汎用性の高い大規模共有型 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† 川原将司† 松岡輝夫†

近年，AmazonEC2 に代表されるクラウドの台頭に伴い，ホスティングサービスの低価格化が進んでいる．そこで，我々は限られたリソースから大多数のホスト(約 12000 ホスト)を処理するための Web ホスティング基盤を開発した．リソースを必要最小限に抑えるために Apache の VirtualHost 機能を用いて，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を採用することにした．本稿では，VirtualHost 採用と大規模対応に伴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．その結果，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た．

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

RYOSUKE MATSUMOTO† MASASHI KAWAHARA† TERUO MATSUOKA†

Recently, it is increasingly important to archive a low price hosting services with marked improvement in Cloud, for example AmazonEC2. Then, we developed the Web based hosting system to process huge number of hosts (about 12000 hosts) using the limited resources effectively. We decided to adopt the configuration using Apache VirtualHost that single process manages multiple hosts and distinguish contents by the hostname with the access for suppressing the resource to the minimum requirement. Here we propose that we clarify and solve the problems of security and operation technology using VirtualHost with large-scale system by developing the new Apache modules and enhancing suEXEC. As a result, We could construct the large-scale, reliable and operationally-effective Web based hosting system.

1. はじめに

近年，インターネットの普及に伴い，自社でインフラ設備をも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共に,ホスティングサービスの低価格化が進んでいる.そこで，我々は限られたリソースから大多数のホスト(約 12000 ホスト)を処理するための Web ホスティング基盤を開発した.

これまで，運用面やセキュリティを重視した場合，OS 上で chroot[2]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．この構成では，サーバを複数のホスティング利用者で共有する場合でも，互いに干渉できない独立した領域でサービスを提供でき，セキュリティが高い．また，独立した領域においては，サーバプロセスの全ての設定が可能であるため，柔軟に運用できる．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，サーバプロセスとハードウェアが紐づくため負荷分散が難しく,汎用性が低い.

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事を考えると，限られたリソースで最大限のセキュリティや運用性を担保し，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仕組みを採用することとした．Web サーバ構築には Apache HTTP Server[3] (以降 Apache とする)の VirtualHost[4]を利用した．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.

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を述べ，6 章で現在主流となっている Web ホスティング基盤製品と比較を行い，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

† ファーストサーバ(株)
Firstserver, Inc.

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．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．図 1 で運用面とセキュリティを重視した場合の構成を示す. 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全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，複数のサーバプロセスをそれぞれ異なるユーザー権限で起動する手法がある[5]．各 chroot 環境などで，ホスティング利用者単位でユーザー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．例えば，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 12000 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ー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 12000 のユーザー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，2000 のユーザー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，ユーザープロセスに依存しない構成をとることができる．図 2 でその構成を示す．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

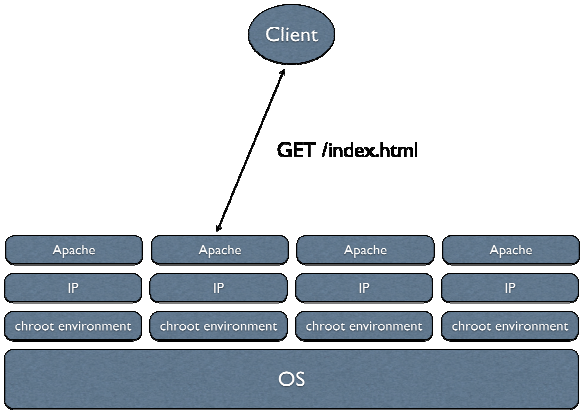

図 1　運用面とセキュリティを重視した構成
Figure 1　The configuration for operation-friendly and secure system

