情報処理学会論文誌 Vol.54 No.3 1077–1086 (Mar. 2013)

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

松本 亮介[1,a)] 川原 将司[1] 松岡 輝夫[1]

受付日 2012年6月28日, 採録日 2012年12月7日

概要：近年，AmazonEC2 に代表されるクラウドの台頭にともない，ホスティングサービスの低価格化が進んでいる．そこで，我々はリソースを必要最小限に抑えるために，アクセスのあったホスト名でコンテンツを区別し，単一のプロセスで複数のホストを処理する方式である Apache の VirtualHost 機能を用いて，ホスト高集積型の Web ホスティング基盤を構築した．しかし，その基盤運用は困難で，運用技術の改善とパフォーマンス劣化の少ないセキュリティ機構の設計が課題であった．本稿では，VirtualHost 採用と大規模対応にともなって生じる運用面とセキュリティ上の課題を明確化し，新しい Apache モジュールの開発と suEXEC の改修によって，それらを解決する手法を提案する．本手法によって，信頼性と運用性の高い大規模 Web ホスティング基盤を構築できる．

キーワード：ホスティングサービス，Apache，セキュリティ，運用技術，大規模

Improvement of Security and Operation Technology for a Highly Scalable and Large-scale Shared Web Virtual Hosting System

Ryosuke Matsumoto[1,a)] Masashi Kawahara[1] Teruo Matsuoka[1]

Received: June 28, 2012, Accepted: December 7, 2012

Abstract: As the recent deployment of Cloud Computing like AmazonEC2, the price of hosting services is falling rapidly. Therefore, we had developed the Web based hosting system to handle huge number of hosts (about 2,000 hosts by a server) using Apache VirtualHost that single process manages multiple hosts and distinguish contents by the hostname with the access for minimizing the use of resources. However, it was difficult to manage the Web based hosting system, and the problems of the system were improvement of operation technology and designing the security mechanism with little performance degradation. In this paper, we clarify the problems of security and operation technology with installing VirtualHost for building a large-scale system, and we propose the approach solving them by developing the new Apache modules and enhancing suEXEC. This approach is help to building a large-scale, reliable and operationally-effective Web based hosting system.

Keywords: hosting service, Apache, security, operation technology, large-scale

1. はじめに

近年，インターネットの普及にともない，自社でインフラ設備を持つことができない企業は，Web やメールの機能を，レンタルサーバと呼ばれるホスティングサービスを介して利用する機会が増大している．その結果，ホスティング企業の間では価格競争が起き，AmazonEC2 [1] に代表されるようなクラウドの台頭とともに，ホスティングサービスの低価格化が進んでいる．

これまで，弊社では，Apache HTTP Server [2]（以降 Apache とする）の VirtualHost [3] を利用することで，メモリ 4 G 程度のサーバ 1 台で約 2,000 ホスト，3 サーバで約 6,000 ホストの高集積型 Web ホスティング基盤の構築が達成できている．しかし，その収容数での運用コストは

[1] ファーストサーバ株式会社
Firstserver, Inc., Osaka 541–0052, Japan
[a)] matsumoto_r@net.ist.i.kyoto-u.ac.jp

非常に高く，また，パフォーマンス劣化の少ないセキュリティ機能を実装することは困難であった．運用性においては，VirtualHost は単一のサーバプロセスで複数の仮想ホストを処理しているため，ある仮想ホストがアクセス集中等によって高負荷になった場合，すべての仮想ホストが影響を受ける．さらに，仮想ホストの数が膨大になってくると，仮想ホスト追加や設定変更等によるプロセスの再読み込み処理の影響が大きくなる．また，セキュリティにおいても，コンテンツを仮想ホスト単位で権限分離するためには，サーバプロセスを root で上げて，コンテンツ実行時にコンテンツの権限にプロセスを変更して，実行後はサーバプロセスを破棄する必要があった．この構成は，たとえば Apache のサーバプロセスそのものの脆弱性を突かれた場合において，任意のコマンドを実行できる脆弱性である場合，root 権限でコマンドを実行できるため非常に危険性が高い．また，サーバプロセスをコンテンツ処理ごとに生成，破棄しなければならないという点で，アクセス集中に非常に弱い構成であった．また，収容数よりも運用面やセキュリティを重視した場合，OS 上で chroot や仮想マシン等を利用して，複数の独立した領域を構築し，各種サービスを領域内部に閉じ込め，稼動させる構成が使われてきた．しかし，大規模を想定した場合，サーバ単位でのプロセス数が多くなるためリソース効率が低い．また，ユーザ専用のサーバプロセスとハードウェアが紐付くため，複数の Web サーバで共有ストレージ上のコンテンツを共有し負荷分散することが難しく，汎用性が低い．このように，大規模に対応した Web ホスティング基盤を構築するにあたって，いかにパフォーマンスを劣化させずにセキュリティを担保し，また，運用者の負担を低減するかが，大きな課題となっていた．

今後，クラウドサービスや低価格ホスティング等が主流になっていくことを考えると，限られたリソースで最大限のセキュリティと運用性を担保し，汎用性のある大規模対応基盤を構築しなければならない．そこで，複数のドメインを単一のサーバプロセスで扱い，リソースを必要最小限に抑えることのできるアーキテクチャを採用することとした．Web サーバ構築には Apache の VirtualHost を利用した．弊社の実績である 1 サーバ 2,000 仮想ホストを目標にした場合，VirtualHost を用いて汎用性のある大規模 Web ホスティング基盤を開発する場合，運用面やセキュリティ上の課題が生じる．それらの課題を解決するために，運用面においては新たに Apache モジュールを開発して機能を拡張し，セキュリティを高めるために suEXEC に対して独自のセキュリティ機構を実装した．その結果，大規模化において課題となっていたセキュリティと運用技術を改善できた．

本稿では，汎用性のある大規模 Web ホスティング基盤を開発するにあたり，VirtualHost を採用することで生じる，セキュリティや運用面の課題を解決する手法を提案する．2 章では Web ホスティングの構成や，運用面とセキュリティを重視した場合に汎用性が低くなる点について述べる．3 章では単一のサーバプロセスで複数ドメインを扱った場合の運用面の課題，4 章ではセキュリティの課題について言及する．5 章でそれらの課題を解決するためのアプローチ，6 章でパフォーマンスについて述べ，7 章でむすびとする．

