リクエスト単位で仮想的にコンピュータ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† 岡部寿男††

大規模な Web サービスの普及に伴い，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 リソースや Disk I/O 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当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事ができない．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事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，各リクエストが，特定のコンテンツに対する突発的なアクセス集中や，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．

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

RYOSUKE MATSUMOTO† YASUO OKABE††

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.

1. はじめに

大規模な Web サービスの普及や Web ホスティングサービス[1]の低価格化に伴い，企業だけでなく個人も 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，Web ホスティングサービス事業者は，いかにセキュリティを担保しながらも安定してアクセスを処理できるか，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一つのアプローチとして，限られたコンピュータリソースで高集積なマルチテナント環境を構築する事が求められてきている．セキュリティを担保しながらも，リソースを柔軟に制御し，効率よく高集積化できれば，高性能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．Web サーバに効率よ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で複数のホストを処理する必要がある．そのような環境で，大量のリクエストを安定して処理するためには，コンピュータリソースを多く消費するリクエストがあった場合，その処理が他のリクエスト処理に影響を与えないように制御する必要がある．さらには，管理者がそのようなリソース制御ルールを自由に記

† 京都大学 情報学研究科
Graduate School of Informatics, Kyoto University
†† 京都大学 学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

ⓒ 2013 Information Processing Society of Japan

述することができれば，多種多様なリソース制御の問題を解決する事ができると考える．しかし，既存の Web サーバの一般的なリソース制御[2][9][10]は，リクエストの同時接続数や単位時間当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事ができない．また，たった一つのリクエスト処理がサーバ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．さらに，Web サーバ運用時も，事前に適切な閾値設定が困難であり，サーバ負荷が高まった事を監視で気付いた後に対応するといった，後手に回りがちなアーキテクチャになっている．

一つのリクエストがコンピュータ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良い．また，管理者がそのようなリソース制御ルールを専用の Domain Specific Language（以降 DSL と呼ぶ）により，自由に記述できれば，柔軟なリソース制御が可能になる．そこで，本論文では，同一のサーバプロセス上で，リクエスト毎に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，各リクエストが，特定のコンテンツに対する突発的なアクセス集中や，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．また，リクエストを処理する時点で，そのリクエストがどのホストに対するリクエストなのかが識別できるため，ホスト単位でリソースを分離する事も可能である．

実装に関して，Linux（Linux Kernel 3.10）上で動作する Apache HTTP Server 2.4.6[3]に mod_mruby[4]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 cgroups[5]を用いたリソース制御モジュールを実装した．制御ルールの記述には，Ruby を用いた専用の DSL で記述できるようにした．

本論文の構成は以下の通りである．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精度評価を行い，5 章でまとめとする．

2. リソース制御アーキテクチャ

限られたコンピュータリソースで複数のホストをできるだけ高集積に管理・運用するためには，リソースの分離と権限分離が重要だと考える．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[2]を行った．その改善において，リソース分離のために，ファイルやホスト単位で同時接続数を制御したり，OS の負荷によってリクエストを拒否したりするような手法をとった．しかし，依然として，CPU や DISK I/O 等のコンピュータリソースは共有のため，特定ホストへのアクセス集中や，多くのリソースを消費するアプリケーションへのたった一つのリクエストによってリソースを専有し，他のホストに影響を与える問題があった．そのような問題を解決するためには，リクエスト単位でリソースを分離し，各リクエストが互いに影響を受けないようなリソース制御アーキテクチャが必要だと考える．また，リクエスト単位でリソース制御ができれば，任意のホストに対するリクエストを制御することにより，ホスト単位でのリソース分離も可能となる．

一方，権限分離においては，セキュリティとパフォーマンスがトレードオフになるという課題があったが，それらは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[6]によって，パフォーマンスを維持しつつセキュリティを担保する事ができた．

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．

2.1 Web サーバのリソース制御

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しながら，複数のホストでサーバのリソースを共有する構成が一般的である．これまで，Web サーバのリソース分離手法は，主に以下の 5 種類に分類される．

