情報処理学会論文誌　コンピューティングシステム　Vol.5　No.5　138–151 (Oct. 2012)

サーバ・チューニング記述のためのスクリプティング環境

相川 拓也[1,a)]　杉木 章義[2]　加藤 和彦[2]

受付日 2012年3月28日, 採録日 2012年6月15日

概要：サーバのチューニングは，性能を左右する重要なタスクである一方で，管理者にとって困難をともなう．パラメータの最適値は多くの場合，1 回のチューニングでは求まらず，さまざまな試行錯誤を行う必要がある．また，1 回の試行においても，サーバの設定を変更するだけではなく，複数台からなるクライアントの設定を変更し，起動するなど，煩雑な手続きが必要である．本研究では，このチューニングにおける試行錯誤の過程を効率化するスクリプティング環境を提案する．本提案では，サーバやクライアントなどチューニングに関連する要素をすべて分散オブジェクト化し，統一的な環境で高水準に試行の過程を記述可能とする．また，自動チューニングアルゴリズムのライブラリ化を行うことで，利便性の向上を図る．実験では，SPECweb2005 ベンチマーク下の Apache ウェブサーバと Hadoop を対象として実験を行い，本環境を利用してチューニングができることを確認した．

キーワード：分散システム，サーバ・チューニング，サーバ管理，スクリプティング環境

A Scripting Environment for Iterative Tuning Process of Server Applications

Takuya Aikawa[1,a)]　Akiyoshi Sugiki[2]　Kazuhiko Kato[2]

Received: March 28, 2012, Accepted: June 15, 2012

Abstract: Although parameter tuning is critical for server performance, that tuning process is error-prone and time consuming. An administrator must repeat many iterations to find an optimal configuration and even at each step non-trivial tasks, including proper configuration of a server and launching benchmark clients, are required. In this paper, we present a scripting environment for efficiently describing such tuning process. We offer distributed objects as ways to manipulate a server and benchmark clients. By this way, tuning process can be described by scripting them in an integrated environment. We also provide automatic tuning as a library for further efficiency. In the experiments, we confirmed that tuning of an Apache web server under SPECweb2005 benchmark and a Hadoop cluster were successfully possible in our tuning environment.

Keywords: distributed systems, server tuning, server management, scripting environment

1. はじめに

サーバアプリケーションのパラメータを調整するチューニングは，管理者の行う作業の中で重要なタスクであることが知られている [1], [2], [3]．同じハードウェア構成であっても，チューニングを行うかどうかでサーバ性能が大きく変化する．

サーバアプリケーションの自動チューニングに関する研究は以前より行われており，ActiveHarmony [4] や，Smart Hill-climbing [5]，進化的アルゴリズム [6] を利用した手法などが提案されている．これらの手法では，自動チューニングのアルゴリズムがライブラリとして提供されており，管理者がそれぞれのサーバアプリケーションに依存する部分のコードを記述することで，さまざまなアプリケーショ

[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, Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 筑波大学システム情報系情報工学域
Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[a)] aikawa@osss.cs.tsukuba.ac.jp

ンに適用可能となっている．

本研究では，これらの研究とは別に，サーバチューニングの全体のプロセスに焦点を当てる．サーバのチューニング過程では，単に，自動チューニングアルゴリズムを 1 度適用すればよいだけでなく，その前後の段階でさまざまな試行を行う必要がある．一般的なチューニング過程は以下のとおりである．

(1) **予備実験：**まず，実験環境上で予備実験を行い，サーバのハードウェア性能に応じて設定可能なパラメータ値の範囲を事前に確かめておく必要がある．これは，ウェブサーバの同時接続数などのパラメータは，CPU 速度やメモリ容量などのハードウェアの物理特性から直接的に値を求めることができない場合が多く，その導出には経験や試行を必要とするからである．

(2) **パラメータのスクリーニング：**続いて，チューニング対象を多数の設定可能な項目から少数の効果的なパラメータに絞り込む，スクリーニングと呼ばれる作業を行う必要がある．これは，近年のサーバアプリケーションでは数十から数百の設定可能なパラメータが存在し，そのすべてを自動チューニングアルゴリズムの探索空間に含めたのでは，現実的な時間でチューニングを終えることができないためである．

(3) **最適パラメータ値の探索（自動チューニング）：**これらの作業の後，初めて自動チューニングアルゴリズムを適用し，パラメータ値の探索を行うことができる．

(4) **チューニング結果の検証：**最後に，チューニングの結果が求める性能を満たしているか検証する．

また，上記の 4 つ過程における 1 回のチューニングサイクル中にも，一般に以下に示すような煩雑な手続きを必要とする．

- **サーバ設定の変更：**まずサーバアプリケーションの設定ファイルを書き換えた後，再起動させるなどして変更を反映させる．
- **クライアントエミュレータの起動：**また同時に，複数台の計算機からなるクライアントエミュレータを適切に設定し，いっせいに起動して管理する．
- **実行結果の取得と解析：**クライアントからサーバ性能の測定結果を取得し，解析結果をもとに次に探索すべきパラメータを決定する．

上記の作業は，これまで多くの場合，Perl，Python や Ruby などの汎用的なスクリプト言語で記述されていた．特に，複数台の計算機の設定を書き換えたり，結果を取得したりする部分に関しては，ssh コマンドと組み合わせるなど，アドホックな手法で記述されていた．これらの操作を行うスクリプトの記述は煩雑であるため，管理者がさまざまなアイデアに基づき試行錯誤を行う際の妨げとなっている．また，前述の自動チューニングライブラリを利用する場合においても，サーバアプリケーション固有の部分については，汎用スクリプト言語に API が提供されており，同じく煩雑な記述を必要としていた．

そこで本研究では，このチューニングの試行錯誤の過程を容易にするためのスクリプティング環境を提案する．本研究ではサーバのチューニングという領域に特化し，必要な機能を提供する．本研究のアプローチは以下のとおりである．まず，サーバアプリケーションやクライアントエミュレータなどのチューニングに必要な構成要素を分散オブジェクト化し，パラメータ値の書き換えやアプリケーションの起動や終了などの操作をメソッド呼び出しで可能とする．また，個別のアプリケーションに対して，利用者が毎回手動でオブジェクトを作成することは困難だと考えられるため，オブジェクトの自動生成機構を提供する．さらに，それらのオブジェクトを操作するための機能をスクリプティング環境へ統合し，高水準なチューニング記述を可能とする．最後に，自動チューニングアルゴリズムのライブラリ化を行う．以上，これらを組み合わせることで，さまざまなチューニングスクリプトを効率的に記述できる環境の構築を目指す．

実験では，提案環境を利用して各種アプリケーションのチューニング記述が可能なことを示すため，SPECweb2005 ベンチマークを使用した Apache ウェブサーバと Hadoop の 2 つに適用した．また，一般的なチューニング過程の記述に対応できることを示すため，予備実験，実験計画法を利用したパラメータのスクリーニング，チューニングライブラリを利用した自動チューニング，およびチューニング結果の検証の 4 種類の実験を行った．実験結果より，提案環境を利用して各種アプリケーションのチューニングを効率的に記述できることを確かめた．

2. 関連研究

本研究は，サーバアプリケーションの自動チューニングに関する研究，および設定作業を容易にする研究の双方に関係がある．

2.1 自動チューニングアルゴリズムに関する研究

計算機を利用したチューニングは数値計算分野などで古くから行われており，サーバアプリケーション分野でも 2000 年代からさまざまな研究がある．

Chung ら [4] は，彼らが継続的に開発している自動チューニングシステム Active Harmony をクラスタ型のウェブサーバのチューニングに適用している．当時の Active Harmony は滑降シンプレックス法 [7] をもとにしており，TPC-W ベンチマークに適用した結果を報告している．また，Xi ら [5] は．大域探索と局所探索を組み合わせた Smart Hill-climbing アルゴリズムを提案している．本アルゴリズムは，ラテン方格を利用して探索空間から探索点をサンプリングしたのち，二次近似曲線を利用した局所探索する操作を繰り

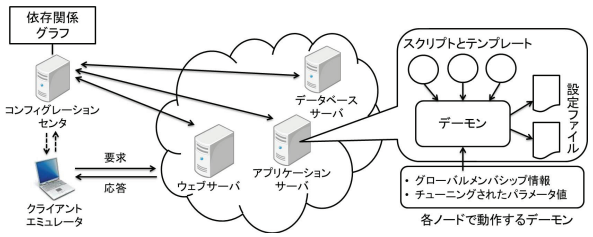

図 1　Zheng らのシステム構成の概要
Fig. 1　Overview of Zheng et al.'s system.

