情報処理学会論文誌 Vol.59 No.3 1016–1025 (Mar. 2018)

リクエスト単位で仮想的にハードウェアリソースを分離するWebサーバのリソース制御アーキテクチャ

松本 亮介[1,a)] 栗林 健太郎[1] 岡部 寿男[2]

受付日 2017年6月26日, 採録日 2017年12月8日

概要：大規模 Web サービスの普及にともない，Web アプリケーションはますます高度化してきている．さらに，スマートフォンの普及によって，Web サーバへのアクセス数は日々増加してきおり，Web サーバの運用・管理が非常に重要になってきている．Web サーバソフトウェアにおいて，CPU リソースを多く消費するリクエストがあった場合，その処理を柔軟に制御する必要がある．しかし，既存の Web サーバのリソース制御は，リクエストの同時接続数や単位時間あたりのリクエスト数が指定の閾値を超えた場合や，管理者があらかじめ設定した CPU 使用時間やメモリ使用量の閾値を超えた場合にリクエストを強制的に切断，あるいは，拒否するようなアーキテクチャになっている．そのため，処理を継続しながらもリソースを制御することができない．そのような状況を解決するためには，リクエスト処理を一時的に限られたリソースの範囲内に分離すれば，複数のリクエスト処理の間でリソース占有の影響を互いに受けることが原理的に生じない．そこで，本論文では，同一のサーバプロセス上で，リクエスト単位で任意のリソース分離が可能なリソース制御アーキテクチャを提案する．このアーキテクチャによって，各リクエスト処理は一時的に仮想的なリソースを割り当てられ，そのリソースの範囲内で処理が行われる．そのため，特定のリクエスト処理のリソース使用超過による影響を受けにくくなる．さらに，大量にリソースを消費するようなリクエストであっても，リクエストを拒否・切断することなく，割り当てたリソースの範囲内で処理を継続させることが可能となる．

キーワード：Web サーバ，運用技術，リソース制御，Apache，cgroup

Resource Control Architecture for a Web Server Separating Computer Resources Virtually at Each HTTP Request

Ryosuke Matsumoto[1,a)] Kentaro Kuribayashi[1] Yasuo Okabe[2]

Received: June 26, 2017, Accepted: December 8, 2017

Abstract: As the increase of large-scale Web services, Web applications are increasingly sophisticated. The increase of smartphones makes traffic to Web servers increase each day. Management of Web servers is increasingly important. When a Web server receives resource-hungry requests, administrators of the web server must control the resources flexibly. However, existing resource control methods process mandatorily such as denial, disconnect or the like of the request when the number of simultaneous connections, request per unit time, CPU used time or memory utilization exceed a certain threshold value. These methods can't control resources processing request. In this situation, if one request has the potential to occupy a majority of computer resources, the server process can process other request in principle by separating resources of the server process temporarily without using up the saving of computer resources. In this paper, we propose a resources control architecture separating computer resources virtually by a HTTP request for a Web server. Each HTTP request processing is allocated virtual computer resources temporarily and the request is process within the limit of the allocated computer resources in our architecture. Therefore, each request processing has a very small effect if clients access to contents heavily or a particular content occupy a majority of computer resources. If a request has the potential to occupy a majority of computer resources, our architecture can process the request continually within the limit of the allocated computer resources without denying or disconnecting requests mandatorily.

Keywords: Web Server, Operation Technology, Resource Management, Apache, cgroup

[1] GMO ペパボ株式会社ペパボ研究所
Pepabo Research and Development Institute, GMO Pepabo, Inc., Fukuoka 810–0001, Japan
[2] 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto Univesity, Kyoto 606–8501, Japan
[a)] matsumotory@pepabo.com

1. はじめに

大規模 Web サービスの普及や Web ホスティングサービス [13] の低価格化にともない，企業だけでなく個人の日記や作品をインターネット上で平易に公開することが可能となり，個人が Web サイトや Web サービスを持つ時代になってきている．さらには，スマートフォンの普及により，インターネットはより身近なものになってきている．その結果，Web サーバへのアクセス数は日々増加してきており，多くの Web サイトを管理する Web ホスティングサービス事業者は，いかに安定してアクセスを処理できるか，また，個人でも利用できるようにいかに安価にサービスを提供できるかが課題となっている．Web ホスティングサービス事業者は，そのような課題を解決するための 1 つのアプローチとして，限られたハードウェアリソースで高集積にホストを収容可能なマルチテナント環境を構築することが求められてきている．多数のユーザに同一サーバ上のハードウェアリソースを共有させることの効率性と，共有することによるセキュリティ上の課題を解決しつつ，リソースを柔軟に制御し，かつ，できるかぎり高集積にホストを収容できれば，安定したユーザ単位のリソース配分が可能となり，ユーザにとっては快適でセキュリティが担保された環境をより安価に提供可能となる．

Web ホスティングサービスにおいて，ドメイン名（FQDN）によって識別され，対応するコンテンツを配信する機能をホストと呼ぶ．ホスティングサービスの個人利用においては，サービスの提供価格を数百円程度に下げるために，筆者が所属する GMO ペパボ株式会社では 1 つのサーバに数万から最大 10 万ホスト程度収容するような，大規模高集積 Web ホスティング基盤の構築が要求される．Web サーバに収容効率良く高集積に複数のホストを収容するためには，プロセス数がホスト数に依存しないように，単一のサーバプロセス[*1]で複数のホストを処理する必要がある．また，ホスティングサービスのようにサービス利用者が自由に Web コンテンツを配置可能なサービスでは，サービス提供側が Web コンテンツの実装まで管理することができないため，OS やミドルウェアの領域でセキュリティやリソースを管理する必要がある．そのような環境で，Web サーバプロセスは高速にリクエストを処理するために複数のリクエストを同一のサーバプロセスを使いまわして順番に処理する特性上，大量のリクエストを安定して処理するためには，CPU といったハードウェアリソースを多く消費する CGI プログラムのような動的コンテンツへのリクエストがあった場合，その特定のリクエストのみを他のリクエスト処理に影響を与えないように制限する必要がある．さらに，特定のリクエストに対する制限が他のリクエストには適用されないように，あらかじめプロセスに制限を課するのではなく，HTTP リクエスト単位でリソース制御する必要がある．

