平成21年度 学士論文

セキュリティ機構の
オフロードを考慮した
仮想マシンの
スケジューリング

東京工業大学 理学部 情報科学科
学籍番号 05-01080
新井　昇鎬

指導教員
千葉 滋 教授

平成19年2月13日

概要

今日、仮想マシンが普及してきたことで、１つの物理マシン上で複数のOSが動作できるようになった。仮想マシンを用いれば、ホスティングを行う場合、物理マシン単位ではなく、仮想マシン単位で利用者に貸し出しを行える。このような仮想マシン間では、パフォーマンスの分離は重要である。パフォーマンスの分離とは、複数の仮想マシンが動作しているときに、仮想マシンが管理者の指定した計算資源の使用量や比率を守っている環境である。仮想マシンへ計算資源を正確に分配できることは、仮想マシン単位で利用者に提供する管理者とって必要不可欠なことである。

しかし、このようなパフォーマンスの分離は、セキュリティ機構を仮想マシンの外へオフロードする構成を採用した場合、困難になる。セキュリティ機構のオフロードとは、セキュリティ機構を攻撃対象の外へ移すことでセキュリティを向上させる方法である。攻撃対象の外へ出すことにより、IDS(侵入探知システム) やファイヤーウォールやアクセスコントロールなどのセキュリティソフトウェアを攻撃されにくいようにする。このように仮想マシンを使用してセキュリティ機構をオフロードすると、オフロードしたセキュリティ機構のCPU使用量がオフロード元の仮想マシンに含まれないので、仮想マシン間のCPU資源の分配に問題が応じる。たとえば、仮想化ソフトウェアとしてXenを使用した場合は、一般ドメインから特権ドメインへセキュリティ機構をオフロードする構成が考えられる。Xenでは一般ドメインと特権ドメインの２種類の仮想マシンが動作している。もしオフロード元のドメインが攻撃されると、特権ドメインにあるセキュリティ機構も仕事が増え負荷がかかるので、大量のCPUを使用してしまう。オフロードしたセキュリティ機構のCPU使用量は特権ドメインに含まれるので、セキュリティ機構とオフロード元のドメインのCPU使用量は、管理者がオフロード元のドメインに指定した分配を超えてしまう。

本研究ではセキュリティ機構のオフロードがなされても、パフォーマンスの分離を実現する仮想マシンのスケジューラを提案する。オフロードしたセキュリティ機構のCPU使用率を計測して、その値に応じてオフロード元の仮想マシンの割り当てCPU時間が減少するようにドメインスケジューラを修正した。本スケジューラでは、オフロードしたセキュリティ機構のCPU使用量はオフロード元の仮想マシンにカウントしている

ように見える。

具体的には、Xen の仮想マシンのスケジューラであるクレジットスケジューラを改良した。クレジットスケジューラは、cap と weight の二つの値により各ドメインの割り当て CPU 時間を計算している。cap は最大 CPU 使用率を表す絶対的な値であり、weight は各ドメインと比較して使用する相対的な値である。よって、オフロード元のドメインの cap を減らすことで最大使用率を下げ、weight を減らすことで他のドメインに対する相対的な CPU 時間を減らしている。実際のオフロード元のドメインの cap と weight は変更されないが、スケジューリング時は計測したセキュリティ機構の CPU 使用量に応じて減少された cap と weight で割り当て CPU 時間が計算される。

本研究では実験を行い、一般ドメインを Web サーバとして動作させ、セキュリティ機構をオフロードしない場合、オフロードした場合、そしてオフロードして本スケジューラを用いた場合の三つときの Web サーバのパフォーマンスを比較した。セキュリティ機構としては snort を、ベンチマークツールに httperf を使用している。snort はオープンソースな侵入探知システムである。本スケジューラにより、一般ドメインからセキュリティ機構をオフロードしても、Web サーバがオフロードしなかった場合と同じパフォーマンスであることを確認した。

謝辞

本研究と進めるにあたり、研究の方針や、論文の書き方について助言をしていただいた指導教員の千葉先生に心より感謝いたします。また、九州工業大学の光来先生にはシステムの設計・実装等の研究全般に渡り指導していただき、心より感謝します。東京工業大学の田所秀和氏には、研究活動において多くの助言をいただきました。心より感謝致します。最後に、ともに研究活動をおこなった研究室の皆様に感謝します。

目 次

図 目 次

表 目 次

第1章　はじめに

今日、**仮想マシン**が普及してきたことで、1つの物理マシン上で複数のOS が動作できるようになった。仮想マシンを用いれば、ホスティングを行う場合、物理マシン単位ではなく、仮想マシン単位で利用者に貸し出しを行える。このような仮想マシン間では、**パフォーマンスの分離**は重要である。パフォーマンスの分離とは、複数の仮想マシンが動作しているときに、仮想マシンが管理者が指定した計算資源の使用量や比率を守っている環境である。仮想マシンへ計算資源を正確に分配できることは、仮想マシン単位で利用者に提供する管理者とって必要不可欠なことである。

しかし、このようなパフォーマンスの分離は、セキュリティ機構を仮想マシンの外へオフロードする構成を採用した場合、困難になる。**セキュリティ機構のオフロード**とは、セキュリティ機構を攻撃対象の外へ移すことでセキュリティを向上させる方法である。攻撃対象の外へ出すことによりIDS(侵入探知システム) やファイヤーウォールやアクセスコントロールなどのセキュリティソフトウェアを攻撃者が攻撃できないようにする。たとえば、仮想化ソフトウェアとして Xen を使用した場合は、Xen では一般ドメインとその一般ドメインを管理する特権ドメインの2種類の仮想マシンが動作しているので、一般ドメインである対象のドメインから特権ドメインへセキュリティ機構をオフロードする構成が考えられる。

このように仮想マシンを使用してセキュリティ機構をオフロードすると、オフロードしたセキュリティ機構の CPU 使用量が対象仮想マシンに含まれないので、仮想マシン間の CPU 資源の分配に問題が応じる。もし対象仮想マシンが攻撃を受けセキュリティ機構が大量の CPU を使用すると、管理者の指定した分配以上の CPU が対象仮想マシンのために使用される。たとえば、Xen を利用した特権ドメインへのオフロードの場合も、対象の一般ドメインが攻撃されると、特権ドメインにあるセキュリティ機構が CPU をその一般ドメインのために大量に使用してしまう。オフロードしたセキュリティ機構の CPU 使用量が特権ドメインに含まれるので、セキュリティ機構と対象の一般ドメインの CPU 使用量は管理者の指定した分配を超えてしまう。

本研究では、仮想マシンを利用してセキュリティ機構をオフロードした構成でも、仮想マシン間のパフォーマンスの分離を実現できる仮想マシン

のスケジューラ、**Yen** を考案した。

Yen は次の特徴をもつ。

- 仮想ソフトウェアとして **Xen**[1] を用いた。Xen はハードウェア層の上で動作し、仮想マシンを監視する仮想マシンモニターである。Xen は各仮想マシンとその上で動作するオペレーティングシステムをドメインとして管理する。ドメインには一般ドメインと特権ドメインに分けられる。特権ドメインは一つであり各ドメインの管理するドメインである。一般ドメインは通常使用されるオペレーティングシステムを動作させるドメインである。

- 現在の Xen のデフォルトのドメインスケジューラであるクレジットスケジューラを対象としている。クレジットスケジューラを修正して、特権ドメインへセキュリティ機構をオフロードしている一般ドメインを認識し、セキュリティ機構の CPU 使用量分の割り当て CPU 時間を減少させている。

- 本研究では、オフロードするセキュリティ機構として snort を使用している。snort とはオープンソースな IDS(侵入探知システム) である。しかし、本システムはプロセス ID を指定するばよいので、他のセキュリティソフトでも対象となる。

仮想マシンを利用したセキュリティ機構のオフロードするアーキテクチャーには、攻撃者されにくく、仮想マシンを利用することで同様の検知力を持つという利点がある。

しかしながら、セキュリティ機構のオフロードにより仮想マシン間のパフォーマンスの分離という点で問題がある。つまりオフロードすることにより他の仮想マシン上のオペレーティングシステムのパフォーマンスに影響を与えることがある。

そこで我々は、セキュリティ機構のオフロードしたアーキテクチャーでも仮想マシン間のパフォーマンスの分離を実現できる **Yen** を提案する。

本稿の残りは、次のような構成からなっている。第 2 章では仮想マシンを用いた IDS のアーキテクチャーとその問題について述べる、第 3 章では **Yen** の提案とその特徴を説明する。第 4 章では **Yen** の実装、第 5 章ではオフロード前とオフロード後と本システムの 3 つアーキテクチャーを比較する実験、第 6 章でまとめを述べる。

第2章　問題点と関連研究

2.1　仮想マシン

2.1.1　仮想マシンとは

仮想マシンという環境は、メインフレームが広く使われていた時代から使われていた。仮想マシン環境とは、ハードウェアを仮想化し、その仮想ハードウェア上でオペレーティングシステムなどのソフトウェアを動作されることである。つまり、仮想マシン環境を用いれば、一つの物理マシン上で複数のオペレーティングシステムを動作させることが可能になる。

