# Harache：ファイル所有者の権限で動作するWWWサーバ

原 大輔[†] 尾崎 亮太[†,☆]
兵頭 和樹[†,☆☆] 中山 泰一[†]

現在のWWWサーバには実行時のユーザ権限に起因する問題が存在する．PHPプログラムがデータをファイルに書き込む場合，データファイルの所有者がWWWサーバを実行するユーザとなるため，サーバを共有する別の利用者のPHPプログラムからそのデータファイルを盗視されたり削除されたりする危険性がある．1台のサーバ計算機を多数の利用者で共有する共有型ホスティングサービスにおいて，これは深刻な問題である．本研究では，この問題を克服するWWWサーバ，*Harache*を提案する．提案するシステムではサーバプロセスをファイル所有者の権限で動作させる．これにより，WebDAVやPHPといったサーバ組み込みのプログラムを用いる際の問題を解決することができる．また，不必要なサーバプロセスを適宜終了することで利用者数に対する高いスケーラビリティを達成できる．本論文では，Haracheの設計および実装法について述べる．HaracheをSELinuxを有効にしたLinux OS上に実現し，評価実験を行った．その結果，ユーザ権限に起因する問題を解決すること，他の実現方式に対してスケーラビリティの面で優位性を持つこと，実用に耐えうるだけの性能を達成していることを確認した．

# Harache: A WWW Server Running with the Authority of the File Owner

Daisuke Hara,[†] Ryota Ozaki,[†,☆] Kazuki Hyoudou[†,☆☆]
and Yasuichi Nakayama[†]

This paper presents a WWW server named *Harache*, that runs under the authority of the file owner. Existing servers have problems that occur because of the user authority during execution. When a PHP program creates data files, the owner of the created files is the special user account that runs a server. Therefore, other users that share the same server can steal and delete these data files. These problems are serious for a hosting service where many users share a server. Harache has server processes that run under the authority of the file owner. Hence Harache can solve these problems that occur because of the user authority. In addition, Harache terminates unnecessary server processes when needed to improve scalability of the number of users. For a proof of concept, we implemented Harache on a Linux OS with a SELinux and performed evaluation experiments. Experimental results show that Harache achieves high performance and scalability.

## 1. はじめに

近年，インターネットの爆発的な普及によりウェブや電子メールが一般に浸透し[1)]，個人でウェブサイトを公開して情報を発信する人が増加している．ウェブサイトを設置する場所として，ISP（Internet Service Provider）の無料スペースやホスティングサービスの有料スペース，自前のサーバなどがあげられる．なかでも低コストで様々な機能を利用できる共有型ホスティングサービス☆☆☆に人気が集まっている．

一方，既存のWWWサーバには実行時のユーザ権限に起因する問題が存在する．WebDAV[2)]やPHP[3)]など，サーバ組み込みのモジュールとして提供されるプログラムはWWWサーバの権限で動作する．PHPプログラムがデータをファイルに書き込む場合，データファイルの所有者がWWWサーバを実行しているユーザとなってしまう．そのため，サーバを共有する別の利用者のPHPプログラムによってデータファイ

† 電気通信大学情報工学科
Department of Computer Science, The University of Electro-Communications
☆ 現在，総合研究大学院大学情報学専攻
Presently with Department of Informatics, The Graduate University for Advanced Studies
☆☆ 現在，東京工科大学コンピュータサイエンス学部
Presently with School of Computer Science, Tokyo University of Technology

☆☆☆ 1台のサーバ計算機を多数の利用者で共有する形態のサービス．

ルが盗視されたり消去されたりする危険性がある．このようなユーザ権限に起因する問題は 1 台のサーバ計算機を多数の利用者で共有する共有型ホスティングサービスでは特に深刻である．

そこで，本研究ではユーザ権限に起因する問題を解決する WWW サーバ，*Harache* の設計および実装，評価を行う．ホスティングサービスで一般的に用いられているリバースプロキシ型のネットワーク構成をとり，1 台のサーバ計算機を多数の利用者で共有する共有型ホスティングサービスを対象とする．本システムではサーバプロセスをファイル所有者のユーザ権限で動作させる[4]．そのため，WWW サーバ組み込みのプログラムを各ユーザの権限で実行でき，ユーザ権限に起因する問題を解決することができる．また，本システムを応用することで利用者ごとに粒度の細かいサービスを提供することができる．

以下本論文では，2 章で本研究の背景について述べる．また，3 章で本論文で提案するユーザ権限に起因する問題を解決する WWW サーバ，Harache の設計について，4 章でその実装法について述べる．5 章では実現した Harache に対して行った評価実験について報告し，6 章では Harache に対する考察について述べる．最後に，7 章でまとめを述べる．

## 2. 背　　景

本章では，本研究の背景として WWW サーバにおけるアクセス制御および，ホスティングサービスの現状について述べる．本論文において，利用者とは WWW サーバを用いて Web コンテンツを発信する人を示し，ユーザとは OS におけるユーザのことを示す．

### 2.1 WWW サーバにおけるアクセス制御

現在の WWW サーバは管理者や一般ユーザとは別の特殊ユーザ☆の権限で動作する（**図 1**）．管理者権限ではなく，特殊ユーザの権限で動作することにより外部からの攻撃やサーバの誤動作に対しての安全性は高まる．しかし，サーバの利用者に対してはユーザ権限に起因する種々の弊害をもたらす．まず，WWW ブラウザなどからのアクセスに対してファイルを公開するためには，特殊ユーザがファイルの読み取りや実行を行えなければならない．つまり，ファイルの読み取り許可や実行許可を所有者，グループとは別のその他に与える必要がある．そのため，HTTP 経由のアクセスに対して認証を用いたアクセス制限をかけているファイルであっても，サーバを共有する別の利用者から盗視されたり実行されたりする危険性がある．

