Vol.2011-OS-118 No.11
2011/7/27

クラウド環境におけるサーバパラメータ調整のためのスクリプティング環境

相川拓也[†1]　杉木章義[†1]　加藤和彦[†1]

近年のクラウドコンピューティングの普及に伴い，効率的なデータセンター管理が求められている．サーバのパラメータ調整は知識と経験を必要とする難易度の高いタスクである一方で，適切に調整を行うかどうかでサーバの性能に大きな差が生じる．本研究では，サーバパラメータ調整のためのスクリプティング環境を提案する．パラメータ調整に必要なアプリケーション，クライアントなど全ての構成要素を分散オブジェクト化し，統一的な環境で，高水準なスクリプティングによりパラメータ調整を可能とする．また，さまざまなアプリケーションに対応するため，インターフェース定義言語から分散オブジェクトの自動生成を行う．実験では，SPECweb2005 を使用した Apache ウェブサーバおよび Hadoop に対して実際にスクリプト記述を行い，パラメータ調整が可能であることを確認した．

A High-level Scripting Environment for Server Parameter Tuning in the Clouds

Takuya Aikawa,[†1] Akiyoshi Sugiki [†1]
and Kazuhiko Kato[†1]

After the rise of cloud computing, there is a sharp increase in demand for effective management in a data center. Whereas parameter tuning is a most difficult task that requires expertise and knowledge, it strongly affects server performance. In this paper, we present a scripting environment for parameter tuning in the cloud environment. The system offers distributed language objects each of which corresponds to a benchmark components such as an application or a client. This allows high-level scripting in an integrated environment, being unconscious of distribution and reducing many tasks. To support a variety of applications, we also provide a compiler which generates distributed objects from Interface definitions. Our experiments with Apache under SPECweb2005 and Hadoop, and the results show that parameter tuning is successfully possible in our scripting environment.

1. 序　　論

近年のクラウドコンピューティングの普及に伴い，効率的なデータセンター管理が求められている．クラウドは, 物理サーバやネットワークなどの計算資源を利用者が保有することなく，ネットワークを介してこれらを必要な分だけ利用する形態である．クラウドでは，計算資源がデータセンタに集約されており，これまで以上に効率的な管理が求められている．

データセンター上のさまざまな管理の中で，サーバのパラメータ調整作業は難易度の高い最も重要なタスクの一つである．この作業は，アプリケーションごとに豊富な知識と経験を必要とし，同じハードウェア構成であってもパラメータ調整を行うかどうかで性能が大きく変化する．

パラメータ調整のための研究は，クラウドだけに限らず，さまざまな分野で既に行われている．最も研究が盛んなのは数値計算分野であり，1)–5) などの研究が存在する．また，インターネットサーバ分野においても，6)–8) などの研究がある．しかしながら，現状，これらの研究成果はクラウド分野では想定したほど普及していない．その原因として，個別のアプリケーションへのチューニング適用の難しさがあると考えられる．

そこで本研究では，クラウド環境におけるサーバパラメータ調整のためのスクリプティング環境を提案する．スクリプティングでパラメータ調整を行う手法は既にさまざまなものが提案されているが，本研究はそのアプローチに特色がある．

本機構では，まず，パラメータチューニングに必要なアプリケーション，ベンチマーククライアントなど，全ての構成要素を分散オブジェクトとして操作可能とする．これらの分散オブジェクトは実際の計算資源と対応しており，オブジェクトに対する代入式のような記述でパラメータ値を変更したり，メソッド呼び出しによりアプリケーションの再起動などの操作を行うことが可能である．次に，それらのオブジェクトに対して高水準なスクリプティング環境を提供する．本環境はプログラミング言語 Scala[9] 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の性質を融合した記述を行うことができる．この環境では，チューニングの構成要素が全て分散オブジェクトとして提供されていることから，統一的な環境で透過的に，分散を意識することなく効率的にチューニング作業を進め