メインフレームが使われていた時代は、古いメインフレームで動いていたアプリケーションを新しいメインフレームで動作させるために仮想マシンという技術が必要であった。当時は古いメインフレームと新しいメインフレームの間に互換性がなく、古いメインフレームのアプリケーションは新しいメインフレームでは動かなかった。古いメインフレーム用に作られたアプリケーションは高価だったために、新しいメインフレームでも動作させたいニーズがあり、仮想マシンという環境が重宝されいた。

過去のメインフレームの時代における仮想マシンの目的が、過去の資産の継承ならば、最近の仮想マシンの目的は資源の有効な利用することである。仮想マシンを用いることで、複数あったサーバー機を一つの物理マシンに集約することができ、また進化の速いハードウェアの資源を十分に利用することもできる。

現在の仮想化の手法として大きく分けて二つある。図 2.1 のようなハイパーバイザ型とホスト型である。ハイパーバイザ型はハードウェアの上にハイパーバイザと呼ばれる仮想マシンを実行する機能だけに特化した最小限のソフトウェアを置く方式ある。ホスト型は仮想ソフトウェアをホストＯＳのアプリケーションとして動かす方式である。本研究で使用している仮想化ソフトウェアの Xen はハイパーバイザ型の方式の代表である。

2.1.2　パフォーマンスの分離

今日、仮想マシンが普及してきたことで、１つの物理マシン上で複数の OS が動作できるようになった。仮想マシンを用いれば、ホスティングを

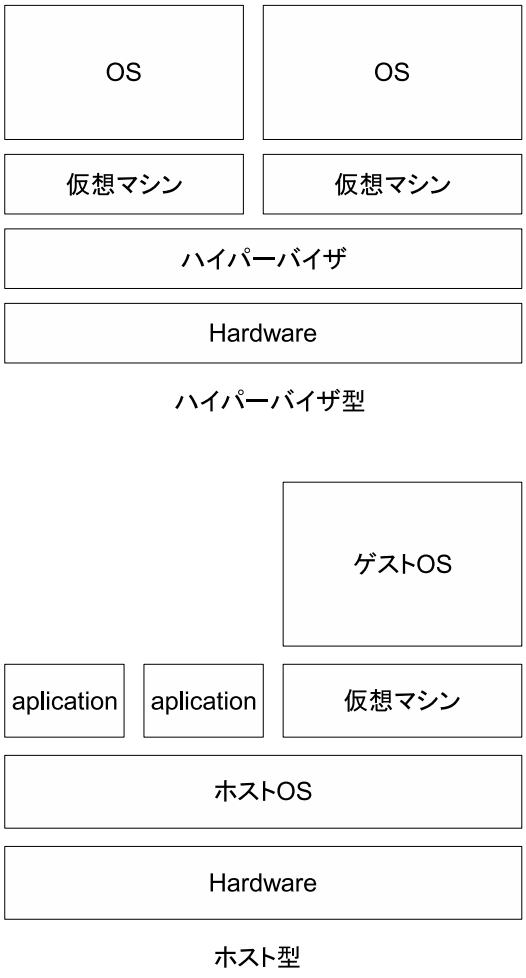
OS
OS
仮想マシン
仮想マシン
ハイパーバイザ
Hardware
ハイパーバイザ型
ゲストOS
aplication
aplication
仮想マシン
ホストOS
Hardware
ホスト型

図 2.1: 仮想化の手法

行う場合、物理マシン単位ではなく、仮想マシン単位で利用者に貸し出しを行える。このような仮想マシン間では、パフォーマンスの分離は重要である。パフォーマンスの分離とは、複数の仮想マシンが動作しているときに、仮想マシンが管理者が指定した計算資源の使用量や比率を守っている環境である。仮想マシンへ計算資源を正確に分配できることは、仮想マシン単位で利用者に提供する管理者とって必要不可欠なことである。

2.1.3　Xen

Xen は一つの物理マシン上に複数の仮想マシンを管理する仮想マシンモニターである。各仮想マシンの上に一つのオペレーティングを置き、それをドメインという単位で管理することで、一つの物理マシン上にマルチオペレーティングシステムを提供する。Xen を用いれば別々マシンに分けていたサーバを一つの物理マシン上の仮想マシンとして統合できるので、データセンタなどのラックスペースを物理マシンで埋め尽くす必要がなくなる。また電力コストも削減できる。

ドメインはドメイン０と呼ばれる特権ドメインと、ドメインUと呼ばれる一般ドメインに分けられる。ドメイン０はドメインUの動作を制御したり、ハードウェアへのアクセスをドメインUから請け負う。ドメイン０はドメインUよりハードウェアへの直接アクセスなど高い権限が与えられているので、ドメイン０の不具合はシステム全体に影響を与える可能性がある。よって通常の使用は、ドメインUのオペレーティングシステムにしたほうが安全である。

Xen の機能とドメイン０への依存

Xen はオペレーティングシステムよりも高い権限で動作するので、Xen 自体の不具合もシステム全体を影響を与える。よって Xen は最小の機能しか備わっていなく、デバイスドライバなどのバグを含むをシステムに危険な影響を与える機能はドメイン０などに依存している。たとえばデバイスドライバは図 2.2 のように、スピリットデバイスドライバがドメインUにフロントエンド、ドメイン０にバックエンドに分けられ、バックエンドから本物のデバイスドライバへ通じる構成になっている。

準仮想化と完全仮想化

Xen には準仮想化 (paravitrualization) と完全仮想化 (fullvirtualization) の２種類の仮想化モデルがある。準仮想化にはインストールするオペレーティングシステムを修正する必要があり、完全仮想化には仮想化を支援す

るハードウェアが必要である。主に x86 アーキテクチャーを中心に説明する。

準仮想化では、オペレーティングシステムのソースコードを修正して、Xen がリング 0 で動作してオペレーティングシステムがリング 1 で動作するようになる。つまりオペレーティングシステムは特権命令を実行できない。類似の機能を提供するために、Xen は特権命令に対応したハイパーコール (hypercall) のセットを用意する。よってオペレーティングシステムは Xen が提供するハイパーコールを使用することで、ハードウェア制御などの命令を実行できる。具体的には、オペレーティングシステムは Xen によってマッピングされる共有メモリページで関数を呼び足す。

完全仮想化では、ハードウェアによる仮想化支援によりオペレーティングシステムのソースコードを修正することなくインストールできます。Intel の仮想化支援は IVT(Intel Virtualization Technology) と呼ばれています。これは Xen をリング-1 で動作させ、リング 0 で動作するオペレーティングシステムが特権命令を実行したらハードウェアがそれを検出し、Xen に制御を渡す。完全仮想化はエミュレーションのためのコストが大きくなるのでパフォーマンスが落ちるが、オペレーティングシステムのソースを修正する必要がないので Winows なども動作できる。

2.2　仮想マシンを利用したセキュリティ機構のオフロード

セキュリティ機構のオフロードとは、攻撃対象の外へ移すことでセキュリティを向上させる方法である。IDS(侵入探知システム) やファイヤーウォールやアクセルコントロールなどのセキュリティソフトが攻撃対象ホストの内にあると攻撃される可能性があるので、仮想マシンを利用することにより、セキュリティ機構を対象仮想マシンの外へオフロードして攻撃者の目から離すことができる。

2.2.1　IDS のオフロード

ネットワークにおけるセキュリティ対策として用いられるのが、ファイアウォールと IDS である。この二つは、基本的に役割が異なる。

IDS(Intrusion Detection System) は侵入探知システムの略で、ネットワークを流れるパケットに対し、システム内の情報と照らし合わせ、攻撃の疑いがあると判定できるパターンを含む場合は警告を発する。IDS は警告を発するだけであり、攻撃を積極的に遮断するものではない。

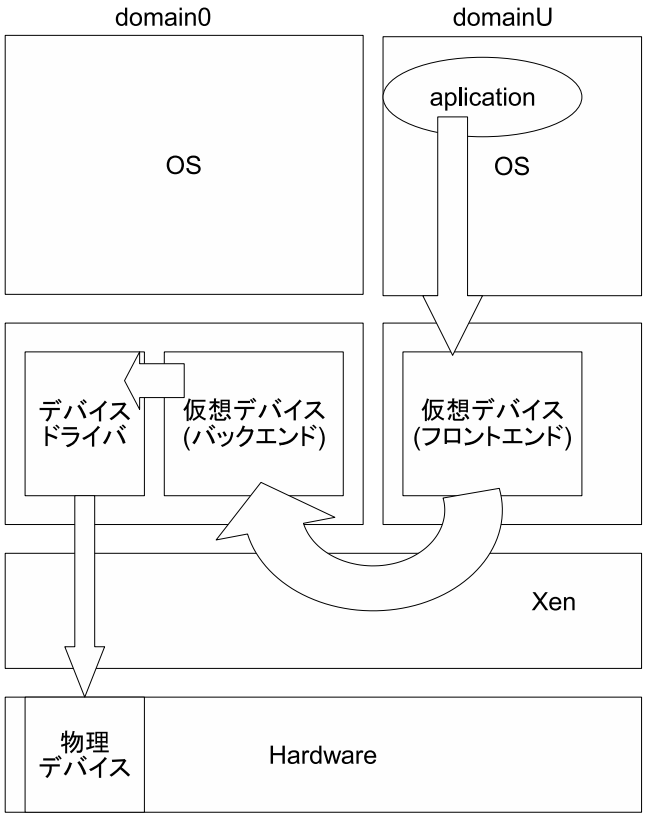
domain0
domainU
aplication
OS
OS
デバイス
ドライバ
仮想デバイス
(バックエンド)
仮想デバイス
(フロントエンド)
Xen
物理
デバイス
Hardware