☆ WWW サーバ専用に用いられ，ログインシェルを持たないなどの特徴がある．*apache*，*www-data*，*www* などの名前が付けられている．

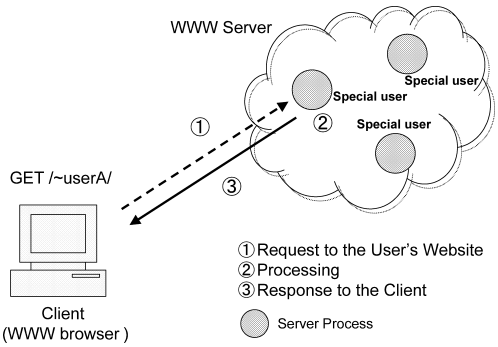


**図 1** リクエスト処理の概要：既存の WWW サーバ
Fig. 1 Overview of request processing: Existing WWW server.

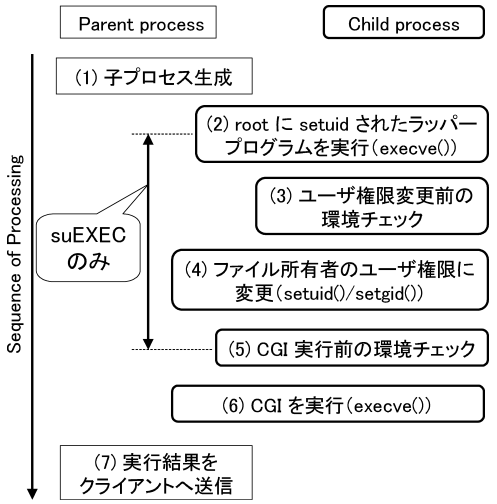


**図 2** CGI 処理の概要
Fig. 2 Overview of CGI processing.

この問題に対して種々のアクセス制御方式が提案されている（**表 1**）．suEXEC[5] は CGI をファイル所有者のユーザ権限で実行させる機構であり，これを用いることで CGI の実行許可をグループやその他へ与える必要がなくなる．**図 2** に CGI 処理の概要を示す．suEXEC は通常の CGI 処理に加え手順（2）〜（5）を行うため，それにともなうオーバヘッドが発生する．(3), (5) の環境チェックは，CGI を実行するユーザが正当なユーザであるか，suEXEC 実行を許可されているかなど 20 項目にも及ぶ[5]．また，suEXEC はテキストファイルの盗視に対して有効でない．

POSIX ACL[6] はファイルシステムレベルのアクセス制御機構である．所有者，グループ，その他という UNIX の伝統的なファイルパーミッションモデルに加

**表 1** アクセス制御方式の比較
Table 1 Comparison of access control approaches.

| | ファイルの種類 | サーバプロセスの実行権限 | ファイル公開に必要なパーミッション (owner/group/other) | ファイルの安全性 |
|---|---|---|---|---|
| 既存の WWW サーバ | テキスト | 特殊ユーザ | rw-/---/r-- | × |
| | CGI | 特殊ユーザ | rw-/---/r-x | × |
| suEXEC | テキスト | 特殊ユーザ | rw-/---/r-- | × |
| | CGI | ファイル所有者 | rwx/---/--- | ○ |
| POSIX ACL | テキスト | 特殊ユーザ | rw-/---/---，特殊ユーザに読み取り許可 | ○ |
| | CGI | 特殊ユーザ | rw-/---/---，特殊ユーザに実行許可 | ○ |
| perchild MPM | テキスト | ファイル所有者 | rw-/---/--- | △（仕様レベル） |
| | CGI | ファイル所有者 | rwx/---/--- | △（仕様レベル） |
| 1 vs 1 方式 | テキスト | ファイル所有者 | rw-/---/--- | ○ |
| | CGI | ファイル所有者 | rwx/---/--- | ○ |

えて，ユーザ単位でのアクセス制御を提供する．そのため，外部に公開したいファイルの読み取り許可や実行許可を特殊ユーザにだけ与えることで，サーバを共有する別の利用者からファイルを盗視されたり実行されたりする危険性が軽減される．

perchild MPM[7] は，バーチャルホストごとに異なるユーザ/グループ権限で WWW サーバを実行する機構である．利用者ごとにバーチャルホストを割り当てることで，ユーザ権限に起因する問題を解決することが期待される．

1 vs 1 方式は利用者ごとに WWW サーバを 1 つずつ割り当てる方式である．各利用者専用の WWW サーバはそれぞれ異なるポートでリクエストを待つように設定する．リバースプロキシ型のネットワーク構成をとり，フロントエンドで動作するプロキシサーバがリクエストをバックエンドで動作する利用者専用の WWW サーバにリダイレクトする．各 WWW をそれぞれ異なるユーザ/グループ権限で実行することで，ユーザ権限に起因する問題を解決することができる．

以上より，テキストファイルと CGI の処理においては POSIX ACL または perchild MPM，1 vs 1 方式を用いることでユーザ権限に起因する問題を解決することができる．

#### 2.1.1 既存技術の問題点

ところが，POSIX ACL を用いてもファイルの安全性や利便性を保障できない場合が存在し，perchild MPM は安定して動作するとの報告がなく，現在開発さえ行われていない．また，1 vs 1 方式を共有型ホスティングサービスに適用するのはスケーラビリティの観点において現実的でない．

WebDAV や PHP といった WWW サーバ組み込みのモジュールとして提供されるプログラムでは，POSIX ACL を適用してもファイルの安全性や利便性を保障できない．WebDAV は HTTP 1.1 を拡張したもので，WWW ブラウザなどのクライアントから WWW サーバに格納したファイルを管理することができる．WebDAV は WWW サーバ同様に特殊ユーザの権限で動作するため，WebDAV 経由で作成したファイルの所有者は特殊ユーザとなってしまう．この状態だと，利用者がサーバにリモートログインしてファイルを直接編集することが困難なため，WebDAV と直接編集の併用はできない．