†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

ることができる．さらに本システムは，並列分散でチューニング作業を行うための分散並列関数機能も提供している．

また，クラウド環境では利用者がさまざまなアプリケーションに対してチューニングを行いたいと考えられるため，この多様性に対応できる能力が必要である．しかし，Apache, Hadoop などの個別のアプリケーションに対して，利用者が対応する分散オブジェクトを手動で記述していたのでは，そのような利用は難しい．そのため本研究では，アプリケーションに対応する分散オブジェクト自動生成機構を提供する．本機構はアプリケーションについてのインターフェース定義言語の記述から，対応する言語オブジェクトのスケルトンコードを自動生成する．

評価では，自動生成機構を用いて対応する言語オブジェクトを作成し，SPECweb2005 ベンチマークを使用した Apache ウェブサーバと Hadoop に対して実験を行った．その結果，まず少量のインターフェース記述と追加のコード記述でアプリケーション対応言語オブジェクトを生成可能であることを確かめた．さらに，実験計画法を利用したパラメータのスクリーニングを実際に行うことができることを示した．

2. サーバのパラメータ調整のためのスクリプティング環境

本研究のスクリプティング環境の構成を図 1 に示す．本システムは，スクリプティング環境のフロントエンドとなるシェルと，データセンタ内の各計算機ごとにインストールされたカーネルから構成される．カーネルでは，サーバアプリケーションなどを分散オブジェクトとして提供する機能，並列分散処理機能，メンバシップ機能などを提供する．

本システムの構成にあたっては，当研究室で開発しているクラウド基盤ソフトウェア Kumoi[10] を利用した．Kumoi は物理計算機や仮想マシンなどを分散オブジェクトとして提供し，クラウド環境を構成するためのスクリプティング環境を提供している．本システムでは，アプリケーション操作への対応にあたって，必要な設計の変更や分散オブジェクトの自動生成機構などの拡張を行う．

2.1 クラウド基盤ソフトウェア Kumoi

Kumoi はクラウド環境，特に Infrastructure-as-a-Service（IaaS）型のクラウド環境を想定して設計されたクラウド基盤ソフトウェアである．仮想マシンをサービスとして提供する IaaS を想定しているため，オペレーティングシステム上で稼動するサーバアプリケーションやベンチマーククライアントなどの操作環境は提供していない．

Kumoi のインターフェースは，プログラミング言語 Scala[9] の対話シェルをもとに実装

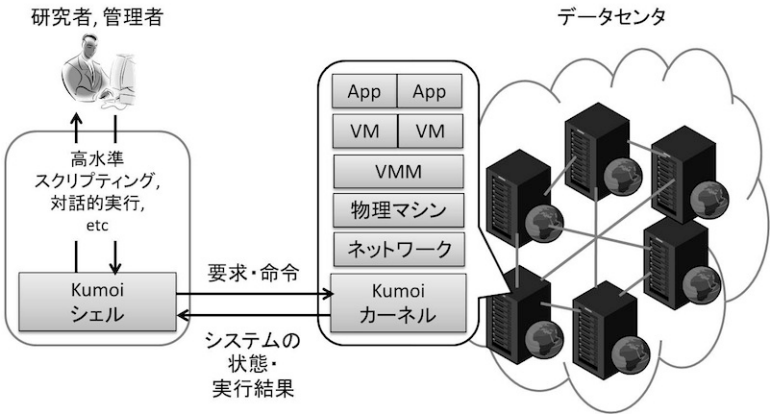

図 1 サーバのパラメータ調整のためのスクリプティング環境

されており，下記のような操作環境を提供する．この例では，CPU 使用率が 90%を超える物理計算機を抽出し，それらの名前を取得している．

```
scala> pms.dfilter(_.cpuRatio > 0.9).dmap(_.name)
res0: List[String] = List(ibm2, ibm3, ibm4, ibm7, ...
```

この例の `pms` は現在参加している物理計算機のリストを返すメソッドである．`dfilter()` や `dmap()` は分散版の `map()` や `reduce()` 関数であり，並列に処理を進めることができる．各物理計算機は，`cpuRatio` メソッドや `name` メソッドなどを提供し，さまざまな情報の取得や操作を行うことができる．Kumoi の設計や実装の詳細は参考文献[10] に記述されている．

2.2 Kumoi の拡張

サーバのパラメータ調整のためのスクリプティング環境を構築するためには，アプリケーションやベンチマーククライアントの設定を変更し，それぞれの設定の反映と再起動を行い，ベンチマークを実行し，結果を観測する，という一連のフィードバックの流れが全てスクリプティング環境上で実現可能でなければならない．