既存の Web サーバの一般的なリソース制御 [3], [17], [23] は，リクエストの同時接続数や単位時間あたりのリクエスト数が指定の閾値を超えた場合や，管理者があらかじめ指定した CPU 使用時間やメモリ使用量の閾値を超えた場合に，リクエストを切断，あるいは，拒否するような手法である．これは，サーバの負荷低減を重視しすぎるあまりに，サイトに何度アクセスしてもレスポンスが受信できないといったようなユーザ体験を損なうアーキテクチャになっている点が課題である．また，1 つの Web サーバに高集積にテナントを収容している場合，特定のテナントに配置されれているループ処理が多数実行されるような CPU 使用率を占有する動的 Web コンテンツに対して，たった 1 つのリクエスト処理がサーバの CPU リソースを占有してしまった場合にも，他のテナントにリソースを分配するためには，強制的に処理を中断するような対応しかとれないという課題がある．数万ホストを単一のサーバに収容するような高集積マルチテナント方式のホスティングサービスにおいて，昨今の Web サーバ用途に使われるようなサーバの CPU コア数が数十コアであることを想定する [20]．その場合，サーバに複数収容されているホストにできるだけ平等にハードウェアリソースを分配すべき状況で，特定のホストに対するたった 1 つのリクエストが 1 つの CPU コアの使用率を 100%占有し続けると，数万ホストで数十コアの CPU を共有して使うことを想定している状況では CPU 使用過多であり，そのような他のホストにリクエスト処理のためのリソースが十分に分配されなくなる状況は回避すべきである．

1 つの HTTP リクエストがハードウェアリソースを大きく占有しようとしても，その他のリクエスト処理が影響を受けず，さらには各リクエストが継続的に処理を行うためには，リクエストを処理中のプロセスを，一時的に限られたリソースの範囲内に分離し，その制限下でリクエストを継続的に処理すればよい．また，管理者がそのようなリソース制御ルールをプログラマブルに記述できれば，柔軟なリソース制御が可能になる．そこで，本研究では，同一のサーバプロセス上で，HTTP リクエスト単位の粒度でハードウェアリソースを仮想的に分離できる Web サーバのリソース制御アーキテクチャを提案する．このアーキテクチャによって，継続的にリクエストを処理しつつも，各リクエスト処理は一時的に仮想的なリソースを割り当てられ，そのリソースの範囲内で処理が行われる．そのため，特定のリクエスト処理が CPU を占有する処理であったとしても，制限したリソース使用量以上には利用できないため，その他のリクエストは制限値以上の影響を受けない．

[*1] ただし，ここでいう単一のサーバプロセスとは，ホストごとにサーバプロセスを起動させるのではなく，複数のホストでサーバプロセスを共有することを指す．

さらに，大量にリソースを消費するようなリクエストであっても，リクエストを拒否・切断することなく，割り当てたリソースの範囲内で処理を継続させることが可能となる．従来手法では高集積にホストを収容すればするほど，互いにリソースの影響を受け合うためにサーバが不安定になりがちであり，それを解決するためにはリクエストを拒否，あるいは，切断という制限手法をとる必要があったが，提案手法によって高集積にホストを収容しても，高負荷時にクライアントにエラーを返すことなく，リクエスト単位でリソースの最大使用量を制限できるため，制限されていてもリクエスト処理が継続されることによるユーザ体験の向上と，より詳細にリソース制御できることによるサーバの安定性を高いレベルで両立できる．

実装に関して，Linux（Linux Kernel 3.10）上で動作する Apache HTTP Server 2.4.6 [16] に mod_mruby [22] を組み込み，mod_mruby 上で動作するモジュールとして，Linux の仮想化技術である cgroup [19] を用いたリソース制御モジュールを実装した．制御ルールは，Ruby で記述できるようにした．

本論文の構成は以下のとおりである．2 章ではリソース制御技術について述べる．3 章では提案するリソース制御アーキテクチャについて説明する．4 章でリソース制御アーキテクチャの評価を行い，5 章でまとめとする．

2. リソース制御アーキテクチャ

限られたハードウェアリソースで複数のホストをできるだけ高集積に収容したうえで管理・運用するためには，リソースの分離と各ホストが安全にコンテンツを配置できるような権限分離の管理が重要である．我々は，大規模共有型 Web バーチャルホスティング基盤のセキュリティと運用技術の改善 [23] を行った．その改善において，リソースの分離のために，ファイルやホスト単位で同時接続数を制御したり [3]，OS の負荷によってリクエストを中断あるいは拒否したりする [17]．しかし，CPU や Disk I/O 等のハードウェアリソースは共有のため，多くのリソースを消費する CGI のような動的コンテンツへのたった 1 つのリクエストによってサーバのリソースを占有し，他のホストに影響を与えるという問題があった．特にホスティングサービスにおいては，利用者が配置する動的コンテンツをサービス提供側が変更することができないため，そのような高負荷状態を引き起こすコンテンツを適切に OS や Web サーバの機能によって制限する必要がある．

2.1 権限分離を考慮した Web サーバのリソース分離

単一のサーバで複数のホストを管理する場合，セキュリティを担保するための権限分離を考慮しながら，複数のホストでサーバのハードウェアリソースを共有する構成が一般的である．これまで，マルチテナントアーキテクチャにおける Web サーバのハードウェアリソースを仮想的に分離する手法は，主に以下の 3 種類が提案されている [2], [11]．

以降では，Web サーバにおける既存のリソース制御技術の問題点を整理する．

(1) KVM [8], [10] や VMware 等の仮想マシンで複数の OS でリソースを分離する手法
(2) OpenVZ [1] や LXC のようにカーネルを共有しながらプロセス権限で OS リソースを分離する方式
(3) 仮想ホスト方式のように単一のサーバプロセスで複数のホストを扱う手法