返し，効率的にパラメータ空間の探索を行う．Saboori ら [6] は，進化的アルゴリズムの一種である Covariance Matrix Adaptation（CMA）アルゴリズムを，多階層のウェブサーバシステムのチューニングに適用した結果を示している．

本研究は，自動チューニングアルゴリズムそのものの提案でなく，これらの研究とは異なる．また，これらの研究を実際のサーバに適用する場合にも，チューニング過程の各段階において，設定ファイルの書き換えや，複数のクライアントエミュレータの操作など，煩雑な手続きの記述が必要であり，本研究ではその問題を解決の対象とする．

2.2　設定作業の負担軽減に関する研究

サーバアプリケーションの設定作業は難易度が高く，またミスも混入しやすいことが知られている [1], [2], [3]．そのため，設定作業の負担を軽減する研究もさまざまに行われている．たとえば，SmartFrog [8] は，サーバシステム管理のためのフレームワークである．宣言的なドメイン固有言語の記述をもとに，システム構成に合わせて設定ファイルの変更を行う．このほかにもさまざまな研究があり，その成果は文献 [1] の中で詳しくサーベイされている．

本研究はこれらとは異なり，パラメータのチューニングのような，より動的なサーバ環境の変更に特化して設計されている．提案環境を利用することで，そのような動的な設定ファイルの書き換え操作を容易に行うことができる．

2.3　両者を統合したシステム

Zheng らは，両者の統合を目指したサーバアプリケーション用の自動チューニングシステムを提案している [9]．**図 1** に Zheng らのシステムの概要を示す．このシステムは，チューニング対象として多階層サーバ群を想定しており，チューニング作業全体を管理するコンフィグレーションセンタと，性能測定のためのクライアントエミュレータを提供する．各サーバ上ではデーモンプログラムが動作しており，メンバシップ情報の取得や，各設定ファイルの更新を行う．

このシステムでは，設定ファイルごとに，対象パラメータの位置や形式を表すテンプレートと，テンプレートをもとにパラメータ値を動的に変更するためのスクリプトを用意し，それらをもとに設定ファイルを自動生成する．

```
# File worker.properties.tmp

# List workers by name
worker.list=[WORKER_LIST] loadbalancer

# Describe properties of each worker
[WORKER_PROPERTIES]

# Describe properties of load balancer
worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=[WORKERS]
```

図 2　Zheng らの研究における設定ファイルのテンプレート例
Fig. 2　Template for worker.properties file.

```
# Global section
$TEMPLATE = “worker.properties.tmp”
$COMMENT_CHAR = “#”

# Program section
ITER($TIER2.COUNT, $index) {
  $name1 = $name1 + “tomcat” + $index + “,” ;
};
[WORKER_LIST] = “worker.list=” + $name1;

ITER($TIER2.COUNT, $index) {
  $port = “worker.tomcat” + $index + “port=8009\n”;
  ...
  $worker_props = $worker_props + $port + ...;
};
[WORKER_PROPERTIES] = $worker_props;

CHOP($name1);
[WORKERS] = $name1;
```

図 3　Zheng らの研究における設定ファイル生成スクリプト例
Fig. 3　Script for worker.properties.

図 2 に設定ファイルの生成に利用するテンプレートの例を示す．図 2 は，Tomcat の `worker.properties` という設定ファイルのテンプレートである．テンプレート中の `[WORKER_LIST]`，`[WORKER_PROPERTIES]`，`[WORKERS]` は空白部分となっている．これら部分に入る文字列を，**図 3** に示すような Perl 言語をもとにしたスクリプト言語の記述から生成し，設定ファイルを完成させる．また，パラメータの依存関係のグラフ生成と滑降シンプレックス法を組み合わせた自動チューニング機能も提供されている．

Zheng らのシステムを利用することにより，典型的な多階層サーバモデルのチューニング過程の大部分が自動化され，効率的なチューニングが可能になると考えられる．しかし，特定の自動チューニング方式に固定されたシステム

であるため，チューニング前の予備実験のような試行錯誤や，チューニング結果の検証といった作業に柔軟に利用できず，それらの作業はシステムの外で行う必要がある．それに対して，本研究では，チューニングに必要な構成要素を分散オブジェクト化し，チューニングアルゴリズムをライブラリとして提供している．これにより，さまざまなスクリプト記述を効率的に行えることに加え，サーバ構成やワークロード，チューニングアルゴリズムなどを自由に組み合わせ，柔軟なチューニングを可能としている．

3. サーバ・チューニング記述のためのスクリプティング環境

本研究では，サーバアプリケーションのチューニングの試行過程を効率的に記述するためのスクリプティング環境を提案する．本研究のアプローチは次の 3 つである．

- **チューニング要素の分散オブジェクト化：**サーバアプリケーションやクライアントエミュレータなどチューニングに必要な要素を，言語オブジェクトを介して操作するようにする．これらの要素をすべて言語オブジェクトとして扱うことで，統一的なインタフェースを与え，またチューニングの試行錯誤の過程を通して，これらのオブジェクトを再利用可能とする．たとえば，Apache や Hadoop などの広く使われるサーバアプリケーションを 1 度オブジェクト化してしまえば，チューニング記述のたびに同じオブジェクトを繰り返し使用することができる．さらに，これらの言語オブジェクトを分散オブジェクトとすることで，多数のクライアントを一斉起動する場合や，サーバが複数台の計算機で構成される場合などにおいても，それらを分散透過に操作できるようになる．
- **高水準なスクリプティング環境への統合：**上記のオブジェクトを高水準な記述をもとに操作するため，Scala 言語の対話環境を利用する．チューニングドメインに特化した独自のスクリプティング言語を提供するという方法も考えられるが，今回は実用性や汎用性および実装のコストを考慮し，既存の言語へ統合する方法を選択した．既存の言語のうち，Scala 言語を採用した理由は大きく 2 つある．1 つは，Scala がオブジェクト指向言語であり，チューニング要素をオブジェクト化する本研究のアプローチと親和性が高い点である．もう 1 つは，関数型言語のパラダイムを取り入れた言語であるため，オブジェクトやその集合に対する操作を，宣言的な記法と手続き的な記法を組み合わせて簡潔かつ柔軟に記述できるという点である．加えて，型推論機能や静的型付けによる型安全性も提供されている．本研究では，Scala 言語のスクリプティング環境を拡張し，サーバのメンバシップ管理機能や並列分散処理機能を提供し，よりチューニングドメインに特化した環境を提供する．
- **自動チューニングライブラリの提供：**サーバアプリケーションを分散オブジェクト化するだけではなく，自動チューニングのアルゴリズムをライブラリ化し，これらを任意に組み合わせることにより，スクリプト記述の利便性や柔軟性が向上する．さらに，これらのアルゴリズムを高階関数として記述しておけば，より再利用性が高まることが期待される．

それぞれの機能の詳細については，以降の節で順に説明する．

3.1 チューニング要素の分散オブジェクト化

本研究では，サーバアプリケーションやクライアントエミュレータなど，チューニングに必要なさまざまな構成要素を分散オブジェクト化する．本研究では，煩雑な操作をオブジェクト内部に隠蔽し，パラメータの値の取得や書き換えをメソッド呼び出しで可能にする．また，アプリケーションの起動や終了などの操作もメソッドを通じて可能にする．

