セキュアかつ高性能なウェブサーバの設計と実装

原 大輔　中山 泰一
hara-d@igo.cs.uec.ac.jp, yasu@cs.uec.ac.jp
電気通信大学 情報工学科

概要

既存のウェブサーバには，共有型ホスティングサービスのような大規模環境において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をセキュアかつ便利に利用できないという問題がある．この問題は，サーバプロセスが単一のユーザ権限で動作することに起因する．この問題に対し，本研究で提案するウェブサーバでは，サーバに格納されたウェブオブジェクトを分割し，パーティション毎に異なるユーザ権限でサーバプロセスを動作させる．これにより，組み込みモジュールを用いる際の問題を解決することが可能となる．また，ウェブサーバに特化したサーバプロセスのスケジューリング方式を提案し，サーバ計算機当たりのサイト格納数に対するスケーラビリティを最大化する．本稿では，提案するウェブサーバの設計及び実装，評価について述べる．評価実験の結果，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．

1　はじめに

近年，インターネットの普及により個人でウェブサイトを公開する人が増加している．サイトの設置場所として，共有型ホスティングサービスに人気が集まっている．これは 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であり，様々な機能を低コストで利用できるという特徴がある．

一方，既存のウェブサーバには，共有型ホスティングサービスのような大規模環境において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（以下，組み込みモジュール）をセキュアかつ便利に利用できないという問題がある．これは，組み込みモジュールをロードしたサーバプロセスが単一のユーザ権限で動作することに起因する．Weblog や Wiki[1]，CMS[2] に代表される動的コンテンツを高速に配信する組み込みインタプリタを用いる場合，サーバを共有する別のユーザからファイルを盗視･改竄される危険性がある．また，ブラウザなどのクライアントから HTTP 経由でサーバ上のファイルを操作する WebDAV[3] を用いる場合，作成されるファイルの所有者はサーバプロセスを実行するユーザとなるため，一般ユーザがサーバにリモートログインしてファイルを直接操作することが困難になる．

これらの問題に対し，我々はセキュアかつ高性能なウェブサーバアーキテクチャ，Hi-sap を提案している [4]．本アーキテクチャでは，サーバに格納された多数のサイト群をサイトやコンテンツなどのパーティションに分割し，パーティション毎に異なるユーザ権限でサーバプロセスを実行する．これにより，組み込みインタプリタを用いた際の盗視･改竄を防止することができ，かつ，WebDAV とファイルの直接編集を併用することが可能となる．このように，本アーキテクチャを用いることで，組み込みモジュールをセキュアかつ便利に利用することが可能となる．また，ウェブサーバレベルでのスケジューラ Content Access Scheduler を提案し，サーバ計算機当たりのサイト格納数に対するスケーラビリティを最大化する．

以下，2 章で本研究の背景について述べる．また，3 章において本アーキテクチャを採用したウェブサーバソフトウェアの設計について，4 章でその実装法について述べる．5 章では実現したシステムに対して行なった評価実験について報告する．6 章で関連研究について述べ，最後に 7 章で本稿をまとめる．

2　背景

本章では，本研究の背景としてサーバ内部のセキュリティ及び，組み込みインタプリタ，

Design and Implementation of a Secure and High-performance Web Server
Daisuke HARA, and Yasuichi NAKAYAMA
Department of Computer Science, The University of Electro-Communications