2. Web ホスティングシステムの構成

Web ホスティングとは，複数の管理者でサーバのリソースを共有し，それぞれの管理者のドメインに対して HTTP サーバ機能を提供するサービスである．サービスを提供された管理者が，コンテンツを置く管理領域をホストと呼び，ホームページアクセス等にホスト名を利用して識別する．

これまで，サービス提供側が Web ホスティングシステムを構築する手法は，主に以下の 4 種類に分類される．

(1) Xen や VMware 等の仮想マシンで管理者領域を分ける手法
(2) chroot 環境のように OS 上に複数の独立したサーバ環境を用意し管理者領域を分ける手法
(3) IP アドレスやポート単位で複数のホストに個別のプロセスを用意して起動させる手法
(4) 単一のプロセスで複数のホストを扱う手法

サーバの運用面やセキュリティを重視した場合は，手法 (1), (2), (3) 等の，管理者それぞれに対して，個別のサーバプロセスや仮想マシンを割り当てる構成がとられてきた．たとえば，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．**図 1** に，運用面とセキュリティを重視した場合に，(2) の手法を採用した構成の概要図を示す．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．また，chroot 環境は OS のシステム領域とほぼ同等のライブラリ群を任意に

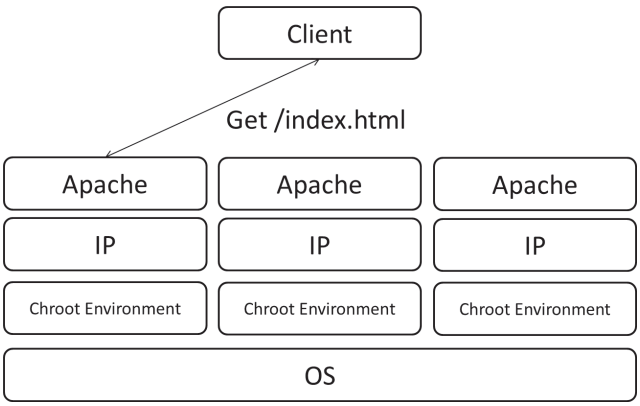

図 1 運用面とセキュリティを重視した構成
Fig. 1 The configuration for operation-friendly and secure system.

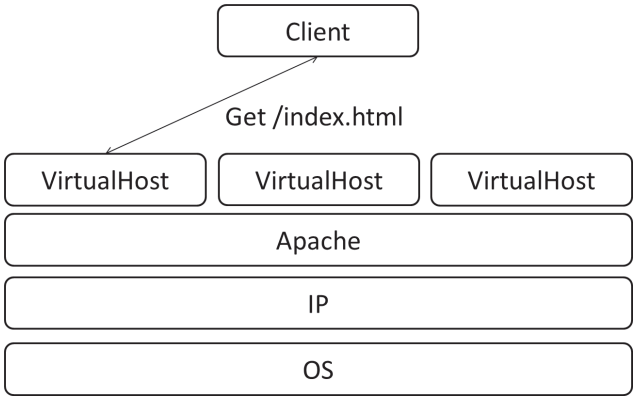

図 2 単一プロセスで複数ホストを扱う構成
Fig. 2 The configuration that single process manages multiple hosts.

配置できるため，ソフトウェアやアプリケーションも OS とは区別された領域で独立して起動することができる．個別の設定も OS と同様に扱うことができるため，サーバプロセスの設定をすべて利用者専用に設定することができ，運用面でも可用性が高い．さらに，不必要なコマンドやライブラリを配置しないことで，セキュリティを高めることもできる．

既存の研究として，(3) の手法を利用して，複数のサーバプロセスをそれぞれ異なるユーザ権限で起動する手法がある [4]．各 chroot 環境等で，ホスティング利用者単位でユーザプロセスを起動させると，そのサーバプロセスが 1 つの物理サーバに紐付く．ここで (1)，(2)，(3) の手法では問題が生じる．たとえば，大規模な Web ホスティング基盤において，ロードバランサを使い，6 台の物理サーバで 12,000 のホスティング利用者領域を処理することを想定する．その場合，コンテンツは共有ストレージを利用し，複数の物理サーバそれぞれを同じ構成にして，コンテンツの処理を負荷分散する．こうすることで，利用者が増えた場合や，処理性能が劣化した場合は，さらに同一のサーバを増やすことで容易にスケールアウト型のリソース追加ができる．しかし，物理サーバに紐付く各ホスティング利用者のユーザプロセスや仮想マシンが起動する以上，共有ストレージによる負荷分散のためには，6 つのサーバを同じ設定にする必要がある．つまり，同様にそれぞれ 12,000 のユーザプロセスや仮想マシンを起動させる必要があり，リソース効率が大幅に低下する．また，2,000 のユーザプロセスに分けてサーバに収容すると，スケールアウト型の負荷分散や容易なリソース追加ができないため，大幅に運用性が低下する．

一方，本稿で採用したのは (4) の手法で，単一のサーバプロセスで複数のホストを処理する構成の場合，VirtualHost を用いることで，ユーザプロセスに依存しない構成をとることができる．**図 2** にその構成を示す．VirtualHost では，アクセスのあったホスト名に対応したドキュメントルートにアクセスするように Apache 内部で制御する．そのため，複数のホストに対して 1 つのユーザ権限でサーバプロセスが起動していればよい．以上の特徴から，ホスティング利用者に依存しないユーザプロセスが起動でき，物理サーバとも紐付かないサーバ設定が可能となる．その結果，複数台の物理サーバを容易に同一の環境にでき，リソースも最小限に抑えることができるため，共有ストレージを用いた効率の良い負荷分散構成も構築できる．

しかし，大規模な共有型 Web ホスティングを想定した場合では，VirtualHost を採用することで，運用面とセキュリティの課題が生じる．そこで，3 章，4 章では現状で考えられる課題について述べる．

