Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

JUnit向け単体テストを対象としたMapReduce型並列分散実行フレームワークの提案

和田祐介[†1]　大森洋一[†2]
日下部茂[†2]　荒木啓二郎[†2]

ソフトウェアの大規模，複雑化に伴い，開発過程のひとつである単体テストフェーズの負荷も高くなっている．一方，クラウドコンピューティング技術の進展により，計算機資源の新しい利用形態が考案されている．我々は，JUnit を用いる単体テストを計算機クラスタ上で並列分散実行するためのテスト実行フレームワークを提案する．フレームワークは Hadoop をベースとし，分散ファイルシステム及び MapReduce を応用する．そして，提案するフレームワーク上で JUnit 向けの単体テストを並列分散実行させる例を示す．

A Parallel Distributed Testing Framework based on MapReduce for Unit Test using JUnit

Yusuke Wada,[†1] Yoichi Omori,[†2] Shigeru Kusakabe[†2] and Keijiro Araki[†2]

As software systems are getting larger and more complicated, the load of the unit test phase that is one of the development processes is getting heavier. On the other hand, emerging cloud computing technologies are changing the form of computer resource usage. With two or more machines, we proposes an elastic parallel and distributed processing approach for unit test execution framework to JUnit. We use the distributed file system and MapReduce programming based on Hadoop. We also show the preliminary evaluation of our prototype the unit test using JUnit on the proposed framework.

1. はじめに

システムに求められる機能をソフトウェアによって実装することで柔軟に再現することが可能である．近年のシステムに求められる機能は増加の一途を辿り，追随する形でソフトウェアの大規模化，複雑化も加速している．開発者は限られた期間でソフトウェアを実装し，品質も確保しなければならない．開発プロセスのひとつである単体テストについて考えた場合，多数のソフトウェア要素で構成される大規模なシステムはテストケースの数も多く，不具合修正や仕様変更による再テスト（リグレッションテスト，回帰テスト）のコストも高くなる．よって，単体テストの効率化はソフトウェア開発の効率化に有用である．

単体テストを支援するフレームワークに JUnit があり，これは主として単体テストの自動化を目的する．クライアントはフレームワークに従ってテストケースを生成することで実行や結果の出力をフレームワークに一任することが可能となる[1)]．我々は，JUnit を用いる単体テストを計算機クラスタ上で並列分散実行するためのテスト実行フレームワークを提案する．

JUnit 向け単体テストを効率化する手段として，コンピュータクラスタ上での並列分散処理技術の利用を考えることが出来，実際にクラスタ上でテストを実行するための GridUnit という技術の提案もなされている[2)]．GridUnit は JUnit 向け単体テストを一度分散できる形式に変換して実行するのに対し，本研究ではテストケース及びその対象には手を加えずに並列分散実行するフレームワークを提案する．単体テストにおけるテストの対象ごとの明確な境界は定義されていないが，本研究ではファイルを境界とした単体テストを想定する．また，クラスタの構成およびフレームワークの実装は Hadoop をベースとする．Hadoop は分散ファイルシステムを実現する Hadoop Distributed File System と，並列分散処理を実現する Hadoop MapReduce から構成されるソフトウェアツールである．Hadoop を用いる利点は，MapReduce プログラミングによって並列分散処理を実現できることと[3)]，フレームワークに従ってクラスタを構築することにより参加する計算ノード数を容易に増減できることである．Hadoop Distributed File System と Hadoop MapReduce を応用・拡張することにより，単体テスト実行の効率化を図る．

†1 九州大学大学院システム情報科学府
Graduate School of Information Science and Electical Engineering, Kyushu University
†2 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

本研究の以降の章の構成は以下のとおりである．まず，第 2 章では JUnit を紹介し，第 3 章では Hadoop について述べる．第 4 章では提案するテスト実行フレームワークを解説し，第 5 章では提案するフレームワークの利用事例として JUnit に同梱されているサンプルを適用し，性能評価を行う．最後に第 6 章で提案するフレームワークの有用性および問題点について考察し，今後の課題を述べる．

