情報処理学会論文誌　コンピューティングシステム　Vol.10　No.1　1–18 (May 2017)

Webアプリケーションサーバにおけるプロセスのふるまいに基づいたDoS攻撃の防御手法

中川 岳[1,†1,a)]　追川 修一[2,†2]

受付日 2016年10月27日, 採録日 2017年2月22日

概要：Denial-of-Service 攻撃（DoS 攻撃）から Web アプリケーションを防衛する手法としては，Web Application Firewall（WAF）を用いる方法，OS のリソース制限機構を用いる方法，クライアントからのリクエスト傾向から DoS 攻撃の可能性を判定する方法などさまざまな方法が提案されてきた．しかしながら，それらの方法は，Web アプリケーションの脆弱性を利用して，大量のリソースを消費させる DoS 攻撃には十分に対処できない．そこで本論文では，Web アプリケーションの DoS 攻撃の防御手法として，プロセスのメモリ消費の傾向を利用したリソース制限を提案する．DoS 攻撃の原因となるリクエストを受け取ると，そのリクエストを処理するプロセスは急速に大量のリソースを消費する．提案手法では，このリソースの急速な消費を検出し，そのプロセスに対してリソースの利用制限を行う．これにより，DoS 攻撃によるリソース浪費を抑制し，正常なリクエストの処理性能の低下を防止する．提案に基づいて，メモリ消費の傾向に基づいた DoS 攻撃への対策機構を設計，実装し，評価実験を行った．結果として，DoS 攻撃下にある Web アプリケーションのリクエスト処理性能を最大で 4.3 倍に改善することができた．また，提案手法による，Web アプリケーションのリクエスト処理性能の性能低下は，最大でも 5.0%程度と，非常に小さいことも確認できた．

キーワード：オペレーティングシステム，リソース管理，Denial-of-Service 攻撃

Mitigating Denial of Service Attacks on Web Application Servers Using Resource Consumption Behavior

Gaku Nakagawa[1,†1,a)]　Shuichi Oikawa[2,†2]

Received: October 27, 2016, Accepted: February 22, 2017

Abstract: There are several previous work to against Denial-of-Service attacks (DoS attacks). They, however, cannot prevent DoS attack that run out system resource via the vulnerabilities of web applications. In this paper, we will propose a new method to attack such DoS attack problem. The new method utilizes the resource consumption behavior to process the requests to the target web applications. When a process receives a DoS attack request, the process consume huge resources, such as RAM or CPU, rapidly. The proposed method detects the signs of the illegal resource consumption, and impose a resource limitation on the cause process. We designed a resource management mechanism based on the discussion and implemented it. The result of the evaluation experiment shows that our management mechanism improves the request processing performance in DoS attack situation more than 4 times.

Keywords: operating system, resource management, Denial-of-Service attack

[1] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, University of Tsukuba, Tsukuba, Ibaraki 305–0006, Japan
[2] 筑波大学システム情報系
Division of Information Engineering, Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305–0006, Japan
[†1] 現在，富士通株式会社
Presently with Fujitsu Limited
[†2] 現在，株式会社フィックスターズ
Presently with Fixstars Corporation
[a)] gnakagaw@cs.tsukuba.ac.jp

1. はじめに

Web アプリケーションはアプリケーションソフトウェアの提供形態の 1 つである．Web アプリケーションは Web サーバによって実行され，その結果は動的な Web ページとしてユーザへ届けられる [1]．この Web アプリケーションはオンラインサービスの提供手段として主流になりつつある．

Web アプリケーションのシステム構成としては，http サーバ，アプリケーションサーバ（AP サーバ），データベースサーバ（DB サーバ）の 3 種類のサーバによる構成が広く用いられている．ユーザからのリクエストはまず http サーバで処理され，その処理に基づいて AP サーバでサービスを提供するプログラムが動作する．AP サーバは必要に応じて DB サーバにアクセスし，必要な情報の取得や更新を実施する．リクエストに関する処理が終了すると，AP サーバはユーザへの応答を生成し，http サーバ経由で送信する．このうち AP サーバは，同時に複数のリクエストを処理するために，1 つのマスタープロセスと複数の子プロセスによって構成されている．http サーバからのリクエストを受信したマスタープロセスは，子プロセスを選択し，リクエスト処理を依頼する．

Web アプリケーションはサービス拒否攻撃（Denial of Service 攻撃：DoS 攻撃）を受ける可能性がある．DoS 攻撃はコンピュータシステムに対する攻撃の一種で，システムの正常なサービス提供を妨害することを目的とする [2]．その形態としては，大量のリクエスト送信によりネットワークを輻輳させるものや，サーバプログラムが不具合を起こすデータを意図的に送信するものなど多岐にわたる．

DoS 攻撃の中でも，Web アプリケーションの脆弱性を利用してリソースを急速に大量消費させる攻撃は特に問題である．Web アプリケーションは，その開発者やシステム管理者の意図に反して，大量のリソースを急速に消費することがある．その原因としては，アプリケーション自体のプログラミングミスや，アプリケーションフレームワーク，ライブラリ，ランタイムといった，実行環境の不具合があげられる．この意図しないリソース消費を引き起こすリクエストを発行することで，対象の Web アプリケーションを提供するシステムの資源を浪費させ，正当なリクエストの処理を阻害する．攻撃が発生した場合は，正当なリクエストの実行遅延や，システムの停止を招き，Web アプリケーションの運用が阻害される．大量のデータを送信してネットワークを輻輳させる量的な攻撃に対して，この脆弱性を利用したリソース浪費攻撃は少ないアクセス規模で問題を引き起こすことが可能である．Web アプリケーションに関するリソースの大量消費を引き起こすソフトウェアの不具合はこれまで多数報告されており [3], [4], [5]，また近年のソフトウェアの更新サイクルの速さを考慮すると，これからもソフトウェア脆弱性を利用した DoS 攻撃は増加していくと考えられる．

Web アプリケーションに対する DoS 攻撃を防ぐ方法としては，オペレーティングシステム（OS）レベルやアプリケーションレベルでのリソース制限機構を利用する方法，Web Application Firewall（WAF）を利用する方法がある．また，Web アプリケーションに対する DoS 攻撃を防ぐことを目的とした先行研究も存在している [6], [7], [8], [9]．しかしながらこれらの方法には，攻撃に関する事前知識が必要である，対象としている DoS 攻撃が異なる，Web アプリケーションのクライアント側での処理が要求されるなどの問題がある．

本論文では，メモリ消費の傾向を利用して，Web アプリケーションを DoS 攻撃から防御する手法を提案する．Web アプリケーションへのリクエストを処理するプロセスには，そのメモリ消費に一定の傾向がある．提案手法では，メモリ消費の傾向に着目して，DoS 攻撃を引き起こすリクエストを検出し，検出したリクエストを処理するためのリソース利用に制限を課す．これにより DoS 攻撃によるリソース消費は一定以下に制限され，正常なリクエストの処理性能の低下を抑制することができる．

提案手法は個々のリクエスト処理のメモリ消費の傾向に基づく DoS 攻撃の検出を行うため，1) 少量のアクセス規模で問題を引き起こす DoS 攻撃に対処できる．また，提案手法は攻撃の特徴などを検出に用いないため，2) 個別の攻撃の知識がなくても，問題に対応できる．さらに，提案手法はこれまでの先行手法とは異なり，DoS 攻撃と判定したリクエストを遮断したり，処理を中断しない．リソース制限を課しながらも，処理を継続することができる．そのため，3) 正常なリクエストを DoS 判定であると誤検出した場合でも，リクエストの処理を継続し，クライアントへ結果を送信することが可能である．

提案手法の実装例として，メモリ消費のふるまいに基づいたリソース制限機構を Linux kernel 3.14.0 を対象に設計し，実装と評価実験を行った．結果として，提案手法を用いることで，DoS 攻撃下にある Web アプリケーションの性能を最大で 4.3 倍に改善することができた．また，提案手法の Web アプリケーションのリクエスト処理性能に対するオーバヘッドは最大でも 5.0%であることが分かった．

本論文の構成は以下のとおりである．第 2 章では，本論文の背景である DoS 攻撃と既存の対策手法について述べる．第 3 章では，本論文で提案するメモリ消費のふるまいに基づくリソース制限手法について述べる．第 4 章では，提案手法の実装例として，メモリ消費のふるまいに基づいたリソース制限機構の設計と実装について述べる．第 5 章では第 4 章で述べたリソース制限機構の評価実験について述べる．第 6 章では本論文をまとめる．

2. 研究の背景

本論文では，Web アプリケーションの脆弱性を利用した DoS 攻撃を防止する方法を提案する．その背景として，本論文が対象とする DoS 攻撃と，既存の対策手法の問題点について述べる．

2.1 DoS 攻撃

DoS 攻撃は，あるサービスを提供するシステムに対して，意図的にそのサービスが正常に利用できないようにする行為である [2]．DoS 攻撃は対象とするシステムや手段によってさまざまな種類があるが，本論文では Web アプリケーションの脆弱性を利用して，急速なリソース消費を引き起こす DoS 攻撃を議論の対象とする．

Web アプリケーションに対する DoS 攻撃は，攻撃の対象，攻撃を引き起こすリクエストの規模によって 4 種類に分類することができる（**図 1**）．ここでは分類した 4 つの攻撃を，分類 A，分類 B，分類 C，分類 D とする．DoS 攻撃は，攻撃対象によってまず 2 つに分類することができる．1 つは Web アプリケーションを提供するプログラムを対象にしたもの，もう 1 つは，ユーザと Web アプリケーションの間の通信を提供するネットワークプロトコル，http サーバなどのネットワークシステムを対象にしたものである．DoS 攻撃は，攻撃を引き起こすリクエストの規模によってさらに 2 つに分類することができる．1 つは攻撃対象に正常なリクエストを大量に送信して，Web サービスの正当な利用者の通信を阻害したり，攻撃対象のリソースを枯渇させるものである．もう 1 つは，攻撃対象の異常を引き起こすリクエストを送信し，攻撃対象を停止させたり，攻撃対象のリソースを枯渇させるものである．本論文が対象にするのは，分類 B の DoS 攻撃である．

本論文が対象とする DoS 攻撃の例として，コンテンツマネジメントシステムとして広く利用されている WordPress [10] に関する脆弱性を取り上げる [4]．WordPress には XML-RPC [11] 経由で記事の操作を行う機能がある．WordPress で構築した Web サイト内の特定の URL に操作内容を指示する XML を送信することで，記事の取得や作成を行うことができる．WordPress の特定バージョンにはこの XML-RPC の処理に関する不具合があり，不適切な XML を送信することで Quadratic Blowup 攻撃 [12] を引き起こすことができる．Quadratic Blowup 攻撃は XML のエンティティ展開を悪用した攻撃である．DTD 内でエンティティとして長大な文字列を定義し，それを XML 本文中で大量に参照させると，XML パーサに大量のメモリ空間と CPU 時間を消費させることができる．この問題のある XML-RPC リクエストを送信することで，WordPress をホストしているシステムのリソースを急速に消費させ，WordPress に対して DoS 攻撃を引き起こすことができる．