図 2.2: Xen の構成

ファイアウォールは外からの来るパケットの IP や TCP,UDP を見て、管理者が設定した条件と照らし合わせ、遮断したり通過させてりする。よって IP や TCP,UDP が条件を満たせば、攻撃の疑いがあるパケットでもファイアウォールを突破できてしまう。IP などは簡単に偽装できてしまうので、ファイアウォールだけで攻撃の疑いのあるパケットを遮断することはできない。

ファイアウォールで足らない部分を IDS で補うことことができる。つまり、ファイアウォールと IDS を両方用いることで、ファイアウォールを突破した攻撃の疑いのあるパケットでも IDS で監視できる。

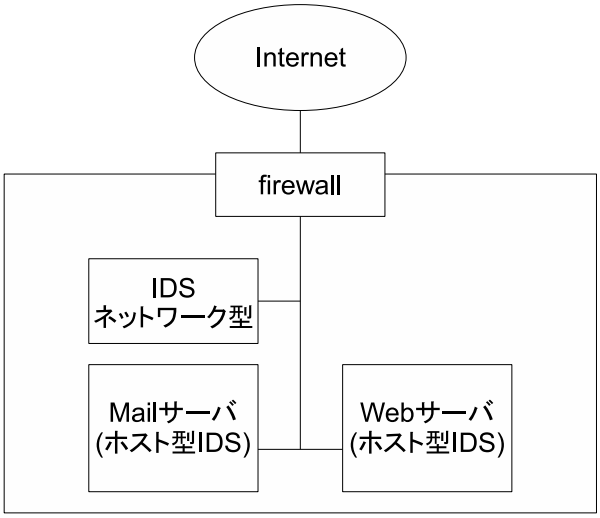

図 2.3: Firewall と IDS

ホスト型 IDS

ホスト型 IDS とは　IDS は配置する場所によって、ホスト型とネットワーク型に分類できる。ホスト型の IDS が監視するのは、IDS を稼働させているコンピュータ自身であり、そのコンピュータ以外は監視対象とならない。

ホスト型の IDS は、そのホストの OS やアプリケーションの活動を監視します。たとえば監視対象ホストの受信したパケット、イベントやログ、

システムコールなどを監視して特定の変化やパターンを検知すると管理者へ通知する。

ホスト型 IDS のメリット・デメリット　ネットワーク型 IDS は攻撃の可能性を検知するのに対し、ホスト型 IDS は すでに成功した攻撃を検知するので偽陽性率が低い。一見、すでに成功した攻撃を検知することは手遅れと思われるかもしれないが、IDS のない場合はその成功した攻撃をしることさえもできない。またホスト型は、監視対象ホスト内にあるのでより攻撃のある行動をより具体的に監視できる。しかし、攻撃者の目に触れやすいことや監視対象ホストの性能オーバーヘッドなどの課題もある。

ネットワーク型 IDS

ネットワーク型 IDS とは　ネットワーク型 IDS は、対象となるネットワーク全体を監視する。ネットワーク型 IDS は、シグネチャーベースと異常ベースのシステムに分類できる。シグネチャーベースの IDS は既知の攻撃シグネチャーのデータベースに基づき警告を発する。異常ベースの IDS は正常なネットワーク活動の基準線を確立し、その基準線を逸脱するような事象が発生した場合に警告を発する。

ネットワーク型 IDS のメリット・デメリット　ネットワーク型の IDS は、ネットワーク全体を見ることができるので、複数のホストが関わる攻撃パターンに対し、重要な監視を行うことができる。また攻撃者と距離があるので攻撃対象となりにくい。しかし、特定のホストの内部の状態を知ることができない。

IDS のオフロード

ホスト型 IDS は監視対象ホスト内の具体的な活動を監視できるが攻撃者の目に触れやすい、それに対しネットワーク型 IDS は攻撃者から距離があるので攻撃されにくいがホスト内の活動を監視することはできない。しかし、ホスト型のメリットである対象ホストの具体的な活動を監視でき、またネットワーク型のメリットである攻撃されにくいアーキテクチャーが仮想マシンを用いると考えられる。

たとえば Xen を用いると、あるドメイン U を対象ホストと考えるとき、そのドメイン上の IDS を導入するとホスト型 IDS になるが、そのドメインから IDS をオフロードしてドメイン 0 上に IDS を配置する。このアーキテクチャーだと監視対象ホスト内に IDS がないので攻撃者の目に触れ

ることはなく、また監視対象ホストが動作している仮想マシンを利用すれば監視対象ホストの具体的な活動を監視することができる。

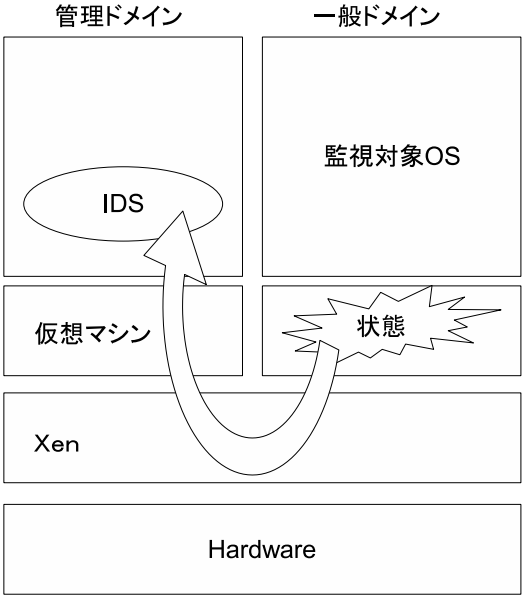

図 2.4: IDS のオフロード

2.2.2　Firewall のオフロード

仮想マシンを利用することにより、ホスティングの場合に仮想マシン単位で利用者に貸し出しを行うことができる。このとき、仮想マシン内の OS の管理者が管理者としての知識が不足している可能性がある。たとえ

知識があったとしても、常時管理することは難しい。そこで、仮想マシンモニタなどで知識の豊富な管理者がまとめて管理することが望ましい。

仮想マシンを利用したファイヤーウォールのオフロードの例として、xFilter[5] では、仮想マシンから Firewall を仮想マシンモニタへオフロードしてパケットフィルタリングを実現している。東京工業大学の田所氏のシステムを利用して仮想マシンモニタのメモリに仮想マシン内のゲスト OS のメモリをマッピングして、ゲスト OS の情報を参照している。仮想マシンモニタはゲスト OS のすべてのパケットを中継するため、ゲスト OS の情報を参照すれば、ゲスト OS 内と同様のフィルタリングを実現できる。

xFilter は、対象仮想マシンのゲスト OS のプロセス単位、ユーザ単位でパケットフィルタリングが可能である。つまり仮想マシンモニタの管理者が、仮想マシン内のゲスト OS で管理するのと同じ粒度でパケットフィルタリングができる。もちろん、通信制御を行うためにゲスト OS の IP アドレスやポート番号などの情報を参照することができる。

xFilter のように、仮想マシンを利用して Firewall を攻撃対象の仮想マシンからオフロードすることにより、Firewall 自体が攻撃者から攻撃できないようになる。また、仮想マシンの情報を利用して、仮想マシン内の OS に Firewall を置いたときと同様の粒度で、バケットを監視できる。

2.2.3　仮想マシンを利用したアクセス制御のオフロード

OS では様々なアクセス制御で、ウィルスやクラッカーからファイルを保護している。しかし、OS 自体に脆弱性があると、これらのアクセス制御が機能しなくなる。たとえば、クラッカーがシステムに侵入して、OS の脆弱性を利用して管理者権限を奪うと、すべてのファイルにアクセスを許してしまう。

SecureAccess[6] では、仮想マシンからアクセス制御を仮想マシンの外へオフロードして、仮想マシン内のファイルをアクセスを管理する。SecureAccess では仮想マシンを利用して作業 OS と認証 OS を動作させる。作業 OS のアクセス制御を認証 OS にオフロードする。このシステムでは、作業 OS が乗っ取られても、作業 OS 内の SecureAccess の管理下のファイルにはアクセスできない。

SecureAccess にはファイルアクセスに関して 2 つのポリシーを記述できる。1 つは空間に関するポリシーであり、もう 1 つは時間に関するポリシーである。空間に関するポリシーでは認証によってユーザーにアクセス許可されるファイルの範囲を限定することが可能である。時間に関するポリシーはアクセスの許可される期間を認証時に指定することができる。

SecureAccess のように、仮想マシンを利用してアクセス制御を攻撃対象の仮想マシンからオフロードすることにより、アクセス制御自体が攻撃者から攻撃できないようになる。たとえ仮想マシン内の OS がクラッカーに乗っ取られても、その OS 内にある SecureAccess の管理下にあるファイルにはアクセスできない。