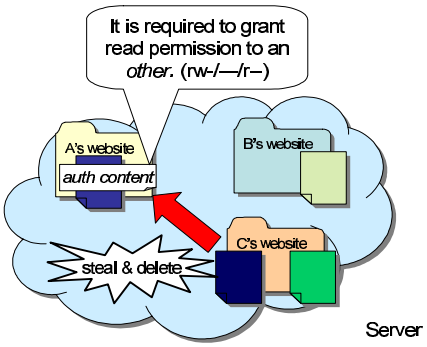

図 1: サーバ内部のセキュリティ

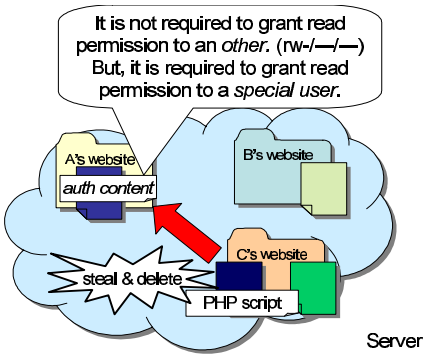

図 2: サーバ内部のセキュリティ：組み込みインタプリタ

共有型ホスティングサービスの現状について述べる．

2.1 サーバ内部のセキュリティ

既存のウェブサーバは特殊ユーザ[1] の権限で動作する．そのため，ウェブコンテンツを公開するためには UNIX のパーミッションモデル，所有者／グループ／その他，における “その他” に読み取り許可や実行許可を与える必要がある．これにより，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掛けているコンテンツであっても，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（図 1）．また，Weblog や Wiki のように CGI がファイルにデータを書き込む場合，“その他” に読み取り許可や書き込み許可を与える必要がある．そのため，別の利用者の CGI によってデータファイルを盗視・改竄される危険性がある．

これらの問題を解決するためには，suEXEC [5, 6, 7] と POSIX ACL [8] を併用する必要がある．まず，POSIX ACL を用いてウェブコンテンツの読み取り・実行許可を特殊ユーザだけに与える．これにより，“その他” に許可を与えることなくコンテンツを公開することが可能となる．その上で suEXEC を用いて，通常は一律に特殊ユーザの権限で動作する CGI をサイトの所有者毎に異なるユーザ権限で実行するようにする．これにより，データファイルをサーバを共有する別の利用者の CGI によって盗視・改竄される危険性がなくなる．

[1] ウェブ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．

2.2 組み込みインタプリタ

一方，最近では Weblog や Wiki，CMS に代表される動的コンテンツが従来の静的コンテンツに取って代わり流行している．セッション毎にサーバプロセスの生成と終了を繰り返す CGI では動的コンテンツの高速配信が難しいため，インタプリタをプロセスに内包する組み込みインタプリタを用いるケースが増加してきている．PHP[9] は通常 CGI ではなく，組み込みインタプリタで実行する方式を取り，Ruby や Perl，Python といった他のスクリプト言語でも同様の方式が一般的に用いられている [10, 11, 12]．

ところが，これら組み込みインタプリタは 2.1 節で述べた suEXEC と併用することができない．suEXEC は CGI であるため，セッション毎にプロセスの生成と終了を繰り返し，プロセスを使い回すことができないからである．そのため，組み込みインタプリタによって実行される組み込みスクリプトは特殊ユーザの権限で動作し，サーバ内部のセキュリティを保持することができない（図 2）．

2.3 Harache

我々はこれまでに，組み込みインタプリタや WebDAV に代表される組み込みモジュールをセキュアかつ便利に用いることを目的としたウェブサーバ Harache[13, 14] を実現した．Harache では，サーバプロセスをサイト毎に異なるユーザ権限で実行する（図 3）．そのため，組み込みスクリプトを含む全てのコンテンツのファイルパーミッションを “所有

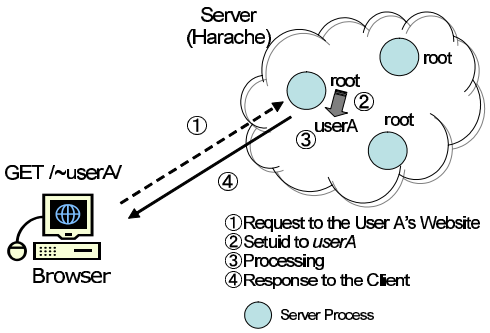

図 3: Harache

者” だけに与えるだけで済み，サーバ内部の高セキュリティを実現することが可能となる．

一方，Harache ではセッション毎にサーバプロセスを終了するため，組み込みインタプリタによる高速化を十分に活用できていないという問題がある．

本システムでは，Harache と同じくサーバ内の高セキュリティを実現しつつも，同時に動的コンテンツの高速配信を実現する．

2.4 共有型ホスティングサービスの現状

共有型ホスティングサービスにおける契約単位はサイトである．利用者は毎月数百円程度を支払うことにより，サーバのディスクスペースにコンテンツを格納し，サイトを開設することができる．