同様の脆弱性は他の Web アプリケーションについても報告されている．ウィキエンジンの実装の 1 つである MediaWiki [13] の特定のバージョンでは，不適切なファイルのアップロードにより，メモリの大量消費を引き起こす脆弱性がある [5]．MediaWiki ではページへの添付ファイルとして SVG 画像をアップロードすることができる．SVG 画像は XML で表現されたベクタ画像データであり，その XML を処理する際に Quadratic Blowup 攻撃が起こる可能性がある．MediaWiki の特定のバージョンにはこの SVG 画像の処理に関する不具合があり，添付ファイルのアップロードにより Quadratic Blowup 攻撃を引き起こすことが可能である．

なお，Web アプリケーションやサーバソフトウェアの脆弱性を利用した DoS 攻撃としては，ソフトウェアのメモリリーク脆弱性を悪用し，長期的にメモリを浪費するものもある．本論文では，この種の攻撃は対象としない．Web アプリケーションにおいては，この種の攻撃は，AP サーバを定期的に再起動することや，1 つのリクエストの処理時間に制限を設けることで十分に対処可能である．本論文では，既存の方法では対策の難しい，急速なメモリ消費をともなう DoS 攻撃からの Web アプリケーションの防衛を議論する．

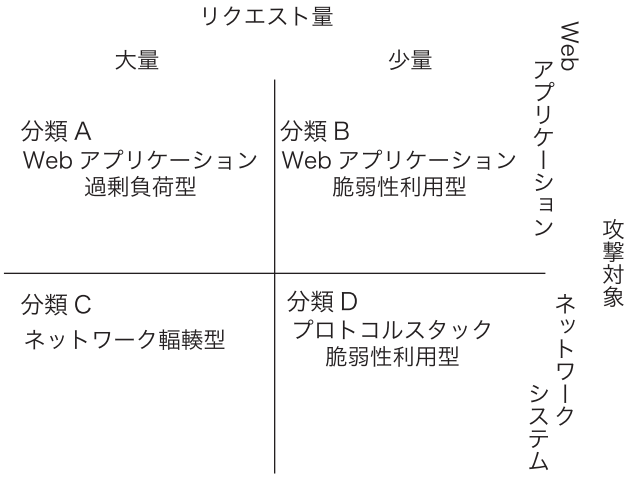


図 1　DoS 攻撃の分類
Fig. 1　Classification of DoS attacks.

2.2 メモリの使用量に基づいたリソース制限

OS には一般的にプロセス単位で利用可能なリソースを制限する機構を持つ．また Web アプリケーションを実行するランタイムがリソース制限機構を持つこともある．これらのリソース制限機構を適切に利用すれば，Web アプリケーションによって急速なリソース消費が起こったとしても，その消費は一定以下に抑えることができる．しかしながら，メモリの使用量だけに基づいたリソース制限だけでは，本論文が問題とする，急速なメモリ消費をともなう

DoS 攻撃から Web アプリケーションを防御することができない．

プロセスごとに使用可能なメモリ量を制限すれば，たしかに，プロセスごとのメモリの浪費は設定値以下に抑えることが可能である．しかしながら，本論文の想定環境では，このメモリの浪費が同時に複数発生する可能性がある．Web アプリケーションサーバでは，リクエストを同時に複数処理するために，AP サーバの子プロセスが複数動作している．この環境下でメモリ浪費を引き起こす複数のリクエストが同時に処理されれば，システム全体としてメモリ使用量が増大し，結果としてメモリ不足を引き起こす．このメモリ不足は大量のスワップ処理を発生させ，正常なリクエストの処理性能の低下を招く．特に，メモリオーバコミットを前提として，AP サーバのプロセス数が多めに設定されている場合は，その性能低下は顕著になる．つまり，本論文が対象とする Web アプリケーションサーバにおいては，単純なメモリ使用量の制限だけでは，DoS 攻撃を防ぐことができない．

プロセスレベルのメモリ使用量制限ではなく，AP サーバ群が合計で使用できるメモリ量を規制する方法や，AP サーバのプロセス数を少なめに設定する方法も DoS 攻撃対策として考えられる．これらの方法は DoS 攻撃発生時のメモリ不足によるシステム全体の安定性確保には寄与するが，AP サーバ群のリクエスト処理性能を保証することはできない．なぜなら，その限定されたリソースを DoS 攻撃を引き起こすリクエストがメモリを浪費すれば，結果として AP サーバ全体が使用できるメモリの量が不足し，正常なリクエストの処理が阻害されるからである．このように，OS やアプリケーションランタイムが提供するメモリ使用量の制限は，個々のプロセスレベルでのメモリの浪費を抑制する効果があるが，本論文が対象とするような DoS 攻撃による Web アプリケーションのリクエスト処理性能の低下を防ぐことができない．

2.3　その他の対策手法

Web Application Firewall（WAF）は，Web アプリケーションのクライアントとサーバの間で，サーバに送られる通信を監視し，問題のあるリクエストを検出する機構である [1]．WAF では HTTP リクエストの内容を検査するため，Web アプリケーションの脆弱性を利用した攻撃を検出することが可能である．しかしながら，WAF はルールベースの検出を行っており，未知の脆弱性を利用した攻撃を防ぐことができない．また，検出に用いられるルールは広く利用される Web アプリケーションについて作成されるため，利用規模の小さい Web アプリケーションには対応できない．本論文の提案手法では，ルールに依らない DoS 攻撃の検出を行うため，この問題は発生しない．

Web アプリケーションに対する DoS 攻撃の防止手法については，いくつかの先行研究がある [6], [7], [8], [9]．このうち Xu らの研究 [6]，Barna らの研究 [9] は対象にしている攻撃の種類が本論文とは異なっている．Ranjan らの研究 [8]，Srivatsa の研究 [7] は本論文と同様に Web アプリケーションレベルのソフトウェア脆弱性を利用した DoS 攻撃の防止手法を議論している．

Ranjan らの提案手法 [8] は Web アプリケーションに対するセッションごとに，DoS 攻撃である確率を表す指標を算出し，その指標に基づいて DoS 攻撃を検出，リクエスト処理の制限を課す．その指標の算出にはクライアントからのリクエスト到達時間，リクエストを処理するためのリソース消費の傾向などを用いる．この手法はリクエスト処理のためのリソース消費に着目する点では，本論文の提案手法と類似している．しかしながら，Ranjan らの方法 [8] は実際にリクエストを処理する前に，DoS 攻撃であるか否かの予測を行う．本論文の提案手法では，それぞれのリクエスト処理時に実際に消費するリソース量に基づいてリソース制限を課す．そのため，より正確に DoS 攻撃の疑いのあるリクエスト処理に関するリソース利用を制限することができる．

Srivatsa ら [7] の提案手法はクライアント，サーバ間でやりとりする TCP パケットに認証情報を埋め込み，正しい認証情報を持たないパケットを破棄する手法である．正しい認証情報を生成するための鍵情報はサーバから取得することができ，この発行する鍵の数を限定しておくことで，同時にアクセス可能なクライアントの数を一定以下に保つことができる．この認証情報の交換処理は HTTP レスポンスとして返される Web ページの中に JavaScript として埋め込まれており，既存の通信プロトコルを変更したり，新しいプロトコルを追加せずにクライアント，サーバ間の認証を行うことができる．Srivatsa [7] の提案手法はクライアント側での処理が必要なのに対して，本論文での提案手法はサーバ側での処理のみで DoS 攻撃を防止することができる．また，Srivatsa [7] の提案手法は JavaScript が実行可能な Web ブラウザによるアクセスを前提としており，ブラウザを用いない Web API を処理するシステムなどには適用できない．これに対して本論文の提案手法はクライアントの種別に依らず，すべての HTTP リクエストに対応することが可能である．

3.　メモリ消費のふるまいに基づく DoS 攻撃の検出と防御

2.3 節で述べたように，既存手法では単一のリクエストでリソースの大量消費を引き起こす DoS 攻撃を防止することは難しい．この問題を解決するために，本論文ではプロセスのメモリ消費のふるまいに着目したプロセスのリソース制限を提案する．この章では，プロセスのメモリ消費のふるまいの定義，その DoS 攻撃対策への有用性について述

べる．

3.1　メモリ消費のふるまいと DoS 攻撃

メモリ消費のふるまいとは，プロセスごとに異なるメモリ消費の傾向である．コンピュータシステムで実行されるプログラムは，それぞれメモリを消費する量や，消費する速度に違いがある．また，同じプログラムでも，処理するデータや処理内容よってメモリ利用の傾向は異なる．このプロセスのメモリ消費の傾向のうち，単位時間あたりのメモリ消費量を，プロセスのメモリ消費のふるまいと定義する．

このメモリ消費のふるまいは，Web アプリケーションの DoS 攻撃の検出に有用である．Web アプリケーションに対するリクエストは，その実行を担うプロセスによって処理される．このプロセスのメモリ消費のふるまいには，処理する Web アプリケーションごとに固有の傾向がある．また，正常なリクエストの種類は有限であるため，そのふるまいは一定に収束する．これに対して，DoS 攻撃を引き起こすリクエストを処理するときは，定常時のふるまいとは異なるふるまいをする．この異常なメモリ消費のふるまいを検出することで，DoS 攻撃を検出することが可能である．

図 2，**図 3** はそれぞれ 2.1 節で取り上げた，WordPress と MediaWiki についてメモリ消費のふるまいを示したものである．青線は，それぞれのアプリケーションで正常なリクエストを処理したときのメモリ消費のふるまいである．この正常なリクエストは，テキストのみの記事（1 KiB，10 KiB，100 KiB）の取得，画像を含んだ記事の取得（100 KB，1 MiB，10 MiB），テキストのみの記事の投稿処理（1 KiB，10 KiB，100 KiB）の 9 種類で，それぞれのリクエストを順に処理したときのメモリ消費のふるまいの変化が示されている．赤線は，それぞれのアプリケーションで DoS 攻撃を引き起こすリクエストを処理したときのメモリ消費のふるまいを示している．青線で示された正常なリクエストを処理するときの，メモリ消費のふるまいに比べると，明らかに異なることが分かる．このふるまいの違いを利用すれば，本論文が対象とする，単一のリクエストで大量のメモリ消費を発生させる DoS 攻撃を検出することが可能である．

このようなメモリ資源を大量に消費するプログラムについて，メモリ消費のふるまいを使わずとも，プロセスのメモリ使用量に上限を設けることで対策をすることも可能である．ユーザプログラムについて，OS レベルやプログラムランタイムレベルでメモリ使用量の制限を課すことは，一般的に行われる．このメモリ使用量の制限を課しておけば，プロセスが大量のメモリ浪費を起こしたとしても，その量は一定以下に抑制することができる．しかしながら，2.2 節で述べたとおり，本論文が対象とする Web アプリケーションサーバに対する DoS 攻撃は，メモリ使用量の制限だけでは十分に防ぐことができない．

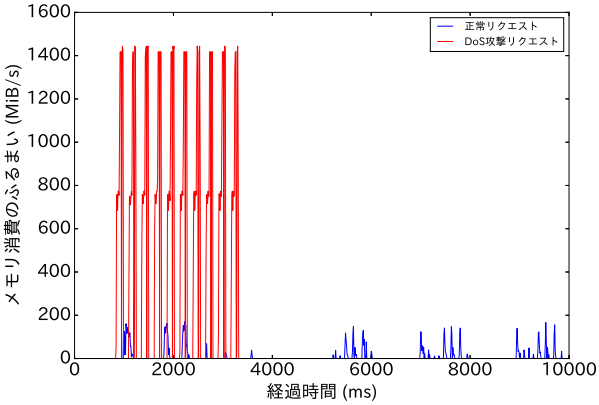


図 2　メモリ消費のふるまい（WordPress）
Fig. 2　Memory consumption behavior (WordPress).