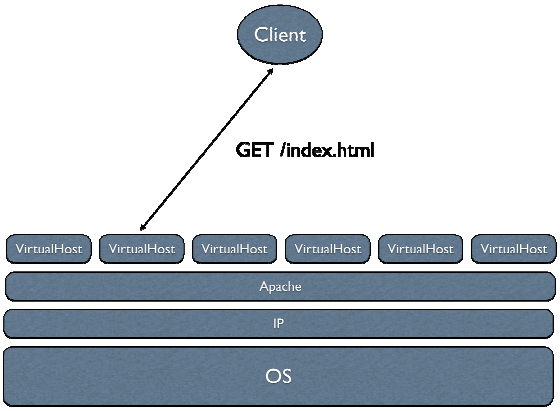

図 2　単一プロセスで複数ホストを扱う構成
Figure 2　The configuration that single process manages multiple hosts

して 1 つのユーザー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ー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伴い, 日々増加してきている．そこで，効率的にサーバ運用を行うことが重要であり，この点において 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[6]や PHP[7]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 版[8]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事が困難である．また，CGI 版 PHP と suEXEC を利用した場合は，プロセス上で uid，gid[9]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と考えられる．

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，ホスト毎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．例えば，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全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伴い，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，リクエストの処理を行うか判断する仕組みが必要だと考えられる．

3.4 新規設定反映時は全てのホストに影響

3.3 で述べたとおり，Apache が停止すると，全てのホスト機能が一時的に停止する．そのため，大規模化に伴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[10]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，

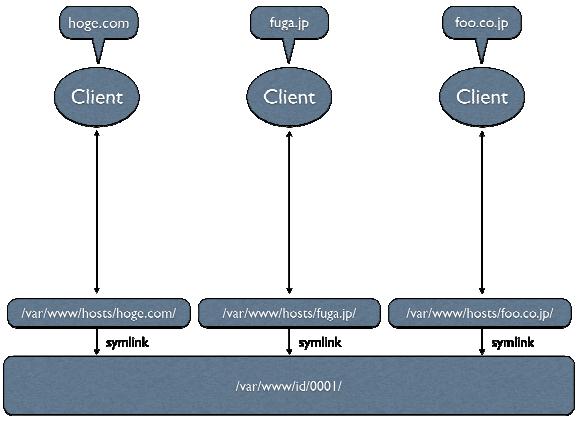

図 3 マルチドメインの構成
Figure 3 The configuration of Multi Domains Service

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 毎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問題や，VirtualHost 毎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問題がある．

3.5 シンボリックリンクへの対応

Web ホスティングでは，あるドメインにアクセスした場合と，別のドメインにアクセスがあった場合に，どちらのドメインも同じドキュメントルートにアクセスできるようにするサービスがある．このサービスをマルチドメインと呼ぶ．マルチドメイン設定はシンボリックリンクを用いて設定することが多い．図 3 に，マルチドメインの仕組みを示す．例えば，Apache の設定を用いてチューニング対象のパスを設定した際，Apache はシンボリックリンクを含むパスとシンボリックリンクを読み替えた完全なリアルパスとを区別しない．とある領域に高負荷対象の CGI 等が存在した場合，リアルパスに対してのみチューニングしただけでは，チューニングがされていないものとして，別のドメイン経由で該当 CGI にアクセスすることができる．該当 CGI を正確にチューニングするためには，シンボリックリンクを含んだパスの数だけ設定を追記する必要がある．そのため，マルチドメインの数が多くなると，チューニング時の運用性が大幅に低下する．

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 章でセキュリティ面における課題を述べる．

4. VirtualHost における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全ての

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 4 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 4 では，index.cgi を Apache の権限である uid500，gid101 で実行する．/var/www/hosts/fuga.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ー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[11]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 のユーザー権限及び一般ユーザーで閲覧可能な/etc 等のシステム領域を覗き見することができる．

4.2 suEXEC 採用に伴う 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とることができる．図 5 に suEXEC の利用例を示す．図 5 では，図 4 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 を閲覧することもできない．

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，その上で 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 で述べた通り，mod_vhost_alias で suEXEC の設定を動的に扱えないという課題が残る．