3.1.1 従来手法との比較

オブジェクトを利用することでチューニング記述を効率的に行えることを示すため，3 つの操作記述を行い従来手法と比較する．具体的には，(1) パラメータ値の取得，(2) パラメータ値の更新，(3) サーバアプリケーションの再起動，というチューニング過程でよく見られる 3 つの操作記述を行い比較する．また，今回は例として，Apache とそのパラメータである `MaxClients` を対象とする．**図 4** に従来手法による操作記述例を示し，**図 5** にオブジェクトを用いた操作記述例を示す．

従来手法で (1) パラメータ値の取得を行う場合，図 4 の例では `getMaxClients()` 関数が該当する．まず，`execSSH()` 関数により SSH を利用した遠隔ログインを行い，続いて文字列検索コマンドを実行し，検索結果からパラメータ値を抽出するという 3 つのステップが必要となる．また，(2) パラメータ値の更新操作は，図 4 の例では `setMaxClients()` 関数がこれに該当し，SSH ログインの後に文字列置換コマンドを実行するという 2 段階のステップが必要となる．同様に，(3) サーバアプリケーションの再起動操作は，図 4 の例では，`restartApache()` 関数が該当し，SSH ログインの後，対応するスクリプトの実行という 2 つのステップが必要である．このように従来手法では，チューニング関連作業を行うため，遠隔ログインと文字列操作コマンドもしくはスクリプトを組み合わせた操作が必要となる．

一方，オブジェクトを利用する場合には，図 5 で示しているように，パラメータ値の取得や更新，サーバアプリケーションの再起動といった操作を，1 回のメソッド呼び出しで実現できている．これは，従来手法で必要となっていた設定ファイルの書き換えのような定型的な作業を行う

```
 1: val serverName = "MyWebServer"
 2: val configFile = "/etc/httpd/conf/httpd.conf"
 3: def execSSH(host: String, s: String) = {
 4:   val command = "ssh" + " " + host + " " + s
 5:   Runtime.getRuntime.exec(command)
 6: }
 7:
 8: /* MaxClients パラメータの取得 */
 9: def getMaxClients =  {
10:   val grep = "grep -e ^MaxClients " + configFile
11:   val src = execSSH(serverName, grep).getInputStream
12:   val pattern = """^MaxClients\s+(\d+)$""".r
13:   Source.fromInputStream(src).getLines.next match {
14:     case pattern(x) => x.toInt
15:     case _ => -1
16:   }
17: }
18: val mc = getMaxClients
19:
20: /* MaxClients パラメータの更新 */
21: def setMaxClients(mc: Int) = {
22:   val sedCond = """'1,/^MaxClients/""" +
23:     """s/^\(MaxClients \+\)[0-9]\+/\1""" +
24:     mc + "/g'"
25:   val sed = "sed -i " + sedCond + " " + configFile
26:   execSSH(serverName, sed)
27: }
28: setMaxClients(512)
29:
30: /* Apache 再起動 */
31: def restartApache =
32:   execSSH(serverName, "/etc/init.d/httpd restart")
33: restartApache
```

図 4　従来のチューニング操作記述例
Fig. 4　Example of a conventional tuning script.

```
 1: /* Apache オブジェクトの取得 */
 2: val apache = pms(0).appm.add("Apache")
 3:
 4: /* MaxClients パラメータの取得 */
 5: val mc = apache.maxClients
 6:
 7: /* MaxClients パラメータの更新 */
 8: apache.maxClients = 512
 9:
10: /* Apache の再起動 */
11: apache.restart
```

図 5　オブジェクトを用いたチューニングの操作記述例
Fig. 5　Example script with application objects.

コードを自動生成し，オブジェクト内に隠蔽しているためである．また，分散オブジェクト技術を利用することで，遠隔ログイン操作に関する記述も不要となっている．このように，オブジェクト化によってチューニングの各段階に頻出する煩雑な操作を抽象化することで，チューニング記述の効率化やスクリプトの可読性向上につながると考えられる．

3.1.2　オブジェクトの自動生成機構

チューニングを行うアプリケーションには，Apache や Hadoop などさまざまなものがあり，それぞれ手動で分散オブジェクトを作成していたのでは効率が悪い．そこで，アプリケーションオブジェクトの自動生成機構を提供する．本機構はアプリケーションの**インタフェース定義言語**（**IDL: Interface Definition Language**）の記述から，対応する分散オブジェクトを自動生成する．また，この生成過程は基本的に CORBA [10] や Java Remote Method Invocation などの分散オブジェクトの生成を参考にしたものであるが，パラメータのチューニングというドメインに特化した機能も提供する．

ドメイン固有の機能は主に設定項目の書き換えに関するものである．サーバアプリケーションでは，設定ファイルの構文に XML 形式を採用する，1 行 1 項目で記述するなど，いくつか共通性が見られる．そこで，設定ファイルの形式ごとに，あらかじめいくつかのテンプレートを用意しておき，煩雑な設定ファイルの書き換えを自動生成で簡略化する．

ただし，分散オブジェクトの自動生成といっても，個別のアプリケーションに依存する部分があるため，すべてを機械的に自動生成できるわけではない．たとえば，統計情報を取得するなどのアプリケーションごとに用意された API を通じて操作しなければならない部分は，どうしても共通化できない．このようなアプリケーションに依存するコードの記述については，オブジェクトのスケルトン生成後に手動で追記する，もしくは Yacc/Lex のように IDL 中に埋め込みコードの形式で記述する，といった方式が考えられる．本論文では実装の簡略化から前者のスケルトン方式を採用している．

図 6 に Apache の IDL 記述例を示す．まず，アプリケーションのクラス名 `Apache` を宣言している．その中で，アプリケーションの操作に関する部分（`Function` 部）と，設定ファイルの操作に関する部分（`Config` 部）で構成される．`Function` 部では，`start`, `shutdown`, `restart`, `logs` メソッドを宣言している．一方で，`Config` 部では，`maxClients`, `keepAliveTimeout`, `apcShmSize` を設定ファイル上のパラメータ項目と対応づけている．

図 7 に IDL 記述から分散オブジェクトの生成過程を示す．IDL 記述を言語オブジェクト生成器に入力すると，この IDL 記述から Apache オブジェクトのスケルトンコードを生成する．この生成の過程で，あらかじめ設定ファイルの種類ごとに用意されたテンプレートを使用する．また，出力先のプログラミング言語ごとにテンプレートを作成しておけば，さまざまな出力言語に対応できる．

```
app Apache("/etc/init.d/httpd") {
   Function {
     start{"start"}
     shutdown{"stop"}
     restart{"restart"}
     logs(path: String): List[String]
   }
   Config {
     httpdConf("/etc/…/httpd.conf") = LineConfig {
      maxClients: Int =
        value("\<IfModule prefork.c>\MaxClients\")
      keepAliveTimeout: Int =
        value("KeepAliveTimeout")
     }
     php_ini("/etc/php.ini") = LineConfig("=") {
       apcShmSize: Int =
         value("apc.shm_size").take("(\\d+)M", 1)
     }
   }
}
```

図 6　Apache の IDL 記述例
Fig. 6　IDL description for generating an apache object.

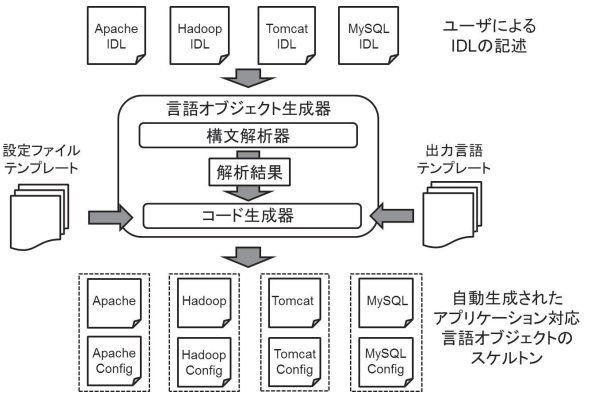

図 7　アプリケーションオブジェクトの生成過程
Fig. 7　Application object generations.