ハードウェアリソースが潤沢にあり，サーバの運用面やセキュリティおよび可用性を重視した場合は，手法 (1) の，ホストそれぞれに対して，個別の仮想マシン [5], [6], [7] を割り当てる構成がとられてきた．しかし，これらはハードウェアリソースの面で非常にコストが高く，オーバヘッドも大きいため，限られたリソースで高集積に数万ホストを収容するにはホストごとに数十から数百 MBytes の仮想マシンが必要となるため不向きである．

(2) の方式では，ホスト単位で chroot() システムコールによるファイルシステムの隔離やプロセスリソース管理技術を組み合わせたコンテナ環境 [4], [14], [18] を構築し，その環境ごとにサーバプロセスを起動させる必要があるため，プロセス数がホスト数に依存する．プロセス数がホスト数に依存すると，収容数が搭載メモリ容量により制約されるため，高集積に数万ホスト収容する場合，Web サーバプロセスが一般的に数十から数百プロセス必要になることを考えると，そのプロセス群が数万ホスト分必要となるため高集積にホストを収容するには向いていない．通常 Web サーバプロセスは性能を担保するために，想定される同時接続数にあたる数のプロセスを起動させる．たとえば 5 万ホスト収容する場合に，ホストごとに同時接続数 20 を想定した場合，20 個のサーバプロセスを起動させたとすると，100 万プロセス起動させる必要がある．一方で，サービスを安価に提供するためには広く使われている一般的なサーバを利用する必要がある．そのため，筆者が所属する GMO ペパボ株式会社では，32 GB のメモリを搭載したサーバを Web ホスティングサービスにおいて採用している．その場合を考えると，1 プロセス最大 32 KB 程度しか割り当てることができず，Apache の 1 プロセスが通常数十 MB 程度消費されることを想定すると，安定稼働は到底不可能である．

(2) の方式のメリットは，ホストごとにサーバプロセスを起動させる方式であるため，(3) の仮想ホスト方式では設定できないような，サーバプロセス全体の設定や，プロセス単位でのリソース制御や隔離機能を利用することができる．また，プロセスとしては完全に他のホストのプロセスと分離しているため，特定ホストのリクエスト処理に時間がかかっていたとしても，プロセスレベルで他のホスト

は影響を受けない．しかし，サーバプロセス数がホスト数に依存するため，高集積は困難である．

Apache の VirtualHost 機能[*2]で採用されている (3) の仮想ホスト方式のように，単一のサーバプロセスで複数のホストを管理するアーキテクチャの場合，数万ホストに対するリクエスト処理は数百から数千個のサーバプロセスで共有して処理するため，ホスト数に依存せず高集積が可能である．たとえば，Apache の VirtualHost 機能を利用すると，2 万ホスト収容した場合でもプロセス数がホスト数に依存せず，プロセスのメモリ使用量は 2 GB 程度となり [24]，5 万ホストであっても十分に，(2) の方式で言及したメモリ 32 GB のサーバでも動作可能となる．しかし，(3) の方式では，単一のサーバプロセスで複数のホストを処理しているため，特定のホストやリクエスト処理がリソースを占有した場合に，他のホストやリクエスト処理が影響を受けやすいという問題がある．

以上より，ホスト単位で適切にリソースを分離するためには (1)，(2) が適切であるが，収容効率は悪い．(2) はプロセスごとにリソース分離できるが，高集積マルチテナントアーキテクチャのリソース制御においては，少なくとも個々のプロセスに適切なリソース制約を与えて，ホスト単位のリソース分離を図る必要がある．(3) は高集積なマルチテナント環境を構築する際には適切であるが，単一のサーバプロセスで複数ホストに対するリクエストを処理するため，最もリソース分離に向いていない．

(3) におけるリソース分離の課題に対して，既存の Web サーバの一般的なリソース制御 [3], [17], [23] は，リクエストの同時接続数や単位時間あたりのリクエスト数が指定の閾値を超えた場合や，管理者があらかじめ指定した CPU 使用時間やメモリ使用量の閾値を超えた場合に，リクエストを切断，あるいは，拒否するような手法であり，CPU 使用率の占有によるサーバの負荷の低減を重視しすぎるあまりに，サイトに何度アクセスしてもレスポンスが受信できないといったユーザ体験を損なうようなアーキテクチャになっている．また，CPU を占有する CGI プログラムのような動的コンテンツに対して，たった 1 つのリクエスト処理がサーバの CPU リソースを占有してしまった場合にも，強制的に処理を中断するような対応しかとれない．

2.2 Linux のプロセスリソース管理技術

Linux には，cgroup [19] と呼ばれるプロセスのリソース管理技術がある．cgroup は，2006 年 9 月から開発が開始され，2008 年 1 月に Linux Kernel 2.6.24 に取り込まれた．cgroup は，プロセスの優先度を変更する nice のような機能から，コンテナである LXC や OpenVZ のような OS レベルの仮想化までの，様々な仮想化の用途に対応するための統一されたインタフェースを持っている．cgroup は，以下のような機能を提供している．