まずは，アプリケーションに対応する分散オブジェクトを取得可能とするために，物理計算機オブジェクトに下記を追加する．`apps` はアプリケーションの一覧であり，`appm` はアプ

リケーションを管理するモニタである．

```
scala> pms.head.apps
res1: List[kumoi.shell.app.Application] = List(Apache, Hadoop, ...
scala> pms.head.appm
res2: kumoi.shell.app.AppMonitor = AppMonitor
```

個別のサーバアプリケーションでは，パラメータ値の変更や変更の反映など全ての操作を言語オブジェクトを通じて可能とする．下記の例では，Apache の設定ファイルに対応する `config` オブジェクトのパラメータ設定関数を利用して `MaxClients` と `KeepAliveTimeout` のパラメータ値の設定を行った後，`save()` を呼び出して変更を反映させている．

```
scala> val config = pms.head.apps.find(_.name == "Apache").get.config
config: kumoi.shell.app.Config = ApacheConfigImpl
scala> config.maxClients = 750
scala> config.keepAliveTimeout = 3
scala> config.save()
```

現在，Kumoi では仮想マシンや物理計算機，ネットワークなどに対応する分散オブジェクトを全て手動で記述している．しかしながら，パラメータチューニングではさまざまなアプリケーションを扱う必要があるため，対応する言語オブジェクトを個別に記述する場合には，その作業のコストが問題となる．よって，本研究では次の章で説明するアプリケーション対応言語オブジェクトの自動生成機構の作成を行った．

3. アプリケーション対応オブジェクトの自動生成

本研究では，さまざまなアプリケーションに対応するため，言語オブジェクトの自動生成機構を作成した．自動生成とは言っても，個別のアプリケーションに依存する部分があることから，全てを自動生成できる訳ではなく，ユーザが実装すべき部分を残した言語オブジェクトのスケルトンを生成する．本機構では，アプリケーションに関するインターフェース定義言語 (IDL: Interface Definition Language) の記述を入力として受け取り，対応する言語オブジェクトのソースコードを出力する Source-to-Source コンパイラである．本機構は，分散オブジェクトにおける自動生成機構，例えば，CORBA[11] の IDL と関連している．

図 2 に IDL 記述の入力からの言語オブジェクト生成過程を示す．ここでは Apache を例に説明する．まず，ユーザは Apache の操作や設定の変更に関する定義を IDL で記述する．次に，言語オブジェクト生成器は IDL 記述と，Apache の設定ファイルテンプレートを元に Apache の言語オブジェクトのスケルトンを生成する．設定ファイルはいくつかの形式に分類されるという我々の分析から，設定ファイルテンプレートを予めいくつか用意しておき，自動生成の際に適合するものを選択してコード出力に利用する．また，プログラミング言語ごとに出力テンプレートを作成しておくことで，様々な言語のオブジェクトを生成できる．図 2 の例では，Scala 言語のオブジェクトを生成している．最終的に，言語オブジェクト生成器は 1 つのアプリケーションにつき 2 種類のスケルトンを生成する．一方は Apache の起動や終了などの操作に対応するオブジェクトであり，もう一方は設定ファイルの変更操作に対応するオブジェクトである．最終的に，ユーザはスケルトンに不足している記述を付け加え，完全なアプリケーション対応言語オブジェクトを作成する．

