されている場合、cap の値 (RC.cap) から IDS の使った CPU 使用率の値 (RC.ids) の値を引いた値をオフロード VM の cap の値としてクレジットの計算を行う。オフロードした IDS の CPU 使用率分だけオフロード元 VM の最大 CPU 使用率を下げれば、その IDS とオフロード元 VM の CPU 使用率の合計は Resource Cage の cap に設定された値を越えることはない。配布するクレジットは以下のようにして計算される。

$$\text{配布 } credit = \text{総 } credit \times \frac{RC.cap - RC.ids}{100}$$

一方、weight のみが設定されている場合、RC.ids の分だけ weight の値を小さくする。weight に関しては、各 VM の weight の総和に対して、設定された weight の値がどのくらいの割合を占めているかによって配布されるクレジットが決まるためである。

$$\text{配布 } credit = \text{総 } credit \times \left(\frac{weight}{\text{総 weight}} - \frac{RC.ids}{100} \right)$$

このように RC.ids の値を使って cap と weight の値を調整した上で値で別々にクレジットを計算し、結果の小さいほうをその VM に配布する。

4.4 OC-Limiter

OC-Limiter は Resource Cage から IDS の CPU 使用率を取得して Resource Cage に設定された上限値を越えていれば IDS の CPU 使用量を制限する。以下の式のように、CPU 使用率からプロセスを停止させる時間、動作させる時間を 100ms ごとに調整する。計算された動作時間だけプロセスを SIGCONT で動作させ、停止時間だけ SIGSTOP でプロセスを停止させる。SIGCONT と SIGSTOP はプロセスに送るシグナルの種類である。このプ

Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

ロセスを制限する方法は cpulimit[9] をベースにして実装している。

$$\text{動作時間} = \min\left(100ms \times \frac{\text{上限値}}{\text{計測値}}, 100ms\right)$$

$$\text{停止時間} = 100ms - \text{動作時間}$$

5. 実　　験

IDS をオフロードした時の VM 間の性能分離を確かめる実験を行った。IDS としてイベント駆動型の Snort とタイマ型の Tripwire である。これらの IDS を Xen のドメイン 0 にオフロードした。また、プロセスの CPU 使用率を計測するのに、VMM で CR3 レジスタを利用する方法とゲスト OS 内の proc ファイルシステムを利用する方法を比較する実験を行った。

CPU は AMD Athlon 2.2GHz、メモリは 2GB、NIC はギガビットイーサ、HDD は SATA 250GB を搭載したマシンで実験を行った。Xen のバージョンは 3.3.0、ドメイン 0 の OS は Linux 2.6.18-8、ドメイン U の OS は Linux 2.6.16.33 であった。

5.1 Snort

Snort にドメイン 0 の仮想インターフェイス vif を監視させることで、ドメイン 0 へのオフロードを可能にした。Xen ではドメイン 0 が VM のネットワークの送受信を仲介している。これは、ドメイン 0 のみ物理 NIC にアクセスできるためである。ドメイン 0 は各 VM がネットワーク通信できるようにそれぞれに対して vif を提供している。例えば、VM 内のネットワークインターフェイスである eth0 に対応する vif1.0 がドメイン 0 内に作成される。

OffloadCage を利用する場合、オフロードした Snort とオフロード元 VM からなる Resource Cage には、最大 CPU 使用率 50 %を設定した。ドメイン 0 には CPU 利用率に関する制限は行わなかった。ドメイン 0 とオフロード元 VM の weight はともに 256 とした。オフロード元 VM では Web サーバを動作させ、外部の別のマシンから httperf[10] を利用してウェブサーバにリクエストを送った。httperf のリクエストレートは毎秒 4000 リクエストとした。この外部のマシンは、CPU が AMDAthlon 2.2GHz、メモリは 1GB、NIC はギガビットイーサであった。