(1) KVM 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[11]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) chroot や Jail[12]環境のようにファイルシステム領域を限定して分離する手法
(4) 単一のサーバプロセスで一つのホストを扱う手法
(5) 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

ハードウェア資源が潤沢にあり，サーバの運用面やセキュリティ，及び，可用性を重視した場合は，手法(1), (2)等の，ホストそれぞれに対して，個別の仮想マシンやコンテナ等の仮想領域を割り当てる構成がとられてきた．しかし，これらはコンピュータリソースの面で非常にコストが高く，オーバーヘッドも大きいため，限られたリソースで，高集積にホストを収容するには不向きである．また，(3)のよう

ⓒ 2013 Information Processing Society of Japan

にファイルシステムを限定する方式がある．例えば，OS のシステム領域とほぼ同等の環境を OS 上に構築し，IP アドレスを個別に設定する．その環境で chroot することで，ファイルシステムを限定する．chroot 環境内部から OS のシステム領域に到達することはできないため，セキュリティ面で堅牢である．しかし，ホスト単位で chroot 環境を構築し，その環境毎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し高集積にはむいていない． (4)は，ホスト毎にサーバプロセスを起動させる方式で，(5)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をすることができる．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，依然としてサーバリソースは共有のため，一つのサーバプロセスがサーバリソースを占有すると，他のホストが影響を受ける問題がある．また，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．それに対し，(5)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，サーバプロセスがホスト数に依存しないため，高集積が可能である．しかし，(5)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)が必要である．また，(3)，(4)はホスト単位でリソース分離しようとしても，せいぜいプロセス毎に分離できる程度である．(5)は高集積な仮想ホス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の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．(5)の方式であってもリソースを適切に分離できるようにするためには，リクエスト単位でリソースを分離するようなアーキテクチャを考えればよい．そうすることで，(5)の方式であってもホストと対応するリクエストのリソースを制御する事により，ホスト単位でのリソース分離が可能である．また，(1)，(2)，(3)，(4)の方式であっても，特定のリクエストが他のリクエストに影響を与えないようなリソース制御も可能になる．

以降では，リクエスト単位，つまりは，プロセス上で細かくリソース制御を行うための技術について，Linux を例に言及する．

2.2 Linux のプロセスリソース管理技術

Linux には，cgroups[5]と呼ばれるプロセスのリソース管理技術がある．cgroups 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．cgroups 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ーフェイスを持っている．cgroups 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 10Mbps にしたり，デバイス I/O の Byte/sec や IOPS を任意の値に制御できる．(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．例えば，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することが容易になる．(4)は異なる名前空間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，再起動無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．

cgroups の機能を利用する事で, システム開発者はシステムリソースの割当，優先順位付，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 で述べた，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，リソース専有の問題を解決するためには，リクエスト処理単位で，リソースを分離する必要があると考える．これにより，一つのリクエスト処理が, 例えば CPU 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原理的に影響を与えない．また，管理者がリソース制御ルールを専用の DSL で自由に記述できれば，柔軟なリソース制御が可能になる．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ため，できるだけ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だと考える．

そこで，リクエスト処理時に，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ることで，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．図 1 に，リソース制御アーキテクチャの概要を

示す．図 1 のように，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，そのリクエスト処理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 resource_controller が, 専用の DSL で記述されたリソース制御ルールから設定値を取得する．そして，そのリソース設定値を元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．例えば，任意のリクエストに対し，最大の CPU 使用率は 10%でディスクへの書き込みは 5MB/sec に制御したいとする．その場合は，DSL の制御ルール記述にのっとり，その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，resource_controller は新規 DSL のルールを解釈し, ルール通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参加させた後，リクエストをそのリソース範囲内で処理する．処理後は，リソース領域からサーバプロセスを退避させて，レスポンスをクライアントに返し，次のリクエスト処理に備える．また，図 1 の resource_controller 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ーフェイスを用意した上で，そのインターフェイス上に実装する．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，resource_controller 自体の拡張・修正が容易となる．