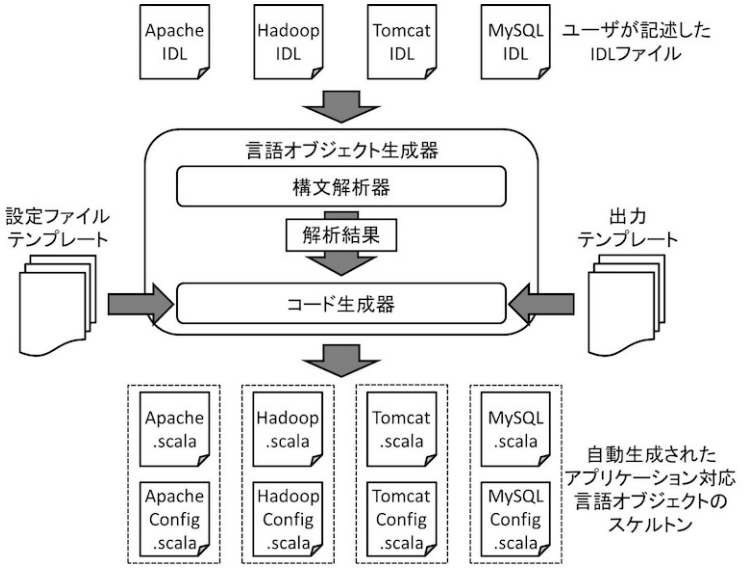

図 2 アプリケーション対応言語オブジェクトの生成過程

本生成過程の各構成要素について，次の節から詳しく述べる．

3.1 インターフェース定義言語

本研究におけるインターフェース定義言語では，アプリケーションについて，設定対象とするパラメータなどの設定項目や，起動や終了などの操作項目を記述する．ここでは，図 3 に示す Apache の IDL 記述例をもとに説明する．

オブジェクト名の定義

IDL では，まずアプリケーションに対応するオブジェクト名の宣言を行う．図 3 では予約語 app に続く Apache が該当する．

```
app Apache {
  Function {
    start{"/etc/init.d/httpd start"}
    stop{"/etc/init.d/httpd stop"}
    restart{"/etc/init.d/httpd restart"}
    checkConfig(path: String): Boolean
  }
  Config {
    httpd_conf("/etc/httpd/conf/httpd.conf"): LineConfig = {
        maxClients: Int = ValueAt("""MaxClients\s+([\d]+)""", 1)
        keepAliveTimeout: Int = ValueAfter("KeepAliveTimeout")
         ...
    }
    php_ini("/etc/php.ini"): LineConfig = {
      apcShmSize: Int =
          ValueAt("""apc\.shm_size\s*=\s*(\d+)[MG]""", 1)
    }
  }
}
```

図 3 Apache の IDL 記述例

関数の定義

本定義では，Apache の起動や終了などのアプリケーションに対する操作を関数として宣言する．Function ブロック内で関数を宣言することができ，start，stop は Apache の起動や終了に対応する操作である．また，引数や戻り値を定義することも可能であり，checkConfig はその例である．この例では，String 型の引数 path をとり Boolean 型を返す関数であることを示している．それぞれの実装では，現在，Unix/Linux のスクリプトの実行に対応する場合には，簡潔な記述法を用意している．

設定パラメータの定義

言語オブジェクトから設定可能なパラメータを Config ブロック内で宣言する．本定義はパラメータが複数の設定ファイルに分散している場合にも対応可能である．図 3 の例は，パラメータが設定ファイル httpd.conf および php.ini に分散している場合の記述である．各設定ファイル内の MaxClients や KeepAliveTimeout，apc.shm_size のパラメータを操作対象としている．

本研究の自動生成機構は，サーバアプリケーションの設定ファイルはいくつかの形式に分類されるという分析に基づいている．例えば，Apache の設定ファイルは基本的に 1 行で 1 つのパラメータを記述する形式をとっている．また，Hadoop などのように XML を設定ファイル記述として使用する形式のものも多い．

本機構では，1 行 1 項目の形式に対応する LineConfig，XML 形式に対応する XMLConfig などの設定ファイルテンプレートを提供している．前者では，設定ファイルから正規表現を利用してパラメータの情報を抽出および設定する．また後者では，XPath 表現を用いてパラメータの場所を記述し，値の取得や設定などを行う．

3.2 言語オブジェクト生成器と構文テンプレート

本研究で使用する言語オブジェクト生成器は Scala で実装されている．このプログラムは入力として受け取った IDL の記述を解析し，得られた抽象構文木をもとに言語オブジェクトのスケルトンコードを出力する．また，コード生成器は出力テンプレートを利用して出力する言語やコードを変更することができる．例えば，出力先のプログラミング言語ごとに出力テンプレートを作成しておけば，コード生成時に与えるテンプレートに応じて任意のプログラミング言語のソースコードを生成することができる．本研究で使用する Kumoi は Scala 言語で実装されているため，Scala 言語用の出力テンプレートを使用する．