図 4 と**図 5** は、Snort をオフロードして OffloadCage を使用した場合と使用しなかった場合の CPU 使用率の変化である。結果から分かるように OffloadCage を使用した場合はオフロードした Snort とオフロード元 VM の合計 CPU 使用率が Resource Cage の制限の 50 %を守ることができている。しかし、OffloadCage を使用しない場合は合計 CPU 使用

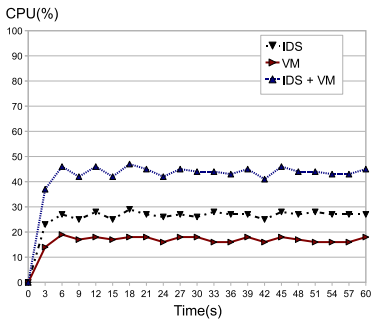

図 4　CPU 使用率 (Snort, OffloadCage)

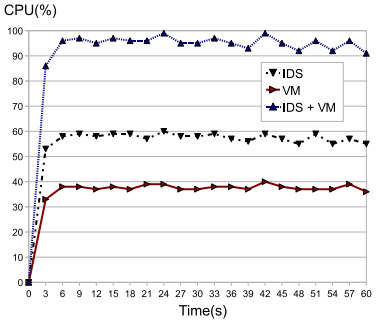

図 5　CPU 使用率 (Snort, OffloadCage なし)

率が Resource Cage の制限を大幅に越えている。オフロード元 VM の CPU 使用率は 50 %以下であるが、オフロードした IDS はその制限と無関係にドメイン 0 の CPU 資源を 60 %程度消費した。

図 6 はオフロード元 VM で動作させた Web サーバのスループットである。単純にオフロードした場合はオフロードした Snort の分だけオフロード元のウェブサーバが使用できる CPU 資源が増える。その結果、オフロードしない場合よりウェブサーバのスループットが増大した。一方、OffloadCage を用いた場合は、オフロードしない場合よりスループットが減少している。この原因を調べるために、Snort をオフロードしない場合の VM の CPU 使用率を測定してみた。(**図 7**) その結果、オフロード元 VM によってドメイン 0 の CPU 資源が使われており、50 %の制限をかけているにも関わらず、80 %程度使ってしまっていることがわかる。これはドメイン 0 がオフロード元 VM のネットワーク処理の一部を行うためである。OffloadCage ではこの分は IDS の CPU 利用率の計測値に含まれており、正味 50 %の CPU 資源しか使っていないためにスループットが減少したと考えられる。

5.2 Tripwire

Tripwire にオフロード元 VM が使う仮想ディスク内のファイルシステムを検査させることで、ドメイン 0 へのオフロードを実現した。Xen では、イメージファイルを仮想ディスクとして仮想マシンを作成することができる。ループバックデバイスを利用して、このイ

Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

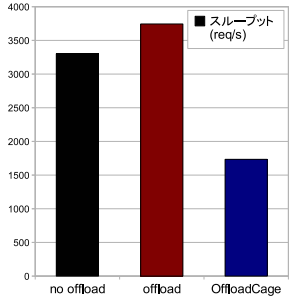

図 6 ウェブサーバのスループット

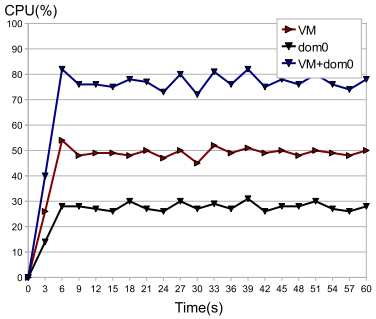

図 7 CPU 使用率 (Snort, オフロードなし)

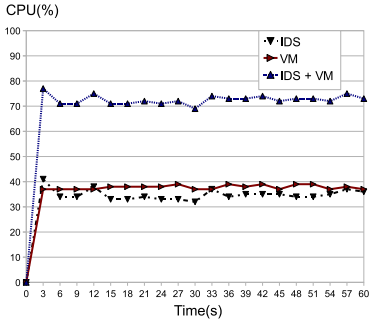

図 8 CPU 使用率 (Tripwire, OffloadCage)