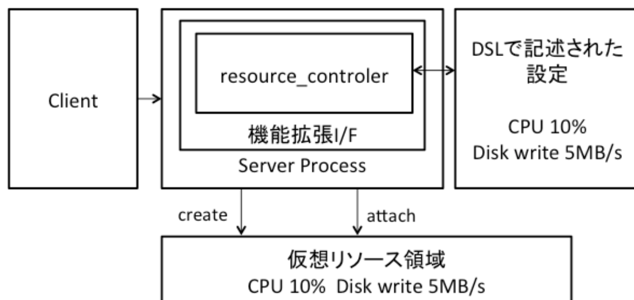

図 1 提案するリソース制御アーキテクチャの概要

上記のようなアーキテクチャを Linux（Linux Kernel 3.10）上で動作する Apache（Apache HTTP Server 2.4.6）に実装する事を考える．仮想リソース領域の作成には cgroups を利用した．2.2 で言及した(2)優先順位の機能を使い，CPU やトラフィック，I/O をリクエスト単位で任意のパラメータによりリソース分離を実現する. 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 サーバに組み込む事で, スクリプト言語で Web サーバの機能拡張が可能になる mod_mruby[4]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[7]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，高速かつ軽量に動作する．また，提案するアーキテクチャでは, リクエスト毎に DSL 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. mod_mruby 上に，cgroups から(2)優先順位機能を操作できる実装を追加し，そのリソース制御ルールを Ruby で記述できるようにした．図 2 に，Linux 上で実装する場合のアーキテクチャの構成を示す．Ruby の DSL で記述する設定は，ファイル，ディレクトリ，ホスト単位等，様々な条件によりリソース対象の記述が可能となっている．また，cgroups によるリソース領域は複数設定する事が可能で，リクエスト時に既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 DSL による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 CPU10%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.

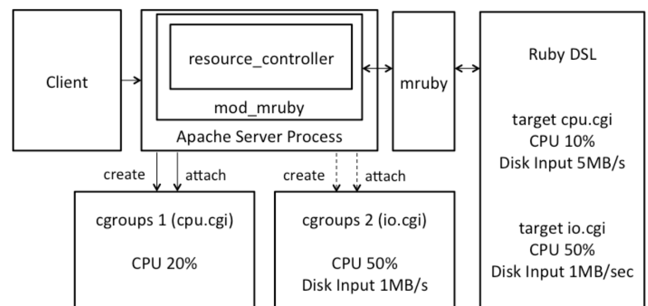

図 2 Linux の実装例

```
r = Apache::Request.new

if r.filename == “/path/to/cpu.cgi”
  cpu = Cgroup::CPU.new “cpu_group”
  if ! cpu.exist?
    cpu.cfs_qupta_us = 10000
  end
  cpu.create
  cpu.attach
end
```

図 3 リソース制御ルール記述例

図 3 のように, Ruby で記述可能な機能拡張インターフェイスによって，Ruby の言語仕様を元に，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，単なるリソース制御パラメータを設定するだけでなく，各種リソースの変化量から相関性を算出し，リソース制御のための適切な条件設定を表現することも可能だと考えている．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正規表現を, 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事

ⓒ 2013 Information Processing Society of Japan

で，サーバプロセスを再読み込みする事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，運用性も高い. 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事でより高速に動作させる事も可能である．なお，mod_mruby は軽量・高速に動作する事も本機構に適合しており，オーバーヘッドは少ない事が分かっている[4].

4. リソース制御アーキテクチャの精度評価

提案するリソース制御アーキテクチャの CPU 制御機能を組み込む事によるオーバーヘッドやリソース制御の精度評価を行った．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程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.