2.3　仮想マシンを利用したセキュリティ機構のオフロードの問題点

今日、仮想マシンが普及してきたことで、１つの物理マシン上で複数の OS が動作できるようになった。仮想マシンを用いれば、ホスティングを行う場合、物理マシン単位ではなく、仮想マシン単位で利用者に貸し出しを行える。このような仮想マシン間では、パフォーマンスの分離は重要である。パフォーマンスの分離とは、複数の仮想マシンが動作しているときに、仮想マシンが管理者が指定した計算資源の使用量や比率を守っている環境である。仮想マシンへ計算資源を正確に分配できることは、仮想マシン単位で利用者に提供する管理者とって必要不可欠なことである。

しかし、セキュリティ機構を仮想マシンからオフロードしたすると、パフォーマンスの分離は難しい。たとえば、仮想化ソフトウェアとして Xen、セキュリティ機構として IDS を用いるとする。一般ドメインから IDS を特権ドメインへオフロードした場合、IDS は特権ドメインにカウントされ、その一般ドメインのために割り当て CPU 資源以上を使用されることになる。

具体的な例を示すと、図 2.5 のように IDS をオフロードした一般ドメインの CPU 割り当てを 40 %としたとき、特権ドメインで動く IDS の CPU の CPU 使用率の Y %はその一般ドメインではなく特権ドメインの使用量としてカウントされるので、オフロードした一般ドメインが使う CPU 使用率は 40 %を超える可能性がある。そうすると、他のドメインにも影響を与えるのでパフォーマンスの分離が実現できていない。

2.4　関連研究

2.4.1　XenMon

Virtual Machine の一つの利点は、一つの物理マシン上でマルチオペレーティングシステムを提供することである。各 Virtual Machine 上にオペレーティングシステムが動作しているが、各 Virtual Machine の環境が互いに分離していることが望ましい。たとえばある Vitual Machine 上の

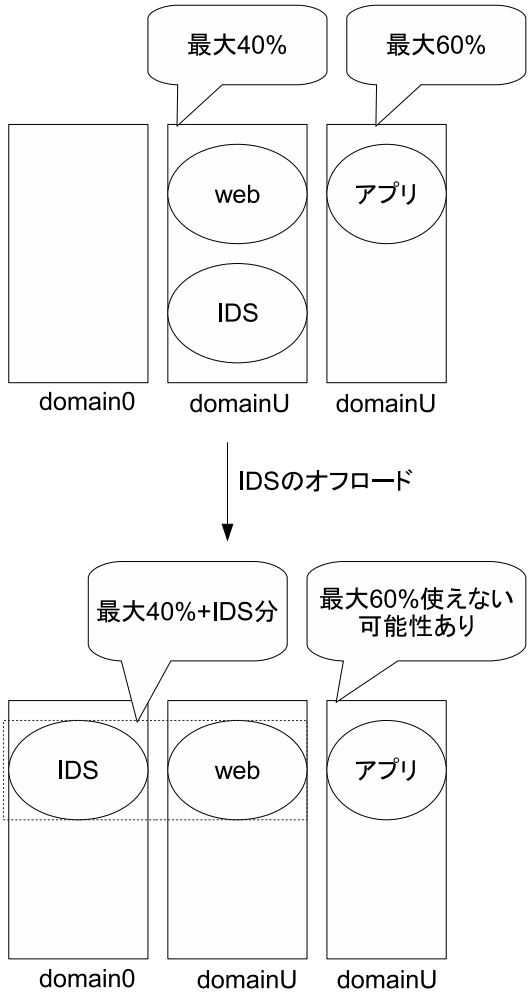
最大40%
最大60%
web
アプリ
IDS
domain0
domainU
domainU
IDSのオフロード
最大40%+IDS分
最大60%使えない
可能性あり
IDS
web
アプリ
domain0
domainU
domainU

図 2.5: 問題点

オペレーティングシステムがダウンしても、他の Virtual Machine に影響を与えてはならない。

XenMon[3] とは、Xen 上の Virtual Macnie ごとにパフォーマンスの分離を提供してくれるシステムである。パフォーマンスの分離とは、互いに決められた資源の量を守り他の Virtual Machine のパフォーマンスに影響を与えないことである。たとえば、Xen 上のあるドメインに決められた CPU 資源が与えられるとき、そのドメインのために動作しているドメイン 0 上のデバイスドライバの CPU 使用量はそのドメインの CPU 使用量としてカウントされず、ドメイン 0 の使用量とされる。このとき、あるドメインは決められた CPU 資源とドメイン 0 上のデバイスドライバ分の CPU 資源を使うので、決められた CPU 資源以上の CPU を利用する。よって、他のドメインに影響を与える可能性がある。XenMon は、このあるドメインのためのドメイン 0 上のデバイスドライバの CPU 使用量をあるドメインの CPU 使用量としてカウントすることで、パフォーマンスの分離を達成するシステムである。

XenMon の仕組みは以下である。Xen 上のネットワーク I/O の処理は、一般ドメインと特権ドメインの間はフロントエンドとバックエンドというスピリットデバイスドライバを通じて通信して、特権ドメインのデバイスドライバを使用する。このときのフロントエンドとバックエンドの通信はメモリページの交換を行うので、この交換回数を利用して特権ドメインのデバイスドライバの CPU 使用量を推測する。

XenMon の問題点は、XenMon はネットワーク I/O を対象としたパフォーマンスの分離を実現するシステムであり、ディスク I/O やメモリなどは対象としていない。また、Xen の現在のデフォルトのスケジューラーである Credit Scheduler は対象としていない。XenMon はネットワーク I/O を対象としたパフォーマンスの分離を実現するシステムであるが、本研究は一般ドメインからオフロードした IDS を対象としたパフォーマンスの分離を実現している。

2.4.2　Livewire

今日の IDS(侵入探知システム) のアーキテクチャーに対し、管理者に難しい選択が迫られる。ホスト型の IDS にすれば、監視対象ホストの具体的な活動が監視できるが攻撃者の目に触れやすくなる。ネットワーク型の IDS にすれば、攻撃されにくくなるがホスト内の具体的な活動は監視できなくなる。

Livewire[2] とは Virtual Machine Monitor を利用して、ホスト型の利点である監視対象ホストの内部状態を検知でき、またネットワーク型の利

点である攻撃されにくい、IDS のアーキテクチャーを可能にしたシステムである。

Livewire の仕組みは以下である。監視対象ホストを VM 上に載せることで IDS を監視対象ホストからオフロードし、攻撃者から目から触れにくい。また監視対象ホストの VM を調べることで監視対象ホストの内部状態が調べられる。Livewire は、VMWareWorkstation でこのシステムを構築している。

Livewire は監視対象ホストの内部状態を調べるために、Virtual Machine Monitor を修正しているので、若干のオーバーヘッドがある。本研究は Livewire で用いられているアーキテクチャーで IDS を対象としたパフォーマンスの分離を実現させるシステムである。

第3章　提案

3.1　セキュリティ機構のオフロードを考慮した仮想マシンのスケジューリング

前章の問題点のように対象の仮想マシンからセキュリティ機構をオフロードすると、オフロードしたセキュリティ機構の CPU 使用量は対象の仮想マシンにカウントされないので、パフォーマンスの分離が実現できていない。よって本システムは、オフロードしたセキュリティ機構の CPU 使用量をオフロード元の対象の仮想マシンにカウントしたように見せるように仮想マシンのスケジューラを修正した。このシステムにより、監視対象ホストはあらかじめ決められた計算資源の比率を守り、仮想マシン間のパフォーマンスの分離が実現できる。仮想化ソフトウェアとして Xen を、セキュリティ機構として snort(IDS) を使用している。

具体的には特権ドメインへオフロードした IDS の CPU 使用率を計り、それに対応してオフロード元ｂの一般ドメインに配布される CPU 利用時間を減らすことでパフォーマンスの分離を実現している。たとえば、図 3.1 のように監視対象のドメインの最大 CPU 使用率を 40 %とする。このとき本システムを使用しない場合、特権ドメインの IDS による CPU 使用率 X %は特権ドメインにカウントされるので、監視対象のドメインは 40 ＋ X %の CPU を使用することができてしまう。しかし、本システムを用いる場合、特権ドメイン上の IDS と監視対象のドメインの CPU 利用率を合わせて 40 %となるようにスケジューリングされるようになる。

3.2　特権ドメインから Xen のドメインスケジューリングを操作

Xen が特権ドメインへ提供しているドメイン管理用のインターフェイスを追加して、特権ドメインから指定したドメインのスケジューリングを操作し、指定したドメインの物理 CPU の利用時間を指定した分だけ減らすことができるようにした。