生成後のスケルトンは，出力言語が Scala の場合，通常の Scala の抽象クラスである．クラス内の中身が空白となっているメソッドを実装することで，完全な分散オブジェクトとすることができる．また，自動生成機構は，提案環境上でオブジェクトを操作するのに必要となるシステム固有の機能のコードも生成するため，スケルトンをもとに作成した分散オブジェクトはそのままシステムに組み込むことが可能である．

3.2　高水準なスクリプティング環境への統合

本論文のスクリプティング環境は Scala をベースとしており，静的型付けされた環境で，オブジェクト指向と関数型言語を融合したパラダイムでチューニングを記述することができる．また，内部の実装には本研究室で開発しているクラウド基盤ソフトウェア Kumoi [11] を使用しており，関数の並列分散実行や計算機の自動メンバシップ管理機能などを提供する．また，チューニングの際に必要となる個別計算機の負荷情報を取得する機能やモニタリング機能なども提供する．

```
 1: /* Apache ウェブサーバの設定 */
 2: val apache = pms(0).appm.add("Apache")
 3: apache.maxClients = 512
 4: apache.keepAliveTimeout = 15
 5: apache.restart
 6:
 7: /* SPECweb ベンチマークの設定 */
 8: val specweb = pms(1).appm.add("SPECwebPrime")
 9: val clients = pms.drop(2).take(8).dmap{
10:    _.appm.add("SPECwebClient")
11: }
12: specweb.testType = specweb.Banking
13: specweb.sessions = 500
14: specweb.webserver = apache
15: specweb.clients = clients
16:
17: /* SPECweb クライアントの一斉起動 */
18: clients.dforeach(_.start)
19:
20: /* SPECweb ベンチマークの実行 */
21: val result = specweb.start
```

図 8　SPECweb2005 による Apache のベンチマーク記述例
Fig. 8　Script for benchmarking apache with SPECweb2005.

3.2.1　SPECweb2005 ベンチマークの記述例

図 8 に Apache ウェブサーバに対して SPECweb2005 ベンチマークを実行するスクリプト記述例を示す．

本スクリプトでは，まず 2 行目から 5 行目で Apache ウェブサーバの設定を行っている．2 行目で，メンバシップ機能により自動的に管理されている計算機オブジェクトのリスト `pms` の先頭のマシンを Apache に割り当て，Apache オブジェクトを取得している．また `appm` は各計算機で動作しているアプリケーション管理モニタであり，`appm` の `add()` メソッドにより，引数で指定したアプリケーションのオブジェクトを取得できる．続いて，3 行目から 5 行目で Apache の `MaxClients` と `KeepAliveTimeout` のパラメータ値を変更し，Apache を再起動させることで，パラメータ値の変更を反映している．

8 行目から 15 行目では，SPECweb ベンチマークの設定を行っている．8 行目では，Apache の場合と同様に，物理マシンリストの 2 台目を割り当てる操作を行い，SPECweb ベンチマーク全体の管理を行うプライマリクライアントのオブジェクトを取得している．また，9 行目から 11 行目で

SPECweb のスレーブクライアントに対応するオブジェクトのリスト clients を取得している．これらの SPECweb クライアントは，プライマリクライアントからの指示に従い実際にワークロードを生成する．今回は，SPECweb クライアントに物理マシン 8 台を割り当てるため，pms に対し drop() 関数でリストの先頭から 2 台を取り除き，take() 関数で 8 台分のオブジェクトを取り出している．そして，これらの物理マシンオブジェクトに dmap() 関数を適用し，SPECweb クライアントオブジェクトを取得している．ここで使用した dmap() 関数は，通常の map() 関数を並列に実行する関数である．続く 12 行目から 15 行目では，SPECweb ベンチマークの設定を行っている．ワークロードの種類や同時セッション数，ベンチマークの対象となるウェブサーバやスレーブクライアントを指定している．

次に，18 行目で SPECweb クライアント 8 台をいっせいに起動させている．dforeach() は foreach() 関数の並列版であり，ここでは 8 台のクライアントの start() メソッドを並列に実行し，一斉起動を実現している．このように，リストに対する並列実行関数を利用することで，複数のクライアントエミュレータを同時に起動する操作を 1 行で簡潔に記述することができる．

最後に，21 行目でベンチマークを開始し，結果の取得を行っている．

Apache ウェブサーバとマスタクライアント，および SPECweb クライアントは，スクリプトを実行する計算機とは異なる管理者の各計算機で実行されているが，これらはすべて分散オブジェクト化されているため，分散環境を特に意識せず，操作することができる．

3.2.2　JMeter ベンチマークの記述例

また，たとえば JMeter ベンチマークを使用する場合は，図 8 のスクリプトを**図 9** のように書き換えることで対応可能である．ただし，JMeter オブジェクトは新たに作成する必要がある．

このほかにも，ロードバランサやアプリケーションサーバ，データベースサーバなどをそれぞれオブジェクト化すれば，提案環境に導入することができる．また，サーバのインスタンス数を調整する操作も，物理マシンのリスト pms から動的に計算機を取得し，各サーバに割り当てることで実現可能である．

3.3　自動チューニングライブラリ

本スクリプティング環境において，自動チューニングのアルゴリズムなどを関数としてライブラリ化しておけば，さまざまなサーバアプリケーションに適用でき，利便性が高まることが期待される．特に，高階関数を利用してアルゴリズムを記述しておけば，チューニング方式に依存しない分散オブジェクトと，特定のアプリケーションに依存しない自動チューニング関数の組合せによって柔軟にチューニングを行えることが期待される．

```
 1:  /* Apache ウェブサーバの設定 */
 2:  val apache = pms(0).appm.add("Apache")
 3:  apache.maxClients = 512
 4:  apache.keepAliveTimeout = 15
 5:  apache.restart
 6:
 7:  /* JMeter ベンチマークの設定 */
 8:  val jmeter = pms(1).appm.add("JMeter")
 9:  jmeter.server = apache
10:  jmeter.testFile = "benchmark.jmx"
11:  jmeter.logFile = "jmeter.log"
12:
13:  /* JMeter ベンチマークの実行 */
14:  val result = jmeter.start
```

図 9　JMeter による Apache のベンチマーク記述例
Fig. 9　Script for benchmarking apache with JMeter.

```
1:  /* 滑降シンプレックス法によるチューニング */
2:  val result = dhSimplex(plist){ params =>
3:      hadoop.params = params
4:      hadoop.restart
5:      val benchResult = hadoop.bench
6:      benchResult.time
7:  }
```

図 10　滑降シンプレックス法のライブラリの利用例
Fig. 10　Script for using downhill-simplex library.

現在，自動チューニングのアルゴリズムをライブラリとして提供できることを示すため，古典的で汎用性の高い滑降シンプレックス法 [7] のライブラリを提供している．このほかにも，焼きなまし法や遺伝的手法などのアルゴリズムをライブラリとして自由に追加していくことが可能である．アルゴリズムを追加する際には，チューニングの構成要素がすべて分散オブジェクト化されており，分散環境を意識せずアクセスできるため，通常の Scala 関数としてアルゴリズムを記述すればよい．また，アルゴリズム間でインタフェースを共通化しておけば，自由に入れ替え可能となる．

図 10 に本環境の滑降シンプレックス法を利用した Hadoop のチューニング記述例を示す．dhSimplex() は，滑降シンプレックス法を実装した関数で，2 つの引数をとる．第 1 引数はパラメータの初期値の組である plist で，第 2 引数は与えられたパラメータの組についての性能評価実験を行い，実験結果を評価値として返すクロージャである．今回は Hadoop の実験であるので，Hadoop のパラメータの値の組を params に設定してベンチマーク実験を

表 1　すべて手動で作成した場合とオブジェクトの自動生成機構を利用した場合とのオブジェクトのコード量比較
Table 1　Comparison of the size of auto-generated objects with those coded by hand.