(1) リソース制限
(2) 優先順位
(3) 説明
(4) 隔離
(5) コントロール

(1) はプロセスグループのメモリ使用量やファイルシステムキャッシュを制限する機能を提供する．この機能により，プロセス単位でメモリ消費量の上限を制限することが可能となる．制限値を超えた場合は，プロセス停止処理が動作する．(2) は CPU やトラフィック，I/O を制御する機能を提供する．この機能により，プロセスグループで利用可能な CPU 使用率の割合を 10%程度にしたり，トラフィックの流量を 10 Mbps にしたり，デバイス I/O を 10 MByte/sec や 100 IOPS に制御したりできる．(3) はプロセスグループのリソース消費の統計値を計測する機能を提供する．たとえば，各プロセスグループのリソース消費量を可視化したり，従量課金の基準を定義したりすることが容易になる．(4) は異なる名前空間，たとえば，プロセス ID やオーナ，ファイルシステム等が別空間には原理的に干渉しないような領域にプロセスグループを分離し隔離する機能を提供する．(5) はプロセスグループをサスペンドしたりリストアしたりする機能を提供する．この機能を利用することで，再起動なしでのカーネルの置き換え，コンテナやプロセスレベルでのサスペンド・レジューム機能，コンテナやプロセスのライブマイグレーションが可能となる．

cgroup の機能を利用することで，システム開発者はシステムリソースの割当て，優先順位付け，モニタリング等，粒度の細かいコントロールが容易に可能となる．

3. 提案するリソース制御アーキテクチャ

2.1 節に基づいて，高集積にホストを収容するためのマルチテナントアーキテクチャのリソース制御の要件をまとめる．2.1 節の (3) のような高集積に収容可能な仮想ホスト方式を採用したうえで，ハードウェアリソースを占有するような動的コンテンツに対するリクエストがあったとしても，同時に処理しているその他のリクエスト処理に対する影響を最小化する必要がある．さらには，各リクエストが継続的に処理を行うためには，リクエストを処理中のプロセスを，一時的に限られたリソースの範囲内に分離してから，その制限されたリソース範囲内でリクエストを処理すればよい．また，管理者がそのようなリソース制御ルールをプログラマブルに記述できれば，柔軟なリソース制御が可能になる．

[*2] http://httpd.apache.org/docs/current/en/vhosts/

3.1 アーキテクチャの詳細

単一のサーバプロセスで複数のホストを高集積に管理するアーキテクチャにおいて，動的コンテンツによるリソース占有の問題を解決するためには，HTTP リクエスト単位という細かい粒度で，CPU のようなハードウェアリソースの最大使用率を制御し，仮想的に分離できればよい．それによって，CPU を大量に消費するような動的コンテンツに対するクエスト処理が，サーバに収容されているホスト間で共有しているリソースを占有しようとしても，分離されたリソースの範囲内で動作するため，他のリクエスト処理に影響を与えない．また，管理者がリソース制御ルールをプログラマブルに記述できれば，柔軟なリソース制御が可能になる．それによって，サービス利用者が配置する Web コンテンツを管理できないサービスのサーバ管理者が，サーバを安定稼働させる際に，リソース制限によってリクエストの拒否を頻繁に生じさせることで，サーバへアクセスする利用者にとってのユーザ体験の低下を引き起こすことなく，多種多様な動的コンテンツのリソース消費量を制御できるようになる．さらに，単一のサーバプロセスで高集積にホストを管理していることを考慮すると，可能な限りサーバプロセスを再起動することなく，リソース制御の変更をできるようにするべきである．

本論文では，HTTP リクエスト処理時に，管理者が記述した内容に従って仮想的に分離されたリソース領域を作成し，サーバプロセスをそのリソース領域内で動作させ，レスポンスを送信後はそのリソース領域内の制限を解除することで，高集積マルチテナント環境においても，リクエスト単位で任意のリソース制御が可能な Web サーバのリソース制御アーキテクチャを提案する．クライアントからサーバプロセスに対してリクエスト処理があると，そのリクエストが制御対象であった場合，サーバプロセス上で動作しているリソースコントローラが，リソース制御ルールからリソースに関する設定値を取得する．そして，そのリソース設定値をもとに確保された仮想リソース領域がなければ，新規で領域を作成する．たとえば，任意のリクエストに対し，CPU 使用率は最大 10%，ディスクへの書き込みは最大 5 MBytes/sec に制限したいとする．その場合は，制御ルールを設定ファイルに記述する．記述後，新しいリクエストを受けた際に，リソースコントローラは制御ルールを解釈し，ルールどおりにリソース領域を生成する．そして，サーバプロセスを，作成したリソース領域に割り当てた後，リクエストをそのリソース範囲内で処理する．処理後は，レスポンスをクライアントに返し，リソース領域への割当てを解除してから，リソース領域を削除し，次のリクエスト処理に備える．また，リソースコントローラは，複数の Web サーバソフトウェア上でも同様に扱えるように，複数の Web サーバソフトウェアに汎用的な機能拡張インタフェースを用意したうえで，そのインタフェース上に実装する．これによって，Web サーバソフトウェアの違いを気にすることなく，リソースコントローラ自体の拡張・修正が容易となる．このようなアーキテクチャをとることにより，リクエストに含まれる情報，たとえば，ホスト名や HTTP メソッド，ユーザ情報等を条件に，管理者が柔軟にリソース制御を行える．

3.2 アーキテクチャの実装

提案するアーキテクチャを Linux（Linux Kernel 3.10）上で動作する Apache（Apache HTTP Server 2.4.6）に実装する．高集積マルチテナント方式の Web サーバでは，収容ホストに対するリクエストを，収容ホストよりも少ない数のプロセスで同時に処理するため，プロセス間でリソース競合が起き，適切にホスト単位でリソースの分配を管理することは困難である．提案するアーキテクチャは，多数のリクエストやリソースを専有するリクエストを同時並行で処理する状況で，各リクエストを制限された仮想リソース領域に隔離することにより，同時に処理しているようなリクエスト間で，特定のリクエストがリソースを占有せず，他のリクエストに十分にリソースを分配できる状態，つまりリソース専有の干渉度合いが低い競合状態を実現できる．その結果，高集積のマルチテナント環境であっても，ホスト間において，たとえば特定のホストに配置された CPU 使用率を占有する CGI プログラムに対してリクエストが生じても，他のホストへのリクエストに必要な CPU リソースを分配可能となり，ホスト間のリソース競合も緩和できる．