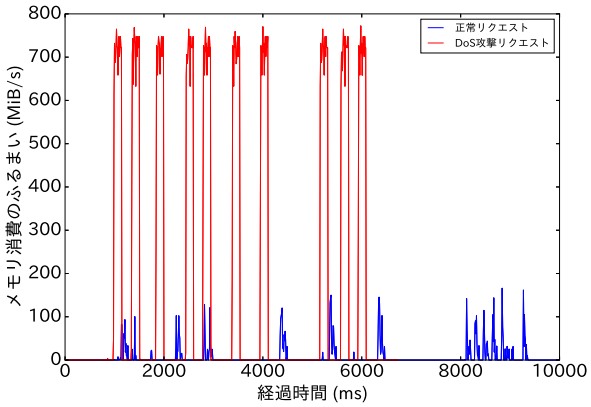


図 3　メモリ消費のふるまい（MediaWiki）
Fig. 3　Memory consumption behavior (MediaWiki).

メモリ消費のふるまいを用いることで，この問題を解決することができる．メモリの使用量に加えて，メモリ消費のふるまいを観察することで，急速なメモリの大量消費の兆しをとらえることができる．つまり，実際にメモリの消費が起こる前に，DoS 攻撃に対する行動が可能となる．

3.2　ふるまいに基づいた DoS 攻撃の検出とリソース制限

本論文では，3.1 節で定義したメモリ消費のふるまいに基づいて異常なメモリ消費を行うプロセスを検出し，そのリソース利用を制限する手法を提案する．これにより本論文が問題とする，単一のリクエストで急速にリソースを消費する DoS 攻撃（図 1 における分類 B）による，正常なリクエスト処理能力の低下を防止することができる．

提案手法は，防御対象の Web アプリケーションへのリクエストを処理するプロセスのメモリ消費を監視し，正常なリクエストを処理する場合と異なるメモリ消費が起きる兆候を検出する．この検出には，プロセスのメモリ消費のふるまいを利用する．実際に使用した大きさではなく，消費する速度に着目することで，実際に大量のメモリ消費が

発生する前に，その可能性を検出することができる．この検出は，防御対象の Web アプリケーションにおける DoS 攻撃でない，正常なリクエストについて，あらかじめメモリ消費のふるまいを計測しておき，そのふるまいと比較することで行う．検出されたプロセスには，リソースの利用制限が課され，DoS 攻撃によるシステムリソースの浪費を抑制する．

この提案手法には，1) リクエスト規模が小さな攻撃にも対処できる，2) 個別の攻撃に関する知識が不要である．3) 正常なリクエストを誤って DoS 攻撃であると検出した際にも，処理の継続が可能である，という 3 つの利点がある．

提案手法では，先行手法のようにアクセスパターンに基づく予測ではなく，個々のリクエスト処理についてのメモリ消費のふるまいに基づいて攻撃を検出する．そのため，本論文が対象とするような，少量のリクエスト規模で問題を引き起こす種類の DoS 攻撃に対処することが可能である．

提案手法には，防御対処の Web アプリケーションのメモリ消費のふるまいに関する知識が必要である．しかしながら，個別の攻撃に関する知識は不要である．また，ルールベースの手法が必要としている，攻撃に関する知識の継続的な更新は不要である．

提案手法や先行手法は，正常なリクエストを DoS 攻撃を引き起こすリクエストであると誤検出する可能性がある（偽陽性検出）．提案手法では，偽陽性検出の場合でも，リクエストの処理を継続することができる．先行手法では，DoS 攻撃を引き起こすリクエストであると判定した場合には，そのリクエストが遮断されるか，処理が中断される．これに対して，提案手法では，DoS 攻撃を引き起こすリクエストを検出しても，その処理はリソース制限を受けながらも継続される．そのため，応答時間が通常より大きくなりながらもリクエストは最後まで処理される．

4. 提案手法の設計と実装

提案手法の具体的な実装例として，メモリ消費のふるまいに基づいたリソース制限を行う機構を設計し，既存の OS に実装した．設計と実装の対象は Linux kernel 3.14.0 である．この章ではその設計と実装について述べる．

本機構は Web アプリケーションに対するリクエストを処理するプロセスのメモリ消費のふるまいを監視し，防御対象である Web アプリケーションのふるまいから逸脱するものを検出する．検出したプロセスは，DoS 攻撃を引き起こすリクエストを処理しているものと判断され，そのプロセスが利用可能なメモリリソースの量が制限される．また，マルチコア環境においては，その制限を受けたプロセスが実行される CPU が限定される．このリソース制限を受けた状態のプロセスを制限対象プロセスと呼ぶ．

リソース利用が制限されたプロセスは，使用していたメモリを解放する際にそのふるまいを検査され，異常なふるまいがなければリソース制限が解除される．

以下では，本機構を構成する処理と対象 OS での実装について述べる．なお本機構では，ふるまいの監視とリソース制限の対象となるプロセスは Web アプリケーションのリクエストを処理するプロセスに限定する．システムで動作するその他のプロセスはこの対象とならない．

4.1 OS カーネルへの実装

提案手法の実現手段としては，ユーザレベルで動作するプログラムとして実装する，OS カーネルに機能を追加するといった選択肢がある．提案手法では，2 つの理由から，その実装先は OS カーネルを前提とする．

理由の 1 つは，DoS 攻撃を可能な限り早く検出し，対策を講じるためである．一定周期でプロセスごとのメモリ消費を監視し，必要に応じてプロセス単位にリソース制限を課すことは，ユーザレベルで動作するプログラムでも可能である．しかしながらそのような定周期の監視では，問題となる急速なメモリ消費の兆候の検出が遅延する可能性がある．この検出が遅延すると DoS 攻撃への対応が遅延し，問題である．この問題に対して提案手法では，その検出の遅延を最小限に抑えるためにページフォルト処理をサンプリングして，プロセスごとのメモリ消費のふるまいを計測し，異常なメモリ消費の検出を行う．このページフォルト処理のサンプリングは．OS カーネルを変更することによって実現可能である．

2 つ目の理由は，提案手法が DoS 攻撃から受ける悪影響を可能な限り防ぐためである．本論文が問題とする DoS 攻撃が発生すると，対象のシステムでは物理メモリ資源が不足する．メモリ資源が不足すると，OS カーネルはプロセスをストレージ上のスワップ領域に書き出して物理メモリを確保しようとするため，ユーザプロセスの実行が遅延される可能性がある．この実行遅延に提案手法を実装したプログラムが影響を受けると，提案手法がその効果を十分に発揮できない可能性がある．このような事態を防ぐためにも，提案手法は OS カーネルへの機能追加として実装することが望ましい．

4.2 メモリ消費のふるまいの算出

提案手法では，メモリ消費のふるまいとして，プロセスごとの物理メモリの消費速度に着目する．メモリの消費速度は，1 秒あたりの物理メモリの使用増加量とする．メモリ消費が減少傾向にあるときは，負の値を取る．図 2，図 3 に示したように，これらのふるまいは，正常なリクエスト処理時と DoS 攻撃を引き起こすリクエストの処理時とで，明らかに異なっている．Web アプリケーションへのリクエストを処理するプロセスについて，このメモリ消費のふるまいを監視することで異常なメモリ消費の兆しを検出する．

このふるまいの計測は，プロセスごとのメモリページの割当て処理のタイミングで行う．メモリページの割当ては頻繁に起こるため，すべてのメモリページの割当てタイミングではなく，一定の回数ごとに行う．この回数をMINFAULT_SAMPLE_RATE と定義する．計測は，計測点間での物理メモリ使用サイズの差分と計測時間の差分を用いて行う．この計測のために，OS のプロセス管理情報に物理メモリ使用量と，それを記録した内部時刻を記録する．このプロセス管理情報に記録された過去の情報と現在の状況との差分を取ることで，メモリ消費のふるまいを算出する．

この実現のため，実装対象 OS のページフォールト処理にメモリ消費のふるまいを計測する処理を追加した．またプロセス管理情報を記録する task_struct 構造体に，直近 1 回分の計測時刻と物理メモリ使用量のための領域を追加した．ふるまいの算出に用いる物理メモリ使用サイズは，プロセス管理情報から取得できる Resident Set Size（RSS）を用いた．また OS カーネルの内部時刻として Linux kernel の関数である ktime_get_ns() 経由で取得可能なナノ秒単位のモノトニック時刻を用いた．

4.3　メモリ消費のふるまいに基づいたリソース制限

提案手法は，メモリ消費のふるまいの算出時に急激なメモリ消費を行うプロセスを検出した場合，そのプロセスは DoS 攻撃を引き起こすリクエストを処理していると判断し，そのプロセスにリソース制限を課す．この検出は，算出したふるまいと基準値を比較して行う．あるプロセスについて，算出したメモリ消費のふるまいが基準値よりも大きければ，そのプロセスはリソース利用が制限される．この基準値を LIMIT_THRESHOLD と定義する．

制限を受けた複数のプロセス群は，それらのプロセスが利用できる物理メモリの総量を一定以下に制限される．この制限値を MEMORY_LIMIT とする．後述するリソース制限の解除に利用するため，制限対象となったプロセスの情報はプロセス管理情報とは独立したデータ構造に記録される．このデータ構造には，対象となったプロセスの情報や，制限を実施したときのメモリ消費のふるまいや，制限実施時の物理メモリの消費量を記録する．

これを実現するために，実装対象 OS のリソース管理機構である cgroups [14] を利用した．cgroups は任意のプロセス群に対して，メモリ，CPU 割当てなどに関するリソース管理を行う機構である．リソース制限を課す管理単位を作成し，それに制限対象となるプロセスを登録することで，プロセス群のリソース利用量の計算やリソース制限を適用することができる．この cgroups の管理単位を作成し，ふるまいが異常であるプロセス群を登録することで，リソース制限を実現する．

この cgroups の管理単位への登録は，制限対象のプロセスを検出したタイミングでは実施されない．前述したとおり，リソース制限対象の検出はページフォルト処理時に行われる．この処理は高頻度で発生することや，割込みコンテキストで実行されるため，検出時点での cgroups の管理単位への登録処理は不適切である．制限対象のプロセスが検出されたときには，カーネルスレッドを作成し処理を遅延させる．リソース制限対象の cgroups の管理単位への登録はそのカーネルスレッドの処理として行われる．この検出から実際の登録の間は，制限対象のプロセスによるリソースの浪費を防ぐために，そのプロセスの実行状態を TASK_UNINTERRUPTIBLE に設定し，プロセスを休止する．作成されたカーネルスレッドで cgroups の管理単位への登録が完了した後は，プロセスの状態は TASK_RUNNING に変更され，提案手法によるリソース制限下で動作を継続する．

4.4　リソース制限の解除（メモリ解放の監視）

提案手法は，制限対象となったプロセスのメモリ解放を監視し，リソース消費のふるまいが正常であると判断すると，そのプロセスに対するリソース制限を解除する．この判断はリソース制限が実施された時点と，メモリ解放処理時点でのメモリ消費のふるまいを比較して行う．もし，メモリ解放処理時に，その時点でのメモリ消費のふるまいが，制限実施時のふるまいを下回っていた場合は，そのプロセスに対するリソース制限を解除する．この判断には 4.3 節で述べた，リソース制限対象のプロセスを管理するデータ構造に記録されているメモリ消費のふるまいを利用する．

これを実現するために，OS カーネルの munmap システムコールおよび mremap システムコール処理に，制限対象のプロセスについて前述した判断を行う処理を追加した．この追加処理では，まず，メモリ解放のためのシステムコールを呼び出したプロセスがリソース制限の対象であるかを調べ，そうでない場合には，本来のシステムコール処理に復帰する．制限対象のプロセスが呼び出した場合は，その時点でのメモリ消費のふるまいを算出し，制限を解除するか，続行するかの判断が行われる．リソース制限を解除されるプロセスは，登録されていた制限用の cgroups の管理単位への登録を解除され，他の制限を受けていないプロセスと同様にリソースを利用することが可能になる．

5.　評価実験

提案手法の効果を検証するために実験を行った．実験の目的は，提案手法により，本論文で対象とする DoS 攻撃によるシステムの性能低下を防止できることを示すことである．そのために，これまで報告されている Web アプリケーションに関する脆弱性を利用した DoS 攻撃を再現し，提案手法の有無による，クライアントからみたリクエスト応答時間を比較する．

表 1　実験環境の諸元
Table 1　Specification of experimental environments.