また，スクリプト言語である PHP は通常 CGI として実行するのではなく，高速化のために WWW サーバ組み込みのインタプリタで実行する方式をとる．同様に，従来の CGI の代わりにインタプリタを WWW サーバに組み込む手法が一般的に用いられている[8),9)]．しかし，これら WWW サーバ組み込みのプログラムを用いる場合，POSIX ACL を用いたアクセス制御を実施することができない．外部に公開するファイルの読み取り許可を特殊ユーザだけに与えたとしても，WWW サーバ同様に特殊ユーザの権限で動作する PHP プログラムからはファイルの内容が読み取れてしまうからである．つまり，サーバを共有する別の利用者の PHP プログラムからファイルを盗視される危険性がある．情報化社会において，個人のプライバシの尊重が叫ばれる今，この問題は大変重大である．また，PHP プログラムがファイルにデータを書き込む場合，データファイルへの書き込み許可を特殊ユーザに与えておかなければならない．そのため，サーバを共有する別の利用者の PHP プログラムにデータファイルを改ざん・削除される危険性がある．

一方，1 vs 1 方式では WebDAV や PHP といった組み込みモジュールが各サイト専用ユーザの権限で動作するため，ユーザ権限に起因する問題が発生しない．しかし，利用者ごとに WWW サーバを割り当てることによって利用者あたりのメモリ使用量が増加してしまう．これはサーバあたりに格納できるサイト数に大

きな影響を及ぼすため，共有型ホスティングサービスに 1 vs 1 方式を導入することは困難だと予想される．本システムと 1 vs 1 方式を対象としたスケーラビリティ評価実験について 5.2 節で述べる．

本研究では 1 台のサーバ計算機を多数の利用者で共有する大規模環境において，サーバ組み込みのプログラムを安全かつ便利に利用できる環境を提供する．

### 2.2 ホスティングサービスの現状

ホスティングサービスでは負荷分散を目的としてリバースプロキシ型のネットワーク構成が一般的に用いられている．これは，リクエストをいったんフロントエンドで動作するプロキシサーバで受け取り，それからバックエンドで動作する WWW サーバにリダイレクトする手法である．実際には，テキストファイルや画像ファイルなどの静的なファイルはフロントエンドでキャッシュしておき，フロントエンドから直接クライアントにレスポンスを返す．バックエンドにリダイレクトされるのはその大半が CGI などの動的なファイルへのリクエストとなる．バックエンドの WWW サーバを CGI など目的の処理に合わせて最適化することで全体的なパフォーマンスの向上が見込まれる．

また，ホスティングサービスでは，1 台のサーバ計算機で WWW サーバ以外にもメールサーバや FTP サーバ，SQL サーバ，アクセス解析ソフトウェアなど多数のサービスを同時に提供しているケースが一般的である．他のサービスへ与える影響を最小化するため，WWW サーバが利用する計算資源を低く抑えることが求められる．

一方，ホスティングサービス業者が直面している一番の問題はデータセンタにおけるサーバの設置面積である．設置面積が小さいほど設備投資にかける費用が少なくて済む．つまり，いかに 1 台のサーバに多くの利用者を格納するかが利益を生む鍵となる．

また，現在のホスティングサービスではデータ転送量に応じた課金方式を採用している場合が多い．これではデータ転送量が少なく，CPU やメモリなどの計算資源を多く使用する CGI を実行しても課金は少なくて済んでしまう．処理の重い CGI の実行によってサーバを共有する他のユーザへ悪影響が及ぶ危険性があるため，この従量課金は平等とはいい難い．そのため，はじめから処理の重い特定の CGI の使用を禁止しているホスティングサービスもある．

本研究ではリバースプロキシ型のネットワーク構成をとり，1 台のサーバ計算機を多数の利用者で共有する共有型ホスティングサービスを対象とする．また，CGI などの動的なファイルの処理において，利用者数に対するスケーラビリティに重点を置く．また，本システムを応用した課金方式について 6.2 節で述べる．

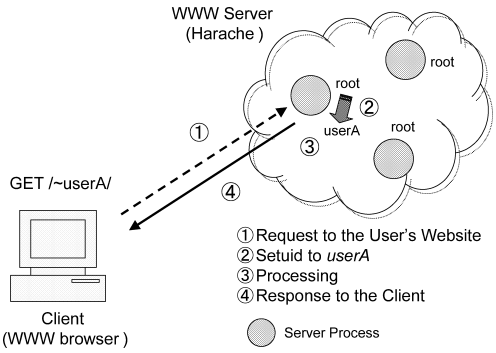


**図 3** リクエスト処理の概要：Harache
Fig. 3 Overview of request processing: Harache.

## 3. 設　　計

本章では，我々が実現した WWW サーバ，Harache の設計について述べる．本設計では UNIX 系 OS を対象とする．

### 3.1 設計方針

我々は，ユーザ権限に起因する問題を解決する WWW サーバ，Harache を実現する．本設計ではリバースプロキシ型のネットワーク構成をとり，1 台のサーバ計算機に 1,000 人程度の利用者を収容する共有型ホスティングサービスを対象とする．本システムでは，ホスティングサービスの利用者 1 人 1 人にアカウントを割り当てる．以下，設計方針について述べる．まず，WWW サーバ組み込みのプログラムの実行に関する問題を解決するために，従来のように WWW サーバを特殊ユーザの権限で実行するのではなく，ファイル所有者のユーザ権限で実行する．そして，利用者数に対するスケーラビリティを高めるためにサーバプロセスを適宜終了するようにする．また，外部からの攻撃に対する安全性を高めるためにセキュア OS[10] を用いたアクセス制御を行う．

### 3.2 WWW サーバ

本システムでは，ユーザディレクトリ内のファイルへのリクエストを受付けるとサーバプロセスをファイル所有者のユーザ権限に変更する．ユーザディレクトリとは，WWW サーバの利用者が各自の Web コンテンツを格納するためのディレクトリのことである．権限変更後は通常のリクエスト処理を行う．リクエスト処理の概要を**図 3** に示す．