表 1　テスト環境

クライアント	
CPU	Intel Core2Duo E6600 3.06GHz
Memory	8GB
NIC	Realtek RTL8111/8168B 1Gbps
OS	Fedora 18
サーバ	
CPU	Intel Core i7-4770K 3.5GHz
Memory	32GB
NIC	Intel I217V 1Gbps
OS	Fedora 19
Middleware	Apache/2.4.6

まず，本アーキテクチャを導入する事で，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[8]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事によるボトルネックはほとんどないと考えられる.

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，単純なループを行う CGI を作成し，そのループ回数を変化させる事で，リクエスト処理時間を変化させた．また，ループ処理であるため，リクエスト処理時間のほとんどが CPU 使用時間と同じであると考えられる. その CGI 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 にリクエストを送信した．CGI 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事にする．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事になる．図 4 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 4 のように，1 リクエストの処理時間が 0msec から 6msec 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わかる．これはリソース制御の際に，リクエストがリソース制御対象であった場合に，仮想的に作成したリソース分離領域にプロセスを参加させる処理が，リクエスト処理時間とくらべて無視できない程度の処理時間となり，ボトルネックになっているためだと考えられる．一方で，6msec 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%に制御できている事がわかる.

以上より，CPU をリソース制御する場合において，リクエスト処理時間が，リソース制御アーキテクチャによってリソースを分離するための処理時間よりも十分大きい場合において，本手法が有効であることが分かった．また，単一のリクエストが大きく CPU リソースを占有するような場合においても，本手法が有効に機能すると考えられる.

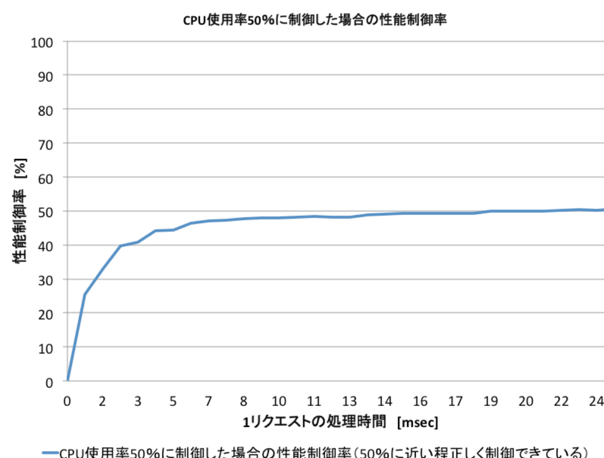

図 4　リクエスト処理時間の違いによるリソース制御精度

5. むすび

本研究では，限られたリソースで高集積にホストを収容する事際に，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，サーバプロセス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，一つのリクエストがコンピュータリソースを大きく専有しようとしても，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・拒否する事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，CPU リソースだけでなく，Disk I/O におけるリソース制御の評価や，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考えていく予定である．さらに，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，汎用的なインターフェイ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) 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, Oct 2009.
2) 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, Mar. 2013.
3) The Apache Software Foundation, "Apache HTTP Server Project", http://httpd.apache.org/.
4) 松本亮介, 岡部寿男, 組み込みスクリプト言語 mruby を利用した Web サーバの機能拡張支援機構, 情報処理学会研究報告 Vol.2012-IOT-18, No.6, 2012 年 6 月.
5) 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, Vol.1, p.58(D-6-11), 2013.
6) 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,pp.-,Oct. 2013.
7) nginx, “nginx”, http://nginx.org/ja/.
8) The Apache Software Foundation, "ab - Apache HTTP server benchmarking tool", http://httpd.apache.org/docs/2.2/programs/ab.html.
9) David J, “mod_limitipconn.c”, http://dominia.org/djao/limitipconn.html.
10) The Apache Software Foundation, "Server – Wide Configuration Limiting Resource Usage", http://httpd.apache.org/docs/2.2/en/server-wide.html#resource.
11) Kolyshkin, Kirill. "Virtualization in linux." White paper, OpenVZ (2006).
12) Kamp, Poul-Henning, and Robert NM Watson. "Jails: Confining the omnipotent root." Proceedings of the 2nd International SANE Conference. Vol. 43. 2000.