	すべて手動で作成した場合の記述量	オブジェクト自動生成機構を利用して作成			
		IDL 記述量	自動生成されたコード量	手動で追加した記述量	分散オブジェクト全体のコード量
Apache	167	26	306	70	376
Besim	102	16	163	22	185
SPECweb	298	35	316	151	467
SPECwebClient	43	10	62	27	89
Hadoop	322	40	422	175	597

行い，実行にかかった時間を評価値として返すクロージャを指定している．以上のようにすると，`dhSimplex()` は最終的に評価関数を極小化するようなパラメータ値の組合せを探索する．図 10 では，Hadoop のチューニングを行っているが，初期値やクロージャを変更すればさまざまなアプリケーションに適用可能である．

4. 実験

実験では，本研究の提案環境を利用して，一般的なサーバアプリケーションのチューニング過程を効率的に記述ができることを示す．まず，アプリケーションを分散オブジェクト化する際の記述量を評価する．続いて，チューニングスクリプトの記述量について評価を行う．さらに，提案環境を利用して記述したスクリプトを用いて，一般的なチューニング過程を実現できることを確かめる．ここでは，(1) 予備実験，(2) スクリーニング，(3) 最適パラメータ値の探索（自動チューニング），および (4) 検証の 4 つの実験がすべて可能であることを確認する．

また，本実験では，SPECweb2005 ベンチマークでの Apache ウェブサーバと，Hadoop の 2 種類のアプリケーションを対象とした．

4.1 実験環境

実験では, Intel Xeon 3.60 GHz のデュアル CPU, 2 GB のメモリ, SCSI 接続の 36 GB HDD で構成された同一サーバを使用し, すべて 1000BASE-T で接続されている．基本となるソフトウェアは CentOS 5.7 (2.6.18-274.12.1.el5xen)，Java SE 1.6.0_24，Scala 2.9.1.final を使用した．Apache ウェブサーバの実験では，上記のサーバから Apache に 1 台，バックエンド・シミュレータ（Besim）に 1 台, SPECweb のプライムクライアントに 1 台, SPECweb クライアントに 8 台を割り当て, 計 11 台使用した．また, 各種サーバソフトウェアは Apache 2.2.3, PHP 5.3.3, FastCGI 2.4.0, `mod_fastcgi` 2.4.6 を使用し，ベンチマークとして SPECweb2005 1.2.0 の PHP 版を使用した．Hadoop の実験では，ネームノード 1 台とデータノード 9 台の計 10 台のサーバを割り当て，すべて Hadoop 0.20.203 を使用した．

4.2 分散オブジェクト作成時の記述量比較

本環境を用いてチューニングを行うためには，分散オブジェクトを作成する必要があるため，その際の記述量について比較した．今回は，以降の実験に使用するアプリケーションのオブジェクトを，すべて手動で作成した場合と自動生成機構を利用して作成した場合のコード記述量について比較した．この結果を**表 1** に示す．なお，自動生成機構を利用してオブジェクト作成した場合の記述量については，IDL の記述に要した行数，IDL から自動生成された行数，手動で追加した行数，および最終的に作成された分散オブジェクト全体の行数を示した．また，これらのオブジェクトには，実験に必要な最低限の機能のみを実装した．

表 1 で，すべて手動で作成したオブジェクトとのコード行数と比較すると，自動生成機構を利用した場合の方が，手動で記述するコード量が少なくなっている．自動生成機構を利用した場合の手動記述量は，IDL の記述量と手動で追加したコード量の合計である．アプリケーションごとにみると，Apache は約 43%，Besim では約 63%，SPECweb では約 38%，SPECwebClient では約 14%，Hadoop では約 33%の記述量を，すべて手動で作成した場合に比べて削減することができている．Besim は，設定ファイルに対する操作が主体となるため，オブジェクト全体のコード量に対し自動生成可能な部分のコードの割合が大きい．そのため，自動生成機構を利用することで，すべて手動で作成する場合に比べて，効率的にオブジェクトを作成できる．一方，SPECwebClient は，設定ファイルに対する操作がなく，自動生成可能な部分が少ない．ただし，このような場合でも，自動生成機構が分散オブジェクトのインタフェースとスケルトンを生成するため，利用者は単にアプリケーション固有の動作に対応するメソッドの内容を実装すればよい．表 1 の比較結果から，この場合に関しても，すべて手動で生成する場合に比べて自動生成機構を利用した場合の方が，わずかではあるがコード記述量が削減できていることが分かる．Apache，SPECweb，Hadoop についても，アプリケーション固有の動作に対応するメソッドの実装の必要性から，Besim に比べて手動記述量の削減割合は小さいものの，自動生成機構を利用することで，それぞれ効率

表 2　チューニング記述のコード行数
Table 2　Numbers of lines in tuning scripts.

アプリケーション	実験共通部分	予備実験	スクリーニング実験	自動チューニング実験	検証実験
Apache	74	8	7	15	6
Hadoop	67	8	6	19	5

的にオブジェクトの作成を行うことができた．

さらに，1 度オブジェクトを作成すれば，チューニング過程の各段階においてそれらを何度も再利用することができ，スクリプト記述作業の効率化に貢献できると考えられる．

4.3　スクリプトの記述量の比較

表 2 に，4 つのチューニング実験の各スクリプトの記述量の比較結果を示す．今回は，サーバの初期化やクライアントエミュレータの管理といったチューニングの過程の各段階において頻出する操作を，実験共通部分としてまとめている．そして，この実験共通部分を利用して各実験のスクリプト記述を行った．

表 2 では，Apache と Hadoop の両アプリケーションのチューニングについて，実験共通部分のスクリプト記述の行数と，(1) 予備実験，(2) スクリーニング，(3) 自動チューニング，および (4) 検証という一般的なチューニング過程の実験スクリプトの記述に要した行数を示している．

まず，今回の実験の共通部分の記述量は，Apache が 74 行，Hadoop が 67 行である．これに対し，予備実験，スクリーニング実験，検証実験に特有の記述は，Apache と Hadoop ともに 10 行未満である．また，自動チューニング実験については，他の実験よりも共通部分に追記する記述量が増えているが，実験に使用した滑降シンプレックス法のライブラリが 100 行以上であることを考えると，こちらもチューニング記述を効率的に行うことができたと考えられる．

このように，オブジェクトの再利用性を利用することで，チューニング過程の各段階に共通する操作をまとめて記述することができる．そして，この共通部分に対し，各実験に特有な操作に関するわずかな記述を加えることで，チューニングの過程の各段階を効率的に記述できる．

以上のことから，アプリケーションの分散オブジェクト化と，その自動生成機構を利用することで，効率的にチューニング記述を行うことが可能であると考えられる．

4.4　(1) 予備実験

はじめに，作成したオブジェクトを利用してサーバチューニングの予備実験を行った．チューニング作業前の予備実験には，たとえばサーバアプリケーションやクライアントエミュレータが正常に動作しているかの確認や，チューニング対象となるパラメータ値の設定すべき範囲を見つける，

```
 1: /*
 2:  * Apache の実験共通部分
 3:  */
 4:
 5: def preExpr(mc: Int, ka: Int) = {
 6:   apache.maxClients = mc
 7:   apache.keepAliveTimeout = ka
 8:   apache.restart
 9:   specweb.start
10: }
11:
12: val result = for(mc <- 100 to 1000 by 100)
13:   yield (preExpr(mc, 1), preExpr(mc, 15))
```

図 11　Apache の予備実験スクリプト
Fig. 11　Script for an Apache pre-experiment.