2. テスティング・フレームワーク JUnit

単体テストを自動化するための枠組みをテスティング・フレームワークと言い，プログラミング言語ごとに実装したツールの総称を xUnit とよぶ．JUnit はその一つであり，他にも C++用の CppUnit，仕様記述言語 VDM に対応した VDMUnit などがある．

2.1 xUnit

テスティング・フレームワークを構成するコンポーネントは以下のようなものがある．

テストフィクスチャ

テストを行うために必要な状態および前提条件の集合．

テストスイート

同じテストフィクスチャを共有するテストの集合．

テストの実行

テストの実行は以下の順で行われる．

(1) テストフィクスチャの初期化を行う
(2) 設定したテストスイートに対してテストを行う
(3) テスト成功の如何にかかわず．テストフィクスチャのクリーンアップを行う

アサーション

テスト対象の関数やクラスに対する，振る舞いや状態を確認するための処理．

上記をまとめたものを単体テストに対するテストケースとよぶ．xUnit のテストケースにはこれらの内容を記述しなければならない．

2.2 JUnit

JUnit を使って Java プログラムに対し単体テストを行うためには，大きく分けて三つの部品が必要となる．

(1) ソースコード
テスト対象となるプログラム．Java における単体テストの場合，単一のクラスをテスト対象にすることが多い．

```
hadoop@ubuntu-vm:~$ java -classpath .:/home/hadoop/junit3/junit.jar junit.textui.TestRunner junittest.TestCalc
.F
Time: 0.004
There was 1 failure:
1) testPlus(junittest.TestCalc)junit.framework.AssertionFailedError: expected:<4> but was:<3>
        at junittest.TestCalc.testPlus(TestCalc.java:16)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

図 1 テキストベースでの JUnit テスト実行結果
Fig. 1 JUnit test result

(2) テストコード（テストケース）
テスト対象プログラムを実行させ，想定する結果と実際の結果が等しいか否かの判定を行うプログラム．

(3) JUnit ライブラリ
テストケースのコーディングおよびテストの実行を支援する．

JUnit テストの実行例を**図 1** に示す．この例では，JUnit ライブラリにある junit.textui.TestRunner クラスの引数にテストケースを指定することでテストを実行している．

3. Hadoop

Hadoop は，Google MapReduce[3] と Google File System[4] のフレームワークを，オープンソースで実装したソフトウェアツールである．本研究では Hadoop をベースとしてテスト実行フレームワークを実装する．使用したバージョンは 0.20.2 である．

3.1 Hadoop MapReduce