	アプリケーションサーバ	データベースサーバ	負荷発生器
CPU	Intel Core i7-2600 (4 core)	Intel Core i7-2600 (4 core)	仮想 CPU (2 core)
RAM	2.0 GiB	2.0 GiB	2.0 GiB
Swap	4.0 GiB	4.0 GiB	5.0 GiB
OS	Linux kernel3.14.0	Linux kernel 3.10.0	Linux kernel 3.10.0

また，提案手法のオーバヘッド検証する実験も行った．提案手法を実現するためには，OS カーネルへ処理を追加する．この追加処理が Web アプリケーションの処理性能を悪化させる可能性がある．そこで，提案手法の有無によるリクエスト処理性能を比較した．

提案手法はパラメータ LIMIT_THRESHOLD によってそのふるまいが変化する．最適な値を設定するためには，事前の見積もりが必要であるが，それが難しいケースや，状況の変化に合わせて最適な値が変化する可能性もある．そこで，LIMIT_THRESHOLD と提案手法の効果の変化についても実験により検証した．

5.1　実験環境

本論文が対象とする Web アプリケーションの動作環境として，アプリケーションサーバ（AP サーバ）とデータベースサーバ（DB サーバ）を構築した．それぞれの諸元を**表 1** に示す．また，Web アプリケーションに負荷を発生させる負荷発生器 A，負荷発生器 B を構築した．AP サーバ，DB サーバ，2 つの負荷発生器のホストサーバは単一のレイヤ 2 スイッチにより接続されている（**図 4**）．

負荷発生器は負荷発生ソフトウェアを用いて Web アプリケーションへリクエストを送信し，AP サーバに負荷を発生させる計算機である．この計算機は仮想マシンとして実装されている．負荷発生ソフトウェアとして，Apache JMeter 3.0 [15] を採用した．負荷発生器 A と負荷発生器 B の構成は同一である．

AP サーバはクライアントからのリクエストを受け付け，処理をするためのサーバである．この処理は http リクエストの受信とレスポンスの生成を担当する http サーバプロセスと，アプリケーションの実際の処理を担当するランタイムプロセスによって提供される．ユーザから送信したリクエストは，http サーバプロセスがまず受け取り，リクエストに応じて，ランタイムプロセスに処理を要求する．ランタイムプロセスは単一のマスタープロセスと複数のスレーブプロセスから構成されており，マスタープロセスが http サーバからのリクエストをスレーブプロセスに委譲して処理を行う．本実験の構成では，スレーブプロセスは 60 プロセス動作する．本実験では，http サーバとして nginx 1.8.1 [16] を採用した．ランタイムソフトウェアには，実験対象の Web アプリケーションが依存する PHP 5.3.8 [17] に含まれる FastCGI Process Manager を採用した．

図 4　実験環境の概要
Fig. 4　Overview of experimental environments.

DB サーバは，Web アプリケーションが利用するデータベース管理システムが動作するサーバである．データベース管理システムとして，MariaDB 15.1 [18] を採用した．

5.2　実験 1：提案手法の有効性

実運用されている Web アプリケーションソフトウェアの過去のバージョンに存在したソフトウェア脆弱性を攻撃例として取り上げ，提案手法の評価を行った．取り上げたのは 2.1 節で述べた，WordPress，MediaWiki に関する脆弱性である [4], [5]．それぞれの実験で，WordPress 3.9.1. MediaWiki 1.24 を使用した．

それぞれの Web アプリケーションに不適切なリクエストを送信すると，Web アプリケーションの処理を行うプロセスにより，メモリや CPU 時間が大量に消費される．Web アプリケーションの実行ランタイムでは利用可能なメモリ量の上限が定められており，問題のあるリクエストはこの上限までメモリを消費して停止する．本実験ではこの上限は 54 MiB に設定した．この値は WordPress と MediaWiki が動作するのに必要なメモリ量を計測し，その結果に基づいて設定した．そのため，上限が適切に設定されていれば，1 つの AP サーバプロセスによるメモリ消費は一定以下に抑制することができる．しかしながら，問題のあるリクエストを並行して処理する場合は，同時にメモリを浪費するプロセスの数が増え，システム全体のメモリ不足を招く．このメモリ不足により，正当なリクエストの処理が阻害される．

このような DoS 攻撃に対する提案手法の有効性を評価するために，負荷生成器 A から Web アプリケーションに正常なリクエストを送信し，その一方で負荷生成器 B から DoS 攻撃を引き起こすリクエストを送信する実験を行った．負荷生成器 B からのリクエストは DoS 攻撃を引き起こし，負荷生成器 A から送信される正常なリクエストへの

表 2　実験条件
Table 2　Experimental conditions.

	負荷生成器 A		負荷生成器 B		提案手法
実験対象	WordPress	MediaWiki	WordPress	MediaWiki	WordPress/MediaWiki
条件 A	正常リクエスト群 A-1	正常リクエスト群 A-2	正常リクエスト群 B-1	正常リクエスト群 B-2	無効
条件 B	正常リクエスト群 A-1	正常リクエスト群 A-2	DoS 攻撃リクエスト 1	DoS 攻撃リクエスト 2	無効
条件 C	正常リクエスト群 A-1	正常リクエスト群 A-2	DoS 攻撃リクエスト 1	DoS 攻撃リクエスト 2	有効

応答時間に影響を与える．AP サーバでの提案手法の有無により，この正常なリクエストへの影響は変化する．この正常なリクエストに与える影響の変化を観察することで，提案手法の DoS 攻撃防御能力を評価することができる．

5.2.1　送信リクエスト

本実験では，負荷生成器 A，B それぞれから AP サーバに対して，さまざまなリクエストを送信する．本節では，そのリクエストとして，正常リクエスト群 A-1，正常リクエスト群 A-2，正常リクエスト群 B-1，正常リクエスト群 B-2，DoS 攻撃リクエスト 1，DoS 攻撃リクエスト 2，の 6 種類のリクエストを定義する．

正常リクエスト群 A-1，A-2 は，負荷生成器 A が送信する，Web アプリケーションに対する通常のリクエストの集合である．正常リクエスト群 A-1 は WordPress を，正常リクエスト群 A-2 は MediaWiki を対象とした実験で用いるリクエストである．それぞれのリクエスト群は，以下の操作を要求する 9 種類のリクエストである．

- テキストで構成された記事の取得
  （1 KiB，10 KiB，100 KiB）
- 画像が添付された記事の取得
  （画像サイズ 1 KiB，10 KiB，100 KiB）
- テキストで構成された記事の投稿
  （1 KiB，10 KiB，100 KiB）

このリクエストで送受信する記事の内容や，画像については，正常リクエスト群 A-1，A-2 で同一とする．

DoS 攻撃リクエスト 1, 2 は負荷生成器 B が送信する，AP サーバで DoS 攻撃を引き起こすリクエストである．DoS 攻撃リクエスト 1 は WordPress を，DoS 攻撃リクエスト 2 は MediaWiki を対象にした実験で用いるリクエストである．2.1 節で述べたとおり，WordPress には XML-RPC 経由で DoS 攻撃を引き起こす脆弱性がある．DoS 攻撃リクエスト 1 は，この脆弱性を引き起こす記事の取得リクエストを XML-RPC 経由で送信する．MediaWiki については，不正なベクタ画像を記事に添付することで DoS 攻撃を引き起こす脆弱性がある．DoS 攻撃リクエスト 2 は，この DoS 攻撃を引き起こすベクタ画像を送信する．

正常リクエスト群 B-1，B-2 は，負荷生成器 B が送信する，Web アプリケーションに対する通常のリクエストの集合である．本実験では，DoS 攻撃リクエスト 1，2 が正常リクエスト群 A-1，A-2 のリクエスト応答時間に与える時間を評価する．その比較の基準とするために，DoS 攻撃が発生しない状況でのリクエスト応答時間の情報が必要である．基準となる状況では，DoS 攻撃が発生すること以外は，DoS 攻撃リクエスト 1，2 が送信される状況と類似する必要がある．そのために，負荷生成器 B から DoS 攻撃リクエスト 1，2 と類似した，しかしながら DoS 攻撃を引き起こさないリクエストを送信する．この類似した内容のリクエストを正常リクエスト群 B-1，B-2 とする．

正常リクエスト群 B-1 は WordPress を，正常リクエスト群 B-2 は MediaWiki を対象として実験で用いるリクエストである．正常リクエスト群 B-1 は，DoS 攻撃リクエスト 1 と同様に記事の取得リクエストを XML-RPC 経由で送信する．ただしこのリクエストは DoS 攻撃を引き起こさず，AP サーバはリクエスト元に指定された記事を送信し，リクエストは成功する．正常リクエスト群 B-1 には，取得する記事のデータ量が異なる 3 種類のリクエストが含まれる．データ量の種類は，1 KiB，10 KiB，100 KiB である．正常リクエスト群 B-2 は，DoS 攻撃リクエスト 2 と同様に WordPress の記事に画像を添付するリクエストである．ただしこの送信する画像は DoS 攻撃を引き起こさず，AP サーバは添付された画像を保存し，リクエストは成功する．

5.2.2　実験条件

本実験では，負荷生成器 A，B が送信するリクエストの種類，AP サーバにおける提案手法の有無を変数とした，3 つの実験条件 A，B，C がある．本項では，その実験条件について述べる．**表 2** は実験条件ごとの変数の組み合わせをまとめたものである．