などのさまざまな作業が考えられる．本論文では，Apache の `MaxClients` と `KeepAliveTimeout` パラメータのおおよその効果を確認した．

図 11 に Apache の予備実験スクリプトを示す．図 11 のスクリプトでは，Apache の `MaxClients` と `KeepAliveTimeout` という 2 つのパラメータの組合せに対する性能の変化を調べている．まず，Apache の実験共通部分で，サーバの初期化や SPECweb2005 ベンチマークの設定を行う．続いて 5 行目から 10 行目で，`preExpr()` 関数を定義している．この関数は，引数で受け取った `MaxClients` と `KeepAliveTimeout` パラメータの組合せにおける Apache の性能を SPECweb ベンチマークを用いて測定し，測定結果を返す関数である．また，12 行目と 13 行目の `for` 文で，`KeepAliveTImeout` の値が 1 の場合と 15 の場合に，`MaxClients` の値を 100 から 1,000 まで 100 ずつ変化させたときの性能測定の結果を取得している．

4.5　(2) スクリーニング実験

続いて，スクリーニング実験を行う．一般的に，チューニングでは探索空間を現実的な範囲に収めるために，非常に多数のパラメータの中から，少数の効果的なパラメータに絞り込む作業（スクリーニング）が必要であることが知られている．

本実験では，提案手法を利用してこのスクリーニング実験ができることを確認する．ここでは実験計画法を使用し，統計的な裏付けに基づき，少ない実験回数でそれぞれのパ

ラメータの効果を確かめている．なお，実験計画の作成と解析には JMP 8 [12] を使用した．また，スクリーニング実験の詳細については，実験計画法の一般的な教科書 [13] に記載されている．

4.5.1　Apache のパラメータスクリーニング

本実験では，まず Apache に対して L16(2^{15}) 計画のスクリーニング実験を行った．L16 計画を使用すると，16 回の実験で，最大 15 個までのパラメータの効果を確かめることができる．実験では SPECweb2005 が提供する `Banking`，`Ecommerce`，`Support` の 3 つすべてのワークロードを対象とし，SPECweb2005 のスコア値である同時接続数の最大化を目指す．

表 3　Apache のスクリーニング対象パラメータ
Table 3　Apache parameter values for screening.

パラメータ	水準 1	水準 2
`Apache.MaxClients`	150	700
`Apache.KeepAliveTimeout`	1	15
`Apache.Timeout`	30	300
`Apache.MaxRequestsPerChild`	0	10
`Apache.MCacheSize`	0	512 MB
`Apache.AllowOverride`	Off	On
`Apache.Modules`	最小	すべて
`Apache.HostnameLookups`	Off	On
`Apache.Logging`	Off	On
`APC.shm_size`	0	512 MB
`Besim.MaxClients`	150	700
`Besim.KeepAliveTimeout`	1	15

表 3 に実験で使用したパラメータとその値の組合せを示す．本実験は L16 計画であるので，パラメータごとに 2 つの値で試行を行う．この 2 つの値は，各パラメータの効果が顕著になるように十分広くとる必要があり，ここでは先行研究の結果 [14] を参考にしている．なお，試行を行う値の見当がつかない場合は，本スクリプティング環境を利用し，事前に確認しておく必要がある．本スクリプティング環境はこのような予備実験にも，役に立つことが期待される．

各パラメータの効果を確認する前に，スクリーニング実験で仮定したモデルについて検証する．**表 4** に，全体の分散分析の結果を示す．12 個のパラメータの効果を確かめているため，モデルの自由度は 12 であり，誤差には自由度 3 が割り当てられている．各ワークロードの p 値を見ると，`Banking` と `Support` は有意水準の 10%（0.1）以下であるから，パラメータの効果が十分解釈可能であり，`Ecommerce` については有意水準以下ではないが，パラメータの効果を確認できないほどではないことが分かる．

各パラメータの効果を**表 5** に示す．表中の F 値は分散分析に基づいて計算した F 分布上の値であり，p 値はその F 値における確率である．通常，p 値が有意水準以下であれば，偶然では起こりえない効果があったと見なす．ここでは，有意水準を 10%（0.1）としている．各ワークロードに対して，10%有意であると判定されたパラ

表 4　Apache における分散分析の結果
Table 4　ANOVA results of Apache parameter screening.

		Banking		Ecommerce		Support	
要因	自由度	平方和	F 値	平方和	F 値	平方和	F 値
モデル	12	1,154,426.5	14.0145	1,175,042.5	2.1810	957,116.25	7.4806
誤差	3	20,593.5	p 値	134,692.5	p 値	31,986.69	p 値
全体	15	1,175,020.0	0.0258∗	1,309,735.0	0.2837	989,102.94	0.0619∗

表 5　Apache パラメータスクリーニングの実行結果
Table 5　ANOVA results for each Apache workload.

	`Banking`		`Ecommerce`		`Support`	
パラメータ	F 値	p 値	F 値	p 値	F 値	p 値
`Apache.MaxClients`	90.917	0.024∗	17.0722	0.0257∗	32.8419	0.0105∗
`Apache.KeepAliveTimeout`	25.881	0.0147∗	7.4668	0.0718∗	22.3124	0.0180∗
`Apache.Timeout`	0.7552	0.4488	0.0281	0.8776	1.0096	0.3890
`Apache.MaxRequestsPerChild`	13.685	0.0343∗	0.4180	0.5640	3.9897	0.1397
`Apache.MCacheSize`	0.1785	0.7012	0.0829	0.7922	2.7185	0.1978
`Apache.AllowOverride`	11.0971	0.0447∗	0.1516	0.7230	5.8267	0.0947∗
`Apache.Modules`	14.7314	0.0312∗	0.2503	0.6513	16.2112	0.0275∗
`Apache.Logging`	9.8857	0.0515∗	0.0009	0.9775	0.0072	0.9378
`Apache.HostnameLookups`	0.1635	0.7131	0.0018	0.6297	0.2863	0.6269
`APC.shm_size`	0.5600	0.5086	0.6551	0.4775	0.4433	0.5532
`Besim.MaxClients`	0.0554	0.8291	0.0018	0.9688	3.6306	0.1528
`Besim.KeepAliveTimeout`	0.2631	0.6434	0.00421	0.8505	0.4896	0.5345

メータが * 付きで示されている．Apache の MaxClients と KeepAliveTimeout はどのワークロードに対しても効果がある．また，AllowOverride や Modules は Banking および Support に効果があり，MaxRequestsPerChild と Logging は Banking のみに効果が現れた．

表 5 の結果は，一般的に知られている経験則や先行研究の結果 [14] とほぼ一致している．スクリーニング実験により得られたパラメータの効果の判定が本当に正しいのか確認するには，パラメータ空間を全数探索する必要がある．しかし，本実験が対象としたパラメータ空間を全数探索するには 32,768 回の実験が必要であり，時間的な制約からそのような実験は現実的に難しく，そもそもそれが可能であれば実験計画法を用いてスクリーニング実験を行う意味がない．そのため，スクリーニング実験では一般的に，表 4 のようなモデルの仮定に一致すれば，一般的に効果が確かめられていると解釈する．

4.5.2 Hadoop のパラメータスクリーニング

Hadoop では，ベンチマークとして Hadoop に付属するサンプルプログラムの wordcount，sort，grep を使用した．また，同じく Hadoop 付属のサンプルプログラムにより作成したランダムデータ 10 GB を入力として使用した．

表 6 にスクリーニング実験の対象としたパラメータとその値の組を示す．今回は，Hadoop の公式ドキュメント [15] で性能に影響を与えるとされるパラメータから 10 個を選択した．Hadoop の実験では，実行時間を最小化することを目標とし，先行研究 [16] を参考に実験計画は L48 を使用した．

表 6　Hadoop のスクリーニング対象パラメータ
Table 6　Hadoop parameter values for screening.