3. VirtualHost における運用面の課題

ホスティングシステムを提供する側が，サーバを運用管理する工数は，インターネットの普及にともない，日々増加してきている．そこで，効率的にサーバを運用することが重要であり，この点で VirtualHost を採用すると，多くの課題が生じる．

3.1 高負荷ホストの特定が困難

Web サーバはしばしば，負荷のかかる CGI [5] や PHP が実行される．PHP の実行方式においては，Apache にモジュールとして組み込み，Apache のサーバプロセス内部で実行する方式（DSO [6] 版 PHP）と，モジュールとして組み込まず新たに別のプロセスを生成して，そこで実行する方式（CGI 版 PHP）がある．DSO 版 PHP で実行された場合は，プロセス情報を取得すると，プロセス名はスクリプトファイル名ではなくサーバプロセス名となる．そのため，サーバプロセスがリソースを大量に消費していた場合は，どのホストのどの PHP スクリプトであるかを特定することが困難である．また，CGI 版 PHP と suEXEC を利用した場合は，プロセス上で uid，gid [7] を特定することはできるが，プロセス名としては php-cgi として表示されるため，高負荷時等，迅速に対応しなければならない状況において，該当の PHP スクリプトを正確に特定できない．以上より，高負荷状況になった場合に，どのスクリプトがどの程度のリソースを消費しているかを迅速かつ適切に調査できる仕組みが必要である．

3.2 ホスト単位でのチューニングが困難

chroot 環境や仮想マシンで Apache を起動している場合は，ホストごとにサーバプロセスが起動しているため，サーバが持つすべての設定を利用することができる．しかし，VirtualHost を利用した構成の場合，Apache の仕様上 VirtualHost 単位での設定は限られている．たとえば，高負荷ホストへの最大同時接続数を設定するために非常に重要な MaxClients は VirtualHost 単位で設定することができ

ない．また，VirtualHost 上でコンテンツ単位の同時接続数を柔軟に設定する方法がないのも高負荷ホストのチューニングを困難にしている．

3.3 リソース変化に対応したチューニングが必要

VirtualHost は複数のホストを 1 つの Apache で管理しているため，サーバプロセスが高負荷で停止してしまうとすべてのホストの機能が停止する．そのため，サーバプロセスが高負荷で落ちないようにチューニングすることが必要となる．NAS や SAN によるストレージの統合化にともない，CPU 使用量，特にファイルシステムとの I/O が大量に発生して高負荷になる場合が多い．Apache のデフォルトのチューニング設定としては，メモリやプロセス数等があるが，CPU のチューニング設定は限られている．CPU のチューニングは，RlimitCPU がある．これは指定した CPU 使用時間を超過した場合，Apache の処理ではなく Kernel の ulimit の機能で処理が強制的に切断される．そのため，契約等が発生するようなサイトにおいては信頼性が低くなる．以上より，サーバ自体の負荷状況に合わせて，クライアントからリクエストを受けた際に，レスポンスを返すための処理を継続するかを判断する仕組みが必要だと考えられる．

3.4 新規設定反映時はすべてのホストに影響

3.3 節で述べたとおり，Apache が停止すると，すべてのホスト機能が一時的に停止する．そのため，大規模化にともなう設定の増加を考慮すると，できる限り Apache のリロードやリスタートを実施しないようにするべきである．そこで，新規ホストの追加設定やチューニングを行う場合に，Apache のリロードを実施しないでよいようにするため，mod_vhost_alias [8] を用いる手法がある．mod_vhost_alias には Dynamically Configured Mass Virtual Hosting（以降 DCMVH とする）という設定記述方法がある．本来，ホスト追加時には新規ホスト用の VirtualHost 設定を追加で記述する．しかし，VirtualHost の設定はホスト名やドキュメントルート名等が異なるだけで，その他の設定は同じ場合が多い．そこで，DCMVH の設定記述法を利用すると，ドキュメントルートにホスト名を含んだパスになるようにディレクトリを作成しておけば，VirtualHost の設定においてパスのホスト名部分を変数で記述することができる．その記述によって，VirtualHost の設定を 1 つ書いておけば，アクセスのあったホスト名で設定を動的に読み替え，該当のドキュメントルートにアクセスできるようになる．また，設定の数がホストの数に依存しないため設定読み込みの負荷も少なくできる．しかし，VirtualHost ごとに個別の設定が必要である suEXEC の設定を，DCMVH では動的に扱うことができないという問題や，VirtualHost ごとに環境変数に保存されるドキュメントルートが，変数を読み替えたパスにならないという問題がある．

以上が，VirtualHost を用いて，大規模対応の Web ホスティング基盤を構築する際に生じる運用面の課題である．

4. VirtualHost のセキュリティの課題

一般的な VirtualHost はサーバプロセス権限ですべてのリクエストを処理する必要があり，コンテンツファイルやディレクトリの権限をサーバプロセス権限で操作可能にしなければならない．そのため，他ホスト領域を覗き見できる．**図 3** で，他ホスト領域のスクリプトファイルを覗き見するための一般的な仕組みとパーミッション設定を示す．図 3 では，index.cgi を Apache の権限である uid500, gid101 で実行する．/var/www/hosts/fuga2.com/ ディレクトリは gid101 からの読み取り権限がある．さらに，ディレクトリ配下のホスト領域内部のファイル群は Apache 権限でアクセスできるように全ユーザに読み取り権限がある．そのため，index.cgi 内でシェル等の外部コマンドを実行することで db.cgi のソースコードを閲覧できる．そこで，CGI 実行時に利用できる suEXEC [9] を用いて，コンテンツの権限で CGI を実行し，適切に各ホスト領域の権限を設定することで，覗き見できないようにする方法がある．以降で，その他，VirtualHost を利用することで生じるセキュリティ上の課題を明らかにする．

4.1 システム領域や他ホスト領域の覗き見

VirtualHost を採用した構成では，一般的に OS のシステム環境で Apache を起動するため，Apache のユーザ権限および一般ユーザで閲覧可能な/etc 等のシステム領域を覗き見することができる．