仮想リソース領域の作成には cgroup を利用した．2.2 節で言及した (2) 優先順位の機能を使い，CPU や I/O をリクエスト単位で制御し，任意の設定パラメータによってリソース分離を実現する．cgroup は，本来同じ基準で集められたプロセスの集合に対してまとめて制御を行うものであるが，その方式に加えて，提案手法では高集積マルチテナント方式における特定のテナント，あるいは特定のリクエストを処理しているプロセスのみの CPU 使用率を一時的に制限する必要があるため，リクエスト単位で cgroup による仮想リソース領域を割り当てるようにした．また，一般的な Web サーバの実装上，サーバプロセスは異なるテナントに対するリクエストを同一のプロセスで順番に処理するため，サーバプロセスをあらかじめ仮想リソース領域に割り当てておくことはできないという問題もあったためである．この方式による性能上のオーバヘッドも実運用上問題ない程度であった．オーバヘッドについては 4 章で言及する．

このようなアーキテクチャと実装によって，たとえばテナント単位でリソース制御をしたい場合は，従来手法の同時接続数制限と組み合わせて，リクエストに含まれるホスト情報により，テナントに割り当てる最大 CPU 使用量を

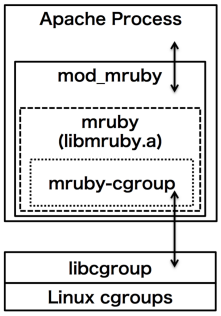

図 1 Apache と mod_mruby と cgroup の関係
Fig. 1 Relationship between Apache, mod_mruby and cgroup.

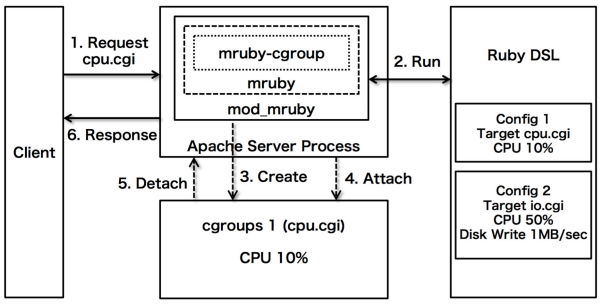

図 2 Linux への実装例
Fig. 2 Implementation example on Linux.

制限することもできる．その場合，1 つのリクエスト，たとえば CGI プログラムを実行するプロセスのみに対する最大 CPU 使用率の割当てを 20%として，最大同時接続数を 5 とし，テナント全体でのプロセス集合に対して最大 CPU 使用率の割当てを 50%とすることもできる．これは，cgroup のプロセスの集合に対してまとめて制御する方式と，プロセス単位でリソース割当て領域を作成する方式を組み合わせることにより実現できる．このように提案手法は，OS のスケジューラのできるだけ平等にプロセスにリソースを配分する戦略を尊重しながら，cgroup の機能を使ってテナントあるいは CGI プログラムに対するリクエストを処理するプロセスの集合，あるいはリクエスト単位に対応する個々のプロセスについて，管理者がその意図するところによりリソースの制御を補助的に行えるようにするものである．

汎用的な Web サーバの機能拡張 I/F には，組み込みスクリプト言語を Web サーバに組み込むことで，スクリプト言語にかかわらず Web サーバを高速かつ省メモリに機能拡張できる mod_mruby [22] を利用した．mod_mruby は，代表的な Web サーバソフトウェアの Apache と nginx [12] の Web サーバ機能拡張を，Ruby によって容易に拡張が可能であり，高速かつ省メモリに動作する．また，提案するアーキテクチャでは，リクエストごとに Ruby で記述された設定を読み込むため，mod_mruby のように Ruby で Web サーバの機能を記述可能で，リクエスト単位で高速に動作する Web サーバ機能拡張は，本アーキテクチャの実装に適している．

図 1 に Apache と mod_mruby と cgroup の関係を示す．Ruby スクリプトから Apache の内部 API を操作できるように，Apache に mod_mruby を組み込む．同様に，Ruby スクリプトから cgroup を操作するために，cgroup を操作するためのシステムライブラリである libcgroup を Ruby で制御できるように mruby-cgroup [*3] という拡張モジュールを実装する．これによって，Ruby スクリプトから Apache 内部の情報を取得したり，その情報をもとに libcgroup を操作したりすることが可能になる．

[*3] https://github.com/matsumotory/mruby-cgroup

```
r = Apache::Request.new

if r.filename = "/path/to/cpu.cgi"
  cpu = Cgroup::CPU.new "cpu_group"
  if ! cpu.exist?
    cpu.cfs_quota_us = 10000
  end

  cpu.create
  cpu.attach
end
```

図 3 リソース制御ルール記述例
Fig. 3 Example of resource control rule descriptions.

図 2 に，図 1 のアーキテクチャ図を Linux 上で動作する Apache に実装する場合のアーキテクチャの構成を示す．リソースコントローラは mruby と mruby-cgroup が担い，その制御を Web サーバ上で可能にする機能拡張機構を mod_mruby が担う．制御ルールは Ruby の構文で記述し，ファイル，ディレクトリ，ホスト単位等，様々な条件からリソース制御対象の記述が可能となっている．また，cgroup によるリソース領域は複数設定することが可能で，リクエスト時にすでに存在する場合はその領域を利用し，存在しない場合は新規で領域を確保する．

Ruby で記述された各種リソース制御ルールに従って，クライアントからのリクエスト処理のリソースを制御する．**図 3** に，cpu.cgi にリクエストがあった場合に，そのリクエスト処理を最大 CPU 使用率 10%のリソース領域に分離する場合のリソース制御ルール記述例を示す．