パラメータ	水準 1	水準 2
io.file.buffer.size	4,096	131,072
dfs.namenode.handler.count	10	40
mapred.job.tracker.handler.count	10	40
mapred.reduce.parallel.copies	5	20
io.sort.factor	10	50
io.sort.mb	100	200
tasktracker.http.threads	40	80
mapred.tasktracker.map.tasks.maximum	1	4
mapred.tasktracker.reduce.tasks.maximum	1	4
mapred.child.java.opts	512 MB	1,024 MB

表 7 にまず，全体の分散分析の結果を示す．どのワークロードにおいても，p 値が 10%の有意水準以下のため，パラメータの効果が十分解釈可能であることが期待される．これは，パラメータの個数 10 個に対して，実験回数を 48 回としているため，誤差の自由度が 37 確保されているからである．

表 8 にパラメータの効果を示す．それぞれのワークロードに対して，10%有意であると判定されたパラメータを * 付きで表す．map.tasks.maximum は，wordcount と grep に対して，また，io.sort.mb は，wordcount と sort に対して効果があると判定された．これら以外に，wordcount では dfs.namenode.handler.count と mapred.child.java.opts に効果があり，sort では reduce.tasks.maximum に効果があると判定された．

以上の結果は，一般的な経験則 [15] や先行研究の結果 [16] ともほぼ一致しており，表 7 の仮定しているモデルにも一致していることから，これらのパラメータに効果があることが期待される．

4.6 （3）自動チューニング実験

スクリーニング実験によって，探索対象が少数の効果的なパラメータに絞り込まれれば，それらを対象として自動チューニングを行い，最終的なパラメータの値を決定することができる．本スクリプティング環境が提供するライブラリを利用して，自動チューニングができることを確かめるため，実験を行った．

本論文では，Hadoop の wordcount を対象に自動チューニングを行った結果を示す．なお，Hadoop の sort と grep については，効果があったパラメータの少なさから，また Apache については実験環境のディスク容量の制限からスコアが 1,000 接続で打ち切りとなるため省略した．

表 8 のスクリーニング実験結果から，dfs.namenode.handler.count，io.sort.mb，map.tasks.maximum および mapred.child.java.opts の 4 つを自動チューニングの対象として選択した．また，入力するデータのサイズは，実験時間の短縮のため 5 GB とした．滑降シンプレックス法の収束判定条件は，実行時間の最小値を best，最大値を worst とした場合に $\frac{2|worst-best|}{|worst|+|best|} < 0.05$ とした．本実験のスクリプトは図 10 の記述をもとにしている．

図 12 に自動チューニングの収束過程を示す．滑降シン

表 7　Hadoop における分散分析の結果
Table 7　ANOVA results of Hadoop parameter screening.

		wordcount		sort		grep	
要因	自由度	平方和	F 値	平方和	F 値	平方和	F 値
モデル	10	770,335.88	64.2976	250,979.21	2.8915	366,986.71	9.3909
誤差	37	44,328.94	p 値	321,152.77	p 値	144,591.77	p 値
全体	47	814,664.81	$< .0001*$	572,131.98	$0.0090*$	511,578.48	$< .0001*$

表 8　Hadoop パラメータスクリーニングの実行結果
Table 8　ANOVA results for each Hadoop workload.

パラメータ	wordcount		sort		grep	
	F 値	p 値	F 値	p 値	F 値	p 値
io.file.buffer.size	1.7254	0.1971	1.3609	0.2508	0.0781	0.7815
dfs.namenode.handler.count	82.867	< .0001∗	1.3179	0.2583	0.0039	0.9506
mapred.job.tracker.handler.count	2.1423	0.1517	0.2630	0.6111	1.5259	0.2245
mapred.reduce.parallel.copies	0.6748	0.4166	1.4342	0.2387	1.2437	0.2720
io.sort.factor	0.8188	0.3714	0.0245	0.8765	0.0003	0.9872
io.sort.mb	249.118	< .0001∗	18.1912	0.0001∗	0.3467	0.5596
tasktracker.http.threads	0.6748	0.4166	0.5461	0.4646	0.0225	0.8815
mapred.tasktracker.map.tasks.maximum	242.580	< .0001∗	1.9745	0.1683	88.6132	< .0001∗
mapred.tasktracker.reduce.tasks.maximum	2.0220	0.1634	3.1909	0.0822∗	1.0940	0.3024
mapred.child.java.opts	60.353	< .0001∗	0.6121	0.4390	0.9811	0.3284

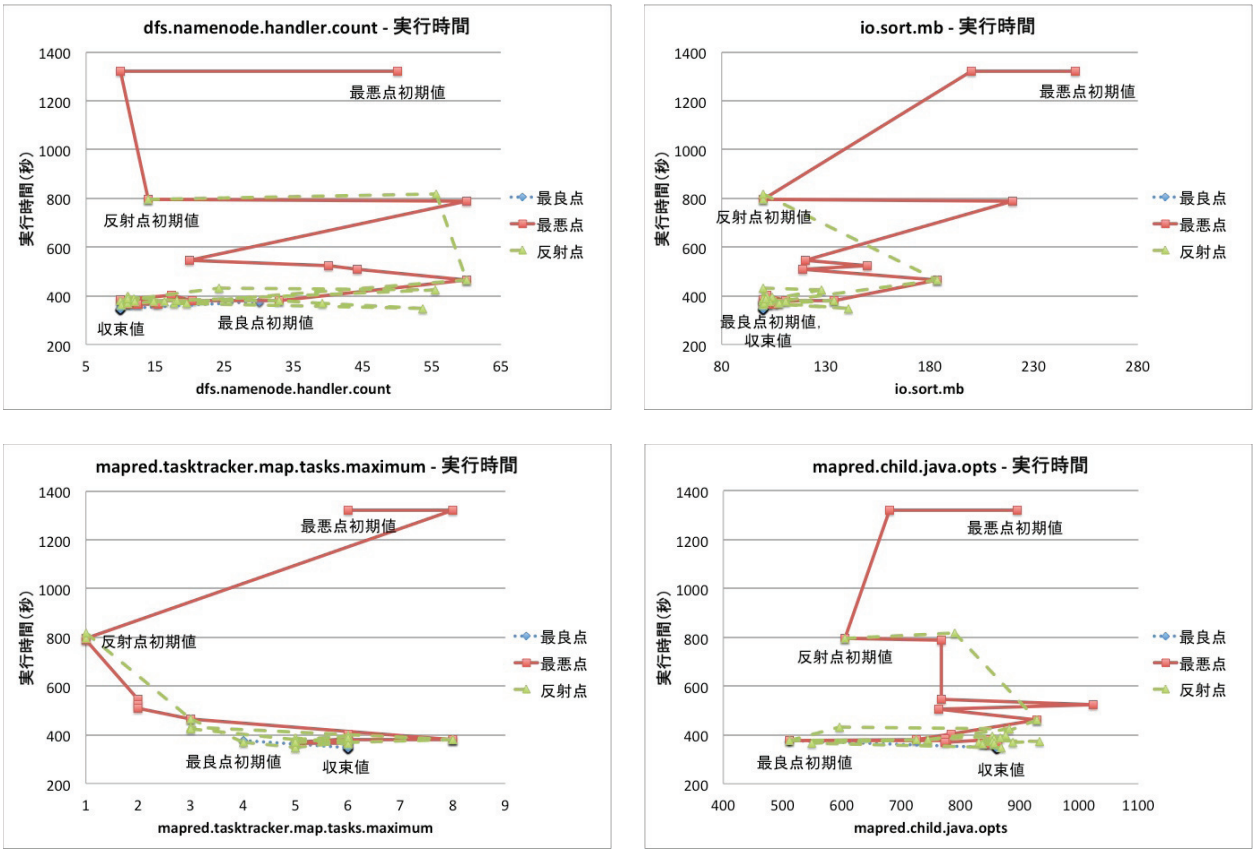

図 12　滑降シンプレックスアルゴリズムによる Hadoop のチューニング過程
Fig. 12　Tuning trails of downhill-simplex algorithm.

プレックス法では，ある時点でのすべての探索点のうち，最良点と最悪点，および最悪点の次に悪い点（最悪次点）の 3 点と，最悪点を除く点群の重心に対して最悪点の点対称となる点（反射点）を比較し，次の探索点を決定していく．このような操作を収束判定条件を満たすまで繰り返し行うために，図 12 のような軌跡となる．今回の実験の場合，アルゴリズム収束までの試行回数は 21 回であった．

図 12 は，滑降シンプレックス法の最良点と最悪点および反射点の推移の軌跡を，各パラメータごとに示している．図 12 のそれぞれの軌跡から，反復によって最悪点のパラメータ値が特定の値に向かって収束するのにともない，評価値である実行時間も小さくなっていることが分かる．また，今回の実験では，初期値に選択した点のうちの 1 点の実行時間がもともと小さかったため，最良点の更新は 1 度しか行われなかった．

また，**表 9** にチューニング開始時点と終了時点での wordcount の実行時間を示す．本論文の滑降シンプレックス法の場合，大局的な最適解が求まる保証はないが，自動チューニングの終了時点での実行時間は，開始当初と比べるとはるかに改善されていることが分かる．

表 9　滑降シンプレックスアルゴリズムによるチューニング結果
Table 9　Tuning results of using downhill-simplex Library.