4. 実　　験

実験では，個別のアプリケーション対応オブジェクト作成にかかる記述コスト，およびそれらのオブジェクトを利用したスクリプティング記述によりパラメータ調整が可能であるかどうかを評価する．SPECweb2005 を使用した Apache ウェブサーバおよび Hadoop の 2 種類に対して本手法を適用し，実験計画法[12] の一部であるスクリーニング計画を実際に行えるかを確認した．

4.1 実験環境

実験は同一性能のブレードサーバを使用して行った．各サーバは，Intel Xeon 3.60GHz のデュアル CPU，2GB のメモリ，SCSI 接続の 36GB HDD で構成されており，全て 1000BASE-T で接続されている．ソフトウェアは CentOS 5.6 (Linux 2.6.18-238.9.1.el5xen), Java SE 1.6.0_24, Scala-2.9.0.1 を使用した．

Apache の実験では，これらのブレードサーバからウェブサーバに 1 台, バックグラウンド・エミュレータ (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．ウェブサーバには，Apache 2.2.3, PHP 5.3.3, Fast CGI 2.4.0, mod_fastcgi 2.4.6 を使用し，ベンチマークには 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 台のブレードサーバを割り当て，全て Hadoop-0.21.0 を使用した．

4.2 オブジェクト生成にかかるコード行数の比較

自動生成機構を使用し，アプリケーション対応言語オブジェクトが効率的に作成できることを示すため，自動生成できた行数，および手動で記述しなければならなかった行数を測定した．なお，言語オブジェクトには，以降で示すスクリーニング実験に必要な機能のみを実装することとした．これらのコード行数を比較した結果を表 1 に示す．各列は左から順にアプリケーション名，インターフェース定義に必要な IDL 記述行数，自動生成された Scala 言語オブジェクトのコード行数，自動生成部分に手動で追加したコード行数，および全体のコード行数とその中で自動生成できた割合を示している．

結果から，Apache と Besim に関しては全体のコード行数のうち 80%以上を自動生成できていることがわかる．一方，SPECwebClient については自動生成の割合が 40%未満である．これは，SPECwebClient は他のアプリケーションと異なり設定ファイルを持たず，起動や終了といった単純な操作のみを行うため，自動生成できる部分が少ないためだと考えられる．SPECwebClient 全体のコード行数を見ても 131 行と少なく，そもそもその実装のコストは低い．

表 1　言語オブジェクトのコード量比較

アプリケーション名	IDL 記述量	自動生成されたコード量	手動で追加したコード量	言語オブジェクト全体のコード量
Apache	33	384	81	465 (82.6%)
Besim	16	166	22	188 (88.3%)
SPECweb	40	318	255	573 (55.5%)
SPECwebClient	18	50	81	131 (38.2%)
Hadoop	52	409	233	642 (63.7%)

また，SPECweb（プライムクライアント）, Hadoop についても自動生成の割合が Apache や Besim に比べて低くなっている．これは，起動や終了といった単純な操作以外に，アプリケーションに固有の動作や実行結果の取得と解析を行う記述の割合が高いためと考えられる．しかしながら，IDL そのものの記述量とその記述から自動生成されたコードの行数を比較すると，約 8 倍のコードをそれぞれ生成できていることから，自動生成機構はこれらのアプリケーションの言語オブジェクト作成作業の効率化に貢献しているといえる．

4.3 スクリーニング計画への適用

本実験では，提案するスクリプティング環境で実際にパラメータ調整が可能であることを確かめる．実験計画法[12] は，予め計画的に実験を構成し，その計画に基づいて実験を行うことで，実験回数あたりの実験効果を最大化することを目的とした手法である．スクリーニング計画は，実験計画法の中でも多数のパラメータの中から性能に大きな効果のある少数のパラメータを統計的および効率的に見つけ出す手法である．本研究では，スクリーニング計画の作成および実験結果の ANOVA による解析に JMP 8[13] を使用した．

4.3.1 Apche のパラメータスクリーニング実験

本実験では，先行研究[14] を参考に Apache の性能に影響を与えると思われる 8 つパラメータについて $L16(2^8)$ 計画によるスクリーニング計画を適用し，各パラメータの性能への影響が確認できるか確かめた．また，実験に使用した SPECweb2005 は Banking，Ecommerce, Support の 3 つのワークロードを提供しており，今回はこれらの 3 種類のそれぞれに対してスクリーニング計画を適用した．実験では，SPECweb2005 スコア値を性能指標として，その値の最大化を目指している．各パラメータ値の組におけるスコア値は，応答時間が基準内に収まるときのクライアントの最大同時接続数をとした．