4.2 suEXEC 採用にともなう CGI 版 PHP の課題

Apache の suEXEC 機能を用いると，VirtualHost を採用していても，他ホスト領域を閲覧できなくする構成を

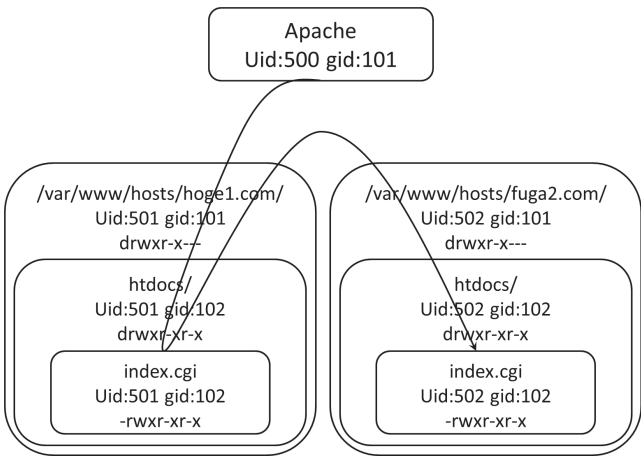

図 3 他ホスト領域の覗き見
Fig. 3 Peeping at other host.

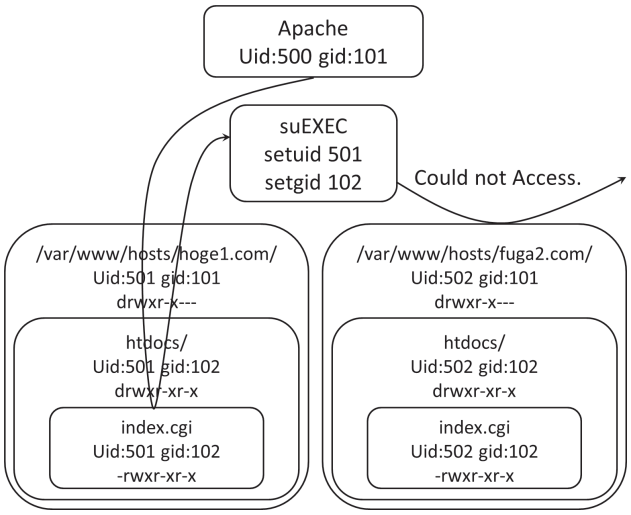

図 4 suEXEC の利用例
Fig. 4 Example for suEXEC.

とることができる．**図 4** に suEXEC の利用例を示す．図 4 では，図 3 と同様のパーミッション設定をしている．suEXEC を採用すると，クライアントから CGI にアクセスがあった場合，Apache によって CGI の実行処理を suEXEC に依頼する．suEXEC は index.cgi を実行する際に，index.cgi の権限である uid501，gid102 を取得する．そして，プロセスの権限を変更するシステムコール（以降 setuid，setgid とする）を実行して，プロセスの権限を変更し，CGI を実行する．そのため，uid501，gid102 の権限では，/var/www/hosts/fuga.com/配下での読み取り権限である uid502，gid101 がない．このように，suEXEC を採用することで，他ホスト領域にアクセスすることができず，db.cgi の閲覧もできない．

セキュリティを高めるために，VirtualHost において suEXEC と同等の機能は必須となる．しかし，suEXEC を利用するためには，各種スクリプト実行形式として CGI 版である必要がある等，Web ホスティングとしてのサービス仕様において様々な制約がある．制約の中，サービス仕様とセキュリティの関係をどのように解決し最善の選択肢をとるかがポイントとなる．以降で suEXEC 採用における制約について述べる．

4.2.1 SuexecUserGroup 設定が必要

suEXEC を採用するためには，仕様上，ホスト単位で SuexecUserGroup という設定を記述する必要がある．SuexecUserGroup で記述された uid と gid が，実行対象の CGI とパーミッションが一致するか確認し，そのうえで CGI を実行する際に setuid，setgid することでセキュリティを高めている．しかし，3.4 節で述べたとおり，mod_vhost_alias では suEXEC の設定の uid と gid を動的に扱う記述がないため，httpd.conf 上にバーチャルホストごとに静的に uid と gid を記述しなければならない．そのため，新規バーチャルホストを追加するたびに，Apache のリロードが必要になるという課題が残る．

4.2.2 CGI 版 PHP の強制

DSO 版 PHP は Apache モジュールとして組み込まれるため，CGI 版 PHP と比べパフォーマンスが大幅に高くなる．また，シェバン行 [10] の記述や権限を細かく設定する必要がない．しかし，Apache に組み込まれて実行される以上，基本的には Apache 権限で実行されるため，図 3 と同様の他ホスト覗き見問題が生じる．

これまで，DSO 版であっても suEXEC と同様の機能を果たすセキュリティ機構が提案されている．4.3 節で，DSO 版に関する既存のセキュリティ機構を述べる．

suEXEC を採用するためには，CGI 版を利用しなければならず，シェバン行の記述や適切な実行権限設定をホスティング利用者に強制する必要がある．しかし，一般的な PHP の利用シーンとして，PHP はシェバン行を記述せず html に組み込む形式になっている．そのため，シェバン行の記述や実行権限設定は，サービス仕様上の問題となる．

4.2.3 CGI 版 PHP と mod_suphp

mod_suphp [11] を利用すると，シェバン行の記述や実行権限設定をホスティング利用者に強制する必要なく，suEXEC と同様に，CGI や CGI 版 PHP をユーザ権限で実行できる．しかし，suEXEC と同様 VirtualHost 単位で uid，gid を設定ファイル内で指定する必要があり，mod_vhost_alias でも動的に扱えない．コンパイル時にオプションを指定することで，uid，gid の設定を省略することができるが，mod_suphp 設計者はそれらをセキュリティ的に推奨していない．また，設定を省略した場合は，他ホスト領域への覗き見を防ぐことができるが，4.1 節で述べた，システム領域の覗き見問題を解決できていない．