本システムではリクエストのない利用者のサーバプロセスを適宜終了する．UNIX 系 OS では，プロセスは実行が終了するまで一定のメモリを占有し続ける．

本システムでは利用者専用のサーバプロセスを用意するため，各サーバプロセスは別の利用者の処理を行うことがない．そのため，リクエストのない利用者のサーバプロセスが増加して，メモリを圧迫し，スラッシングが発生する可能性がある．そこで，サーバプロセスをセッションごとに生成・終了することにより，メモリの節約を図り，スラッシングによる性能低下を防止する．これにより，サーバにより多くの利用者を格納することが可能となる．

### 3.3 OS におけるアクセス制御

本システムではセキュア OS を用いたアクセス制御を行う．セキュア OS は情報ネットワーク法学会セキュア OS と基盤ソフトウエアに関する研究会[11] によって強制アクセス制御☆と最小特権☆☆の機能を有する OS であると定義された OS である[10]．

UNIX 系 OS では，プロセスの権限を変更するために管理者権限が必要である．そのため，本システムではサーバプロセスを管理者権限で起動する．実際にリクエスト処理を行う際にはファイル所有者のユーザ権限に変更しているため，管理者権限で起動することに対する危険性は小さい．しかし，インターネット上の不特定多数からアクセスされる WWW サーバはセキュリティホールや設定ミスなどによって管理者権限を奪われる危険性がある．そこで，本システムではセキュア OS を用いてサーバプロセスに必要最小限の権限だけを与える．

WWW サーバの実行に必要な権限は以下のとおりである．

- WWW サーバ実行に必要なライブラリの実行
- 設定ファイルの読み取り
- ログファイルの書き込み
- Web コンテンツの読み取り，実行
- CGI の実行に必要なインタプリタの実行
- CGI のデータファイルの書き込み

セキュア OS を用いてサーバプロセスに対し上記以外の処理を制限する．これにより，万一管理者権限を奪われた場合でも被害を WWW サーバで完結させることができ，OS の他の部分には被害が拡大しない．

### 3.4 適 用 条 件

サーバ組み込みのインタプリタに状態を保持させるアプリケーションには本システムの適用が難しい．たとえば，データベースサーバ（以下 DB）への接続コストを削減するために接続状態を保持しておく持続的データベース接続[12] がこれに該当する．本システムではセッションごとにサーバプロセスの生成と終了を繰り返すため，DB への接続状態を保持することができない．そのため，アプリケーションが実行されるたびに DB への接続処理を行う必要があり，性能面での損失が大きい．

ただし，WWW サーバと DB との間にデータベース接続をプールするゲートウェイ[13] を別途設置することでこの問題を軽減することができる．

☆ アクセス制御ポリシをすべてのユーザとプロセスに例外なく適用すること．

☆☆ ユーザとプロセスに必要以上の権限を与えないこと．

## 4. 実 装

本章では，前章の設計に基づいた Harache の実装について述べる．本実装において，Harache は SELinux[14] を有効にした Linux OS 上に実現しており，Apache HTTP Server[15]（以下 Apache）をシステムのベースとして用いた．本システムの中核部分は Apache モジュール *mod_harache* として実現した．以下，SELinux のポリシおよび WWW サーバの実装詳細について述べる．

### 4.1 SELinux ポリシ

本システムでは，SELinux を用いてアクセス制御を実施する．SELinux はセキュア OS に該当するため[10]，3.3 節で述べたサーバプロセスに対するアクセス制御を実現することができる．

本実装では以下に述べるポリシを用いた．まず，サーバプロセスが読み取ることのできる範囲を Apache の設定ファイルに，書き込むことのできる範囲をログファイルにそれぞれ限定した．また，サーバプロセスが実行することのできる範囲を Apache の実行に必要なライブラリディレクトリおよび CGI の実行に必要なバイナリディクトリに限定した．そして，Web コンテンツが格納されている各ユーザディレクトリからの読み取り許可をユーザディレクトリの所有者のみに与えた．同じく，CGI などがデータを書き込むディレクトリへの書き込み許可をユーザディレクトリの所有者のみに与えた．

### 4.2 WWW サーバ

本実装では Apache HTTP Server 1.3.33 をシステムのベースとして用いた．

mod_harache は本システムの中核をなす Apache モジュールであり，主にサーバプロセスをファイル所有者のユーザ権限へ変更する機能を提供する．テキストファイルへのリクエストに対する処理手順を**図 4** に示す．処理手順のうち，mod_harache は手順（2）〜（5）の処理を行う．まず，リクエスト URI からユーザ ID とグループ ID を特定する（手順（2），(3)）．次に

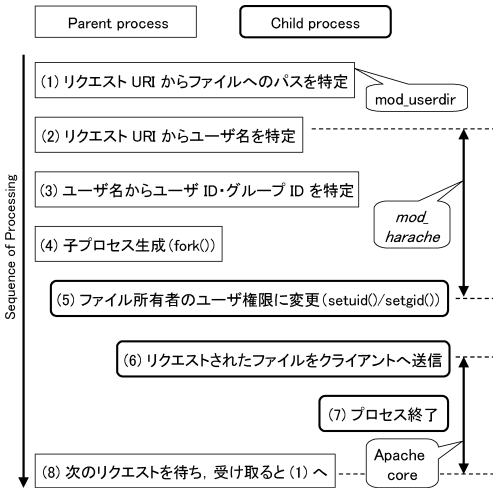


**図 4** リクエストに対する処理手順：テキストファイル
Fig. 4 Procedure for request: Text file.

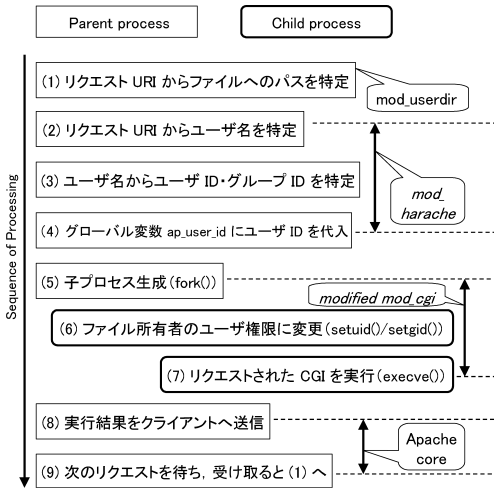


**図 5** リクエストに対する処理手順：CGI
Fig. 5 Procedure for request: CGI.

*fork()* システムコールを用いて子プロセスを作成する（手順（4））．その子プロセスが権限を変更し（手順（5）），リクエストに対する処理を行うようにした（手順（6））．権限変更には *setuid()*，*setgid()* システムコールを用いた．そして，セッションごとにその子プロセスを終了するようにした（手順（7））．逐一サーバプロセスを終了することで不要なサーバプロセスがメモリを圧迫することを防止することができる．これにより，利用者数に対して高いスケーラビリティを達成できる．