一般ドメインを対象とした IDS が特権ドメイン上で動くと、その対象となる一般ドメインは DEBT 状態という新しく追加した状態になる。そ

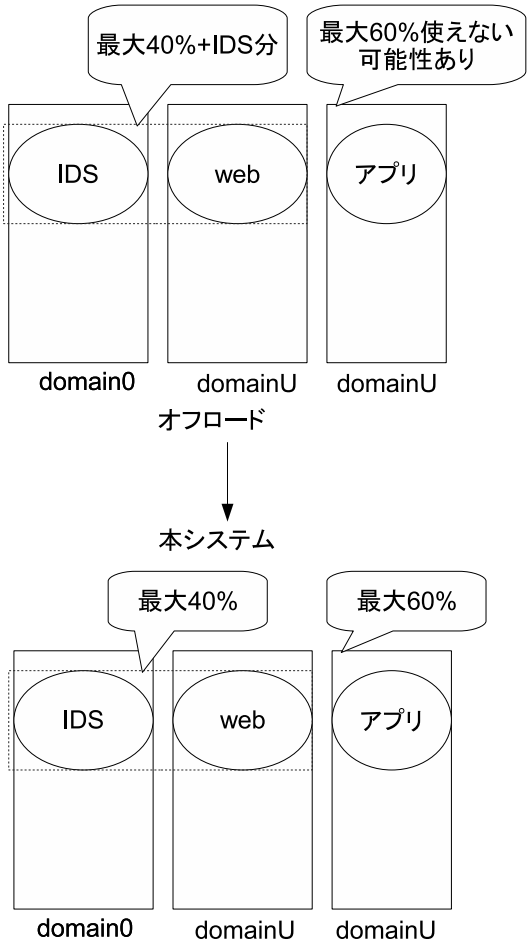
最大40%+IDS分
最大60%使えない
可能性あり
IDS
web
アプリ
domain0
domainU
domainU
オフロード
本システム
最大40%
最大60%
IDS
web
アプリ
domain0
domainU
domainU

図 3.1: 本システム

の状態になると、対象となる一般ドメインはスケジューリングされるときに、配布される物理 CPU の利用時間が使用している IDS の CPU 使用率の応じて減少される。

本システムのために変更した点は以下である。

- ドメインに新しい状態 (DEBT 状態) を追加
- ドメインを表す構造体に新しいメンバー (debt) を追加
- Xen から特権ドメインに提供されているインターフェイスを追加
- ドメインスケジューリングを変更、新しい状態を認識してスケジューリングを変化

基本的な動作をまとめると以下のようになる。

1. 指定されたプロセス番号から特権ドメイン上の IDS の CPU 使用率を計算
2. Xen の管理用インタフェースから指定したドメインを DEBT 状態へ、また 1 の CPU 使用率をドメインの debt メンバーに保存
3. スケジューリングが DEBT 状態を認識して、そのドメインの CPU 利用時間を debt メンバーに応じて減少させる
4. 1 秒ごとに特権ドメイン上の IDS の CPU 使用率に対応して debt メンバーを変化させる

3.3　管理用インターフェイス

Xen は特権ドメインにドメインの管理用のライブラリを提供していて、このライブラリの中でハイパーコールを使っている。よって特権ドメインはこのライブラリを用いて、一般ドメインの作成、削除、また変更などの動作をすることができる。

例えばこの管理用ライブラリにある

- xc_domain_create(xc_handle, ssidref, handle, flags, pdomid)

は内部で以下のように関数を呼ぶ

1. xc_domain_create
2. do_domctl(xc_handle, &domctrl)
3. do_xen_hypercall(xc_handle, *hypercall)

4. do_privcmd(xc_handle, cmd, data)

5. ioctl(xc_handle, cmd, data)

引数に共通している xc_handle は/proc/xen/privcmd へファイルディスクリプタである。ここにアクセスすることでハイパーコールを呼ぶことができる。

新しく追加した特権ドメインの管理用インターフェイスは

- xc_sched_credit_domain_putdebt(fd, domid, *sdom)
- xc_sched_credit_domain_repaydebt(fd, domid, *sdom)

である。具体的な関数の動作は次の章に説明する以下は管理用ドメインのオペレーティングシステムの/usr/include/xenctrl.h の中で宣言されている。よって一般ドメインからは使用できない。この 2 つの新しいインターフェイスを通して管理ドメインはドメインを新しく作った状態である DEBT へ変化させることができる。

3.4　本システムの使用例

まず、特権ドメイン上の IDS のプロセス ID を取得する。ps コマンドなどを用いれば簡単にプロセス ID はわかる。監視対象ドメインのドメイン ID を Xen が提供する xm 管理コマンドで取得する。取得した pid とドメイン ID を引数にして以下のコマンドを打つ。

- ./debt pid domainID

終了するときは、上記のプログラムのプロセスを停止させ、以下のコマンドを打つ。そうすると本システムが動作する以前の状態に戻る。

- ./repay domainID

3.5　利点

本システムの利点をまとめる。

IDS のオフロード時においてもパフォーマンスの分離を実現

本システムでは、Xen 上の一般ドメインからセキュリティ機構のオフロードをしても、オフロード前と同じ仮想マシン間のパフォーマンス関係を実現できる。本システムを用いないと、特権ドメインにオフロードしたセキュリティ機構の CPU の利用分は、特権ドメインにカウントされ、他の一般ドメインのパフォーマンスに影響を与える可能性がある。

IDS 以外のオフロードでも対応

本システムではオフロードしたプロセスを指定して動作するので、IDS 以外でもオフロードしたアプリケーションは本システムの対象となることができる。

3.6　欠点

マルチドメインに対して非対応

本システムを Xen 上の複数のドメインに対して用いるとき、各ドメインに対して IDS と本システムを動かさなくてはならない。今後、特権ドメイン上の一つの IDS で複数のドメインを監視できるように修正する必要がある。

マルチプロセッサに対応していない

本システムはシングルコア、つまり物理 CPU が一つのときという条件で構成しているので、物理 CPU が複数への対応は今後の課題である。

特権ドメイン上のデバイスドライバは考慮していない

ドメインUがネットワーク I/O 処理を行うとき、Xen のデバイスドライバの構成上、特権ドメインの本物のデバイスドライバを使用する。しかし、このデバイスドライバは特権ドメイン上で動いているので、ネットワーク処理を行っているドメインUではなく、特権ドメインの CPU 使用量としてカウントされる。この状態だと、ネットワーク処理を行っているドメインUが想定している以上の CPU を使用する場合があり、ドメイン間のパフォーマンスの分離が実現できない。

これは関連研究の XenMon で研究されている。本システムもより良いドメイン間のパフォーマンスの分離するには、このデバイスドライバの点も考慮する必要がある。

第4章　実装

4.1　ドメインと仮想CPU

Xen は、各ドメインを domain 構造体、各仮想 CPU を vcpu 構造体で表している。また domain 構造体は自分が持つ仮想 CPU のポインタを配列で管理しているし、仮想 CPU も自分が所属するドメインのポインタを保持している。図 4.1 は domain 構造体と vcpu 構造体である。

```
1  /* 省略あり */
2  struct domain
3  {
4      /* ドメインID */
5      domid_t            domain_id;
6      /* 共有データページへのポインタ */
7      shared_info_t    *shared_info;
8      /* スケジューリングのデータ、
9      デフォルトはcsched_dom構造体へのポインタ */
10     void               *sched_priv;
11     /* 所持しているVCPUへのポインタの配列 */
12     struct vcpu *vcpu[MAX_VIRT_CPUS];
13 };
14 struct vcpu
15 {
16     /* どの物理CPUに束縛されているか */
17     int                 processor;
18     /* このvcpuを所持しているドメインへのポインタ */
19     struct domain    *domain;
20     /*ドメインスケジューラのデータ、
21     デフォルトはcsched_vcpu構造体へのポインタ */
22     void               *sched_priv;
23     /* アーキテクチャー依存 */
24     struct arch_vcpu arch;
25 };
```

図 4.1: domain 構造体と vcpu 構造体

Xen 上で各ドメインをスケジューリングするのが、ドメインスケジューラである。図 4.2 のようにドメインスケジューラはそれぞれ物理 CPU ごとに独立して動作し、仮想 CPU 単位でスケジューリングを行う。仮想 CPU が各物理 CPU ごとに用意された RUN キューに登録される。RUN キュー

に登録された仮想 CPU が物理 CPU に対応づけられたとき、その仮想 CPU を含むドメイン内のオペレーティングシステムが実行されます。

また各物理 CPU はハイパーバイザースタックというスタックを持っていて、ハイパーコール、割り込み、例外などが発生したとき、実行していたドメインのコンテキストを退避して、Xen に実行権を渡します。Xen の実行が終了したとき退避していたコンテキストを復帰させ、ドメインの処理を再開します。

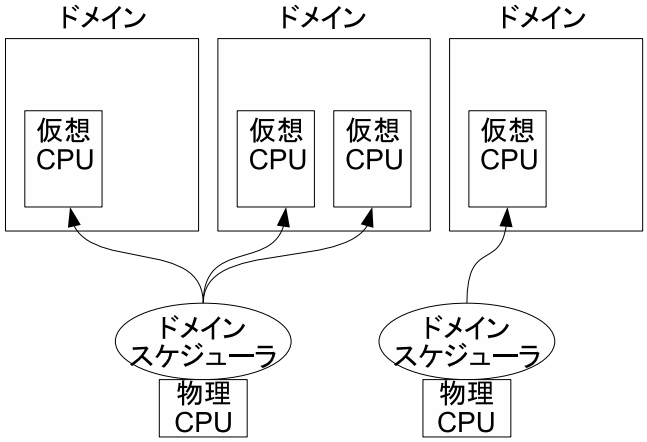

図 4.2: ドメインスケジューラ

4.2　クレジットスケジューラ

クレジットスケジューラは、現バージョンの Xen のデフォルトのドメインスケジューラである。基本的な特徴は以下である。

- 各ドメインは weight と cap が指定されている。weight は他のドメインの weight と比較する相対的な値であり、cap は一つの物理 CPU の最大利用率を 100 とした絶対的な値である。デフォルトでは weight が 256、cap が 0。cap が 0 の場合は制限なしを意味する。
- 定期的に各ドメインには credits が weight に応じて振り分けられる。そのドメイン内で振り分けられた credits が、そのドメインが所持している仮想 CPU に分配される。
- 定期的に物理 CPU 上で実行している vcpu の credits を減らす。