また，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．

一方，共有型ホスティングサービスでは，一日当たりのデータ転送量が制限されていることが一般的である．アクセス数の少ない個人サイトが多く，中には全くアクセスされないサイトも存在する可能性がある．

以上より，アクセスのないサイトにかかる計算資源量を最小化し，サーバ計算機当たりのサイト格納数をいかに増やすかが利益を生む鍵となる．

また，現在の共有型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別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言い難い．そのため始めから処理の重い特定の CGI の使用を禁止している共有型ホスティングサービスもある．

3 設計

本章では，我々が提案するウェブサーバ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

まず，サーバが格納するウェブオブジェクト[2] を複数のパーティションに分割する．各パーティションの粒度は分割方法によりサイトやコンテンツ[3]，QUERY_STRING[4] などになる．そして，Harache と同じく，サーバ内部の高セキュリティを実現するために，パーティション毎に異なるユーザ権限でサーバプロセスを実行する．

また，共有型ホスティングサービスでは，2.4 節で述べた通り，サーバ計算機当たりのサイト格納数に対するスケーラビリティが重要な要素である．ウェブサーバの場合，サイト格納数に対するスケーラビリティはメモリ使用量に強く左右される [13]．そのため，本システムでは，ウェブサーバレベルでのスケジューラ，Content Access Scheduler を用いて，サーバプロセスを動的に生成・終了し，メモリの有効利用を図る．

3.2 アーキテクチャ

本システムで提供する，サーバプロセスのユーザ権限によるアクセス制御は，攻撃者によって管理者権限を乗っ取られないことを前提としている．管理者権限を乗っ取られてしまうと任意のユーザ権限への遷移を許し，OS 全体を乗っ取られてしまう危険性がある．

そこで，本システムでは，セキュア OS[15] と連携し，管理者権限の乗っ取りによる被害を最小化する．セキュア OS では，ユーザ権限の遷移をセキュアに行なうことが難しい．本

[2] 外部に公開するファイルやディレクトリ，HTTP の環境変数などの集合．
[3] 例えば，Weblog や Wiki など．
[4] CGI などのスクリプトに与えられる引数のこと．

システムの場合，サーバプロセスを管理者権限で動作させ，リクエストに応じて各パーティション専用の一般ユーザの権限に遷移して処理を行なう方式が考えられる．しかし，セキュア OS のポリシにおいてこの動作を許可してしまうと，サーバプロセスの管理者権限を乗っ取られた場合，任意のユーザ権限に遷移できることになってしまい，セキュア OS を使う意味がない．そこで，本システムでは，全サーバプロセスを一般ユーザの権限で動作させる．

本システムのアーキテクチャを図 4 に示す．リバースプロキシ型のネットワーク構成を取り，(1)dispatcher プロセスがサイト閲覧者からのリクエストを受け付け，(2) 各パーティション専用の worker プロセス（サーバプロセス）へ転送する．(3)worker プロセスはリクエスト処理を行ない，(4)dispatcher を経由して，(5) 閲覧者にレスポンスを返す．

3.3 Content Access Scheduler

サーバ計算機当たりのサイト格納数に対するスケーラビリティを高めるために，ウェブサーバレベルでのスケジューラ，Content Access Scheduler を提案する．OS のスケジューラとは異なり，worker プロセスの生成と終了を司るものである．Content Access Scheduler の基本的な方針は以下の通りである．

- worker プロセスはサーバ起動時に生成するのではなく，クライアントからのリクエストを契機として生成する[5]
- 性能に影響が生じる高負荷状態に陥った場合，動作中の worker プロセスを適宜終了する

特に，終了する worker プロセスの選択アルゴリズムをコンテンツによって最適化することで，高スケーラビリティを実現することが可能となる．

3.4 パーティション

2.4 節で述べた通り，共有型ホスティングサービスにおける契約単位はサイトである．まず，ウェブオブジェクトをサイトに分割する（図 5）．そして，各パーティション（サイト）を異なるユーザ権限で実行する．