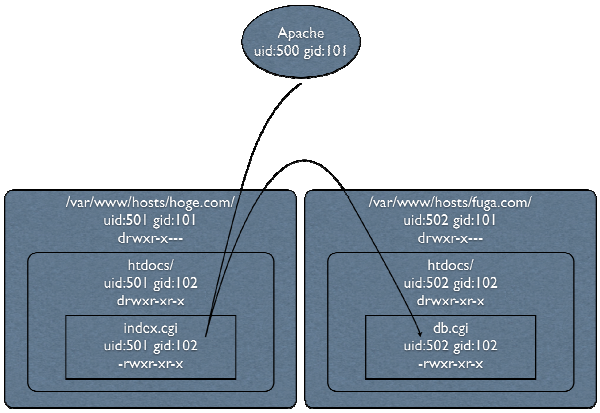

図 4 他ホスト領域の覗き見
Figure 4 Peeping at Other Host

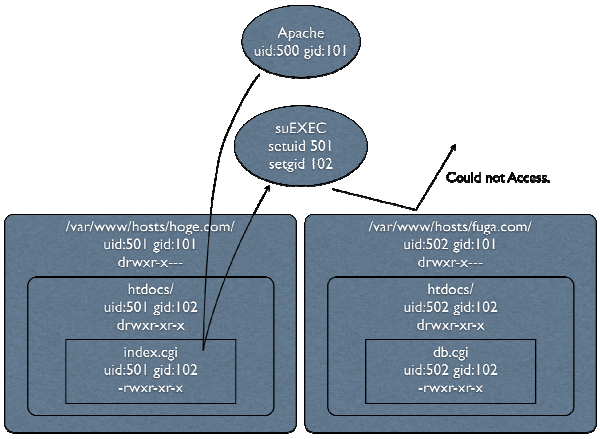

図 5 suEXEC の利用例
Figure 5 Example for suEXEC

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[12]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 4 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 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[13]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 をユーザー権限で実行できる．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 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[14]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設定なく実行できるラッパープログラムに渡す設定ができる．この設定により，suEXEC の機能を利用した上で PHP を実行できる．しかし，ラッパープログラムを 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，ラッパープログラムは 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 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機構

4.2.2 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[15]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無い問題や，多数のオープンソースアプリケーションが正常に動作しなくなる問題等が存在する．

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[16]や mod_ruid[17]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[18]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度にユーザー権限に 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ー権限であるため，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ー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，ユーザー権限であっても setuid 等を実行可能にする手法[19]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[20]利用すると，一時的にユーザー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[21]と呼ばれる機構の内，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 のように全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．例えば，関連研究[22]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[23]等のセキュア OS を用いて制限したとする．しかし，サーバプロセスが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，及び，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上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 等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．以降で，それらの課題を同時に解決するための手法を提案する．

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 リクエスト毎でリソース消費量測定機能を実装

3.1 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，ユーザー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，ユーザー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上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繋がる．しかし，3.2 で述べた通り, 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実装

3.3 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，全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挙げられる．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[24]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 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 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問題は，各 VirtualHost の suEXEC のユーザー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ー名とグループ名を解釈できるように suEXEC を改修した．5.2 で suEXEC 改修の詳細を述べる．この改修と DCMVH によって，全ての VirtualHost の設定を一つの設定に書き直すことができた．また，.htaccess と同じ機能をも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 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 のリロード無くホスト単位で新たな設定を反映させることができる．