表 2 に調査を行ったパラメータを示す．今回は，2 水準で実験を行うため．各パラメータの低水準側と高水準側の 2 種類の値を示す．各パラメータの水準値は，パラメータ値として設定可能な範囲の中からおおまかに最小値と最大値となるような値を選択した．

表 2 Apache のスクリーニング対象パラメータ

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

図 4 に実験で使用したスクリプトを示す．本スクリプトでは，まず最初に Apache，Besim, SPECweb のプライムクライアントとクライアントに対応する分散オブジェクトを取得している．実験計画は `exprWith()` 関数に `design` として渡しており，それぞれのパラメータ値の組に対して `expr()` 関数で実験を進める．この関数の中では，再帰的な二分探索アルゴリズムによってスコアの最大値を求めている．図 4 の例から，パラメータ値の設定をあたかも変数への代入のように簡潔に記述できていることがわかる．また，アプリケーションに対して計算機を割り当てる操作や，実験結果のパターンマッチングといった操作も，対応するオブジェクトやそのリストを用いて高水準に記述できていることが示された，

このスクリプトを使用して行ったスクリーニング実験の結果を表 3 に示す．それぞれのワークロードに対して，5%有意であると判定されたパラメータが * 付きで示されている．Apache の `MaxClients` はどのワークロードに対しても効果があり，Ecommerce ワークロードにおいては，`KeepAliveTimeout` も効果があると判定されている．

4.3.2 Hadoop のパラメータスクリーニング

Hadoop では，先行研究[15] を参考に，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 sort, grep, wordcount を使用した．これらのプログラムは単純である反面,

```
val apacheConf = pms(0).appm.create("Apache")
val apache = pms(0).appm.add(apacheConf)
val besimConf = pms(1).appm.create("Besim")
val besim = pms(1).appm.add(besimConf)
val specwebConf = pms(2).appm.create("SPECweb")
val specweb = pms(2).appm.add(specwebConf)
val clients = pms.drop(3).take(8).map(_.appm.add("SPECwebClient"))

def exprWith(design: List[List[Param]]) = {
    def setParams(params: List[Param]) = params.foreach{
        p => p.key match {
            case "Apache.MaxClients" => apacheConf.maxClients = p.value
            case "Apache.KeepAliveTimeout" =>
                apacheConf.keepAliveTimeout = p.value
            ...
        }
    }
    def expr(smax: Int, smin: Int, result: Result): Result = {
        val ss =  (smax + smin) / 2
        apache.restart
        besim.restart
        clients.dforeach(_.restart)
        specwebConf.sessions = ss
        val r = specweb.add(specwebConf).start
        r.state match {
            case Success if (smax <= ss + 1) => r
            case Failed if(ss - 1 <= smin) => result
            case Success => expr(smax, ss + 1, r)
            case Failed => expr(ss - 1, smin, result)
            case Incompleted => expr(smax, smin, result) //retry
        }
    }
    design.map{ p =>
        setParams(p)
        expr(SESSION_MAX, SESSION_MIN, null)
    }
}
```

図 4 Apache パラメータスクリーニング実験のスクリプト例（抜粋）

表 3 Apache パラメータスクリーニングの実行結果

パラメータ	Banking		Ecommerce		Support	
	F	p	F	p	F	p
Apache.MaxClients	10.8793	0.0458∗	59.5136	0.0045∗	26.0824	0.0145∗
Apache.KeepAliveTimeout	4.8106	0.1159	26.2530	0.0144∗	7.9633	0.0666
Apache.Timeout	1.6312	0.2914	0.0560	0.8282	0.1901	0.6923
Apache.MaxRequestsPerChild	0.6709	0.4727	1.9017	0.2617	0.0038	0.9546
Apache.MCacheSize	0.4475	0.5514	0.8954	0.4138	0.4791	0.5386
Apache.AllowOverride	1.4890	0.3095	0.5801	0.5017	1.0060	0.3898
Apache.Modules	1.7047	0.2828	1.3855	0.3240	0.3601	0.5908
Apache.HostnameLookups	0.9494	0.4017	0.0767	0.7998	0.2912	0.6269
Apache.Logging	0.5536	0.5108	1.1149	0.3685	0.1163	0.7556
APC.shm_size	0.3406	0.6005	2.0808	0.2448	0.1368	0.7361
Besim.MaxClients	0.2534	0.6493	1.3451	0.3301	1.1493	0.3623
Besim.KeepAliveTimeout	0.1661	0.7109	2.5916	0.2058	0.5621	0.5079