- 仮想 CPU には 4 状態あり、BOOST、UNDER、OVER、IDLE である。BOOST は I/O 待ち後などの wakeup したとき、UNDER は credits がプラスとき、OVER は credits がマイナスのとき、IDLE はドメインがアイドル状態のときである。

またクレジットスケジューラの情報として、図 4.3 のように domain 構造体と vcpu 構造体にはそれぞれ csched_dom 構造体、csched_vcpu 構造体が対応付けられる。

```
1  struct csched_dom {
2      struct list_head active_vcpu;
3      struct list_head  active_sdom_elem;
4       /* 対応する domain構造体のポインタ */
5      struct  domain *dom;
6      uint16_t active_vcpu_count;
7      /* weight */
8      uint16_t weight;
9      /* cap */
10     uint16_t cap;
11 }
12  struct csched_vcpu {
13     struct list_head runq_elem;
14     struct list_head active_vcpu_elem;
15     /* 対応するドメインの csched_dom構造体へのポインタ */
16      struct csched_dom *sdom;
17     /* 対応する vcpu構造体へのポインタ */
18     struct vcpu *vcpu; atomic_t credit;
19     uint16_t flags; int16_t pri;
20 #ifdef CSCHED_STATS
21     struct {
22         int credit_last;
23         uint32_t credit_incr;
24         uint32_t state_active;
25         uint32_t state_idle;
26         uint32_t migrate_q;
27         uint32_t migrate_r;
28     } stats;
29 #endif
30 };
```

図 4.3: csched_dom 構造体と csched_vcpu 構造体

tick ごとの credis の増減について

各 vcpu が持つ credit がどのように増減するか説明すると、tick(10ms) ごとに csched_tick(void *cpu) が呼ばれる。この csched_tick() は基本的には以下の二つの動作をする。

- 毎回 csched_vpuc_acct(int cpu) を呼び、この関数の中で現在実行されている vcpu の credit から一定の値 (100) 引く
- また三回に一回 csched_acct(void) を呼び、各 vcpu の credit を増やす

csched_acct(void) は動作は以下であり、図 4.4 は本体である。

1. このドメインスケジューラが管理している各ドメインに対し、ドメインの weight に応じて credit を振り分ける
2. 1 でもらった credit をそのドメイン内の各 vcpu に均等に分ける
3. 各 vcpu は 2 でもらった credit を現在の credit に足す
4. 3 で足した credit の値が負なら OVER 状態へ、正ならば UNDER 状態へ

4.3　本システムの実装

4.3.1　実装の方針

一般ドメインで動作している IDS を特権ドメインへオフロードする。その一般ドメインを対象ドメインと呼ぶことにする。

1. 特権ドメインへオフロードしている対象ドメインは DEBT 状態へ
2. その特権ドメインで動く IDS の CPU 使用率を計る
3. ２で計った値を対象ドメインに保存する
4. ドメインスケジューラは DEBT 状態のドメインに対して 3 の値を考慮して credit を分配する

上記の動作を実現するために Xen に以下の機能を追加した。

- IDS を特権ドメインへオフロードした一般ドメインを表す状態である DEBT 状態を追加
- 特権ドメインで動いている、一般ドメインからオフロードした IDS の CPU 使用率を表す debt メンバーをその一般ドメインに追加
- 指定したドメインを DEBT 状態にして、そのドメインの debt メンバーを更新する特権ドメイン用のインタフェースを追加

```
1  /* 省略部分あり */
2  static void csched_acct(void)
3  {
4  /* 宣言 */
5
6   weight_total = csched_priv.weight; /* ドメインの総weight数 */
7   credit_total = csched_priv.credit; /* creditの合計 */
8
9   /* 各ドメインに対して  */
10  list_for_each_safe(iter_sdom, next_sdom,
11                     &csched_priv.active_sdom)
12  {
13   /* 一つのドメイン */
14   sdom = list_entry(iter_sdom, struct csched_dom,
15                     active_sdom_elem);
16
17   /* そのドメインのweight/すべてのドメインの総weight
18      に応じてcreditをそのドメインへ分配 */
19   credit_fair = ((credit_total * sdom->weight)+(weight_total-1)
20                  ) / weight_total;
21
22   /* そのドメインに配されたcreditをvcpuの数で分ける */
23   credit_fair = (credit_fair + (sdom->active_vcpu_count - 1)
24                  ) / sdom->active_vcpu_count;
25
26   /* そのドメインのvcpuに対して */
27   list_for_each_safe(iter_vcpu, next_vcpu, &sdom->active_vcpu)
28   {
29    /* 一つのvcpu */
30    svc = list_entry(iter_vcpu, struct csched_vcpu,
31                     active_vcpu_elem);
32
33    /* そのvcpuのcreditを更新 */
34    atomic_add(credit_fair, &svc->credit);
35
36    /* そのvcpuの更新後のcredit */
37    credit = atomic_read(&svc->credit);
38
39    if(credit < 0){ /* creditが負  */
40
41     /* OVER状態へ */
42     svc->pri = CSCHED_PRI_TS_OVER;
43
44    }else{  /* creditが正  */
45
46     /* UNDER状態へ */
47     svc->pri = CSCHED_PRI_TS_UNDER;
48
49    }
50   }
51  }
52 }
```

図 4.4: csched_acct 関数

- デフォルトのドメインスケジューラを DEBT 状態を認識するように修正

なお、debt はオフロードした IDS の CPU 使用率のために使用するが、この CPU 使用率は Xen 全体での CPU 使用率である。

4.3.2　DEBT 状態と debt メンバー

一般ドメインが特権ドメインへオフロードしているかどうかを区別したい。またオフロードしているならば、オフロードした IDS がどのくらい CPU を使用しているかを対象ドメインの情報として保持させたい。前者は DEBT 状態を追加することで、後者は debt メンバーを追加することで解決した。

オフロードされた対象ドメインを表す状態 DEBT は図 4.5 に定義してある。CSCHED_DEBT_YES ならば、そのドメインは特権ドメインへオフロードしている。

```
1  #define CSCHED_DEBT_NO      0
2  #define CSCHED_DEBT_YES     1
```

図 4.5: DEBT 状態の定義

そして csched_dom 構造体も図 4.6 に変更した。このように csched_dom

```
1  struct csched_dom {
2      struct list_head active_vcpu;
3      struct list_head active_sdom_elem;
4      struct domain *dom;
5      uint16_t active_vcpu_count;
6      uint16_t weight;
7      uint16_t cap;
8      /* 特権ドメインへオフロードしている状態を表す */
9      uint16_t state;
10     /* 特権ドメインで動作しているIDSのCPU使用量 */
11     uint16_t debt;
12 }
```

図 4.6: csched_dom 構造体を変更

構造体を変更すれば、state メンバーが CSCHED_DEBT_YES ならば、そのドメインは特権ドメインへ IDS をオフロードしているわかる。またオフロードした IDS がどのくらい CPU を使用しているかは debt メンバーからわかる。

4.3.3　管理用インターフェイスの追加

追加した特権ドメイン用のインターフェイスは図 4.7 の 2 つである。

```
1  int
2  xc_sched_credit_domain_putdebt(
3      int xc_handle,
4      uint32_t domid,
5      struct xen_domctl_sched_credit *sdom);
6
7  int
8  xc_sched_credit_domain_repaydebt(
9      int xc_handle,
10     uint32_t domid,
11     struct xen_domctl_sched_credit *sdom);
```

図 4.7: 新しいインタフェース

xc_sched_credit_domain_putdebt() 関数

引数は 3 つ、共有メモリページ/proc/xen/privcmd へのファイルディスクリプタとドメイン ID と状態を変更するための情報である。状態を変更する情報とは、DEBT 状態にして debt メンバーを設定するための値が入っている。この関数を呼ぶと指定されたドメインは DEBT 状態となり、debt メンバーが設定される。

xc_sched_credit_domain_repaydebt() 関数

引数は 3 つ、共有メモリページ/proc/xen/privcmd へのファイルディスクリプタとドメイン ID と状態を変更するための情報である。状態を変更する情報とは、DEBT 状態から通常の状態へ変化させる値が入っている。この関数を呼ぶと指定されたドメインは DEBT 状態から通常の状態になる。

ハイパーコールへ

追加したインターフェースからどのようにハイパーコールが呼ばれ、csched_dom 構造体のメンバーを更新するかを見てみる。xc_sched_credit_domain_putdebt() 関数から以下のような関数が呼ばれる。なお、返り値と引数は省略している。

1. xc_sched_credit_domain_putdebt()

2. do_domctl()
3. do_xen_hypercall()
4. do_privcmd()
5. ioctl()

2 の do_domctl の中で、HYPERVISOR_domctl に対応するハイパーコールを呼ぶように引数に設定される。すべての関数の引数に/proc/xen/privcmd へのファイルディスクリプタがあり、そこに 5 の ioctl() でアクセスすると、引き数の情報に対応したパイパーコールが呼ばれる。