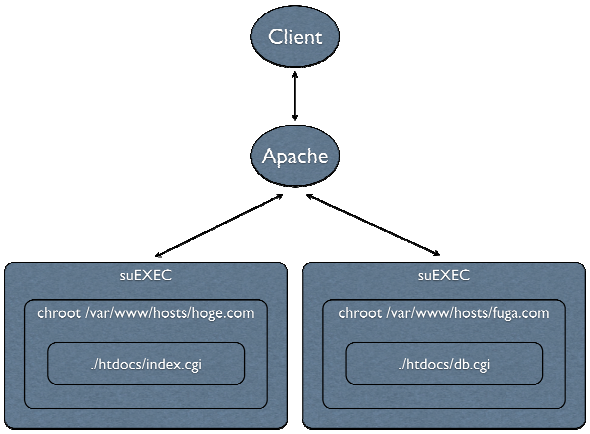

図 6　suEXEC 実行時に chroot
Figure 6　chroot on suEXEC

以上の手法で, ホスト数に依存した煩雑な設定無く，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．

5.1.5 シンボリックリンク解釈機能実装

3.5 において，マルチドメインサービスから生じるシンボリックリンクを用いたドキュメントルートの課題を述べた．そこで，5.1.4 の動的設定を導入した場合は，ホスト名を表すディレクトリ名に対して，別のホスト名のシンボリックリンクを張ることで簡単にこのサービスを実現することができる. 5.1.1, 5.1.2, 5.1.3, で実装したモジュールでチューニングを行う場合，任意のスクリプトに対するフルパスを指定することがある．そこで，新たに実装した各モジュールには，シンボリックリンクを含むパスでアクセスがあった場合，リアルパスに解釈し直す機能を実装した. これにより，リアルパスのみを対象にチューニング設定を行っておけば，マルチドメインサービスを利用していたとしても，リアルパスで設定を解釈するため，設定記述は 1 つでよく，運用性が向上する．

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 する機能実装

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 6 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 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，ラッパー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．

5.2.3 SuexecUserGroup 設定無効化

5.1.4 で述べた通り，ホスト毎の SuexecUserGroup パラメータに記述するユーザー名とグループ名は全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．顧客ホスト領域毎に 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．

5.3 パフォーマンス

suEXEC を改修したことによるパフォーマンスへの影響について述べる．改修前の suEXEC 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，同時接続数 10 で合計 1000 リクエストを処理する性能を 3 回測定し，その平均値を比較した．同環境のサーバにおいて，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 54 リクエストであった．それに対して，DSO 版 PHP と mod_suid2 を利用した場合は，4.3.2 で述べた通り，約 17 リクエストにまで低下していた．

改修後の suEXEC に対して，同様のテストを行った．その結果，単位時間当たりのリクエスト数は 53 で，改修前の suEXEC とほぼ同等の数値となり，chroot やその他のエラー処理を追加実装しても，性能劣化はほとんど見られなかった．

6. 他の Web ホスティングと比較

本章では，容易にホスティング機能を提供するための，ホスティングシステムパックとして提供されている Parallels Business Automation Standard[25]（以降 PBAS とする）の Web ホスティング機能と本稿の提案手法を比較した．

PBAS の Web ホスティング機能は VirtualHost を利用しており，柔軟な設定が可能で多くの企業で採用されている信頼のあるシステムパックである．Parallels Plesk Panel と呼ばれるコントロールパネルを含め，ホスティングシステムパックのデファクトスタンダードといっても過言ではない．表 1 では，運用面の課題に関しての比較を行い，表 2 ではセキュリティの比較を行った．運用面では，2 章で挙げた課題を全て解決することができておらず，全ての面で本手法に優位性がある．また，セキュリティ面に関して，PBAS は DSO 版 PHP を自由に選択可能なホスティングサービス仕様になっており，それをコントロールパネル上で選択できないように HTML を変更する（仕様上，システムで無効にすることできない）ことで PHP スクリプトによる他ホスト領域の覗き見を防げる．しかし，システム領域は覗き見可能である．また，CGI 版 PHP のシェバン行や実行権限付加の強制に関しては，mod_actions におけるラッパースクリプトを採用することで解決できているが，顧客の URL を一部専有している問題があるので△とした．リロードも頻繁に発生する仕様となっている．全ての点で本手法に優位性がある．さらに，本稿で提案した仕組みは Apache モジュールで作成しているため，PBAS の構成にも組み込むことができ，本稿で提案した機能を容易に反映させることができる．

表 1　運用面の比較
Table 1　Comparing with PBAS for Operation Technology

機能(節番号)	PBAS	本手法
リソース測定機能(5.1.1)	×	○
同時接続数制限機能(5.1.2)	×	○
リソース変化による制限機能(5.1.3)	×	○
動的設定機能(5.1.4)	×	○
シンボリックリンク解釈機能(5.1.5)	×	○