[5] リクエストのないパーティション用の worker プロセスは生成されない．

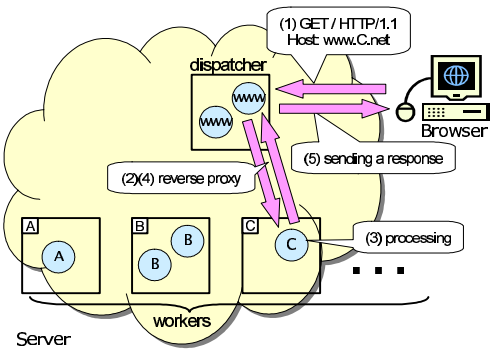

図 4: アーキテクチャ

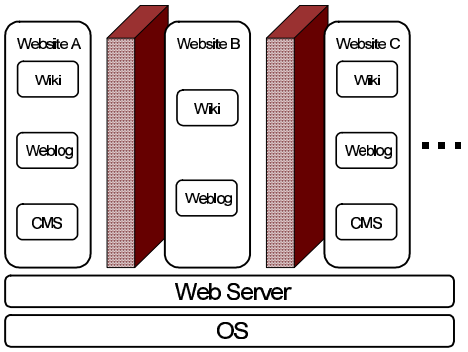

図 5: パーティション：サイト

また，各サイトには Weblog や Wiki，CMS などのコンテンツが 1 つ以上設置されている．本システムでは，サイトを更にコンテンツに分割し，各パーティション（コンテンツ）をそれが属するサイトとは別の権限で実行できるようにする．図 6 では，サイト A に Wiki，Weblog，CMS という 3 つのコンテンツが設置されている．Wiki と Weblog が同一権限（例えばユーザ A）で実行されるのに対し，CMS は別の権限（例えばユーザ A2）で実行される．

また，Wiki や Weblog などの動的コンテンツは静的コンテンツのようにファイル名に応じてページを切替えるのではなく，主に与えられた QUERY_STRING に応じてページを切替えたり，閲覧・編集・プレビュー・保存などの操作を行なったりしている．このため，QUERY_STRING に応じてアクセス制御を行なう必要がある．本システムでは，パーティションの粒度を QUERY_STRING に設定することにより，現在の動的コンテンツに対応したアクセス制御を提供する．

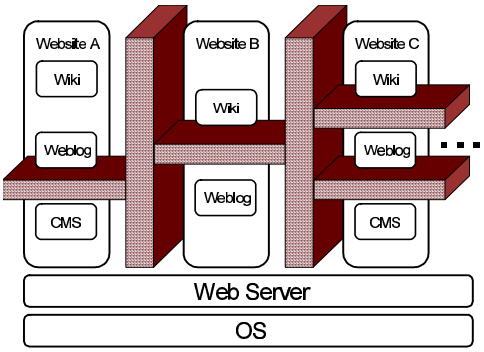

図 6: パーティション：コンテンツ

3.5 課金・利用制限

2.4 節で述べた通り，現在の共有型ホスティングサービスでは，課金に計算資源使用量が加味されていない．また，2.1 節で述べた通り，既存のウェブサーバはサーバプロセスが単一ユーザの権限で動作するため，サイト毎の計算資源使用量を取得することが難しい．一方，本システムでは，パーティション毎に異なるユーザ権限でサーバプロセスを実行するため，各パーティションが利用した計算資源の量を容易に取得することが可能である．そのため，計算資源使用量に応じた課金を実現することが可能となる．

また，本システムでは，サーバプロセスの利用制限も容易に行なうことが可能である．各パーティション毎にサーバプロセスの各種制限値[6] を設定することにより，特定のパーティションだけが計算資源を占有することを防ぎ，平等なサービスを提供することが可能となる．