図 3 のように，Ruby で記述可能な機能拡張インタフェースによって，Ruby の言語仕様をもとに，リソース制御ルールを柔軟に記述することが可能となる．また，条件として利用するホスト名やファイル名等は条件分岐として記述するだけでなく，数万単位のホストを扱う場合には，ホストと関連する設定データを Key-Value Store（KVS）のようなデータベースに保存しておいて，リクエスト時にホスト名をキーに設定を取得するような記述も Ruby で実装可能であるため，制御ルール行数がホスト数に依存して長大にな

```
targets = %w(

target1.example.com
target2.example.com
target3.example.com
target4.example.com

)

r = Apache::Request.new

if targets.include? r.hostname
  c = Cgroup::CPU.new "httpd-limiterd"
  c.shares = 25
  if c.exist?
    c.modify
  else
    c.create
  end
  c.attach

  b = Cgroup::BLKIO.new "httpd-limited"
  b.throttle_write_bps_device = "8:0 30000000"
  b.throttle_read_bps_device = "8:0 30000000"
  if b.exist?
    b.modify
  else
    b.create
  end
  b.attach
else
  c = Cgroup::CPU.new "httpd"
  c.shares = 75

  if c.exist?
    c.modify
  else
    c.create
  end
  c.attach

  b = Cgroup::BLKIO.new "httpd"
  b.throttle_write_bps_device = "8:0 100000000"
  b.throttle_read_bps_device = "8:0 100000000"
  if b.exist?
    b.modify
  else
    b.create
  end
  b.attach
end
```

図 4 ホスト単位でリソースを分離
Fig. 4 Separating resources for each host.

らない．また，制御ルールをフックしておくためのフェーズは，サーバのレスポンスを生成するまでのリクエスト解析，アクセスチェック，レスポンス生成直前，レスポンス生成後等である．

図 4 にリソース制御ルールの応用例を示す．図 4 では，あらかじめ高負荷ユーザが所属するホストを対象に，リクエストが対象のホストであれば高負荷ユーザと判定し，全体の CPU の 25%，DISK I/O の最大読み書き帯域を 30 MByte/sec 使えるようにリソースを分離する．対象外のホストに対しては CPU を 75%，DISK I/O の読み書きの帯域を最大 100 MByte/sec 使用できるように設定した例である．cgroup により，プロセスが利用できる最大メモリ使用量を制限することは可能であるが，本研究ではリクエストを継続的に処理しながら制限も行うことを目的としているため，今回は対象としない．また，本研究では，高集積マルチテナント方式により高集積にテナントを収容しているようなシステムで，特定のテナント上で CPU を 100%消費するような CGI プログラムに対して最大使用率を制限することにより，プログラムの動作に依存することなく CPU 使用率を制限でき，かつ継続的に処理を行えるため，ホスティングサービスのように多種多様なプログラムが動作する環境においても，特定のテナントのみを制限して，他のテナントにリソースを分配することができるため有用である．CPU 使用率を制限されたリクエストは，リクエスト処理にかかる時間が増加する．そのため，テナント単位での同時接続数を制限する従来手法と組み合わせ，並行処理されるリクエストの数を一定以下に抑えるような運用を行うことが必要である．また，制限対象となったテナントについては，同時接続数による制限だけではなく，CPU 使用率との組合せによって制限が可能となるため，これまでは CPU100%使用するレスポンスに対して同時接続数 1 としか制限できなかった状況で，CPU33%かつ同時接続数 3 というような，CPU リソースは制限しながらも同時接続数をできるだけ増やしたいといった要求にも応えられるようになる．

従来手法では，閾値を超過した場合にリクエストを拒否，あるいは，中断していたが，本手法では閾値を超えた場合には，あらかじめプログラマブルに記述されたルールに従って，使用可能なリソース使用量を限定しつつも継続的にリクエストを処理することが可能となる．さらに，Web サーバソフトウェア内部の情報や OS の負荷状況を考慮した記述や，仮想ホスト，ファイル，ディレクトリ単位の制御も，Ruby の制御構造によって柔軟に記述でき，汎用性が高い．また，Ruby スクリプトを変更することで，サーバプロセスを再読み込みすることなくリアルタイムで条件記述を変更できる機能も実装しているため，運用においても負担が少ない．一方で，スクリプトの変更を必要としない場合は，事前に中間コードにコンパイルしておいて，リクエスト時に中間コードを実行することでより高速に動作させることも可能である．なお，mod_mruby は高速にかつ省メモリで動作することも，本手法のプログラマブルにルールを記述する機能に適しており，オーバヘッドは少ないことが分かっている [22]．

4. リソース制御アーキテクチャの実装の評価

提案するリソース制御アーキテクチャの評価を行うために，CPU 制御機能を組み込むことによるオーバヘッドやリソース制御の精度評価を行った．リクエスト単位で CPU のリソース制御精度が十分であれば，HTTP リクエスト単位でリクエスト処理の最大 CPU 使用率が制限できる．それにより，高負荷時においても，高負荷の原因となるリクエスト以外の他のリクエスト処理に CPU リソースを残すことができ，かつ，制限した最大 CPU 使用率の範囲内で制限対象のリクエスト処理を継続的に行うことができる．HTTP リクエスト単位でリソース制御が達成できれば，高集積ホスティング環境のように，プロセスを共有しながら

表 1 実験環境
Table 1 Experimental environment.

	項目	仕様
Client	CPU	Intel Core2Duo E6600 3.06 GHz
	Memory	8 GBytes
	NIC	Realtek RTL8111/8168B 1 Gbps
	OS	Fedora 18
Server	CPU	Intel Core i7-4770K 3.5 GHz
	Memory	32 GBytes
	NIC	Intel I217V 1 Gbps
	Middle-Ware	Apache/2.4.6
	OS	Fedora 19

複数のホストに対するリクエストを処理するような場合であっても，各リクエストのリソース使用量を限定できるため，ホスト間のリソース競合も低減することが可能となる．

性能評価として，本アーキテクチャの導入前後でリソース制御対象でないリクエストの性能にどの程度違いがあるか，リクエスト処理時間の違いによってリソース制御の精度に差がどれほど生じるのかを評価した．**表 1** にテスト環境のマシンスペックを示す．また，Apache の仮想ホストの数は 10 万ホストとした．Apache の起動プロセス数は Apache のバージョン 2.4.6 のデフォルト設定の 250 プロセスとした．