また，サーバプロセスのユーザ権限を変更した本システムでは，そのままの状態で CGI を実行することができない．Apache は設定ファイル（*httpd.conf*）の *User* ディレクティブ☆で指定したユーザのユーザ ID と CGI へのリクエスト処理を行っているサーバプロセスのユーザ ID が等しい場合にだけ CGI を実行する．本システムでは，User ディレクティブで root を指定し，サーバプロセスはファイル所有者のユーザ権限で動作するため，CGI を実行することができない．そこで，User ディレクティブで指定された値を保持するグローバル変数 *ap_user_id* の値をファイル所有者のユーザ ID に変更することで CGI の実行を可能にした．

また，CGI の実行を司る Apache 同梱の *mod_cgi* では mod_harache と同様に子プロセスを生成し，その子プロセスが execve() システムコールを用いて CGI を実行している．その結果，mod_harache と mod_cgi で各 1 回，計 2 回子プロセスを生成することになり無駄である．そこで，CGI へのリクエストの場合，mod_harache では子プロセスの生成と権限変更はせず，mod_cgi において，権限変更を行うようにした．CGI へのリクエストに対する処理手順を**図 5** に示す．処理手順のうち，mod_harache は手順（2）〜（4）の処理を行う．また，mod_cgi に手順（6）の処理を追加した．

また，ユーザディレクトリ以外へのリクエストを受け取った場合は，従来の WWW サーバと同様に特殊ユーザの権限に変更して処理を行うようにした．

## 5. 評　　価

本章では，Harache に対して行った評価実験について報告する．評価実験は基礎性能評価実験，スケーラビリティ評価実験からなる．実験環境として，1 台のサーバ計算機および計測用の 1 台のクライアント計算機からなるネットワークを構築した．実験で使用するハードウェアおよび OS については**表 2** にまとめた．サーバ計算機とクライアント計算機はスイッチングハブを介して接続されている．サーバ計算機，スイッチングハブ間は Gigabit Ethernet で接続され，クライアント計算機，スイッチングハブ間は Fast Ethernet で接続されている．

### 5.1 基礎性能評価実験

本節では，本システムに対して行った CGI へのリクエスト処理における性能評価実験について述べ，性能が実用に耐えうることを確認する．

比較対象として，初期状態の設定でコンパイルした Apache HTTP Server 1.3.33 と suEXEC を有効にした Apache HTTP Server 1.3.33 を用いた．本システ

☆ Apache の設定ファイルにおける設定項目のこと．

**表 2** 実験環境
Table 2 Hardware configuration of experimental environments.

| ネットワーク | |
|---|---|
| スイッチングハブ | CentreCOM FS909GT V1<br>1000 BASE-T ポート ×1<br>100 BASE-TX ポート ×8 |

| クライアント | |
|---|---|
| プロセッサ | Intel Pentium 4 1.6 GHz |
| メモリ | 512 MBytes |
| OS | Fedora Core 2（カーネル 2.6.10） |
| NIC | DEC DECchip 21140 100 Mbps |

| サーバ | |
|---|---|
| プロセッサ | Intel Pentium Ⅲ Xeon 500 MHz ×4 |
| メモリ | 256 MBytes（スワップ 512 MBytes） |
| OS | Fedora Core 2（カーネル 2.6.10） |
| NIC | Intel PRO/1000XT PWLA8490XT 1 Gbps |

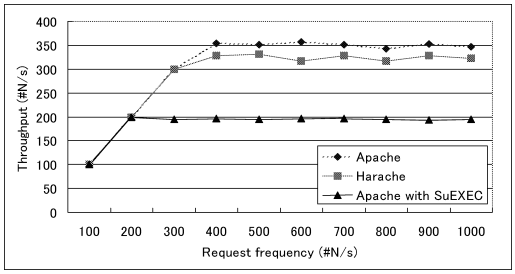


**図 6** 基礎性能評価実験：CGI
Fig. 6 Experiment of basic performance evaluation: CGI.

ム，2 種類の Apache とも設定ファイルにおいて初期状態の設定値を用いた．性能評価には *httperf* ベンチマーク[16] を用いた．

リクエスト頻度を変えながら CGI に対してリクエストを送信し，レスポンスの際のスループットを計測した．実験に用いた CGI は 10 KBytes のテキストを表示するものであり，C 言語で記述されている．**図 6** に測定結果を示す．横軸はサーバ計算機へ 1 秒間に送信したリクエストの回数を表しており，縦軸はサーバ計算機から 1 秒間に受信したレスポンスの回数を表している．本システムは通常の Apache と比較して平均 5.2%，最大 10.9%の性能低下が見られた．しかし，suEXEC を有効にした Apache と比較して大幅に高い性能を達成しており，本実装の有効性が示されたといえる．suEXEC を有効にした Apache の性能が低い原因は図 2 で示したとおり，通常の CGI 実行に加えてラッパプログラムの実行とユーザ権限変更，環境チェックが必要なことにある．