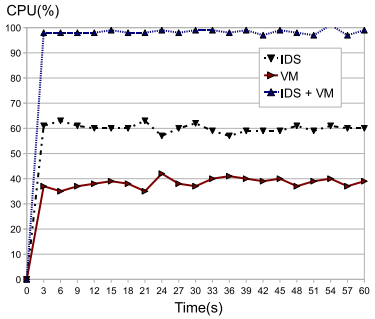

図 9 CPU 使用率 (Tripwire, OffloadCage なし)

メージファイルを読み込み専用でマウントすることにより、ドメイン 0 上でオフロード元 VM のファイルを参照することが可能になる。ただし、ドメイン 0 の OS が仮想ディスク内のファイルシステムを認識できなければならない。今回の実験ではオフロード元 VM でも Linux 標準の ext3 ファイルシステムを用いた。

オフロードした Tripwire とオフロード元 VM からなる Resource Cage には最大 CPU 使用率 70 %を設定した。また、OC-Limiter で Tripwire は最大 30 %までしか CPU を利用できないように制限した。その他の設定は 5.1 と同様であり、オフロード元の Web サーバに httperf でリクエストを送った。

図 8 と**図 9** は、Tripwire をオフロードして OffloadCage を使用した場合と使用しなかった場合である。グラフから分かるように、単純にオフロードした場合、Resource Cage の制限である 70 %を大幅に越えている。一方、OffloadCage を用いた場合はほぼ 70 %の Resource Cage の制限を守れていることが分かる。

5.3 CPU 使用率の計測方法の比較

まず、ある VM の中で動作する無限ループするプログラムの CPU 使用率を計測するのに VMM 内で CR3 レジスタを利用した場合と VM 内で proc ファイルシステムを利用した場合を比較した。途中で別の VM で同じように無限ループするプログラムを動作した。2 つの VM でそれぞれ CPU バウンドなプロセスが動作しているので、50 %ずつの CPU 使用率になるはずである。CR3 レジスタを利用して計測する方法では別の VM で無限ループが動作すると CPU 使用率が正しく 50 %まで下がる。しかし、proc ファイルシステムを利用して計測する方法では、O(1) スケジューラを用いたカーネルの場合、別の VM で無限ループするプログラムが動作を開始しても CPU 使用率はほぼ 100 %のままで下がらなかった。これは、O(1) スケジューラを動作させている Linux カーネルが VM の切り替えを考慮していないことが原因である。一方、CFS を用いた Linux カーネルでは VM を切り替えを考慮されているので、proc ファイルシステムで計測しても CR3 レジスタで計測した場合と同じ結果になる。しかし、CR3 レジスタを利用すれば OS に依存しないで計測できる。

図 11 と**図 12** はある VM の中で動作する Snort と Tripwire の CPU 使用率を CR3 レジスタと proc ファイルシステムを利用して計測した結果である。CR3 レジスタを利用したほうが CPU 使用率が高く計測されていることがわかる。これは先述したように O(1) スケジューラでは I/O バウンドなプロセスが使用した CPU 使用量を適切に課金できないことが原因である。CFS スケジューラで計測すれば、CR3 レジスタと同じ結果がでることが予想される。しかし、現在のドメイン 0 の標準のスケジューラは O(1) スケジューラであり、CFS スケジューラは使用できない。

Vol.2010-ARC-189 No.6
Vol.2010-OS-114 No.6
2010/4/21

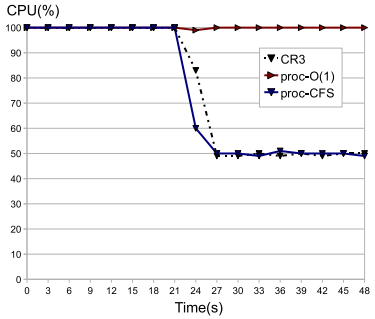

図 10 CR3 と proc で測定した CPU 使用率 (loop)