実験条件 A

実験条件 A は，DoS 攻撃が発生しない実験条件である．この実験条件は本実験が問題とする AP サーバに対する DoS 攻撃による，リクエスト応答性能の変化を観測するための基準状態をつくる実験条件である．実験条件 A では，負荷生成器 A は，WordPress を対象にした実験について正常リクエスト群 A-1 を送信する．また，MediaWiki の実験について正常リクエスト群 A-2 を送信する．負荷生成器 A では，複数のスレッドでリクエストを AP サーバに送信する．送信するスレッドの並列数は 1，5，10，15，20，25，30 と変化させ，それぞれの並列数でリクエスト応答時間を計測する．それぞれの送信スレッドは正常リクエスト群

A-1 から無作為に 1 つ送信し，Web アプリケーションからのレスポンスを待つ．レスポンス受信完了後は，すぐに次のリクエストを選択し，送信に移る．

実験条件 A では，負荷生成器 B は，WordPress を対象にした実験について，正常リクエスト群 B-1 を送信する．MediaWiki の実験については正常リクエスト群 B-2 を送信する．リクエストを送信するスレッドの並列数は 30 である．負荷生成器 A と異なり，負荷生成器 B ではリクエストを送信するスレッドの数は変化させない．それぞれのスレッドは正常リクエスト群 B を無作為に 1 つ送信し，Web アプリケーションからのレスポンスを待つ．レスポンスを受信完了後は，他のスレッドの終了を同期して次のリクエスト送信に移る．負荷生成器 A と異なり，スレッドの並列数が固定であること，リクエスト送信スレッドが同期する理由は，実験条件 B で述べる DoS 攻撃の形態に近づけるためである．

実験条件 A では，AP サーバでの提案手法は無効状態である．AP サーバでは実験対象の Web アプリケーションが動作し，従来のメモリ使用量にのみ基づいたメモリ管理が行われる．

実験条件 B

実験条件 B は，DoS 攻撃が発生し，既存のメモリ使用量にのみ基づいたリソース制限を用いて DoS 攻撃に対応する状態をつくる実験条件である．実験条件 B では，負荷生成器 A は正常リクエスト A-1 または A-2 を送信する．負荷生成器 A に関しては，実験条件 A と同一である．

実験条件 B では，負荷生成器 B は，WordPress の実験については DoS 攻撃リクエスト 1 を送信する．また，MediaWiki の実験については DoS 攻撃リクエスト 2 を送信する．リクエストを送信するスレッドの数は 30 である．DoS 攻撃を引き起こすリクエストの量を一定にするため，このリクエスト送信スレッド数は変化しない．それぞれのリクエスト送信スレッドは DoS 攻撃リクエスト 1 または 2 を送信し，Web アプリケーションからのレスポンスを待つ．レスポンスを受信完了後は，他のリクエスト送信スレッドの終了を同期して，次のリクエスト送信に移る．この同期処理によって，それぞれのスレッドは同時に攻撃対象に問題のあるリクエスト送信を開始する．この同期したリクエスト送信により，攻撃対象の AP サーバにおいて問題のあるリクエストの処理が同時多発的に発生し，攻撃対象のシステムをより不安定な状態にすることができる．

実験条件 B では，AP サーバの挙動は実験条件 A と同様である．提案手法は無効であり，従来のメモリ使用量にのみ基づいたメモリ管理が行われる．ただし，この実験条件 B では実験条件 A とは異なり，DoS 攻撃リクエスト 1，2 によって DoS 攻撃が発生する．DoS 攻撃リクエストは AP サーバにおいて大量のメモリ消費を引き起こす．この大量のメモリ消費を引き起こすリクエスト処理は，AP サーバの OS やアプリケーションランタイムが設定したメモリ利用量の上限に達した時点で中止される．実験条件 B は問題となる DoS 攻撃に対して，メモリ使用量の制限のみで対応を試みる．

実験条件 C

実験条件 C は，DoS 攻撃が発生した状況で，実験状態 B の状態に加えて，提案手法を用いて DoS 攻撃に対応する状態をつくる実験条件である．実験条件 C では，負荷生成器 A は正常リクエスト A-1 または A-2 を送信する．負荷生成器 A に関しては，実験条件 A，B と同様である．

実験状態 C では，負荷生成器は DoS 攻撃リクエスト 1 または 2 を送信する．負荷発生器 B に関しては，実験状態 B と同様である．

実験状態 C では，AP サーバでは提案手法を有効とする．つまり，実験状態 B での AP サーバの状況に提案手法を適用する．この状態では，AP サーバはメモリ使用量制限に基づいた DoS 攻撃の防御に加えて，提案手法である，メモリ消費のふるまいに基づいた DoS 攻撃の防御を実施する．

5.2.3　評価指標

本実験では，評価指標として負荷生成器 A で計測したサービス応答時間を用いる．ここでのサービス応答時間は，Web アプリケーションにリクエストを送信してから，そのリクエストに対する応答ヘッダ，コンテンツの受信が終了するまでの経過時間（単位：ミリ秒）とする．DoS 攻撃の目的は，Web アプリケーションの利用者のサービス利用を阻害することであり，クライアント側から計測したサービス応答時間を利用することで，提案手法の有効性を評価することができる．

なお，実験条件 A においては，負荷生成器 B は負荷生成器 A と同様に正常なリクエストを送信するが，そのリクエストに対する応答時間は評価指標としては用いない．条件 A における負荷生成器 B は，条件 B，C における DoS 攻撃リクエストと類似した正常リクエストを送信し，条件 A において DoS 攻撃時と同程度のリクエスト送信を模倣することを目的としている．この模倣するリクエストは，検証対象の Web アプリケーションに対する一般的な負荷として見なすことができる．本実験では，この一般的な負荷を背景にした状態（実験条件 A）を基準とし，その状況下での DoS 攻撃防御手法の効果について検証する．

5.2.4　パラメータ設定

4 章で述べたように，提案手法にはその動作を変化させる 3 つのパラメータがある．本実験では**表 3** のように設定した．MINFAULT_SAMPLE_RATE は 4.2 節で述べたメモリ消費のふるまい算出のための，メモリページ割当て処理のサンプリング周期である．このパラメータにより，メモリ消費のふるまいの粒度が変化する．本実験では AP サーバの物理メモリ（2048 MiB）の約 0.2%にあたる 4000 KiB のページ割当てごとにサンプリングを行う．実

表 3　パラメータ設定
Table 3　Experimental parameters.

パラメータ名	設定値	パラメータの意味
LIMIT_THRESHOLD	170 MiB/s	リソース制限を実施する閾値
MINFAULT_SAMPLE_RATE	1000	メモリ消費の監視に用いるマイナーページフォールトのサンプリング周期
MEMORY_LIMIT	256 MiB	リソース制限対象のプロセス群が利用可能なメモリ量

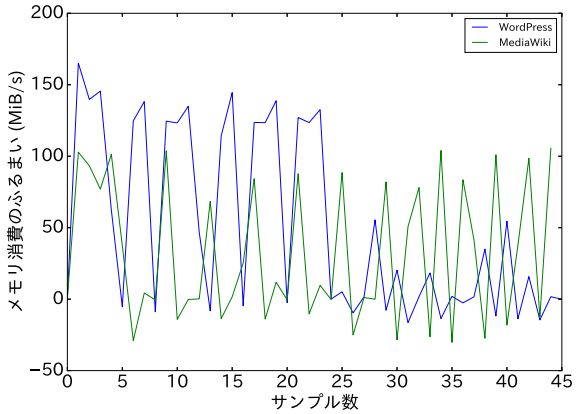


図 5　正常リクエスト時のメモリ消費のふるまいの変化
Fig. 5　Memory consumption behavior for processing legimate requests.

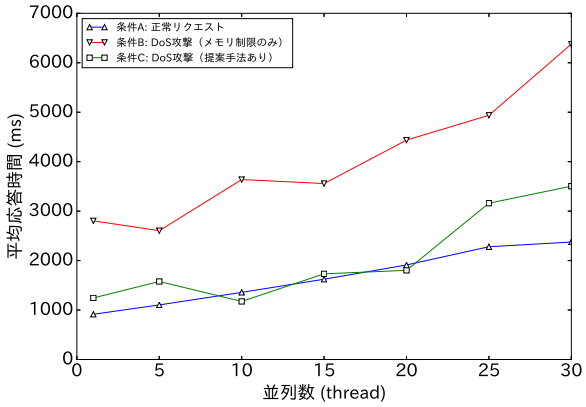


図 6　リクエスト応答時間（WordPress）
Fig. 6　Request response time (WordPress).

装対象 OS のページサイズは 4 KiB であるので，本実験では MINFAULT_SAMPLE_RATE を 1000 とした．MEMORY_LIMIT は DoS 攻撃を引き起こすリクエストを処理しているとしてリソース利用を制限されたプロセス群が利用できるメモリの最大サイズである．これは，AP サーバの物理メモリの 12.5%である，256 MiB とした．

3 つのパラメータのうち LIMIT_THRESHOLD は，リソース制限の決定に直接関与する重要なパラメータである．防衛対象の Web アプリケーションによって最適な値が異なるため，実験対象の Web アプリケーションのメモリ消費のふるまいを計測する実験を行い，それに基づいて設定値を決定した．

実験ではそれぞれの Web アプリケーションに対して，実運用時に送信頻度が高いと推定されるリクエストを送信し，そのリクエストを処理する際の AP サーバプロセスのメモリ消費のふるまいを計測した．計測には，提案手法によるメモリ消費のふるまいの計測と同じ機構を用いた．WordPress，MediaWiki は共にコンテンツマネジメントシステムであり，記事の取得，投稿を行うリクエストが，全体のうち大きな部分を占めていると考えられる．よって，それぞれの Web アプリケーションについて，記事の取得を行うリクエスト，記事の投稿を行うリクエストを送信し，リクエスト処理に関するメモリ消費のふるまいを計測した．

図 5 はそれぞれの Web アプリケーションのメモリ消費のふるまいの計測結果を表したものである．青線は WordPress，緑線は MediaWiki のメモリ消費のふるまいである．提案手法では，監視対象プロセスがマイナーページフォルトを一定回数起こすごとに，そのメモリ消費のふるまいを算出する．このグラフはその計測結果を記録し，左から計測順に並べたものである．縦軸は計測したメモリ消費のふるまい，横軸は計測した順番である．この結果は監視対象のプロセスのページフォルトが一定回数起きるタイミングで計測したものであり，一定の時間間隔で記録されたものではない．したがって横軸は，実際の経過時間には等間隔では対応していない．

分析の結果，WordPress についてメモリ消費のふるまいの最大は 165.1 MiB/s，最小は −16.4 MiB/s，平均は 50.6 MiB/s であった．MediaWiki についてメモリ消費のふるまいの最大は 105.6 MiB/s，最小は −30.2 MiB/s，平均は 31.8 MiB/s であった．ここでの負の値は，使用するメモリ量の減少傾向を示している．

この結果から，2 つのアプリケーションについて通常のアクセスの範囲では最大で約 165 MiB/s のメモリ消費のふるまいを示すことが分かった．本実験では，観測された最大値（165 MiB/s）にマージンを持たせ，170 MiB/s を LIMIT_THRESHOLD に設定した．メモリ消費のふるまいの計測時にこの値を超えるメモリ消費のふるまいを示したプロセスは提案手法によるリソース制限の対象となる．