4 実装

本章では，前章の設計に基づいた本システムの実装について述べる．本実装において，本システムは Linux OS 上に実装を行なった．dispatcher の機能は Apache HTTP Server 2.0.55[5] 上に Apache モジュール *mod_hisap* として実装した．worker には Apache HTTP Server 2.0.55 を 1000 個用いて，それぞれ異なるポートでリクエストをリスンするよう設定を行なった．また，worker の動的な起動・停止などの管理を行なう機能をデーモン *hisapd* として実装した．以下，dispatcher と hisapd の実装詳細及び，本実装における Content Access Scheduler のスケジューリング・アルゴリズムについて述べる．

[6] CPU 時間，メモリ使用量，オープンできるファイルディスクリプタの個数，生成できるプロセス数など．

4.1 dispatcher

dispatcher では，閲覧者からのリクエストを受け取ると，まず，hisapd にリクエスト先のパーティションを担当する worker が起動しているか確認を行なう．dispatcher と hisapd 間の通信は UNIX ドメインソケットを通して行なうようにした．次に，リクエストを受けた worker の識別子 (*Worker ID*) を専用のログファイル *Worker Request Log* に記録する．hisapd から担当 worker の準備が整った旨の通知を受け取ると，リクエストを担当 worker に転送する．worker のリクエスト処理が終了すると，worker からレスポンスを受け取り，閲覧者に転送を行なう．

4.2 hisapd

hisapd では，dispatcher からの要請に基づき，worker を起動する．worker を起動した際，Worker ID を専用のログファイル *Active Worker List* に記録する．また，Content Access Scheduler による worker 停止機能は hisapd に実装した．worker を停止した際，Worker ID を専用のログファイル *Idle Worker List* に記録する．dispatcher から，worker の生存確認要請を受け取ると，Active Worker List と Idle Worker List を基に生存を確認し，停止している場合は起動処理を行なう．

4.3 スケジューリング・アルゴリズム

本実装における Content Access Scheduler のスケジューリング・アルゴリズムはスラッシングに注目した．ウェブサーバの性能はスラッシングが起きると急激に低下する [13]．そのため，スラッシングが起きると判断した場合，worker を停止するようにした．スラッシングが起きると判断する条件は以下の通り．

- スワップアウトが発生していること
- メモリ使用量が 99 % を越えていること

現実装では，5 秒おきに ALRM シグナルを発生させて上記の値をチェックし，両方の条件をともに満たす場合，worker を停止するよう

表 1: 実験環境

スイッチングハブ	
製品名	DELL PowerConnect 2724
ポート	1000 BASE-T ×24

クライアント	
CPU	Intel Pentium Ⅲ Xeon 500 MHz ×4
Memory	256 MBytes (swap 512 MBytes)
OS	Fedora Core 4 (Linux 2.6.14)
NIC	Intel PRO/1000XT 1 Gbps

サーバ	
CPU	AMD Opteron 240EE 1.4 GHz ×2
Memory	4 GBytes (swap 8 GBytes)
OS	Fedora Core 4 (Linux 2.6.14)
NIC	Broadcom BCM5704C 1 Gbps

にした．また，停止する worker の選択条件は以下の通り．

- 起動していること
- Worker Request Log の最近 10000 アクセスに記録されていないこと

Worker Request Log を検索する時間を短縮するために，完全な LRU ではなく，検索範囲を制限した．

5 評価

本章では，本システムに対して行なった基礎性能評価実験について報告する．実験環境として，1 台のサーバ計算機及び計測用の 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及び OS については表 1 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．各計算機とスイッチングハブ間は Gigabit 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なった動的コンテンツ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，性能が実用に耐え得ることを確認する．

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 2.0.55 と suEXEC を有効にした Apache HTTP Server 2.0.55 を用いた．本システム，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．