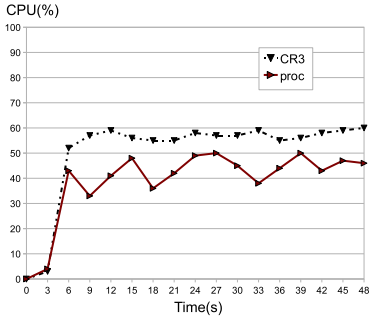

図 11 CR3 と proc で測定した CPU 使用率 (Snort)

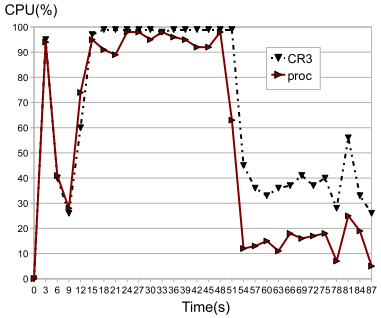

図 12 CR3 と proc で測定した CPU 使用率 (Tripwire)

6. マルチコア CPU と Resource Cage

シングルコアの CPU が 1 つだけしかない場合は、Resource Cage を導入しても効率よく CPU 資源を使い切ることができる。すべての VM で 1 つの CPU を共有するため、Resource Cage によって VM や IDS への CPU 資源の配分が制約されても、他の VM や IDS がその分を使うことができる。しかし、マルチコア CPU の場合、VM へのコアの割り当てには様々な方法が考えられるため、必ずしも効率よく CPU を使い切ることができない。

IDS-VM とオフロード元 VM がコアを共有している場合、コアが一つの場合と同様に、CPU を使い切ることができる。しかし、IDS-VM とオフロード元 VM がコアを共有していない場合、オフロード元 VM の CPU 資源を使い切ることができない場合がある。IDS が使った CPU 使用率の分をオフロード元 VM から差し引くと、これらの間でコアを共有していないため、差し引いた分だけオフロード元 VM の CPU 資源が余ってしまう。使われない CPU 資源は無駄になるため、一般的にはこのような制御は無意味である。ただし、CPU 使用率に対して料金体系が変わるような場合や、省電力を考えた場合にはこのような制御も有用かもしれない。

7. 関 連 研 究

Xen における I/O 処理を考慮した VM の性能分離を行うスケジューラとして SEDF-DC[11] がある。Xen ではドライバがドメイン 0 側のバックエンドとドメイン U 側のフロントエンドで分かれている。しかし、ドメイン 0 のバックエンドドライバによる CPU 資源の消費は、I/O 処理を行っているドメイン U に課金されない。そこで SEDF-DC では、ドメイン U のための I/O 処理で使用された CPU 消費量を測定し、そのドメイン U の使用量として課金する。また、ShareGuard という機構で各ドメイン U のための I/O 処理に使われるドメイン 0 の CPU 使用量を制限する。これらは OffloadCage の OC-Scheduler と OC-Limiter と類似している。しかし、SEDF-DC はネットワーク I/O 処理に特化している点と SEDF をベースに開発している点が異なる。SEDF-DC ではパケット数から CPU 使用量を見積り、パケットフィルタによって CPU 使用量を制限する。

LRP[12] はカーネル内のネットワーク処理に使われる CPU 消費量をプロセスに課金できるようにしている。ネットワーク処理をよく行うプロセスがその処理の大部分をカーネルで行うことになる。しかし、カーネル内のネットワーク処理による資源の消費はそのプロセスのものとして適切に課金されない。LRP では、カーネル内のネットワーク処理をプロセス