MapReduce は単一マシンで処理できないような大量のデータを複数のマシンを用いて分散処理するために Google が開発したフレームワークである．MapReduce フレームワークは，ひとつのジョブを Map 関数と Reduce 関数に対応づけることから，このように呼ばれている．Map 関数と Reduce 関数は，KeyValue ペアという形式のデータを対象とした処理を行う．KeyValue ペアは MapReduce が規定する基本的なデータ形式であり，フレームワーク内でのデータ授受を円滑に進めるために設定されたものである．ゆえに，Map 関数と Reduce 関数は，ユーザー任意の形式で実装するわけではなく，フレームワークが想定す

Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

る形式に則って実装しなければならない．Map 関数はジョブの対象である大量のデータを分解し KeyValue へ割り当て Map タスクとよばれるタスクを生成する．Reduce 関数では Map 関数が出力した KeyValue 情報の集約および Reduce タスクの生成を行い，最終的な計算結果を KeyValue 形式で出力する．

Hadoop MapReduce は上記 Google MapReduce を Java で実装したオープンソースクローンであり．Map 関数や Reduce 関数も Java で記述することができる．

3.2 Hadoop Distributed File System

Hadoop Distributed File System（以下 HDFS）は，MapReduce の背景と同様に単一のストレージで対応できないようなデータを対象とした分散ファイルシステムである．こちらは Google File System（以下 GFS）のオープンソースクローンであり，HDFS を利用してデータの分散格納が可能である．特徴としては，データ複製（レプリケーション）による耐障害性の高さやストレージ追加の容易さなどが挙げられる．

3.3 Hadoop クラスタ

Hadoop によって構成される並列分散環境を Hadoop クラスタとよぶことにする．Hadoop は合計 4 種のデーモンを起動し，デーモンによってマシンを 2 種類に区別する．一つはクラスタ全体の管理を担当するマシンでマスタとよび，もう一つはマスタの指示に従い処理を行うマシンでスレーブとよぶ．また，Hadoop MapReduce や HDFS に処理を要求するユーザーのことをクライアントとよぶ．マスタには JobTracker と NameNode という二つのデーモンが割り当てられ，スレーブには TaskTracker と DataNode という二つのデーモンが割り当てられる．これらの関係をまとめた図を**図 2** に示す[5]．

4. 単体テスト実行フレームワークの提案

本章では JUnit 向け単体テストの Hadoop ベース並列分散実行フレームワークを設計し適用する．フレームワークは Hadoop MapReduce プログラムとして Java で実装する．その概略図を**図 3** に示す．

4.1 設　　計

設計時特に考慮する点として以下を挙げる．

(1)　テストデータの分割
(2)　テストデータの参照
(3)　MapReduce ジョブへの入力
(4)　Map 関数でのテスト実行コマンド生成と結果の出力

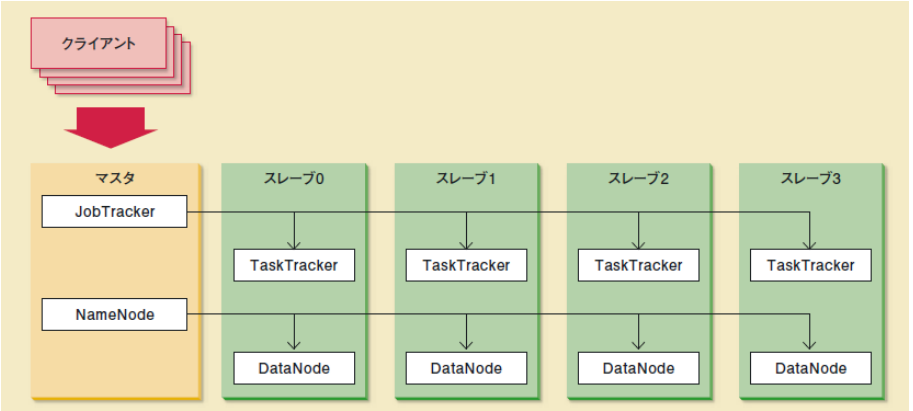

図 2　Hadoop の構成
Fig. 2　Hadoop Architecture

4.1.1 テストデータの分割