ハイパーコールの中へ

xc_sched_credit_domain_putdebt() から HYPERVISOR_domctl に対応するハイパーコールである do_domctl() が呼ばれる。なお、これは前に書いた do_domctl() とは別の関数である。以下が do_domctl ハイパーコールからの流れである。

1. do_domctl()
2. sched_adjust()
3. csched_dom_cntl()

1 の関数中で、引数の情報から 2 の関数が呼ばれる。2 の sched_adjust は関数ポインタであり、デフォルトのドメインスケジューラはクレジットスケジューラなので 3 の関数が呼ばれることになる。3 の csched_dom_cntl() 関数の中で図 4.8 のように csched_dom 構造体のメンバーである state と debt を設定している。

4.3.4　ドメインスケジューラの修正

ドメインは DEBT 状態となることができるようになったので、あとはドメインスケジューラにその状態を認識させるために csched_acct() 関数を変更した。この関数の中では、cap が設定してあれば cap で計算した credit と weight で計算した credit の 2 つ方法で credit を求め、小さいほうをドメインに分配する。cap が設定してなければ weight で計算した credit をドメインに分配する。

もしドメインが DEBT 状態ならば、cap と weight で 2 つの計算方法をともに変えることで、ドメインに配布する credit を減少させる。cap を用

```
1  static int
2  csched_dom_cntl(
3      struct domain *d,
4      struct xen_domctl_scheduler_op *op)
5  {
6      /* 省略 */
7      else if (op->cmd == XEN_DOMCTL_SCHEDOP_putdebt){
8          /* DEBT状態へ */
9          sdom->state = CSCHED_DEBT_YES;
10         /* ここでsdom->debtを引数から設定 */
11     }
12     else if (op->cmd == XEN_DOMCTL_SCHEDOP_repaydebt){
13         /* DEBT状態を解除 */
14         sdom->state = CSCHED_DEBT_NO;
15     }
16     /* 省略 */
17     return 0;
18 }
```

図 4.8: csched_dom_cntl() 関数

いての計算をどのように変更し、また weight を用いての計算をどのように変更するかを説明する。ただし、変更するのはドメインの vcpu に分ける前、つまりドメインに credit を分配する計算を変更する、vcpu に分ける計算は変更する必要はない。

cap を用いての計算

cap が設定してあるとする、つまり cap が 0 でない。このときは、通常図 4.9 のように計算している。cap とは物理 CPU1 つの最大利用率を 100 としたときの絶対的な値である。計算方法は総 credit から cap %分だけドメインに与えるという単純なものである。

```
1  if ( sdom->cap != 0)
2  {
3    /* 総creditからsdom->cap%分を計算   */
4    credit_cap = ((sdom->cap * CSCHED_CREDITS_PER_ACCT)+99)/100;
5    if ( credit_cap < credit_peak )
6      credit_peak = credit_cap;
7
8    /* 省略 */
9  }
```

図 4.9: cap を用いての credit の計算

ドメインが DEBT 状態であるときをを考慮するために図 4.10 のように変更した。この計算方法はドメインの debt %分だけドメインの cap を小さく見せ、credit を計算させるというものである。もとの cap の値は変わらない。

```
1  if ( sdom->cap != 0U )
2  {
3    if(sdom->state == CSCHED_DEBT_YES){ /* DEBT状態 */
4      /* capをdebt分、小さく見せる */
5      temp = sdom->cap - sdom->debt;
6      /* 小さく見せてcreditを計算 */
7      credit_cap = ((temp * CSCHED_CREDITS_PER_ACCT) + 99)/100;
8    }else{ /* デフォルトの計算方法 */
9      credit_cap = ((sdom->cap*CSCHED_CREDITS_PER_ACCT)+99)/100;
10   }
11
12   if ( credit_cap < credit_peak )
13     credit_peak = credit_cap;
14
15   /* 省略 */
16 }
```

図 4.10: DEBT 状態を考慮した、cap を用いての credit の計算

weight を用いての計算

通常は図 4.11 のように weight を用いてドメインに credit を分配する。weight_total は各ドメインの weight の和なので、weight が相対的な値で他のドメインと比較して credit をドメインに分配することがわかる。

```
1 credit_fair=((credit_total * sdom->weight)+(weight_total - 1)
2             ) / weight_total;
```

図 4.11: weight を用いての計算

ドメインが DEBT 状態であるときを考慮するために図 4.12 のように変更した。この計算方法は各ドメインの weight の和の debt %を計算して、その値を対象ドメインの weight から引き、引いた値で credit を計算させるものである。もとの weight は変わらない。

```
1  if(sdom->state == CSCHED_DEBT_YES){ /* DEBT状態ならば */
2    /* weightの総和のdebt%分を計算 */
3    temp = (sdom->debt * weight_total + 99) / 100;
4    /* 上記の値を対象ドメインのweightから引く */
5    credit_fair=((credit_total*(sdom->weight-temp))+
6                 (weight_total-1)) / weight_total;
7  }else{ /* 通常 */
8    credit_fair=(( credit_total * sdom->weight)+(weight_total-1)
9                ) / weight_total;
10 }
```

図 4.12: DEBT 状態を考慮した、weight を用いての計算

DEBT 状態なら対象ドメインへの credit 分配は減少

以上より、もしドメインが DEBT 状態ならば、そのドメインの debt メンバーに応じて分配される credit が減少する。credit が減少すれば、そのドメインの CPU 利用時間は少なくなり、パフォーマンスに影響がでる。理想は特権ドメインにオフロードした IDS の CPU 利用分だけ、対象ドメインの CPU 利用分を減らすことである。

第5章　実験

実験では、3つのアーキテクチャーを比較している (図 5.1)。その3つとはオフロードなし、オフロードあり、オフロードかつ本システムである。オフロードなしとはIDSを対象ドメインの中に置くアーキテクチャーのことであり、オフロードありとはIDSを対象ドメインから特権ドメインへオフロードしたアーキテクチャーである、本システムは動作していない。もうひとつは、オフロードありの状態で、かつ本システムを動作させている状態である。

5.1　実験１

目的

この実験では、オフロードしない場合、オフロードした場合、オフロードかつ本システムを使用した場合のオフロード元のドメインのパフォーマンスがどのように変化するかを調べる。またオフロードあり、かつ本システムを動作させるとオフロードなしの状態のときのパフォーマンスと同じようになるかどうかも調べる。

設定と環境

図 5.2 ドメイン U をウェブサーバとして動作させ、外部のマシンからhttperf[4] を使用して負荷をかける。このとき図 5.1 の 3 つのパフォーマンスを比較する。

具体的な設定は以下である。

- ドメイン U は一つ、ウェブサーバとして動作 (apache を使用)
- IDS は snort を使用
- ドメイン U には cap を 40 と設定、つまり最大 CPU 使用率は 40 %
- httperf で 2000requests/second、10000connections を指定してウェブサーバへ負荷 (Web サーバに/../../etc/passwd を要求)

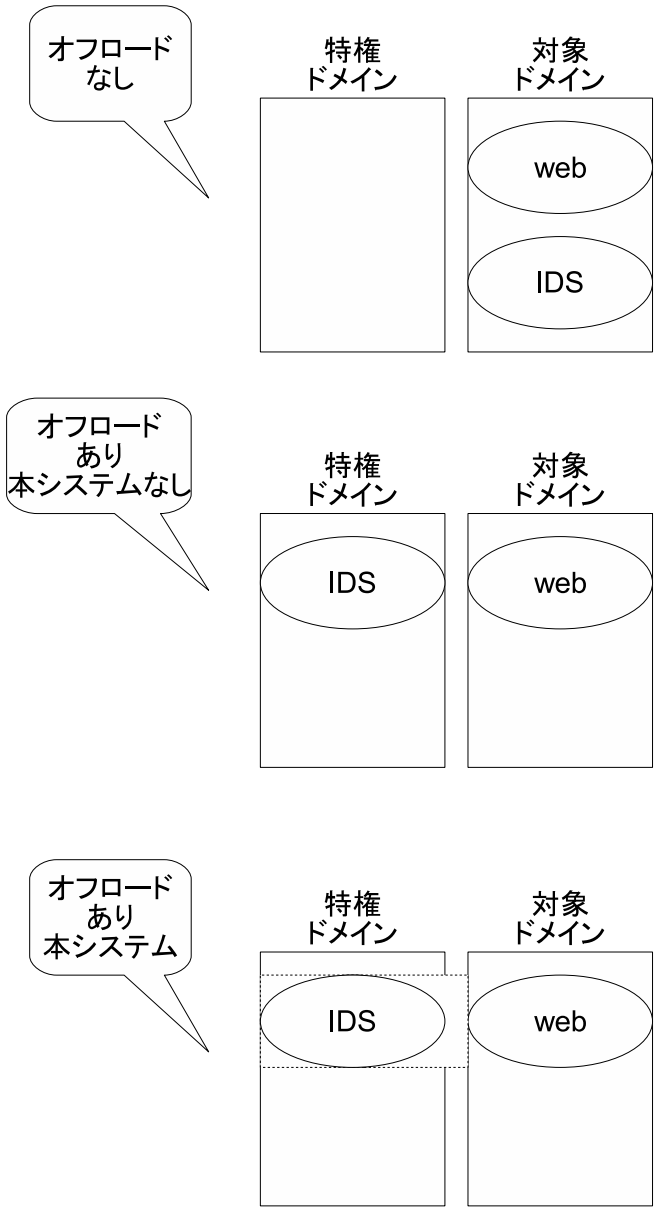

図 5.1: 本システムとの比較

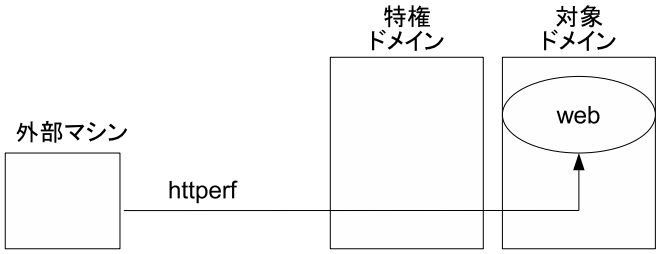

図 5.2: 設定

- 図 5.1 の 3 つ場合のウェブサーバのパフォーマンスを比較

図 5.1 の 3 つで動作する snort は同じ version で、同じルールが適用してある。なお、apache の version は 2.2.8 で、snort の version は 2.8.3.1 である。また httperf と snort については後述する。