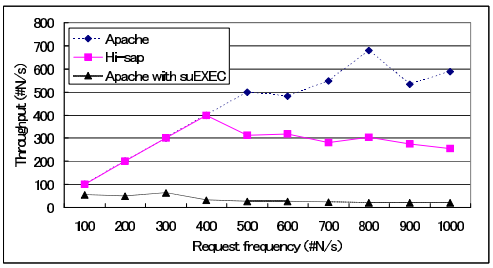

図 7: 基礎性能評価実験

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．リクエスト対象には，サーバ計算機にインストールされている PHP 処理系に関する様々な情報[7] を出力する phpinfo() 関数を呼び出す PHP スクリプトを用いた．1 回のリクエストにおけるデータ転送量は 40 KBytes 程度である．図 7 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 28.0 %，最大 56.5 %の性能低下が見られた．本システムでは，リクエストが一旦 dispatcher を経由する．性能低下の要因はこのオーバヘッドによるものである．一方で，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言える．

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得る性能を達成していることを確認した．

6 関連研究

本章では，本研究に関連する OS 及び，言語処理系，ウェブサーバに関する既存技術について述べ，本システムとの比較を行なう．

6.1 サンドボックス・VM

サーバソフトウェアやそれを搭載する OS を安全に実行することを目的として，サンドボックス [17] や仮想計算機 (VM)[18] が数多く提案されている．これらは，サーバソフトウェアや OS がクラックされたり誤動作したりしても，同じ計算機で動作する他のサーバ

[7] コンパイルオプション，バージョン，環境変数，OS のバージョン，HTTP ヘッダなど．

表 2: ウェブサーバ実現方法の比較

	サーバ内部のセキュリティ	動的コンテンツの処理性能	サイト格納数に対するスケーラビリティ	汎用性
通常のウェブサーバ	×	◎	○	○
suEXEC & POSIX ACL	○	×	○	○
サンドボックス	◎	◎	△	○
VM	◎	◎	×	○
PHP セーフモード	○	◎	○	×
Apache perchild MPM	○	–	△	○
本システム	◎	○	○	○

ソフトウェアや OS に影響を与えないようにする機構である．

サーバに格納するサイト毎にサンドボックスや VM を割り当てることにより，サーバ内の高セキュリティを実現することができる．しかし，これらの機構を適用することで各サイトの運用に必要な計算資源が大幅に増加してしまう．そのため，共有型ホスティングサービスのように，1 台のサーバに数百〜1000 程度のサイトを格納することは難しい．

一方，本システムでは，Content Access Scheduler を用いた worker の動的な起動・停止を行なうことで，サイト格納数に対する高いスケーラビリティを実現する．

6.2 言語処理系

動的コンテンツを記述するスクリプト言語の処理系のレベルでセキュリティを高める機構が存在する．PHP[9] のセーフモードでは，スクリプトに以下の制限を課すことでサーバ内部の高セキュリティの実現を目指している．

- スクリプトとそのスクリプトが操作しようとしているファイルの所有ユーザ（または所有グループ）が一致している場合のみスクリプトの実行を許可
- ファイル操作を特定のディレクトリ配下のみに制限
- 変更できる環境変数の制限
- 特定の関数・クラスの無効化

しかし，この機構は処理系に依存し，標準的なものではない．また処理系毎に設定が必要になり，作業が煩雑になるという欠点もある．

本システムでは，処理系に依存しない汎用的なセキュリティ機構を提供する．これにより，セーフモードのないスクリプト言語であってもサーバ内の高セキュリティを実現することができる．

6.3 Apache 関連

Apache HTTP Server[5] には perchild という MPM (Multi-Processing Module) が存在していた．これは，サイト毎に異なるユーザ／グループ権限でウェブサーバを実行する機構である．仕様上はサーバ内部の高セキュリティを実現できる可能性がある．しかし，サーバ起動時に各サイト専用のサーバプロセスを固定数ずつ生成する必要があり，リクエストのないサイトであっても専用プロセスが常駐する．その結果，サイト格納数が増えるに従ってサーバプロセスの数が増大し，スケーラビリティが低くなってしまう．また，perchild が安定して動作するとの報告はなく，開発は中止された．

6.4 ウェブサーバ実現方法の比較

本節では，ウェブサーバの各種実現方法について比較を行ない，本システムの有効性を確認する．表 2 にウェブサーバ実現方法の比較結果を示す．なお，動的コンテンツの処理性能の項目は，組み込みインタプリタが利用できない suEXEC & POSIX ACL 以外，組み込みインタプリタを利用した場合の評価を算出した．