実験結果から原因の分析が容易である．これらのプログラムに対し，同じサンプルプログラムである randomwriter と randomtextwriter によってそれぞれ生成したランダムなバイナリデータとランダムなテキストデータを入力として使用した．また，各ファイルのサイズはそれぞれ 10GB とした．Hadoop 実験での最適化基準は，実行時間の最小化である．

表 4 に調査の対象としたパラメータとその値の組を示す．これらパラメータは Hadoop の公式ドキュメント[16] などで効果があるとされているパラメータを取り上げている．

表 4 Apache のスクリーニング対象パラメータ

パラメータ	水準: 低	水準: 高
io.file.buffer.size	4,096	131,072
dfs.namenode.handler.count	10	40
mapreduce.jobtracker.handler.count	10	40
mapreduce.reduce.shuffle.parallelcopies	5	20
mapreduce.task.io.sort.factor	10	50
mapreduce.task.io.sort.mb	100	200
mapreduce.tasktracker.http.threads	40	80
mapred.tasktracker.map.tasks.maximum	1	4
mapred.tasktracker.reduce.tasks.maximum	1	4
mapreduce.{map|reduce}.java.opts	512MB	1,024MB

表 5 に Hadoop のスクリーニング実験の結果を示す．それぞれのワークロードに対して，5%有意であると判定されたパラメータを ∗ 付きで表す．mapred.tasktracker.map.tasks.maximum は，wordcount と sort の両ワークロードに対して効果があると判定された．それ以外に，wordcount では dfs.namenode.handler.count と mapreduce.task.io.sort.mb に効果があり，sort では mapreduce.{map|reduce}.java.opts に効果があると判定された．grep については，有意水準 5%では効果があると判定されたパラメータがなかったが，mapreduce.reduce.shuffle.parallelcopies や mapreduce.tasktracker.http.threads は 10%有意である．

5. 関 連 研 究

パラメータ調整は伝統的な研究分野であり，特に数値計算分野で広く研究が行われている．ATLAS[1], FFTW[2], OSKI[3] は行列演算のような数値計算に用いられる自動チューニングライブラリである．それぞれ，線形代数演算や離散フーリエ変換 (FFT) などに用いられ，ライブラリのインストール時やプログラムの実行時にチューニングが行われる．ABCLibScript[4] は，自動チューニングを機能をもつ数値演算ソフトウェア開発を効率的に行うために開発された言語である．ソフトウェアの開発者による ABCLibScript 言語の記述をもとに自動チューニングのためのコードを生成する点が本研究と類似するが，本研究とは対象とする領域が異なる．また，Kamil ら[5] は，マルチコア並列計算環境での自動調整フレームワークを提案している．このフレームワークでは，ドメイン特化言語の記述をもとにコンパイラが対象環境向けのソースコードとテスト用のコードを出力する．そして，チューニング用のエンジンを用いてテストを行い最適なパラメータを設定する．

インターネットサーバ分野でも，効率的にチューニングを行うため，ActiveHarmony[6] , Smart Hill-climbing[7] アルゴリズム, シンプレックス法[8] などを利用したチューニング手法が提案されている．これらの研究に対し本研究では，ベンチマークに必要な各要素を分散オブジェクトとすることで，任意のアルゴリズムによるパラメータ調整を容易にする．

6. まとめと今後の予定

本研究では，クラウド環境におけるサーバパラメータ調整のためのスクリプティング環境を提案した．本システムは，サーバアプリケーションやベンチマーククライアントなどのパラメータ調整に必要な全ての構成要素を分散オブジェクト化し，統一的な環境で，高水準なスクリプティングにより，パラメータ調整を行うことを可能としている．実験では，SPECweb2005 ベンチマークを使用した Apache ウェブサーバおよび Hadoop に対して本スクリプティング環境を適用し，パラメータ調整が可能であることを確認した．