以上の基礎性能評価実験により，本システムが実用に耐えうる性能を達成していることを確認した．

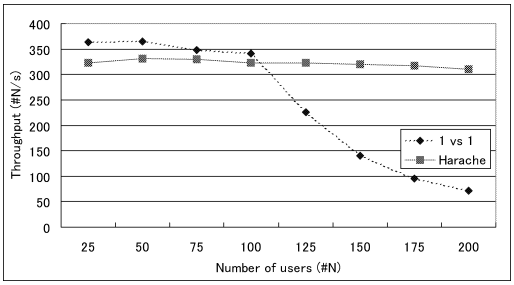


**図 7** スケーラビリティ評価実験：CGI
Fig. 7 Experiment of scalability evaluation: CGI.

### 5.2 スケーラビリティ評価実験

本節では，本システムに対して行ったスケーラビリティ評価実験について述べ，本実装の有効性を確認する．

2.1 節で述べたとおり，本システムに競合する実現方式として，利用者ごとに WWW サーバを 1 つずつ割り当てる 1 vs 1 方式がある．本システムが独自の Apache モジュールの開発を必要としたのに対し，1 vs 1 方式は初期状態の Apache を利用できるという利点がある．本システムと 1 vs 1 方式に対して性能評価実験を行った．1 vs 1 方式は初期状態の設定でコンパイルを行った．また，本実験では動的コンテンツに対する処理性能を測定するため，1 vs 1 方式においてリバースプロキシ型の構成はとらなかった．

#### 5.2.1 CGI 実験

まず，ユーザディレクトリ内に設置した CGI を対象に，リクエストに対するレスポンスのスループットを計測した．実験に用いた CGI は 5.1 節で用いたのと同じものである．各利用者のサイトに対しリクエストを 100 回ずつ送信する実験を逐次的に全サイトに対して行った．本システム，1 vs 1 方式とも設定ファイルにおいて初期状態の設定値を用いた．性能評価には Apache 同梱の Apache ベンチマーク・バージョン 2.0.41-dev を用い，KeepAlive を有効にした．**図 7** に測定結果を示す．横軸はサーバ計算機が格納する利用者数☆を表しており，縦軸はサーバ計算機から 1 秒間に受信したレスポンスの回数を表している．本システムはユーザ数 100 以下で 1 vs 1 方式に対し平均 8.4%，最大 12.8%低い性能を示した．しかし，ユーザ数 125 以上で性能は逆転し，1 vs 1 方式の性能低下の度合いは急激である．また，1 vs 1 方式は利用者数 300 以上でメモリ不足により OS がハングアップしてしまい，実験が継続できなかった．本システムは利用者数増加

☆ 1 vs 1 方式では起動している Apache の数に等しい．

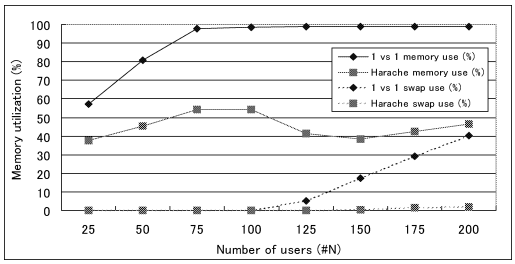


**図 8** スケーラビリティ評価実験：CGI（メモリ利用率）
Fig. 8 Experiment of scalability evaluation: CGI (memory utilization).

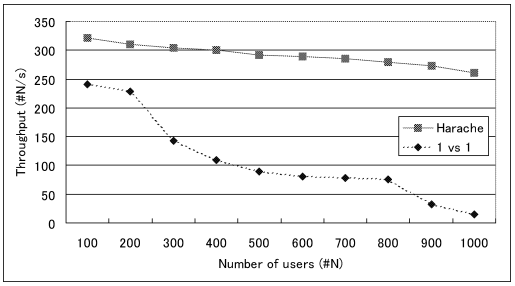


**図 9** スケーラビリティ評価実験：CGI（スケーラビリティ重視）
Fig. 9 Experiment of scalability evaluation: CGI (emphasis on scalability).

**表 3** 変更したディレクティブ
Table 3 Changed directives.

| ディレクティブ名 | 初期値 | 1 vs 1 方式 |
|---|---|---|
| StartServers | 5 | 1 |
| MinSpareServers | 5 | 1 |
| MaxSpareServers | 10 | 1 |

にともなうスループットの低下が小さく，高いスケーラビリティを示している．

本実験中のメモリ，スワップ利用率の変化を**図 8** に示す．本システムと 1 vs 1 方式の性能が逆転したユーザ数 125 において，1 vs 1 方式でスワッピングが始まっているのが分かる．ユーザ数増加にともないスワップ利用率は上昇し，性能低下につながっていると考えられる．一方，本システムではユーザ数が増加した場合でもメモリ利用率，スワップ利用率の上昇が小さい．これにより，ユーザ数増加にともなう性能低下が小さいと考えられる．

#### 5.2.2 CGI 実験（スケーラビリティ重視）

1 vs 1 方式において，ユーザ数増加に対応するために Apache の設定ファイルにおけるディレクティブの値を一部変更した．変更した設定値は**表 3** のとおりである．初期状態のサーバプロセス数を指定する StartServers と余分に起動するサーバプロセスの最小値を指定する MinSpareServers，同じく最大値を指定する MaxSpareServers の値を小さく設定した．5.2.1 項の実験ではユーザ数 300 以上で OS がハングアップしたが，これらの設定変更を行うことでメモリ不足が軽減され，ユーザ数増加に対応できる．

設定を変更した 1 vs 1 と本システムに対し，5.2.1 項と同様の実験を行った．**図 9** に測定結果を示す．1 vs 1 方式はユーザ数 1,000 まで実験を継続できたが，終始本システムよりも低いスループットを示し，その低下の度合いも急激である．一方，本システムは利用者数増加にともなうスループットの低下が小さく，高いスケーラビリティを示している．これにより，1 vs 1 方式は性能とユーザ数に対するスケーラビリティがトレードオフの関係に陥っていることが確認された．