通常のウェブサーバは，サーバ内部のセキュリティが低い．suEXEC と POSIX ACL を併用する方法は，サーバ内部の高セキュリティを実現するが，性能が低い．6.1 節で述べたサンドボックスや VM は，サーバ内部の高セキュリティを実現するが，サイト格納数に対するスケーラビリティが低い．6.2 節で述べた PHP のセーフモードは汎用性に問題がある．6.3 節で述べた Apache の perchild MPM はスケーラビリティに問題がある．また，性能

は不明である．

一方，本システムは全ての項目で高い評価が期待でき，バランスが良い．まず，パーティション毎に異なるユーザ権限でサーバプロセスを実行し，その上，セキュア OS と連携するため，極めて高いセキュリティを実現する．また，dispatcher を経由することによるオーバヘッドはあるが，組み込みインタプリタを用いるため，性能が高い．perchild とは異なり，Content Access Scheduler を用いた worker の動的な起動・停止を行なうため，スケーラビリティが高い．以上の議論により，本システムの有効性を確認することができた．

7 おわりに

本研究では，共有型ホスティングサービスなどの大規模環境を対象に，セキュアかつ高性能なウェブサーバの設計を行ない，Linux OS 上に実現した．実現したシステムに対する評価実験及び考察を行なった結果，本システムが実用に耐え得る性能を達成していることを確認した．

今後，サイト格納数に対するスケーラビリティに関する評価実験を行なう予定である．また，今後の課題として，性能向上のための dispatcher 改良及び多数の worker を管理するためのツールの作成，組み込みインタプリタの代わりに FastCGI[19] を用いた実装の検討が挙げられる．

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．貴重な御助言を下さった並木美太郎 PM に感謝致します．

参考文献

[1] WikiWikiWeb. http://c2.com/cgi/wiki?WikiWikiWeb

[2] Steve Goodwin, et al.: Content, content, everywhere...time to stop and think? The processof Web content management, *IEE Computing & Control Engineering Journal*, Vol.13, No.2, pp.66–70 (2002)

[3] Yaron Y. Goland, et al.: HTTP Extensions for Distributed Authoring – WEBDAV, *RFC 2518* (1999).

[4] 原大輔，中山泰一：セキュアかつ高性能なウェブサーバアーキテクチャ Hi-sap の提案，日本ソフトウェア科学会第 7 回インターネットテクノロジーワークショップ (WIT 2005) (2005).

[5] Apache HTTP Server. http://httpd.apache.org/

[6] Nathan Neulinger: CGIWrap: User CGI Access. http://cgiwrap.unixtools.org/

[7] Sebastian Marsching: suPHP. http://www.suphp.org/

[8] Andreas Grunbacher: POSIX Access Control Lists on Linux, *Proc. FREENIX Track: 2003 USENIX Annual Technical Conference*, pp.259–272 (2003).

[9] PHP: Hypertext Preprocessor. http://www.php.net/

[10] mod_ruby. http://modruby.net/

[11] mod_perl. http://perl.apache.org/

[12] mod_python. http://www.modpython.org/

[13] 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 (2005 予定).

[14] Daisuke Hara, et al.: Design and Implementation of A Web Server for A Hosting Service, *Proc. the 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, pp.69–74 (2005).

[15] Peter Loscocco, et al.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. FREENIX Track: 2001 USENIX Annual Technical Conference*, pp.29–40 (2001).

[16] David Mosberger, et al.: httperf—A Tool for Measuring Web Server Performance, *Proc. 1st Workshop on Internet Server Performance*, pp.59–67 (1998).

[17] Poul-Henning Kamp, et al.: Jails: Confining the omnipotent root, *Proc. of the 2nd International System Administration and Networking Conference (SANE)* (2000).

[18] Paul Barham, et al.: Xen and the Art of Virtualization, *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, pp.164–177 (2003).

[19] FastCGI. http://www.fastcgi.com/