まず，本アーキテクチャを導入することで，リソース制御対象でないリクエストの性能にどの程度差異があるかを評価した．評価方法として，リソース制御アーキテクチャの処理の影響を最大にするため，リクエスト対象のファイルは hello world を出力するだけの静的な HTML ファイルとした．そのファイルに対し，表 1 のテスト環境のリソースを十分に使用できるように，予備実験から同時接続数 100，総接続数 10 万のリクエストパターンを決定し，評価を行った．ベンチマークソフトウェアには ab コマンド [15] を利用した．その結果，リソース制御アーキテクチャを導入していない場合は，1 秒間に 32915.46 リクエスト処理できており，リソース制御アーキテクチャを導入している場合は，32322.07 リクエスト処理できていた．この結果から，リソース制御アーキテクチャを導入することによるボトルネックはほとんどないと考えられる．また，CGI プログラムのような動的コンテンツの場合には，静的コンテンツの処理よりも多くの CPU 使用時間を使用することが多いため，静的コンテンツで無視できる程度の処理であれば，動的コンテンツの場合も無視できる．

次に，リクエストの処理時間の違いによって，リソース制御の精度にどれほど差異が生じるのかを評価した．評価方法として，シェルスクリプトで単純に while でループを行い，ループ終了後に hello world を出力するだけの CGI プログラムを作成し，そのループ回数を変化させることで，リクエスト処理時間を変化させた．また，CGI プログラムが単なるループ処理であるため，CGI プログラムに対するリクエスト処理時間のほとんどは，Web サーバ上で CGI プログラムのレスポンスを返す Web サーバプロセスの CPU 使用時間とおおむね同じであると考えられる．その CGI プログラムに対して，リソース制御アーキテクチャにより CPU 使用率を 50%に制限し，表 1 のテスト環境のリソースを十分に使用できるように予備実験から決定した同時接続数 10，総接続数 1000 で，CGI プログラムにリクエストを送信した．CGI プログラムのリクエスト処理時間を変化させながら，1 リクエストの処理にかかった時間が，リソース制御をしていない場合にかかった処理時間と比較して，性能がどれだけ低下しているかを測定した．その値を性能制御率と呼ぶことにする．50%にリソース制御している場合，性能制御率が 50%に近ければ近いほど，正確にリソース制御できていることになる．

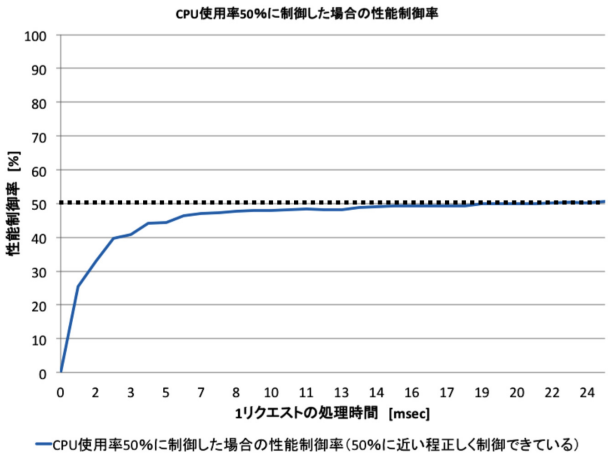

図 5 リクエスト処理時間の違いによるリソース制御精度
Fig. 5 Resource control accuracy regarding the difference between request processing time.

図 5 に 1 リクエストの処理時間を変化させた場合の，リソース制御アーキテクチャの性能制御率を示す．図 5 のように，1 リクエストの処理時間が 0 msec から 6 msec 程度の場合は，表 1 のテスト環境において，本来制御したい 50%よりも，より低く性能が制御されていることが分かる．これは，CGI プログラムが CPU を 50%以上消費できていないことによって生じており，CPU を 50%以上消費するプログラムであれば，正確に 50%に制御できることが分かる．実験環境においては，6 msec 以上リクエスト処理に時間がかかるような処理は，正確に性能を 50%に制御できていることが分かる．

以上より，リクエスト単位でリソースの最大使用率を制限する場合において，本手法が有効に機能すると考えられる．制度評価により，リソースを占有するようなプログラムに対してリクエストがあったとしても，管理者が設定した CPU 使用率を超えることはない．すなわち，高負荷時でも他のリクエストへ与える影響は，リクエスト処理時間

の大小にかかわらず限定できる．2.1 節で述べたように，高集積マルチテナント方式における従来の制限手法は処理を継続しながらリクエスト単位でリソースを制御することができない粗い制限手法であり，制限によってサーバを安定させたとしても，ユーザ体験の質は大きく低下するという課題があったが，リクエストを拒否，あるいは，切断することなく継続的な処理とリソースの制限が同時に実現できるため，処理は低速になるもののレスポンスを送信可能となり，本研究の目的を達成できる．

5. むすび

本研究では，限られたハードウェアリソースで高集積にホストを収容する際に，単一のサーバプロセスで複数のホストを管理する場合に生じるリソース制御の問題を解決するために，リクエスト単位で仮想的にリソースを分離する Web サーバのリソース制御機構を提案した．本手法により，1 つのリクエストがハードウェアリソースを大きく専有しようとしても，同時にリクエストを処理しているプロセス間で，リソース専有の競合の少ない状態を実現することができるため，他のリクエストが処理できなくなるようなことが生じない．また，リクエスト処理を切断・拒否することなく継続的にリソースを制御できる．さらに，システム管理者が Ruby で柔軟にリソース制御ルールを記述可能であるため，多種多様なリソース制御に関する問題に対し，対応できると考えている．

今後，より複雑なリソース管理の問題を解決するための方法論について検討していく予定である．実装面では，このアーキテクチャを，マルチスレッド型や非同期 I/O 型の Web サーバアーキテクチャにも対応させ，汎用的なインタフェースとして実装することで，複数の Web サーバソフトウェアでも同様にリソースを制御できるようになると考えている．

参考文献

[1] Ben-Yehuda, M., Mason, J., Xenidis, J., Krieger, O., Van Doorn, L., Nakajima, J. and Wahlig, E.: Utilizing IOMMUs for Virtualization in Linux and Xen, *The 2006 Ottawa Linux Symposium* (*OLS'06*), pp.71–86 (July 2006).