5.2.5　実験結果：WordPress

WordPress の脆弱性についての実験結果を図 6 に示す．このグラフは，それぞれの条件下において負荷生成器 A で計測したリクエスト応答時間の平均値を表す．横軸は負荷生成器 A でリクエストを並列処理するスレッドの数を表している．縦軸は横軸が示すそれぞれの並列数における，リクエスト応答時間を平均した数値である．青線，赤線，緑

線はそれぞれ，条件 A，B，C における AP サーバでのリクエスト処理性能である．

条件 A（青線）と条件 B（赤線）の結果を比較すると，正常リクエスト群 A-1 についての平均リクエスト応答時間が増加している．条件 B は条件 A と同規模の正常なリクエストを処理した際のリクエスト処理性能であり，この 2 つの条件におけるリクエスト応答時間の増加を DoS 攻撃による Web アプリケーションの処理低下とみなすことができる．この結果から，条件 B における DoS 攻撃によって正常リクエスト群 A-1 の処理が妨害されていることが分かる．条件 A でのリクエスト応答時間に対して，条件 B での反応時間は最大で 3.1 倍，最小で 2.1 倍に増加した．

条件 A（青線）と条件 C（緑線）の結果を比較でも，正常リクエスト群 A-1 についての平均応答時間が増加している．しかしながら，条件 B（赤線）に比べて条件 C（緑線）ではリクエスト応答時間の増加が少ない．この結果から，条件 B で用いたメモリ制限のみによる DoS 攻撃の防御に提案手法を加えることで，DoS 攻撃による影響が緩和できていることが分かる．条件 A でのリクエスト応答時間に対して，条件 C での応答時間の増加は最大でも 1.5 倍，最小で 0.9 倍であった．また，条件 B でのリクエスト応答時間と比較すると，条件 C での応答時間は最小で 0.2 倍に抑制することができた．これは条件 B に対する条件 C の 4.3 倍の性能改善にあたる．

また図 6 のグラフからは，並列数 10，20 において，条件 C（緑線）のリクエスト応答時間が条件 A（青線）を下回っていることが分かる．これは，条件 C では提案手法により，負荷発生器 B からのリクエストの処理が抑制され，負荷生成器 A からのリクエスト処理により多くのリソースが割り当てられるためと考えられる．条件 A では負荷発生器 B が正常リクエスト群 B-1 を AP サーバに送信する．条件 C では負荷発生器 B は正常リクエスト群 B-1 に代わり同規模の DoS 攻撃リクエスト 1 を AP サーバに送信する．この DoS 攻撃リクエスト 1 は提案手法によって検出され，その処理に割けるリソースは厳しく制限される．その結果として，条件 C では負荷生成器 A によるリクエストの処理により多くのリソースが割り当てられ，リクエスト処理性能の向上につながったと考えられる．

平均リクエスト応答時間による評価に加えて，条件 A，B，C それぞれにおいて計測したリクエスト応答時間の分布を分析した．**図 8** は，それぞれの条件ごとに色分けしてリクエスト処理時間のヒストグラムを表したものである．青棒，赤棒，緑棒はそれぞれ条件 A，B，C に対応する．横軸はリクエスト応答時間によるヒストグラムの区間を表す．それぞの区間は 500 ms ごとの幅である．縦軸は，区間ごとにあてはまるリクエストの数である．つまり左端の棒がリクエスト応答時間が 0 から 499 ms であったリクエスト数，その 1 つ右の棒がリクエスト応答時間が 500 から 999 ms であったリクエスト数，そのさらに 1 つ右の棒が応答時間が 1,000 から 1,499 ms であったリクエストの数を表現している．この図を用いることで，図 6 に示した平均リクエスト反応時間のデータには現れない，それぞれの条件ごとのリクエスト応答時間の傾向を観察することができる．

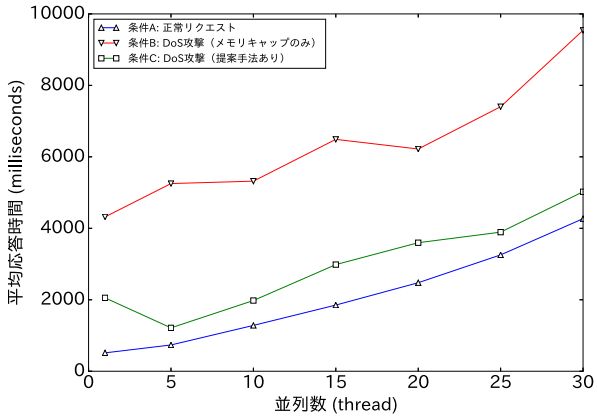


図 7　リクエスト応答時間（MediaWiki）
Fig. 7　Request response time (MediaWiki).

図 8 において条件 A（青棒）と条件 B（赤棒）を比較すると，DoS 攻撃の状況下にある条件 B では，条件 A（青棒）に対して，リクエスト応答時間が全体的に大きい傾向にあることが分かる．条件 B（赤棒）と条件 C（緑棒）の分布を比較すると，条件 B とは反対に，条件 A に対してリクエスト応答時間が全体的に小さい傾向にあることが分かる．この結果からも，条件 C では提案手法により DoS 攻撃による影響が緩和できていることが分かる．

5.2.6　実験結果：MediaWiki

MediaWiki の脆弱性についての負荷生成器 A におけるリクエスト応答時間の計測結果を**図 7** に示す．グラフの意味については，図 6 に示した，WordPress の脆弱性についての実験と同様である．

条件 A（青線）と条件 B（赤線）の結果を比較すると，正常リクエスト群 A-2 についての平均リクエスト応答時間が増加している．この結果から，条件 B における DoS 攻撃によって正常リクエスト群 A-2 の処理が妨害されていることが分かる．条件 A でのリクエスト応答時間に対して，条件 B での応答時間は最大で 8.4 倍，最小で 2.2 倍に増加した．

条件 A（青線）と条件 C（緑線）の結果の比較でも，正常リクエスト群 A-2 についての平均リクエスト応答時間が増加している．しかしながら，条件 B（赤線）に比べて条件 C（緑線）ではリクエスト応答時間の増加が少ない．条件 A でのリクエスト応答時間に対して，条件 C でのリクエスト応答時間は最大でも 1.5 倍，最小で 0.9 倍であった．また，条件 B でのリクエスト応答時間と比較すると，条件 C での応答時間は最小で 0.3 倍に抑制することができた．

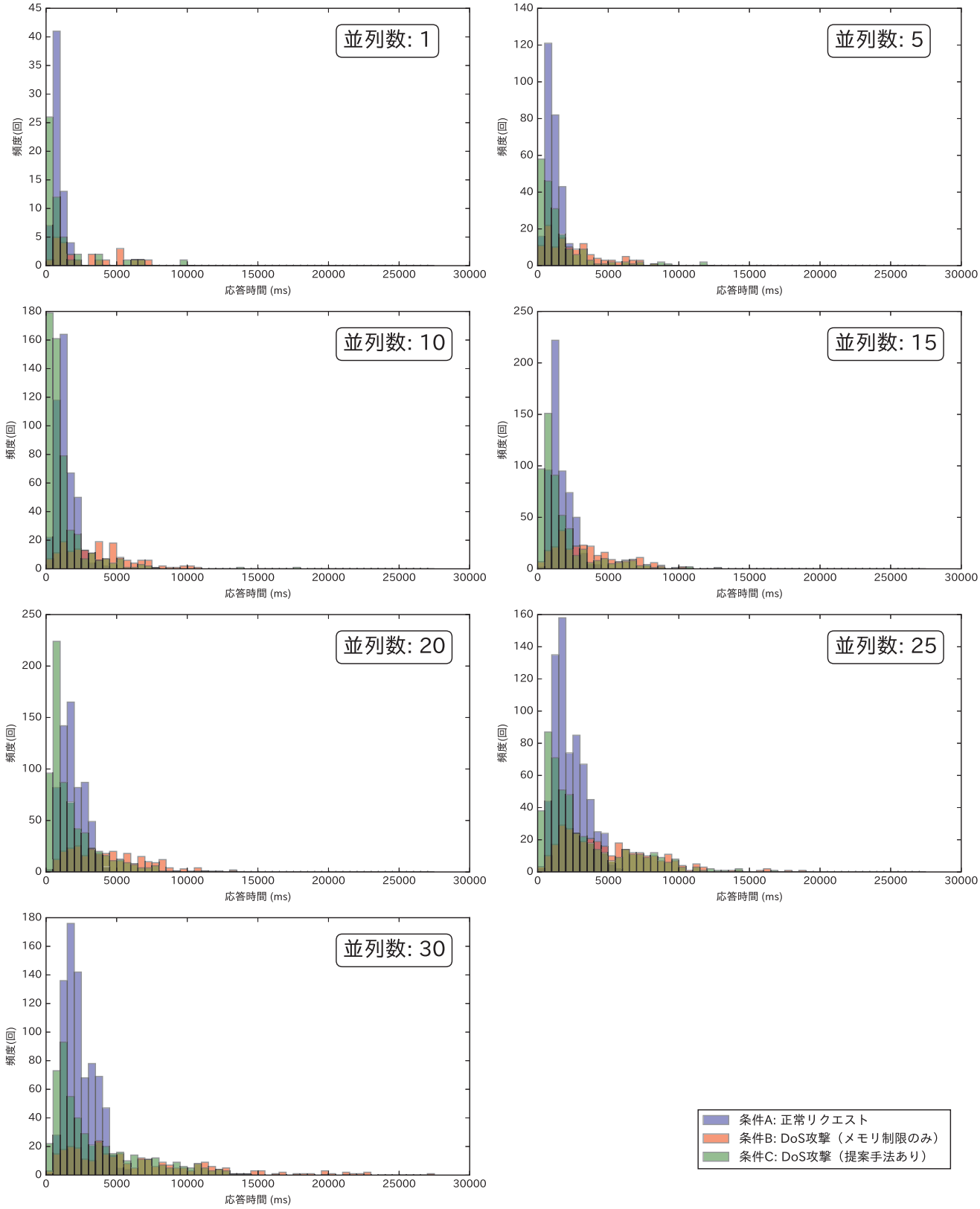


図 8　リクエスト応答時間の分布（WordPress）

Fig. 8　Histogram of request response time (WordPress).