実験環境は以下である。

- Xen の version は xen-3.3.0(x86_64)
- ドメイン 0 は linux2.6.18.8-xen(x86_64)
- ドメイン U は linux2.6.16.33-xenU(x86_64)
- CPU は AMD Athlon(tm) 64 Processor 3500+
- 搭載メモリは 2Gbytes

結果

実験結果は図 5.3 のようになっている。グラフは、左からオフロードしなかった場合、オフロードした場合、オフロードかつ本システムの場合である。オフロードした場合のパフォーマンスが一番よく、オフロードしなかった場合とオフロードかつ本システムの場合はほぼ同じであった。

考察

ウェブサーバとして動作させているドメインを対象ドメインを呼ぶことにする。以下に三つ場合の対象ドメインで動作する実験のためのアプリケーションをまとめる。

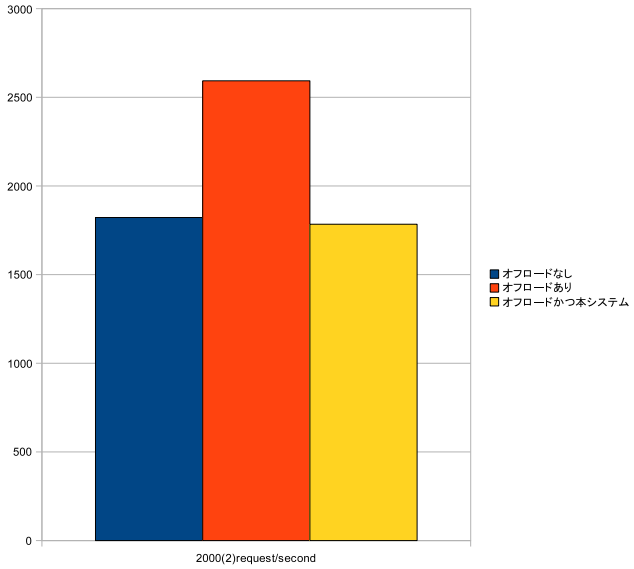
3000
2500
2000
1500
1000
500
0
2000(2)request/second
オフロードなし
オフロードあり
オフロードかつ本システム

図 5.3: 実験

- オフロードしない場合…apache(ウェブサーバ) + snort
- オフロードした場合…apache(ウェブサーバ) のみ
- オフロードかつ本システム場合…apache(ウェブサーバ) のみ

まず対象ドメインが snort をオフロードしなかった場合とオフロードした場合を比べると、オフロードした場合のほうがウェブサーバのパフォーマンスがよい。これは、オフロードした場合はウェブサーバだけがほぼこの対象ドメインに分配された CPU 資源を使うことができ、一方オフロードしなかった場合はウェブサーバと snort が分配された CPU 資源を分け合うことが理由と考えられる。この対象ドメインには cap40、つまり最大 CPU 使用率 40 %の制限がかかっている。

また、オフロードかつ本システムを用いた場合のウェブサーバのパフォーマンスは、ほぼオフロードしなかった場合と同じである。つまり、本システムを用いれば、特権ドメインにオフロードした snort は対象ドメインにカウントされているように見える。

5.2　snort について

snort(http://www.snort.org) はオープンソースの侵入探知システム (IDS) である。snort の特徴は以下である。

- フリーソフトである
- パケットのスニファやロガーとしても使用できる
- 豊富なプリプロセッサが用意されているため、柔軟な解析ができる
- さまざまな形式で、アラートを出力できる
- Linux,Windows,UNIX などで動作できる
- さまざまなプロトコルに対応してる

以下は使用例である。

- snort -D -u snort -g snort -c /etc/snort/snort.conf

-D で snort をデーモンと機動し、-u,-g でユーザ、グループは snort で実行、-c で設定ファイルの Path を指定する。設定ファイルには rule ファイルがインクルードしてあり、rule ファイルには監視するルールやポリシーを記載する。

5.3　httperfについて

httperf はウェブサーバのパフォーマンスを測定するツールである。１秒当たりのリクエスト数や総コネクション数を指定して使用することで、ウェブサーバに一定の負荷を与えることができる。以下のようにして使用する。

- httperf –server www –port 80 –uri /index.html
  –rate 150 –num-conn 27000
  –num-call 1 –timeout 5

このコマンドは、www というホストの 80 ポートにアクセスして index.html ファイルを要求する、また総コネクション数が 27000 に達するまで１秒当たり 150 リクエストで一定の負荷を与えるという指示のコマンドである。
本実験は以下のコマンドで行っている

- httperf –server 192.168.16.1 –port 80 –uri /../../etc/passwd
  –rate 2000 –num-conn 10000
  –num-call 2 –timeout 5

このコマンドは、192.168.16.1 というホストの 80 ポートにアクセスして /etc/passwd ファイルを要求する、また総コネクション数が 10000 に達するまで１秒当たり 2000 リクエスト (x2) で一定の負荷を与えるという指示のコマンドである。１つのコネクションに対して、2 回リクエストを送る。

第6章　まとめ

パフォーマンスの分離の実現

仮想マシンを利用してセキュリティ機構を仮想マシンの外へオフロードすることで、セキュリティ機構が攻撃されない構成ができる。このとき、オフロード元の仮想マシンを、対象仮想マシンと呼ぶことにする。しかし、対象仮想マシンからオフロードしたセキュリティ機構は対象仮想マシンの CPU 使用量としてカウントされないので、管理者が指定した比率以上の CPU が対象仮想マシンのために使用される可能性がある。本研究では、仮想マシンをセキュリティ機構をオフロードしたアーキテクチャーに対し、パフォーマンスの分離を実現できるシステムを考案した。仮想化ソフトウェアとして Xen、セキュリティ機構として snort を使用している。

特権ドメインへオフロードした IDS の CPU 使用率を計測して、その値に応じてオフロード元のドメインの割り当て CPU 時間を減少させるようにドメインスケジューラが修正した。ドメインスケジューラにオフロード元のドメインとオフロードした IDS の CPU 使用率を知らせるため、ドメインに新しい DEBT 状態と debt 変数を与えた。

本システムを使用すれば、Xen を用いたセキュリティ機構を特権ドメインへオフロードしても、オフロード元のドメインはオフロードしない場合と同じパフォーマンスを実現できることが実験から確かめられた。この状態であれば、パフォーマンスの分離が守られ、他の一般ドメインがセキュリティ機構のオフロードによるパフォーマンスの影響を受けることはない。

今後の課題

現在のシステムでは、特権ドメイン上で動作する IDS のプロセスを利用しているので、複数のドメインを対象とするには各ドメインに対しそれぞれ IDS と本システムを動かす必要がある。つまり、Xen の複数のドメインを対象とした特権ドメイン上の IDS を作成し、それに対応したシステムに修正する必要がある。

Xen 上でドメイン間のパフォーマンスの分離を実現するには、ほかにも特権ドメイン上で動作するデバイドライバなども考慮しなければならない。一般ドメインは特権ドメインのデバイスドライバを利用する構成になっている。このデバイスドライバの利用分もその一般ドメインに反映させれば、より良いパフォーマンスの分離が実現できる。

参考文献

[1] 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, *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, pp. 164–177 (2003).

[2] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *In Proc. Network and Distributed Systems Security Symposium*, pp. 191–206 (2003).

[3] Gupta, D., Cherkasova, L., Garder, R. and Vahdat, A.: Enforcing Performance Isolation Across Virtual Machines in Xen (2006).

[4] 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).

[5] 安積武志: Xen によるゲスト OS の監視に基ずくパケットフィルタリング (2008).

[6] 滝澤祐二: 仮想計算機を用いて OS を介さずに行う安全なファイルアクセス制御 (2008).

付 録A　システムの実行方法

特権ドメインから以下のようにして利用する。

1. xm list
2. ps aux
3. ./yen domainID PID

まず、xm コマンドで、オフロード元のドメイン ID を取得する (1)。次に、ps コマンドでオフロードしたセキュリティ機構である snort のプロセス ID を取得する (2)。最後に、取得したドメイン ID と snort のプロセス ID を引数として yen を実行する (3)。

- ./repay domainID

本システムを終了する場合は、yen のプロセスを停止させ、上記ののコマンドを利用する。