	パラメータ				実行時間（秒）
	dfs.namenode.handler.count	io.sort.mb	map.tasks.maximum	mapred.child.java.opts (MB)	
開始時の最悪点	50	250	6	896	1,322
終了時の最良点	11	100	6	850	347

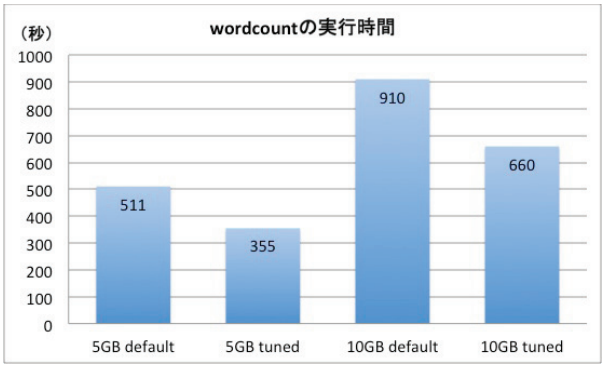

図 13　実行時間の比較によるチューニング結果の検証
Fig. 13　Verification results of Hadoop wordcount tuning.

4.7 （4）検証実験

最後に，自動チューニングの結果に対して検証実験を行い，実際にサーバ性能が向上していることを確かめた．具体的には，自動チューニング実験で得られたパラメータ値を設定した場合と，デフォルトのパラメータ値を使用した場合の性能を比較した．Hadoop の `wordcount` ワークロードに対するチューニングの検証結果を**図 13** に示す．

図 13 のグラフでは，ランダムデータ 5 GB と 10 GB に対して，デフォルトパラメータ（default）とチューニング結果（tuned）のパラメータにおける Hadoop の `wordcount` の実行時間を比較している．なお，この実行時間は，Hadoop の実行における誤差を考慮し，同条件で 3 回実験を行った場合の中央値である．グラフより，両データサイズにおいて，チューニング後の実行時間がデフォルトパラメータの場合に比べ約 30%短縮されている．このことから，実際にサーバ性能が向上していることを確認した．

これらの実験結果より，本研究の提案環境を利用することで，一般的なチューニング過程におけるさまざまなスクリプトを効率的に記述し，実際にチューニングが行えることが確かめられた．

5. まとめ

本研究では，サーバアプリケーションのチューニング過程を記述するためのスクリプティング環境を提案した．一般的にチューニングの過程では，さまざまに条件を変えながら，何度も試行を繰り返す必要がある．また，1 つのチューニングのサイクルにおいても，サーバの設定変更だけではなく，複数台のクライアントエミュレータの設定を書き換え，起動するなど煩雑な手続きを必要とする．

本環境は，チューニングに必要な構成要素を分散オブジェクト化し，それらを操作するための高水準なスクリプティング環境を提供する．これらによって，サーバやクライアントエミュレータなどのチューニング環境の構築作業を効率化する．また，自動チューニングのアルゴリズムをライブラリ化しておくことで，さまざまなサーバアプリケーションに対して，簡単に自動チューニングを適用できることが期待される．

実験では，本手法を利用して SPECweb2005 ベンチマーク下の Apache ウェブサーバおよび Hadoop のチューニングを行った．その結果，本手法を通じてこれらのサーバアプリケーションがチューニングできることを確認した．

謝辞　本研究の一部は，総務省 SCOPE「ディペンダブルな自律連合型クラウドコンピューティング基盤の研究開発」の支援，および科研費（2230006）と科研費（22700023）の助成を受けている．

参考文献

[1] Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L.N. and Pasupathy, S.: An Empirical Study on Configuration Errors in Commercial and Open Source Systems, *ACM SOSP '11*, pp.159–172 (2011).
[2] Nagaraja, K., Oliveira, F., Bianchini, R., Martin, R.P. and Nguyen, T.D.: Understanding and dealing with operator mistakes in internet services, *USENIX OSDI '04*, pp.61–76 (2004).
[3] Oppenheimer, D., Ganapathi, A. and Patterson, D.A.: Why do internet services fail, and what can be done about it?, *USENIX USITS'03*, Vol.4, p.1 (2003).
[4] Chung, I.-H. and Hollingsworth, J.: Automated Cluster-Based Web Service Performance Tuning, *IEEE HPDC'04*, pp.36–44 (2004).
[5] Xi, B., Liu, Z., Raghavachari, M., Xia, C.H. and Zhang, L.: A Smart Hill-Climbing Algorithm for Application Server Configuration, *WWW'04*, pp.287–296 (2004).
[6] Saboori, A., Jiang, G. and Chen, H.: Autotuning Configurations in Distributed Systems for Performance Improvements Using Evolutionary Strategies, *IEEE ICDCS '08*, pp.769–776 (2008).
[7] Nelder, J.A. and Mead, R.: A Simplex Method for Function Minimization, *The Computer Journal*, Vol.7, No.4, pp.308–313 (1965).
[8] Anderson, P., Goldsack, P. and Paterson, J.: SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control, *USENIX LISA '03*, pp.213–222 (2003).
[9] Zheng, W., Bianchini, R. and Nguyen, T.D.: Automatic

Configuration of Internet Services, *ACM EuroSys '07*, pp.219–230 (2007).
[10] Vinoski, S.: CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments, *Communications Magazine*, Vol.35, No.2, pp.46–55, IEEE (1997).
[11] Sugiki, A., Kato, K., Ishii, Y., Taniguchi, H. and Hirooka, N.: Kumoi: A High-Level Scripting Environment for Collective Virtual Machines, *IEEE ICPADS '10*, pp.322–329 (2010).
[12] SAS Institute Inc.: JMP (online), available from ⟨http://www.jmp.com/⟩ (accessed 2011-12-28).
[13] 山田　秀：実験計画法—方法編，日科技連 (2004).
[14] 杉木章義，加藤和彦：実験計画法を利用したウェブサーバの主要なパラメータ選択手法，情報処理学会 OS 研究会報告（2008-OS-108 (35)），pp.33–40 (2008).
[15] Apache Hadoop: Cluster Setup (online), available from ⟨http://hadoop.apache.org/common/docs/r0.20.203.0/⟩ (accessed 2011-12-02).
[16] 杉木章義，加藤和彦：実験計画法を利用した MapReduce のパラメータ設定，日本ソフトウェア科学会 DSW'09 Summer (2009).

相川 拓也

2011 年筑波大学第三学群情報学類卒業．現在，同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博士前期課程に在学．分散システムやシステムソフトウェア工学の分野に興味を持つ．

杉木 章義　（正会員）

2002 年電気通信大学情報工学科卒業．2004 年同大学大学院情報工学専攻博士前期課程修了．2007 年同博士後期課程修了．博士（工学）．2007 年科学技術振興機構 CREST 研究員，2009 年筑波大学システム情報工学研究科助教．分散システム，オペレーティングシステムの研究に従事．2009 年度山下記念賞受賞．日本ソフトウェア科学会，ACM，IEEE-CS，USENIX 各会員．

加藤 和彦　（正会員）

1989 年東京大学理学部情報科学科助手，1993 年筑波大学電子・情報工学系講師，1996 年同助教授，2004 年筑波大学大学院システム情報工学研究科教授，現在に至る．1992 年博士（理学）（東京大学）．オペレーティングシステム，分散システム，仮想計算環境，セキュリティに興味を持つ．1990 年情報処理学会学術奨励賞，1992 年同研究賞，2005 年同論文賞，2004 年日本ソフトウェア科学会論文賞，各受賞．