これは条件 B に対する条件 C の 3.1 倍の性能改善にあたる．この結果から，5.2.5 項で述べた WordPress の事例と同様に，条件 B で用いたメモリ制限のみによる DoS 攻撃の防御に提案手法を加えることで，DoS 攻撃による Web アプリケーションのサービス性能低下が緩和できていることが分かる．

条件 A，B，C で計測したリクエスト応答時間の分布を，**図 9** に示す．グラフの意味については，図 8 に示した

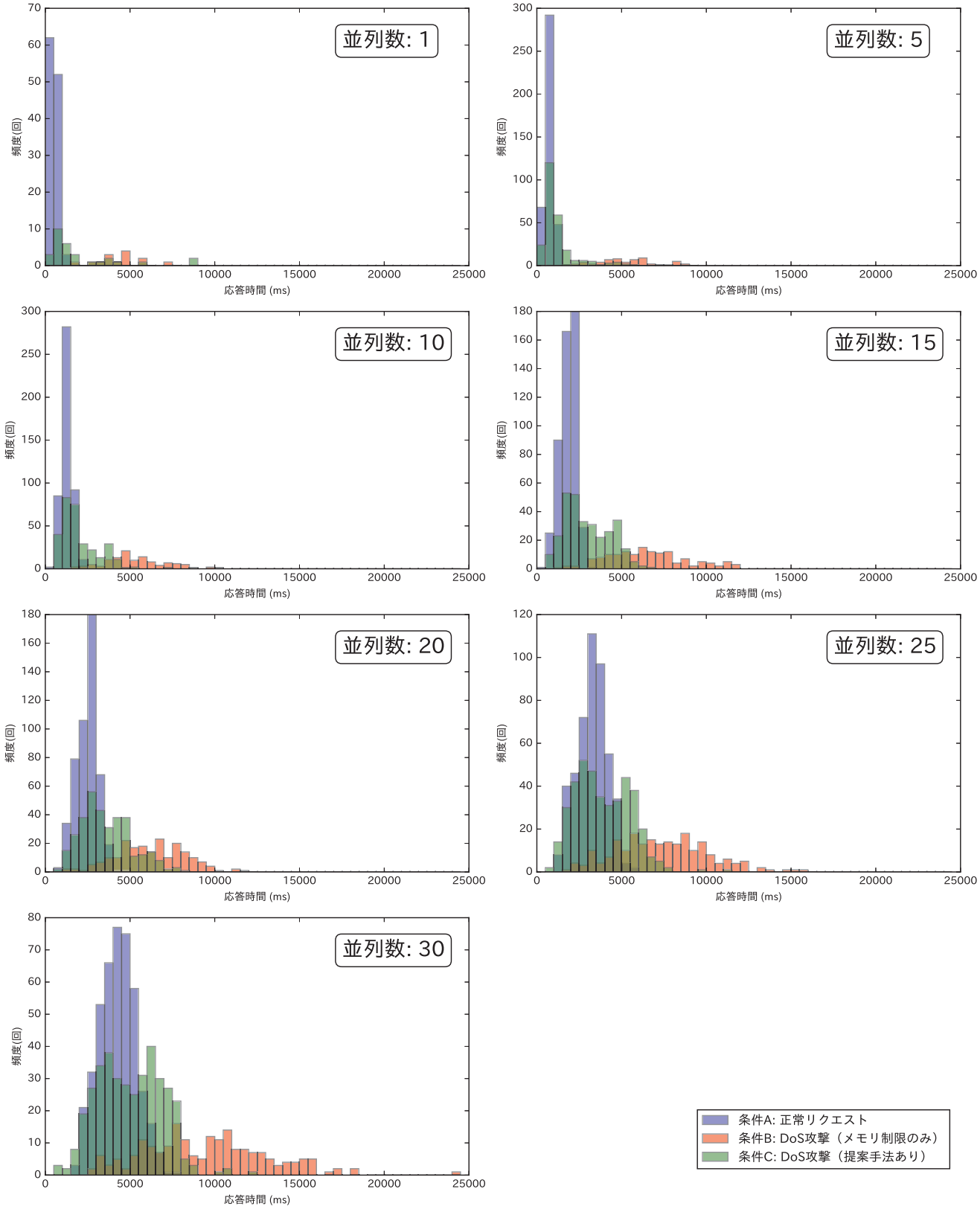


図 9　リクエスト応答時間の分布（MediaWiki）
Fig. 9　Histogram of request response time (MediaWiki).

WordPress の脆弱性についての実験と同様である．

条件 A（青棒）と条件 B（赤棒）の分布を比較すると，DoS 攻撃の状況下では応答時間の分布がより大きな方へ拡がっていることが分かる．条件 B（赤棒）と条件 C（緑棒）の分布を比較すると，その偏りが抑えられ，より条件 A に近い分布になっていることが分かる．この結果からも，平均応答時間による評価と同様に，提案手法により DoS 攻撃による影響が緩和できていることが分かる．

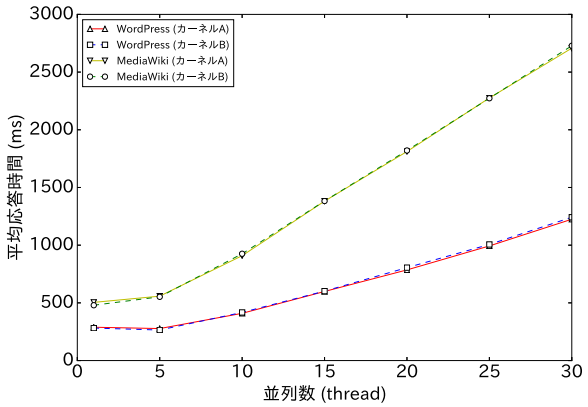


図 10　提案手法の有無による正常リクエスト処理時間の変化
Fig. 10　Overhead of the proposed method.

5.3　実験 2：提案手法によるオーバヘッド

提案手法を実現するためには，OS カーネルに対して処理の追加が必要である．この追加した処理が，Web アプリケーションに対する通常のリクエスト処理を遅延させる可能性がある．そこで本節では，提案手法が Web アプリケーションにおける通常のリクエスト処理に対して発生させるオーバヘッドを評価する．

提案手法のために追加した処理の中でも，プロセスごとのメモリ消費のふるまい計測する処理はリクエスト処理性能に与える影響が大きい．この処理はプロセスへメモリ割当てを行うページフォルト処理に処理を追加しており，多数の呼出しが起こる．そのため，この追加した処理がシステム全体の実行性能を悪化させる可能性は大きい．

評価のために 2 つの OS カーネルを構成した．OS カーネル A は提案手法を有効にした OS カーネルである．OS カーネル B は OS カーネル A から，前述したシステム性能に影響を与える可能性のある処理を取り除いたカーネルである．つまり，OS カーネル B は，OS カーネル A から，ページフォルトを利用したメモリ消費のふるまいの計測処理を取り除いたものである．

評価は，構成した 2 つの OS カーネルでそれぞれ Web アプリケーション環境を構築し，負荷生成器 A から正常なリクエストを送信したときのリクエスト応答時間を計測することで行った．計測対象の Web アプリケーションは実験 1 で取り上げた WordPress と MediaWiki である．環境の構成は，実験 1 と同等である．送信するリクエストの内容は実験 1 における正常リクエスト群 A-1（WordPress），正常リクエスト群 A-2（MediaWiki）である．それぞれのアプリケーションについて，リクエストを送信するスレッドの数を 1，5，10，15，20，25，30 と可変させて計測を行った．計測はそれぞれの並列数条件ごとに 60 秒間実施した．

図 10 に計測結果を示す．赤線，青線はそれぞれカーネル A，B を用いたときの WordPress に対するリクエスト応答時間の平均値を表している．黄線，緑線はそれぞれカーネル A，B を用いたときの MediaWiki に対するリクエスト応答時間の平均値を表している．WordPress，MediaWiki，いずれの場合でも，リクエスト処理性能に大きな変化は見られなかった．分析の結果，WordPress を対象にした計測では，カーネル A でのリクエスト応答時間に対するカーネル B での応答時間の増加は最大でも 1.05 倍であった．MediaWiki を対象にした計測では，カーネル A でのリクエスト応答時間に対するカーネル B での応答時間の増加は最大でも 1.02 倍であった．この結果から，提案手法が DoS 攻撃がない状態での Web アプリケーションの処理性能に与える影響は，非常に小さいことが分かった．

5.4　実験 3：しきい値の変化と処理性能

LIMIT_THRESHOLD はこの提案手法の効果を左右する重要なパラメータである．このパラメータの最適値は保護対象となる Web アプリケーションごとに異なる．実験 1 では 5.2.4 項で述べたとおり，対象のアプリへ想定されるリクエストを仮定してメモリ消費のふるまいを計測し，その結果に基づいて LIMIT_THRESHOLD を設定した．

この実験のように，最適な値がいつでも維持できるとは限らない．十分な事前調査ができない場合や，ソフトウェアの改修により，最適な値が変わる可能性がある．Web アプリケーションの中には，ユーザにより機能拡張できるプラグイン機構を持つものがあり，同じ Web アプリケーションでも環境によってメモリ消費のふるまいが異なることもある．

そこで，本実験で用いた LIMIT_THRESHOLD（170 MiB/s）を基準に，そこから設定値にずれが生じると，処理性能にどのような変化を与えるのかを検証した．5.2 節で述べた条件 C を用いて，LIMIT_THRESHOLD を変化させながら，負荷生成器 A から正常リクエストを送信し，そのリクエストごとの応答時間を計測した．計測対象の Web アプリケーションは実験 1 で取り上げた WordPress である．環境の構成は，実験 1 と同等である．送信するリクエストの内容は実験 1 における正常リクエスト群 A-1 である．それぞれのアプリケーションについて，リクエストを送信するスレッドの数を 1，5，10，15，20，25，30 と可変させて計測を行った．計測はそれぞれの並列数条件ごとに 60 秒間実施した．

図 11 は，LIMIT_THRESHOLD を基準値（170 MiB/s）から下の方向にずらしたときの，リクエスト応答時間を示している．橙の実線は 5.2.5 項で述べた，WordPress に関する実験のうち，条件 B（メモリ制限のみ）でのリクエスト応答時間を示したものである．それ以外の破線は，条件 C（提案手法有効）の状態で，LIMIT_THRESHOLD を変化させたときのリクエスト応答時間を表している．

LIMIT_THRESHOLD を実際のメモリ消費のふるまいの計測に基づいた基準値（170 MiB/s）から下の方向にずらした場合には，正常リクエストを DoS 攻撃として誤検出す

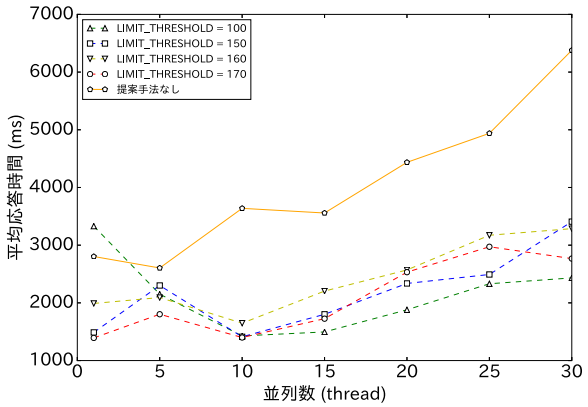


図 11　しきい値の変化とリクエスト処理性能（100 から 170）
Fig. 11　The relationship between the threshold and the performance (from 100 to 170).

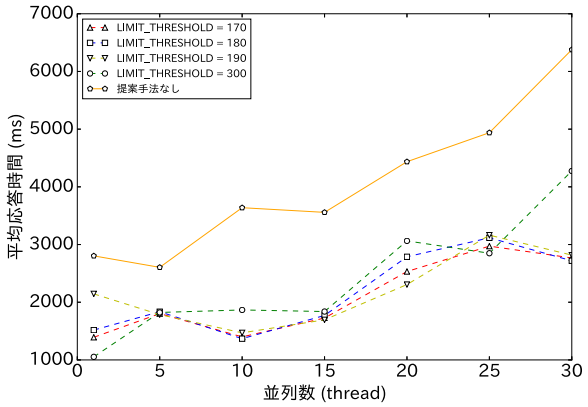


図 12　しきい値の変化とリクエスト処理性能（170 から 300）
Fig. 12　The relationship between the threshold and the performance (from 170 to 300).