4.2.4 CGI 版 PHP と mod_actions

mod_actions [12] を利用すると，CGI 版 PHP であっても，PHP スクリプトをシェバン行や実行権限を設定しなくても実行できるラッパプログラムに渡す設定ができる．この設定により，suEXEC の機能を利用したうえで PHP を実行できる．しかし，ラッパプログラムを Apache や各ホストの権限からアクセス可能なディレクトリに安全に配置する必要があり，設定や構成が煩雑になりがちである．また，ラッパプログラムは URL からアクセスできる領域に配置する必要があり，顧客の URL を一部専有するのも問題がある．4.1 節で述べた，システム領域の覗き見問題も解決できない．

4.3 既存の DSO 版 PHP に関わるセキュリティ

4.2.2 項において，通常 DSO 版 PHP は共有型 Web ホスティング基盤で安全に利用することができないと述べた．これまで，DSO 版 PHP を安全に利用するためのセキュリティ機構が提案されている．それらに関して，本稿での見解を述べる．

4.3.1 DSO 版 PHP のセーフモード

他ホストの領域が見えないようにするため，PHP にはセーフモードという機能がある．セーフモード機能を利用すると，DSO 版 PHP であっても他のファイルを覗き見できない．しかし，PHP 特有のセキュリティ機構であり汎用性が低いという問題や，共有サーバ上の OS やファイルシステム上のセキュリティ問題を PHP アプリケーションのレイヤで解決しようと試みるのはアーキテクチャ上正しくないと考えられる．また，PHP5.3.0 では使用が非推奨となり，PHP5.4.0 では削除された [13]．

4.3.2 DSO 版 PHP と mod_suid2 や mod_ruid

DSO 版 PHP を採用した場合でも，mod_suid2 [14] や mod_ruid [15] というモジュールを利用すると，他ホスト領域の閲覧を防ぐことができる．mod_suid2 や関連研究 [16] では，Apache のサーバプロセスを root 権限で起動しておき，リクエストを処理するたびにユーザ権限に setuid，setgid する．これによって，Apache の権限とは別の権限でプロセスを実行できるため，suEXEC と同様，他ホスト領域を閲覧できなくなる．しかし，処理後はサーバプロセスが一般ユーザ権限であるため，権限を元の root 権限に戻すことができない．そのため，setuid，setgid されたプロセスをコンテンツ処理後に破棄する必要がある．その結果，プロセスを再利用できず，DSO 版を利用していたとしても，suEXEC よりもパフォーマンスが大きく低下する．

Apache のプロセスがユーザ権限で起動している場合は，setuid や setgid を実行することができない．アクセスするクライアントが正確に特定できるようなイントラネットの環境において，ユーザ権限であっても setuid 等を実行可能にする手法 [17] は存在するが，不特定多数のクライアントには対応していない．そこで，mod_ruid や関連研究 [18] を利用すると，一時的にユーザ権限で起動しているサーバプロセスに，root の特権を細分化した Capability [19] と呼ばれる機構のうち，CAP_SETUID，CAP_SETGID の特権を与えられる．その結果，特権を与えられたサーバプロセスは，root でなくても setuid，setgid を実行可能となる．その後，mod_suid2 同様に Apache のサーバプロセス自体を任意の uid，gid に権限変更してから処理を実行し，再度，元の uid，gid に戻す．この仕組みによって，DSO 版 PHP であっても，PHP スクリプトは他ホスト領域を閲覧できない．また，実行後でも，元のサーバプロセスの権限に戻すことで，プロセスの再利用も可能にしているため，DSO 版のパフォーマンスを維持できる．しかし，このようなプロセスは，root のようにすべての権限を持たないものの，setuid，setgid を実行できる Capability を保持している．たとえば，関連研究 [20] のようにサーバプロセスのアクセスできるファイルを，SELinux [21] 等のセキュア OS を用いて制限したとする．しかし，アプリケーションの脆弱性をつかれ悪意のある者にのっとられた場合，setuid，setgid による権限変更を利用し，他ホスト領域のファイル閲覧や変更，および，不正プログラムの配置や配布等が可能となる．サーバプロセスに権限を変更できる特権の保持を許すことは，同時に数多くの脆弱性を許すことになる．このように，パフォーマンス向上のための，サーバプロセスのアクセス制御を設定後，再度解除するというアプローチは，共有型の大規模 Web ホスティング基盤のセキュリティを考えるうえで非常に危険であり，脆弱性をつかれた場合の利用者や閲覧者への被害は甚大であると考える．そのため，本稿では suEXEC 程度のパフォーマンスを満たしていれば，このような危険性は除外すべきであると考えた．そこで，setuid，setgid した後に CAP_SETUID，CAP_SETGID の Capability を放棄し，処理後にプロセスを復帰できないようにすれば安全であるが，やはりプロセスが再利用できなくなり，mod_suid2 同様，パフォーマンスは著しく低下する．

以上から，現状，本稿の要件を満たす Web バーチャルホスティング基盤において，モジュールとして Apache に組み込まれる DSO 版 PHP を安全に利用する手法は存在しないと考えられる．以上より，汎用性のある大規模共有型 Web ホスティング基盤を構築する際に生じる課題を明らかにできた．

5. 課題解決へのアプローチ

Apache の VirtualHost 機能を用いて大規模対応基盤を構築する場合に，運用面とセキュリティ面の課題が生じることを，3 章と 4 章で明らかにした．既存の手法において，これらをすべて同時に解決している手法は存在しない．そこで，明らかにした課題を解決するための手法を提案する．

5.1 運用面の課題解決

5.1.1 リクエスト毎でのリソース消費量測定機能

3.1 節で，プロセス情報からの高負荷対象の特定や，リソースの測定は困難だと述べた．そこで，クライアントから Apache へのリクエストがあり処理を行ってからレスポンスを返すまでに，プロセスが消費したリソース量を測定するモジュールを開発した．このモジュールによって，実行されたプログラムが任意のシステム CPU 時間，ユーザ CPU 時間，メモリ使用量を超えていた場合に，プログラムのフルパス，バーチャルホスト名，システム CPU 使用時間，ユーザ CPU 使用時間，メモリ使用量を計測しファイルに記録する．運用者は VirtualHost を気にすることなく，高負荷ホストやスクリプトをリアルタイムで調査，検知し適切にチューニングすることができる．また，この機能はコンテンツを設置する側の開発者にとっても，スクリプトのリソース消費量を知るうえで役立つと考えられる．