表 5 Hadoop パラメータスクリーニングの実行結果

パラメータ	wordcount		sort		grep	
	F	p	F	p	F	p
io.file.buffer.size	3.3012	0.0750	1.0741	0.3067	1.4814	0.2313
dfs.namenode.handler.count	6.2607	0.0155∗	1.3795	0.2477	0.2469	0.6222
mapreduce.jobtracker.handler.count	0.4236	0.5180	3.1197	0.0856	0.0206	0.8865
mapreduce.reduce.shuffle.parallelcopies	0.3610	0.5506	0.1508	0.7000	3.0741	0.0879
mapreduce.task.io.sort.factor	1.1787	0.2826	1.9299	0.1731	0.3620	0.5511
mapreduce.task.io.sort.mb	7.0521	0.0105∗	0.0658	0.7900	2.5917	0.1159
mapreduce.tasktracker.http.threads	1.6593	0.2034	0.0639	0.8018	3.8047	0.0587
mapred.tasktracker.map.tasks.maximum	2565.927	$<$ .0001∗	4.3126	0.448∗	0.0033	0.9544
mapred.tasktracker.reduce.tasks.maximum	2.2717	0.1378	0.0600	0.8079	0.1231	0.7276
mapreduce.{map|reduce}.java.opts	1.8900	0.1751	142.0826	$<$ .001∗	0.0744	0.7866

今後の予定として，自動調整のためのアルゴリズムを関数型言語の性質を利用して高水準に記述しておき，さまざまなアプリケーションに対して適用できるライブラリとすることが考えられる．また，よりクラウドらしい性質を活用し，余剰計算機をクラウドから確保し，大規模に並列チューニングを行うことなどが考えられる．

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) R.C. Whaley, A. Pitiet, and J. Dongrra: Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, pp. 3-35, 2001.
2) M. Frigo and S.G. Johnson: FFTW: An Adaptive Software Architecture for the FFT, *In Proc. of ICASSP'98*, Vol. 3, pp. 1381-1384, 1998.
3) R. Vuduc, J.W. Demmel, and K.A. Yelick: OSKI: A Library of Automatically Tuned Sparse Matrix Kernels. *Journal of Physics: Conference Series*, Vol.16, No.1, pp. 521, 2005.
4) T. Katagiri, K. Kise, H. Honda, and T. Yuba: ABCLibScript: A Directive to Support Specification of an Auto-tuning Facility for Numerical Software, *Parallel Compututing*, Vol.32, pp. 92-112, 2006.
5) S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams: An Auto-Tuning Framework for Parallel Multicore Stencil Computations, *In Proc. of IEEE IPDPS'10*, pp.1-12, 2010.
6) I.-H. Chung and J.K. Hollingsworth: Automated Cluster-Based Web Service Performance Tuning, *In Proc. of IEEE HPDC'04*, pp. 36-44, 2004.
7) B. Xi, Z, Liu, M. Raghavachar, C.H. Xia, and L. Zhang: A Smart Hill-climbing Algorithm for Application Server Configuration, *In Proc. of WWW'04*, pp. 287-296, 2004.
8) W. Zheng, R. Bianchini, and T.D. Nguyen: Automatic Configuration of Internet Services, *In Proc of EuroSys'07*, Vol. 41, pp.219-229, 2007.
9) Martin Odersky. Scala. http://www.scala-lang.org/.
10) A. Sugiki, K. Kato, Y. Ishii, H. Taniguchi, and N. Hirooka: Kumoi: A High-Level Scripting Environment for Collective Virtual Machines, *In Proc. of IEEE ICPADS'10*, pp.322-329, 2010.
11) Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, Vol.14, No2, 1997.
12) 山田秀: 実験計画法 - 方法編 -, 日科技連, (2004)
13) SAS: JMP 8. http://www.jmp.com/.
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) 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).
16) Apache Foundation: Apache Hadoop Cluster Setup. http://hadoop.apache.org/common/docs/r0.21.0/cluster_setup.html