のコンテキストで行うことで CPU 資源の消費を各プロセスに適切に課金する。

Resource Container[13] は、LRP を拡張して OS に新しい資源管理の方法を導入している。アプリケーションの資源管理を行うとき、プロセス単位の管理が適切であるとは限らない。そこで Resource Container はプロセスとは異なる単位で CPU やメモリなどのシステムの資源を管理することができる。スケジューラはプロセスの属する Resource Container の資源の消費量など情報を利用してそのプロセスのスケジューリングを行う。我々の Resource Cage は Resource Container を VMM に応用したものである。

8. まとめと今後の課題

本論文では、IDS のオフロード時の VM 間の性能分離を実現する Resouce Cage を提案した。Resource Cage は IDS とオフロード元 VM をひとまとまりとして新しい CPU 資源管理の単位である。Resouce Cage に対して CPU 資源に関する制限を設定することでオフロードした IDS を考慮した性能分離が可能になる。我々は Resource Cage を用いた OffloadCage を開発し、Snort と Tripwire を実際にオフロードして Resource Cage の制限を守れていることを実験で確認した。

今後の課題としては、Resource Cage 内の IDS と VM の間でうまく CPU 資源の分配を行えるようにする必要がある。現状では Resource Cage 内の最大 CPU 使用量を設定できるなどの静的な制限しかできない。また、OC-Limiter を VMM の中で動作させることを検討している。VMM からゲスト OS プロセススケジューリングを変更する手法[14] を利用すれば、VMM の中でオフロードした IDS のプロセスを制御できる。OC-Limiter は VMM の中で動作すれば、OffloadCage を VMM の中のみで動作させることができる。また、監視する資源を CPU 使用量だけではなく、メモリやディスクなどの他の資源にも拡張することも今後の課題である。

参 考 文 献

1) Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Network and Distributed Systems Security Symp*, pp.191–206 (2003).
2) 滝澤裕二，光来健一，　千葉滋，柳澤佳里：SAccessor: デスクトップ PC のための安全なファイルアクセス制御，情報処理学会論文誌:コンピューティングシステム（ACS），pp.124–135 (2008). in Japanese.
3) 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, *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, ACM, pp.164–177 (2003).
4) *Credit Scheduler*, http://wiki.xensource.com/xenwiki/CreditScheduler.
5) Roesch, M.: Snort - Lightweight Intrusion Detection for Networks, *LISA '99: Proceedings of the 13th USENIX conference on System administration*, Berkeley, CA, USA, USENIX Association, pp.229–238 (1999).
6) Kim, G. H. and Spafford, E. H.: The design and implementation of tripwire: a file system integrity checker, *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, New York, NY, USA, ACM, pp.18–29 (1994).
7) Jones, S.T., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Antfarm: tracking processes in a virtual machine environment, *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, Berkeley, CA, USA, USENIX Association, pp.1–1 (2006).
8) VMware, Inc: *Timekeeping in VMware Virtual Machines*, `http://www.vmware.con/pdf/vmware\_timekeeping.pdf` (2008).
9) *cpulimit: CPU Usage Limiter for Linux*, http://cpulimit.sourceforge.net.
10) Mosberger, D. and Jin, T.: httperf—a tool for measuring web server performance, *SIGMETRICS Perform. Eval. Rev.*, Vol.26, No.3, pp.31–37 (1998).
11) Gupta, D., Cherkasova, L., Gardner, R. and Vahdat, A.: Enforcing performance isolation across virtual machines in Xen, *Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, New York, NY, USA, Springer-Verlag New York, Inc., pp.342–362 (2006).
12) Druschel, P. and Banga, G.: Lazy Receiver Processing(LRP): A Network Subsystem Architecture for Server Systems, *Operating Systems Design and Implementation*, pp.261–275 (1996).
13) Banga, G., Druschel, P. and Mogul, J.C.: Resource containers: a new facility for resource management in server systems, *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, Berkeley, CA, USA, USENIX Association, pp.45–58 (1999).
14) 田所秀和，光来健一，　千葉滋：仮想マシン間にまたがるプロセススケジューリング，情報処理学会論文誌:コンピューティングシステム（ACS），pp.124–135 (2008). in Japanese.