5.1.2 コンテンツへの同時接続数チューニング

人気サイトでは，しばしば同一スクリプトやメディア，ホ

ストに対して，複数のクライアントから同時に大量のアクセスが発生する．その結果，高負荷となってサーバのレスポンスが大きく低下し，場合によってはサーバダウンへとつながる．しかし，3.2 節で述べたとおり，VirtualHost 単位での設定は限られている．そこで，高負荷ホスト全体や高負荷スクリプトへのアクセスをチューニングするために，リクエスト対象への同時接続数を設定するモジュールを開発した．設定対象は，任意のホストやファイル名，ファイルへのフルパス，任意のディレクトリ，正規表現にマッチしたファイルやフォルダ等である．同時接続数の設定値を超えた場合は，リターンコード 503（Service Unavailable）を返す．また，同一クライアント IP からの同時接続数も設定できるようにした．

5.1.3 リソース変化に対応したチューニング機能

3.3 節において，なんらかの原因で親プロセスが停止した場合は，すべてのホストが停止することになると述べた．最も多い原因として，サーバ高負荷によるプロセス停止があげられる．そこで，CPU 処理や I/O 処理の負荷の目安となるロードアベレージ [22] の数値に着目し，リクエストを受けた後，ロードアベレージの数値によってレスポンスを返すかどうかを Apache が判断するモジュールを開発した．ロードアベレージの数値を閾値として設定し，その数値を超えた場合はリターンコード 503 を返すことで安全に処理を中断させる．設定対象は 5.1.2 項のモジュールと同様で，1 分平均のロードアベレージを閾値に設定できる．

5.1.4 mod_vhost_alias と.htaccess による動的設定

DCMVH で設定した際には 3.4 節で述べた課題が生じる．そこで，mod_vhost_alias，suEXEC を改修することで解決した．環境変数に保存されるドキュメントルートが，変数を読み替えたパスにならない問題があったが，それらを正しいドキュメントルートで保存されるようにした．また，mod_vhost_alias で suEXEC の動的設定ができないという問題は，各 VirtualHost の suEXEC のユーザ名とグループ名を同一のダミーとして設定し，suEXEC プログラム内部で実行ファイルからユーザ名とグループ名を解釈できるように suEXEC を改修した．5.2 節で suEXEC 改修の詳細を述べる．この改修と DCMVH によって，すべての VirtualHost の設定を 1 つの設定に書き直すことができた．また，.htaccess と同じ機能を持った各ホスト専用のチューニングファイルを用意した．5.1.1，5.1.2，5.1.3 項で述べた Apache モジュールは，チューニングファイルに設定を記述することで Apache のリロードなくホスト単位で新たな設定を反映させることができる．

以上の手法で，ホスト数に依存した煩雑な設定なく，また，Apache のリロードをせずにチューニング設定や新規ホスト設定を反映させることができた．その結果，大幅に運用性とサービス性が向上したといえる．

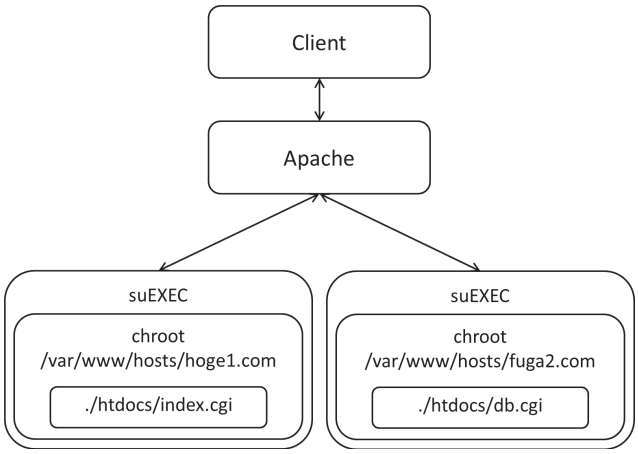

図 5 suEXEC 実行時に chroot
Fig. 5 chroot on suEXEC.

5.2 セキュリティの課題解決

4 章で述べた課題を解決するために，suEXEC を改修した．以降で，suEXEC の改修内容を詳細に述べる．

5.2.1 suEXEC 時にホスト領域で chroot する機能

システム領域や他ホスト領域を覗き見できないように，suEXEC 時に各ホスト環境で chroot してからスクリプトを実行するように改修した．**図 5** は suEXEC プログラム内部で chroot する仕組みの概要図である．これにより，CGI や PHP はホスト領域内の隔離された領域で実行されるため，利用しているホスト領域外のファイルを閲覧することができない．

5.2.2 php は suEXEC 内部でインタプリタより実行

4.2 節で述べた課題を解決するため，.php ファイル実行時は，suEXEC プログラム内部で PHP インタプリタに直接渡し実行するように改修した．これによって，.php のシェバン行や実行権限の有無をホスト管理者が意識することなく，CGI 版 PHP を suEXEC で実行できる．この改修によって，複雑な構成や複数のモジュール，ラッパ等を組み込む必要がないという点で非常にシンプルな構成になっている．また，ホスティング利用顧客が DSO 版 PHP で PHP スクリプトを扱うために，シェバン行を記述しなかった場合でも，suEXEC 内部で CGI 版 PHP として実行されるため，CGI 版か DSO 版かをホスティング利用顧客が気にする必要がない．セキュリティを担保した状況での，DSO 版 PHP と CGI 版 PHP のパフォーマンスに関しては，6 章で言及する．

5.2.3 SuexecUserGroup 設定無効化

5.1.4 項で述べたとおり，ホストごとの SuexecUserGroup パラメータに記述するユーザ名とグループ名はすべて同一のダミー名を設定し，実行ファイルが存在するディレクトリの uid と gid に suEXEC プログラム内部で読み替えるように改修した．また，読み替えの際には，root の uid と gid であった場合はエラー処理を実装した．実行対象プロ