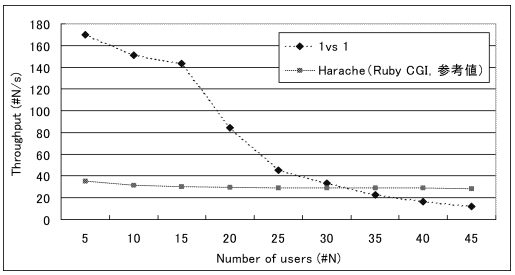


**図 10** スケーラビリティ評価実験：組み込みインタプリタ
Fig. 10 Experiment of scalability evaluation: Embedded interpreter.

#### 5.2.3 組み込みインタプリタ実験

また，1 vs 1 方式では mod_ruby や mod_perl などサーバ組み込みのインタプリタモジュールを用いることで Ruby [17]，Perl などで記述されたプログラムを高速に実行できる可能性がある．一方，本システムではセッションごとにサーバプロセスを終了させているため，組み込みインタプリタによる高速化は期待できない．

5.1 節で用いた 10 KBytes のテキストを表示する CGI と同等のプログラムを Ruby によって記述し，mod_ruby を有効にした 1 vs 1 方式で実行する実験を行った．その他の実験環境は 5.2.1 項と同様である．参考として，Ruby で記述した同等の CGI を本システムで実行した結果も示す．測定結果および本実験中のメモリ，スワップ利用率の変化をそれぞれ**図 10**，**図 11** に示す．1 vs 1 方式はスワッピングが顕著になり始めたユーザ数 20 から急激にパフォーマンス低下が起きており，ユーザ数 35 で本システムと逆転している．また，mod_ruby を有効にした 1 vs 1 方式は利用者数 50 以上でメモリ不足により OS がハングアップしてしまい実験が継続できなかった．一方，本システムは利用者数 1,000 まで実験を行うことができた．

5.2.1 および 5.2.2 項で述べた C CGI を用いた実験と比較してスループットが低い原因は Ruby 処理系によるものである．

以上のスケーラビリティ評価実験により，本システムは競合する 1 vs 1 方式に対しスケーラビリティにおいて圧倒的な優位性を持つことが確認できた．よって，Apache モジュールの開発をともなう本実装は有効な実現方法である．

## 6. 考　　察

本章では，共有型ホスティングサービスを対象とした WWW サーバの実現方式についての考察および，課金システムへの応用について述べる．

### 6.1 WWW サーバの実現方式

WWW サーバの実現方式の比較を**表 4** に示す．比較項目のうち基礎性能とは CGI 処理におけるピーク性能を示す．スケーラビリティとは，1 台のサーバ計算機に格納する利用者数の増加にともなうスループットの低下が小さいことを示す．基礎性能およびスケーラビリティは 5 章の実験結果を基に算出した．ユーザ権限に起因する問題の解決とは，2.1 節で述べたサーバを共有する利用者間のプライバシやセキュリティに関する問題を解決できるかどうかを示す．可用性とは，その実現方式が現在安定して利用可能な状態であるかどうかを示す．

以下，各実現方式について考察を行う．まず，本システムのベースとして用いた通常の Apache は，ユーザ権限に起因する問題をかかえている．suEXEC を有効にした Apache は CGI 処理における性能が通常の Apache より低い．また，CGI 処理に限ってはユーザ権限に起因する問題を解決しているが，テキスト処理においては解決していない．perchild MPM を有効にした Apache は仕様レベルではユーザ権限に起因する問題の解決が期待できるが，2.1 節で述べたとおり現在安定して利用できる状態にない．1 vs 1 方式はユーザ権限に起因する問題を解決しているが，スケーラビリティの面で不利である．

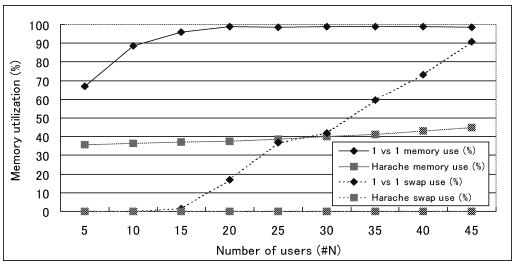


**図 11** スケーラビリティ評価実験：組み込みインタプリタ（メモリ利用率）

Fig. 11 Experiment of scalability evaluation: Embedded interpreter (memory utilization).

本システムは実用に耐えうる性能と高いスケーラビリティを達成しつつ，ユーザ権限に起因する問題を解決しており，有効な実現方式といえる．また，2.2 節で述べたとおり，ホスティングサービスなどの実環境では WWW サーバが利用する計算資源を低く抑えることが求められる．そのため，本システムのメモリ使用量の低さは実環境において大きなメリットとなる．

### 6.2 本システムの応用

本節では，Harache の応用例として，利用者ごとのアカウンティング情報の取得とアカウンティング情報を基にした従量課金について紹介する．

#### 6.2.1 アカウンティング情報の取得

本システムを用いることで利用者ごとにアカウンティング情報を取得することが可能となる．UNIX 系 OS には，各プロセスが利用した計算資源の情報（アカウンティング情報）を記録する機構が備わっている[18]．記録される情報の一例を以下に示す．

- プロセス名
- CPU 時間
- 平均メモリ使用量
- ユーザ ID

これらの情報をユーザ ID ごとにまとめると，各利用者が利用した CPU 時間，メモリ使用量などを算出することができる．既存の WWW サーバは特殊ユーザの権限で動作するため，利用者ごとにアカウンティング情報を取得することができない．一方，本システムではサーバプロセスをファイル所有者のユーザ権限

**表 4** 実現方式の比較

Table 4 Comparison of realization approaches.

| | 基礎性能（CGI） | スケーラビリティ | ユーザ権限に起因する問題の解決 | 可用性 |
|---|---|---|---|---|
| Apache | ○ | ○ | × | ○ |
| suEXEC | △ | ○ | △（CGI のみ） | ○ |
| perchild | - | - | △（仕様レベル） | × |
| 1 vs 1 方式 | ○ | × | ○ | ○ |
| 本システム | ○ | ○ | ○ | ○ |