[2] Che, J., Shi, C., Yu, Y. and Lin, W.: A Synthetical Performance Evaluation of Openvz, Xen and KVM, *IEEE Asia Pacific Services Computing Conference* (*APSCC*), pp.587–594 (Dec. 2010).

[3] David, J.: mod_limitipconn.c, available from ⟨http://dominia.org/djao/limitipconn.html⟩.

[4] Felter, W., Ferreira, A., Rajamony, R. and Rubio, J.: An Updated Performance Comparison of Virtual Machines and Linux Containers, *IEEE International Symposium Performance Analysis of Systems and Software* (*ISPASS*), pp.171–172 (Mar. 2015).

[5] Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D.: Terra: A Virtual Machine-based Platform for Trusted Computing, *ACM SIGOPS Operating Systems Review*, Vol.37, No.5, pp.193–206 (Oct. 2003).

[6] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *NDSS*, Vol.3, pp.191–206 (Feb. 2003).

[7] Goldberg, R.P.: Survey of Virtual Machine Research, *Computer*, Vol.7, No.6, pp.34–45 (1974).

[8] Habib, I.: Virtualization with KVM, *Linux Journal*, No.166, p.8 (2008).

[9] Poul-Henning, K. and Watson, R.N.M.: Jails: Confining the omnipotent root, *Proc. 2nd International SANE Conference*, Vol.43 (2000).

[10] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: The Linux Virtual Machine Monitor, *The Linux Symposium*, Vol.1, pp.225–230 (July 2007).

[11] Kovári, A. and Dukan, P.: KVM & OpenVZ virtualization Based IaaS Open Source Cloud Virtualization Platforms: OpenNode, Proxmox VE, *IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics*, pp.335–339 (Sep. 2012).

[12] nginx: nginx, available from ⟨http://nginx.org/ja/⟩.

[13] Prodan, R. and Pstemann, S.: A survey and taxonomy of infrastructure as a service and web hosting cloud providers, *Grid Computing, 2009 10th IEEE/ACM International Conference*, pp.17–25 (October 2009).

[14] Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A. and Peterson, L.: Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors, *ACM SIGOPS Operating Systems Review*, Vol.41, No.3, pp.275–287 (Mar. 2007).

[15] The Apache Software Foundation: ab–Apache HTTP server benchmarking tool, available from ⟨http://httpd.apache.org/docs/2.2/programs/ab.html⟩.

[16] The Apache Software Foundation: Apache HTTP Server Project, available from ⟨http://httpd.apache.org/⟩.

[17] The Apache Software Foundation: Server – Wide Configuration Limiting Resource Usage, available from ⟨http://httpd.apache.org/docs/2.2/en/server-wide.html#resource⟩.

[18] Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T. and De Rose, C.A.: Performance Evaluation of Container-based Virtualization for High Performance Computing Environments, *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp.233–240 (Feb. 2013).

[19] 富樫荘太, 片山吉章, 桐村昌行, 瀧本栄二, 毛利公一：Linux の Cgroups における CPU throttling の性能評価，2013 年電子情報通信学会総合大会 情報・システム講演論文集, No.1, p.58 (2013).

[20] 松本亮介：博士学位論文 Web サーバの高集積マルチテナントアーキテクチャに関する研究，入手先 ⟨https://repository.kulib.kyoto-u.ac.jp/dspace/handle/2433/225954⟩, 京都大学 (May 2017).

[21] 松本亮介, 岡部寿男：スレッド単位で権限分離を行う Web サーバのアクセス制御アーキテクチャ，電子情報通信学会論文誌，Vol.J96-B, No.10 (Oct. 2013).

[22] 松本亮介，岡部寿男：mod_mruby：スクリプト言語で高速かつ省メモリに拡張可能な Web サーバの機能拡張支援機構，情報処理学会第 6 回インターネットと運用技術シンポジウム IOTS2013, pp.79–86 (Dec. 2013).

[23] 松本亮介，川原将司，松岡輝夫：大規模共有型 Web バーチャルホスティング基盤のセキュリティと運用技術の改善，情報処理学会論文誌，Vol.54, No.3, pp.1077–1086 (2013).

[24] 松本亮介，三宅悠介，力武健次，栗林健太郎：高集積

マルチテナント Web サーバの大規模証明書管理，情報処理学会研究報告インターネットと運用技術（IOT），Vol.2017-IOT-37, No.1, pp.1–8 (May 2017).

松本 亮介 （正会員）

2015 年京都大学大学院情報学研究科博士課程単位取得認定退学．同年 GMO ペパボ株式会社入社．2016 年よりペパボ研究所主席研究員．京都大学博士（情報学）．OS やサーバソフトウェア，インターネットの運用技術やセキュリティ等に興味を持つ．電子情報通信学会会員．

栗林 健太郎 （正会員）

1999 年東京都立大学法学部政治学科卒業．2002 年奄美市役所入所，2008 年株式会社はてな入社を経て，2012 年株式会社 paperboy&co.（現，GMO ペパボ株式会社）入社．2016 年よりペパボ研究所所長，2017 年より GMO ペパボ株式会社取締役 CTO．インターネットサービスの開発，技術経営等に興味を持つ．人工知能学会会員．

岡部 寿男 （正会員）

1988 年京都大学大学院工学研究科修士課程修了．同年京都大学工学部助手．同大学大型計算機センター助教授等を経て，2002 年より同大学学術情報メディアセンター教授．博士（工学）．2005 年より国立情報学研究所客員教授．インターネットアーキテクチャ，ネットワーク・セキュリティ等に興味を持つ．システム制御情報学会，日本ソフトウェア科学会，IEEE，ACM 各会員．