グラムと実行対象プログラムが存在するディレクトリ，ドキュメントルートディレクトリの uid と gid がそれぞれ互いに 1 つでも異なっていた場合もエラーを返すように実装した．顧客ホスト領域ごとに chroot することを考慮に入れると，suPHP の個別設定を省略する方法よりも，PHP プログラム経由でシステム領域を覗き見できないという観点から，セキュリティ面で優れており，システム領域を覗き見できるという問題や複雑な実行権限設定も同時に解決したと考えられる．

6. パフォーマンス

単一のサーバプロセスで複数のホストを処理する方式をとった場合に，サーバプロセスをリロードすることなく，動的コンテンツ実行時のセキュリティを担保するためには，DSO 実行方式と mod_suid2 または mod_ruid2 を採用し，サーバプロセスそのものを動的コンテンツ処理ごとに破棄する必要があると述べた．サーバプロセスそのものの生成破棄は，CGI 方式における fork されたプロセスの生成破棄と比べて非常に処理に時間がかかる．しかし，本手法によってそれを suEXEC 内部で動的コンテンツファイルの権限情報から動的に権限分離を行うことで，CGI 実行方式程度のパフォーマンスに抑えることができたと考えられる．以上から，CGI 実行方式，サーバプロセスを生成破棄する方式についてプログラム実行方式の差についてのパフォーマンスの比較を行うことで本手法の優位性を計るには十分だと判断した．実験方法は，使用する言語は CGI 実行方式および DSO 実行方式両方で実行可能な PHP とし，実行方式の差をより顕著にするために，プログラム上の処理はサーバ情報を文字出力するだけの簡易な処理で比較を行った．

また，Apache のバーチャルホスト方式では，アクセスのあったホスト名から，それぞれに紐付くファイルパス名を導出する．これは，単一のホストに同時接続する場合と，複数のホストに同時に接続する場合で，アクセス先のファイルのパスが違うだけで，本質的な差はないと考えられる．また，本稿で改良した suEXEC においては，バーチャルホスト方式そのもののアーキテクチャは変更していない．そのため，変更した実装である suEXEC の内部処理における性能劣化の評価は，1 つのホスト名に対してのみ行っても十分であると考えた．

以上から，それぞれの実行方式に対して，Apache のベンチマークコマンドを用いて，文字列を出力する PHP スクリプトに対し，DSO 実行方式でサーバプロセスそのものを破棄する方式，本手法で suEXEC を改良した CGI 方式，および，通常の suEXEC を使った CGI 方式での比較を行った．同環境のサーバにおいて，同時接続数 50，総接続数 10,000 で負荷をかけたところ 1 秒間に処理できるリクエスト数は，CGI 版 PHP と suEXEC を利用した場合は約

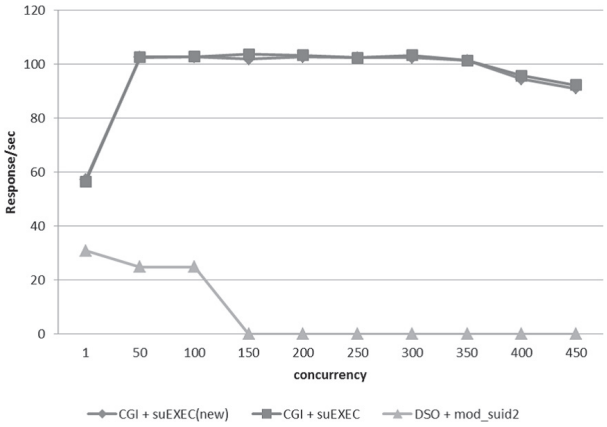

図 6 実行方式別のパフォーマンス比較
Fig. 6 Performance comparison by executing methods.

102.56 リクエストであった．しかし，この構成では仮想ホスト追加ごとに Apache のリロードが必要となったり，仮想ホストの数に比例して設定行数が増加したりするため，運用性と品質が低い．一方で，DSO 版 PHP と mod_suid2 を利用した場合は，Apache のリロードは必要なく，設定も動的に記述できるものの，約 24.88 リクエストにまで低下していた．

次に，改修後の suEXEC を CGI 版 PHP に対して適応した場合において，同様のテストを行った．その結果，単位時間あたりのリクエスト数は 102.44 で，改修前の suEXEC とほぼ同等の数値となった．仮想ホスト追加時の Apache のリロードや，設定も動的に記述できるため，運用性や品質が高い．また，同時接続数を変動させても，改修前と同等の性能が得られ，chroot やその他のエラー処理の追加実装による性能劣化はほとんど見られなかった．

図 6 に同時接続数を変化させた場合の評価結果をグラフで示した．横軸は同時接続数，縦軸は 1 秒間でサーバ側が処理できたレスポンスの数を示している．DSO 実行方式で mod_suid2 を使ってサーバプロセスを破棄する方式では，同時接続数 100 以降ではコネクションタイムアウトとなりレスポンスを返すことができなくなった．この結果より，共有サーバにおいて DSO 版 PHP を使う場合に，セキュリティを担保するために mod_suid2 が必要であるが，mod_suid2 を適応した結果，CGI 版 PHP よりも大きく性能が落ちることが分かった．そのため，ホスティング利用顧客が DSO 版 PHP で動作するように，シェバン行を書かずに PHP スクリプトを記述した場合でも，Apache を経由して suEXEC により CGI 版 PHP として動作するように改修した本稿の suEXEC の改修設計は有効性があると考えられる．

以上より，これまで高集積型の Web バーチャルホスティング基盤を運用した場合生じるセキュリティと運用上の課題を解決できた．これによって，運用者の負担が大幅に低減され，実運用に耐えうる大規模構成をとることができる

と考えている．

7. むすび

本稿では，汎用性の高い大規模 Web バーチャルホスティングサービスの基盤技術において，Apache の VirtualHost を採用した場合に生じる運用面とセキュリティの課題を改善する手法を提案した．本稿で明らかにした VirtualHost の課題に対して，既存の手法が数多くあるが，現状，汎用性が高くスケーラビリティがあり，3 章，4 章であげた課題をすべて同時に解決している手法を見つけることができなかった．既存の手法や関連研究をもとに，とりうる最善の選択をし，新たな機能追加や改修によって多くの課題を解決することができた．セキュリティと運用のバランスをとるために行った suEXEC の改修も，既存手法にない，オリジナリティのあるアプローチだといえる．パフォーマンスも実績のある suEXEC と同等であり，実用に耐えうると考えられる．