で動作させるため，利用者ごとにアカウンティング情報を取得することが可能となる．

#### 6.2.2 計算資源に基づいた従量課金

本システムを用いることでホスティングサービスにおいて平等な従量課金を実現することができる．2.2節で述べたとおり，現在のホスティングサービスなどで用いられている従量課金はデータ転送量に応じたものが主であり，平等とはいい難い．そこで，データ転送量に加えて利用者ごとのアカウンティング情報を課金の指標とすることで，平等な従量課金を実現することが可能となる．

#### 6.2.3 スケジューリング優先度・利用制限

本システムではサーバプロセスがファイル所有者のユーザ権限で動作するため，利用者ごとにスケジューリング優先度の設定や計算資源の利用制限を容易に行うことができる．これにより，ホスティングサービスにおいてこれまで画一的であったサービス品質を利用者ごとにカスタマイズすることが可能となる．

UNIX系OSには，ネットワークの受信処理に優先度が反映されないという問題が存在する．そのため，従来はネットワークの負荷が高いときにプロセスのスケジューリング優先度を実際のネットワーク処理に反映させることが難しかった．しかし，最近ではネットワークの受信処理にスケジューリング優先度を反映させる動きがある[19]．本システムとこれらの技術を組み合わせることを想定すれば，利用者ごとにスケジューリング優先度を設定することに意義があるといえる．

## 7. おわりに

本研究では，ユーザ権限に起因する問題を解決するWWWサーバ，Harache の設計を行い，SELinux を有効にしたLinux OS上に実現した．実現したシステムに対する評価実験および考察を行った結果，本システムが他の実現方式に対してスケーラビリティの面で優位性を持つこと，実用に耐えうるだけの性能を達成していることを確認した．また，本システムの応用について述べ，本システムの有用性を示した．

今後の課題として組み込みインタプリタへの対応があげられる．

**謝辞** 本研究は，一部，独立行政法人情報処理推進機構（IPA）「未踏ソフトウェア創造事業」の支援による．

## 参考文献

1) 総務省：平成16年度版情報通信白書 (2004).
2) Goland, Y.Y. and Whitehead, E.J. Jr.: HTTP Extensions for Distributed Authoring – WEBDAV, RFC 2518 (1999).
3) PHP version 4. http://www.php.net/
4) 原 大輔，兵頭和樹，中山泰一：ファイル所有者のユーザ権限で動作するHTTPサーバの実現と評価，SEAソフトウェアシンポジウム2004論文集，pp.43–47 (2004).
5) Apache suEXEC Support. http://httpd.apache.org/docs/suexec.html
6) Grunbacher, A.: POSIX Access Control Lists on Linux, *Proc. FREENIX Track: 2003 USENIX Annual Technical Conference*, pp.259–272 (2003).
7) Apache MPM perchild. http://httpd.apache.org/docs-2.0/mod/perchild.html
8) mod_perl. http://perl.apache.org/
9) mod_ruby. http://modruby.net/
10) 大森敏行：セキュリティの切り札「セキュアOS」の正体，日経コンピュータ，No.602, pp.138–143 (2004).
11) 情報ネットワーク法学会セキュアOSと基盤ソフトウエアに関する研究会. http://in-law.jp/secure-os/
12) PHP: Persistent Database Connections, PHP Manual. http://www.php.net/manual/features.persistent-connections.php
13) SQL Relay. http://sqlrelay.sourceforge.net/
14) Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. FREENIX Track: 2001 USENIX Annual Technical Conference*, pp.29–40 (2001).
15) Apache HTTP Server. http://httpd.apache.org/
16) Mosberger, D. and Jin, T.: httperf—A Tool for Measuring Web Server Performance, *Proc. 1st Workshop on Internet Server Performance*, pp.59–67 (1998).
17) Ruby version 1.8.1. http://www.ruby-lang.org/
18) Frisch, A.（著），飯塚正樹，下田みどり（訳）：UNIXシステム管理第3版VOLUME 2，オライリージャパン (2003).
19) 尾崎亮太，中山泰一：Linuxにおけるプロセス優先度に基づく受信処理の実現，情報処理学会論文誌，Vol.45, No.3, pp.785–793 (2004).

(平成17年 2 月14日受付)
(平成17年10月11日採録)

**原　大輔**（学生会員）

1981 年生．2004 年電気通信大学情報工学科卒業．現在，同大学院電気通信学研究科情報工学専攻博士前期課程在学中．サーバ・アーキテクチャ，WWW，システム・ソフトウェアに興味を持つ．2005 年度 IPA 未踏ソフトウェア創造事業に採択され，Harache を発展させたウェブサーバ・アーキテクチャHi-sap の研究開発に従事．第 66 回情報処理学会全国大会学生奨励賞受賞．

**尾崎　亮太**（学生会員）

1979 年生．2003 年電気通信大学大学院電気通信学研究科情報工学専攻博士前期課程修了．同年総合研究大学院大学数物科学研究科（現複合科学研究科）情報学専攻博士後期課程入学．並列・分散処理，オペレーティング・システムに興味を持つ．

**兵頭　和樹**（正会員）

1976 年生．1999 年電気通信大学情報工学科卒業．2004 年同大学院電気通信学研究科情報工学専攻博士後期課程単位取得退学．2004 年 4 月より，東京工科大学コンピュータサイエンス学部助手．博士（工学）．並列・分散処理，ネットワーク，システム・ソフトウェアに興味を持つ．電子情報通信学会，ACM の会員．

**中山　泰一**（正会員）

1965 年生．1988 年東京大学工学部計数工学科卒業．1993 年同大学院工学系研究科情報工学専攻博士課程修了．工学博士．1993 年 4 月より，電気通信大学情報工学科助手．現在，同学科助教授．オペレーティング・システム，並列・分散処理，ゲーム・プログラミングに興味を持つ．日本ソフトウェア科学会，電子情報通信学会，IEEE CS 等の会員．