る可能性がある（偽陽性検出）．この偽陽性検出が発生した場合は，正常リクエストの処理が妨げられ，リクエスト応答性能が低下し，メモリ制限のみでの DoS 攻撃防御を行った場合よりも性能が悪化する可能性がある．実際に図 11 では，LIMIT_THRESHOLD が 100 のとき，並列数 1 で大きな性能低下が発生している．これは偽陽性検出により，正常リクエストのためのメモリ利用が制限されたために，リクエスト応答時間が遅延したものと考えられる．しかしながら，その他の並列数条件，LIMIT_THRESHOLD の条件では，リクエスト応答時間は LIMIT_THRESHOLD が基準値（170 MiB/s）であった場合（赤破線）と似た傾向を示している．また，DoS 攻撃に対する防御性能はメモリ制限のみである場合（橙実線）よりも優れていた．これは 4.4 節で述べた，リソース制限の解除が適切に働き，偽陽性検出の場合でも速やかにリソース制限が解除されているためと考えられる．

図 12 は同様に LIMIT_THRESHOLD を基準値（170 MiB/s）から増加させた場合のリクエスト処理性能を示している．図 11 と同様に橙の実線は，5.2.5 項で述べた，WordPress に関する実験のうち，条件 B（メモリ制限のみ）でのリクエスト応答時間を示したものである．

LIMIT_THRESHOLD を計測に基づいた基準値（170 MiB/s）から上の方向にずらした場合には，LIMIT_THRESHOLD が DoS 攻撃リクエストを処理する際のメモリ消費のふるまいを超過し，正しく検出できない可能性がある（偽陰性検出）．この場合，提案手法の効果はなくなり，メモリ制限による DoS 攻撃対策のみの場合の防御性能と同等になると考えられる．しかしながら，図 12 の結果では，実際のアプリケーションのメモリ消費のふるまい計測に基づいた基準値（170 MiB/s）からの差が最大の LIMIT_THRESHOLD が 300 の場合でも，メモリ利用量制限のみによる DoS 攻撃対策を実施した場合（橙実線）よりも高い効果を発揮している．

この理由は，正常リクエストを処理した場合のメモリ消費のふるまいと，DoS 攻撃リクエストを処理したときのメモリ消費のふるまいに大きな差があるためだと考えられる．図 2 で示したように，WordPress に関して，正常なリクエスト処理時のメモリ消費のふるまいと，DoS 攻撃リクエストを処理した際のメモリ消費のふるまいには最大で 8 倍以上の差がある．そのため，正常リクエストのメモリ消費のふるまいに対して LIMIT_THRESHOLD がある程度大きい場合でも，偽陰性検出を起こすことなく，DoS 攻撃が検出可能である．

以上に述べたように，LIMIT_THRESHOLD が事前計測に基づいた最適値から外れた場合でも，提案手法は効果を維持することが分かった．LIMIT_THRESHOLD が最適値よりも小さい場合は，正当なリクエスト処理を DoS 攻撃と検出し，その正当な処理を妨げる危険性があるが（偽陽性検出），実験の結果，一部の例外を除いて LIMIT_THRESHOLD が最適値を下回っても，それが最適値であるときと近しい防御性能を発揮することが分かった．これは提案手法に実装されているリソース制限の解除が適切に動作しているためである．反対に LIMIT_THRESHOLD が最適値よりも大きい場合でも，DoS 攻撃を見逃すことなく（偽陰性検出），それが最適値であるときと同等の DoS 攻撃からの防御性能を発揮することも分かった．これは正当リクエストを処理する際のメモリ消費のふるまいと，DoS 攻撃リクエストを処理する際のメモリ消費のふるまいに大きな差があるためである．このようなパラメータに柔軟性がある特徴は，提案手法を用いたシステム管理に有用である．

5.5　提案手法の制約

本節では，ここまでの実験結果では明らかになっていない提案手法の制約について述べる．

提案手法を用いた環境では，リソース制限されたメモリ空間不足により，制限状態に置かれたプロセスが強制終了されることがある．提案手法では，DoS 攻撃の可能性があると判定された AP サーバのプロセスがリソース制限状態

に置かれる．リソース制限状態に置かれたプロセス群は，パラメータ MEMORY_LIMIT で許可された物理メモリを使用することができる．その制限状態に置かれたプロセス群が，MEMORY_LIMIT を超えてメモリが必要になった場合には AP サーバのスワップが使用される．もし，このスワップも使い切ってしまった場合には，制限状態に置かれたプロセス群はメモリの割り当てに失敗したり，OS カーネルに強制終了される可能性がある．本論文でのシステム構成では，AP サーバのメモリ制限設定によって，提案手法によるリソース制限を課せられた後に，アプリケーションの実行を中断するようになっている．そのため前述の問題は発生せずに，実験結果にこの制約は影響していない．

前述したとおり，提案手法ではリソース制限に置かれたプロセスのためにスワップ領域が使われる．このスワップ領域の処理によって，システム全体の処理性能が低下する可能性がある．この制約については，すでに実験結果に影響をしている可能性もあるが，これまでの実験ではその影響がどの程度であるか明らかになっていない．

5.6　評価実験のまとめ

5.2 節で述べた実験 1 では，提案手法による，Web アプリケーションに対する DoS 攻撃の緩和効果について検証した．WordPress，MediaWiki に関して実際に報告された脆弱性 [4], [5] を対象に，実験を行った．結果として，提案手法を用いることで，DoS 攻撃により低下したリクエスト処理性能を，最大で 4.3 倍に改善することができた．

5.3 節で述べた実験 2 では，提案手法によるオーバヘッドを評価した．実験では，提案手法が有効な OS カーネルと，提案手法の主要な処理を無効化した OS カーネルを用いて，Web アプリケーションに対する正常なリクエストの処理能力を比較した．結果として，提案手法の実装による Web アプリケーションのクライアントに対するリクエスト応答時間の増加は，最大でも 1.05 倍であった．

5.4 節では，提案手法のふるまいを左右するパラメータ LIMIT_THRESHOLD の設定柔軟性について評価した．LIMIT_THRESHOLD は，DoS 攻撃を検出するしきい値である，提案手法の重要なパラメータである．この LIMIT_THRESHOLD が防御対象の Web アプリケーションにとって理想的な値から外れると，偽陽性検出による正当なリクエストの処理遅延や，偽陰性検出による DoS 攻撃の取りこぼしが発生する可能性がある．そこで事前のメモリ消費ふるまいの計測を実施することによって算出した値を基準として（170 MiB/s），その基準からパラメータ LIMIT_THRESHOLD を上下に変化させたときの，DoS 攻撃の緩和効果を評価した．結果としては，1 つの例外を除いて，LIMIT_THRESHOLD を変化させたときに，DoS 攻撃防御性能の大幅な低下は確認できなかった．

6.　まとめ

本論文では，Web アプリケーションの脆弱性を利用した DoS 攻撃の検出とその攻撃によるリクエスト処理性能低下の防止手法として，リクエストを処理するプロセスのメモリ消費のふるまいに着目したリソース制限を提案した．Web アプリケーションに対する DoS 攻撃の防止手段には，WAF や OS の機能を用いた方法や，先行研究で提案された方法があるが，いずれも本論文が対象とする Web アプリケーションの脆弱性を利用した DoS 攻撃を防ぐためには不十分であった．提案手法では，プロセスのリソース消費のふるまいに着目することで，既存手法では対処が難しかった，DoS 攻撃への対応が可能になる．

提案手法の実装例として，メモリ消費のふるまいに基づいたリソース制限機構を設計，実装した．その評価実験を行ったところ，提案手法を用いることで，DoS 攻撃の状況下でのリクエスト処理性能の低下を低オーバヘッドで抑制することができた．今後の課題としては，より広い種類の DoS 攻撃防御への適用，他の手法と提案手法を組み合わせた手法の検討などがあげられる．

謝辞　本研究は JSPS 科研費 14J01818 の助成を受けたものです．

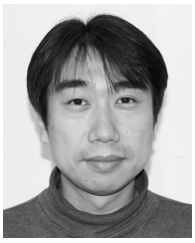
参考文献

[1] Web Application Security Consortium: Web Security Glossary (2004), available from ⟨http://www.webappsec.org/projects/glossary/⟩.
[2] M, H., E, R. and IAB: Internet Denial-of-Service Considerations (RFC 4732) (2006), available from ⟨http://www.rfc-editor.org/info/rfc4732⟩.
[3] The MITRE Corporation: CVE-2012-0789, available from ⟨http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0789⟩.
[4] The MITRE Corporation: CVE-2014-5266, available from ⟨http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5266⟩.
[5] The MITRE Corporation: CVE-2015-2942, available from ⟨http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2942⟩.
[6] Xu, J. and Lee, W.: Sustaining availability of Web services under distributed denial of service attacks, *IEEE Trans. Comput.*, Vol.52, No.2, pp.195–208 (2003).
[7] Srivatsa, M., Iyengar, A., Yin, J. and Liu, L.: Mitigating Application-level Denial of Service Attacks on Web Servers: A Client-transparent Approach, *ACM Trans. Web*, Vol.2, No.3, pp.15:1–15:49 (2008).
[8] Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A. and Knightly, E.: DDoS-shield: DDoS-resilient Scheduling to Counter Application Layer Attacks, *IEEE/ACM Trans. Networking*, Vol.17, No.1, pp.26–39 (2009).
[9] Barna, C., Shtern, M., Smit, M., Tzerpos, V. and Litoiu, M.: Model-based Adaptive DoS Attack Mitigation, *Proc. 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp.119–128 (2012).
[10] Matt Mullenweg, et al.: WordPress.ORG, available from

⟨https://wordpress.org/⟩.
[11] Winer, D.: XML-RPC Specification (1999), available from ⟨http://xmlrpc.scripting.com/spec.html⟩.
[12] Roger L., C.: XML Risks and Mitigations (2013), available from ⟨https://www.mitre.org/publications/technical-papers/xml-risks-and-mitigations⟩.
[13] Wikimedia Foundation, Inc: MediaWiki, available from ⟨https://www.mediawiki.org/wiki⟩.
[14] Menage, P.: CGROUPS, available from ⟨https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt⟩.
[15] Apache Software Foundation: Apache JMeter, available from ⟨http://jmeter.apache.org/⟩.
[16] Igor Sysoev: nginx, available from ⟨http://nginx.org/⟩.
[17] The PHP Group: php, available from ⟨ http://php.net/⟩.
[18] MariaDB Foundation: MariaDB, available from ⟨https://mariadb.org/⟩.

中川 岳　（正会員）

2017 年筑波大学より博士（工学）．2017 年 4 月より富士通株式会社に勤務．オペレーティングシステム，システムソフトウェアに関する研究開発に興味を持つ．



追川 修一　（正会員）

1996 年慶應義塾大学より博士 (工学)．2004 年筑波大学大学院システム情報工学研究科助教授．2016 年筑波大学システム情報系教授．2017 年 4 月より株式会社フィックスターズに勤務．システムソフトウェアに関する研究開発に従事．IEEE 会員．