表 2　セキュリティ面の比較
Table 2　Comparing with PBAS for Security

機能(節番号)	PBAS	本手法
chroot 機能(5.2.1)	×	○
php 実行機能(5.2.2)	△	○
suEXEC 設定無効化(5.2.3)	×	○

その他既存の手法において，VirtualHost を利用して大規模対応基盤を構築する場合，本手法よりも優位性のある手法を見つけることはできなかった．しかし，実運用における評価結果は得られていないため，今後の課題として，12000 ホストを処理できるとする設計目標を達成できているか評価していく必要がある．

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新たな手法を提案した．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 章で挙げた課題を全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元に，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思われる．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，現時点における一つの答えであると考えている．

今後の課題として，セキュリティ面では，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 に留まらず 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．運用面では，クラウドの普及に伴って仮想環境が爆発的に増えた場合，それらを管理し運用する仕組みが重要になると考えられる．そのためには，リソース変化や同時接続数の変化を時系列のデータとして動的にとらえ，時系列データが大きく変化した際に自動でチューニングがされるような仕組みを考えていく必要がある．

謝辞 技術開発並びに本論文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．本論文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", http://aws.amazon.com/jp/ec2/.
2) Wikipedia, “chroot”, http://ja.wikipedia.org/wiki/Chroot.
3) The Apache Software Foundation, "Apache HTTP SERVER PROJECT", http://httpd.apache.org/.
4) The Apache Software Foundation, " Apache Virtual Host documentation”, http://httpd.apache.org/docs/2.2/en/vhosts/.
5) Daisuke Hara, Ryohei Fukuda, Kazuki Hyoudou, Ryota Ozaki, and Yasuichi Nakayama, “Hi-sap: Secure and Scalable Web Server System for Shared Hosting Services”, Proc. of 7th International ICST Conference on Broadband Communications, Networks, and Systems (BROADNETS 2010), (Oct. 2010).
6) The Apache Software Foundation, "Apache Tutorial: Dynamic Content with CGI”, http://httpd.apache.org/docs/2.2/en/howto/cgi.html.
7) The PHP Group, "PHP: Hypertext Preprocessor", http://www.php.net/.
8) The Apache Software Foundation, "Dynamic Shared Object (DSO) Support”, http://httpd.apache.org/docs/2.2/en/dso.html.
9) Wikipedia, “User identifier”, http://en.wikipedia.org/wiki/User_identifier_(Unix).
10) The Apache Software Foundation, "Apache Module mod_vhost_alias, http://httpd.apache.org/docs/2.0/mod/mod_vhost_alias.html.
11) The Apache Software Foundation, "suEXEC Support”, http://httpd.apache.org/docs/2.2/en/suexec.html.
12) Wikipedia, “Shebang (Unix)”, http://en.wikipedia.org/wiki/Shebang_(Unix).
13) Sebastian Marsching, "suPHP Homepage", http://www.suphp.org/Home.html.
14) The Apache Software Foundation, "Apache Module mod_actions”, http://httpd.apache.org/docs/2.0/en/mod/mod_actions.html.
15) The PHP Group, "Safe Mode”, http://www.php.net/manual/en/features.safe-mode.php.
16) Hideo NAKAMITSU, “mod-suid2”, http://code.google.com/p/mod-suid2/.
17) Hideo NAKAMITSU and Pavel Stano, "mod_ruid", http://websupport.sk/~stanojr/projects/mod_ruid/.
18) 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).
19) 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).
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～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).
21) 日本 Linux 協会, “JM Project CAPABILITIES”, http://archive.linux.or.jp/JM/html/LDP_man-pages/man7/capabilities.7.html.
22) 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).
23) National Security Agency, “Security-Enhanced Linux”, http://www.nsa.gov/research/selinux/.
24) Wikipedia, “Load (computing)”, http://en.wikipedia.org/wiki/Load_average.
25) Parallels, "Parallels Business Automation Standard", http://www.parallels.com/jp/hspcomplete/.