本手法を採用すれば，今後増えるであろうクラウドサービスや低価格ホスティング等に対応するために，共有ストレージを用いたスケールアウト型の負荷分散構成をとることができる．そして，コンピュータリソースの効率化や運用業務の省力化が促進され，その結果，最小限の人員リソースでの運用の仕組みが構築できると考えている．本稿で提案した Web ホスティング基盤は，今後のクラウドや低価格ホスティングを支える Web サービスプラットフォームとしての，現時点における 1 つの答えであると考えている．

今後の課題として，suEXEC の改修による脆弱性が生じていないかを再度検証し，suEXEC やセキュリティ関連のモジュールをさらに発展させ，CGI にとどまらず DSO 版 PHP を含めた多くのプログラムを区別せず対応できる，より安全で効率の良い柔軟なセキュリティアーキテクチャを考える必要がある．

謝辞 技術開発ならびに本稿の作成に協力されたファーストサーバ（株）の諸氏に感謝する．本稿の草稿に対して有益なコメントをいただいた京都大学岡部寿男教授に深謝する．

参考文献

[1] Amazon.com: Amazon Elastic Compute Cloud, available from ⟨http://aws.amazon.com/jp/ec2/⟩.
[2] The Apache Software Foundation: Apache HTTP SERVER PROJECT, available from ⟨http://httpd.apache.org/⟩.
[3] The Apache Software Foundation: Apache Virtual Host documentation, available from ⟨http://httpd.apache.org/docs/2.2/en/vhosts/⟩.
[4] Hara, D., Fukuda, R., Hyoudou, K., Ozaki, R. and Nakayama, Y.: Hi-sap: Secure and Scalable Web Server System for Shared Hosting Services, *Proc. 7th International ICST Conference on Broadband Communications, Networks, and Systems* (*BROADNETS 2010*) (Oct. 2010).
[5] The Apache Software Foundation: Apache Tutorial: Dynamic Content with CGI, available from ⟨http://httpd.apache.org/docs/2.2/en/howto/cgi.html⟩.
[6] The Apache Software Foundation: Dynamic Shared Object (DSO) Support, available from ⟨http://httpd.apache.org/docs/2.2/en/dso.html⟩.
[7] Wikipedia: User identifier, available from ⟨http://en.wikipedia.org/wiki/User_identifier_(Unix)⟩.
[8] The Apache Software Foundation: Apache Module mod_vhost_alias, available from ⟨http://httpd.apache.org/docs/2.0/mod/mod_vhost_alias.html⟩.
[9] The Apache Software Foundation: suEXEC Support, available from ⟨http://httpd.apache.org/docs/2.2/en/suexec.html⟩.
[10] Wikipedia: Shebang (Unix), available from ⟨http://en.wikipedia.org/wiki/Shebang_(Unix)⟩.
[11] Sebastian Marsching: suPHP Homepage, available from ⟨http://www.suphp.org/Home.html⟩.
[12] The Apache Software Foundation: Apache Module mod_actions, available from ⟨http://httpd.apache.org/docs/2.0/en/mod/mod_actions.html⟩.
[13] The PHP Group: PHP Manual セーフモード，入手先 ⟨http://php.net/manual/ja/features.safe-mode.php⟩.
[14] Nakamitsu, H.: mod-suid2, available from ⟨http://code.google.com/p/mod-suid2/⟩.
[15] Nakamitsu, H. and Stano, P.: mod_ruid, available from ⟨http://websupport.sk/~stanojr/projects/mod_ruid/⟩.
[16] 原 大輔，尾崎亮太，兵頭和樹，中山泰一：Harache：ファイル所有者の権限で動作する WWW サーバ，情報処理学会論文誌，Vol.46, No.12, pp.3127–3137 (2005).
[17] 鈴木真一，新城 靖，光来健一，板野肯三，千葉 滋：ユーザ権限変更機構を利用した安全なイントラネットサーバの実現，情報処理学会論文誌コンピューティングシステム，Vol.44, No.SIG 10 (ACS 2), pp.86–96 (2003).
[18] 原 大輔，中山泰一：Hussa：スケーラブルかつセキュアなサーバアーキテクチャ—低コストなサーバプロセス実行権限変更機構，第 8 回情報科学技術フォーラム（FIT 2009）講演論文集，RB-002 (Sep. 2009).
[19] 日本 Linux 協会：JM Project CAPABILITIES，入手先 ⟨http://archive.linux.or.jp/JM/html/LDP_man-pages/man7/capabilities.7.html⟩.
[20] 原 大輔，中山泰一：Hussa スケーラブルかつセキュアなサーバアーキテクチャ—セキュア OS の適用と評価，日本ソフトウェア科学会第 26 回大会論文集，6C-1 (Sep. 2009).
[21] National Security Agency: Security-Enhanced Linux, available from ⟨http://www.nsa.gov/research/selinux/⟩.
[22] Wikipedia: Load (computing), available from ⟨http://en.wikipedia.org/wiki/Load_average⟩.

松本 亮介 （学生会員）

ファーストサーバ株式会社．2008 年大阪府立大学工学部情報工学科卒業．同年ファーストサーバ株式会社入社．レンタルサーバ（ホスティング）の基盤技術に関わる研究・開発・運用に従事．2012 年 4 月より，京都大学情報学研究科博士課程．電子情報通信学会学生会員．

川原 将司

ファーストサーバ株式会社．1996 年大阪電気通信大学経営工学部卒業．同年クボタシステム開発株式会社入社，レンタルサーバ（ホスティング）システムを開発．2000 年にファーストサーバ株式会社としての独立に参加．現在も同社にて研究開発に従事．

松岡 輝夫

ファーストサーバ．1994 年龍谷大学理工学部電子情報学科卒業．独立系システム会社を経て，2006 年ファーストサーバ入社．レンタルサーバ（ホスティング）事業の運用部門の管理に従事．