MapReduce フレームワークを用いて単体テストの並列分散実行を行うためには，テストデータを分割するための境界を設定する必要がある．しかし，1 節で述べたように単体テストの対象範囲は明確な定義が成されていない．よって，フレームワークは「単体テストの対象とする最小単位はファイルである」と想定する．つまり，Hadoop クラスタのスレーブにはそれぞれテストデータのファイルが分散されて行き渡るようにする．

4.1.2 テストデータの参照

テストデータはファイルごとに分割するように設定するが，それらは相互に依存するものを含んでいる可能性が高いため，Hadoop クラスタを構成するマシンのローカルディスクへ移動させるのは好ましくない．そのファイルが他のファイルを参照している場合，参照先のファイルを見つけることができなくなるからである．テストデータが参照先を損失しないためにデータを HDFS 上にまとめ，クラスタのマシンが HDFS 上のデータを参照する，という方法をとる．この方法ならば，テストデータが参照先を失うことなくクラスタのマシンは必要なファイルを読み込むことができる．

4.1.3 MapReduce ジョブへの入力

4.1.2 節で述べたように，実際にはファイルを分散させる必要はない．しかし，MapReduce ジョブは入力データを要求する．そこで，フレームワークは「テストデータのファイル名」

Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

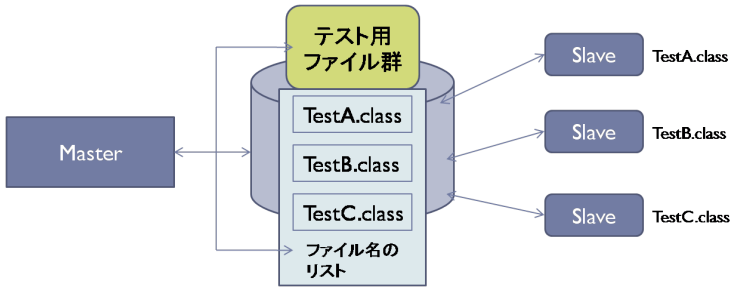

図 3 フレームワーク概略
Fig. 3 framework overview

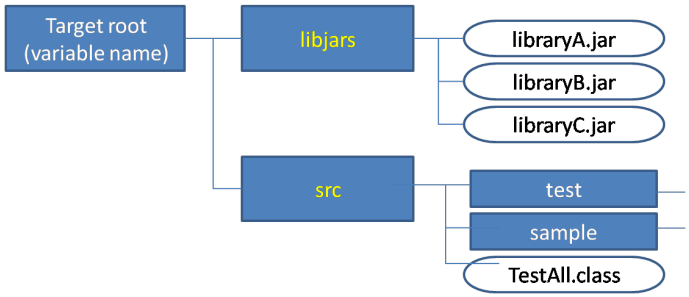

図 4 JUnit 用フレームワークのテストデータ保存先
Fig. 4 testdata directory tree for framework

を含むテキストデータ生成し，MapReuce の入力とする．これにより，クラスタ上のスレーブにはファイル名のテキストデータが行き渡ることになる．スレーブは，受け取ったファイル名のデータをもとに HDFS から目的のファイルを見つけ出し，読み込む．

4.1.4 Map 関数でのテスト実行コマンド生成と結果の出力

Map 関数では単体テストを実行し中間結果を出力する．このときファイルを対象としたテストを実行するためのコマンドを生成し，Java に処理させる．クライアントの関心は「テストに失敗したプログラム」であることが予想できるので，テストの結果は失敗したもののみを採用し Map 関数の出力とする．

4.2 JUnit 向けフレームワーク

JUnit を対象としたフレームワークを，4 章に従って実装した．このフレームワーク上ではクライアントは 3 つのステップを踏むことにより，JUnit 向け単体テストの並列分散実行が可能となる．

4.2.1 テストデータの保存

テストデータは Map 関数から直接参照および実行されるため，パッケージツリーとディレクトリ構造が一致した状態で HDFS へ保存されている必要がある．クライアントは**図 4** のような構造に従って実行に必要なライブラリおよびテストデータのクラスファイルを保存する．フレームワークは図 4 の libjars 下にあるファイルすべてと src にクラスパスを通す．

4.2.2 テストの実行

Hadoop の起動を確認し，テストデータの保存も完了したならば，テストを実行することができる．保存する際に図 4 の”Target root”と表記しているルートディレクトリを指定し，

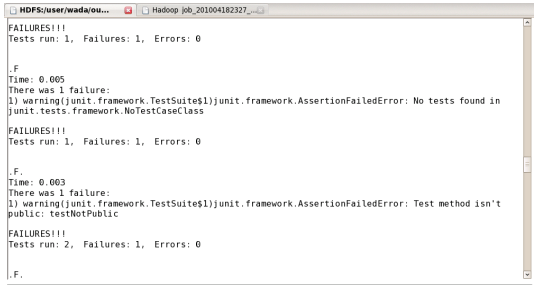

図 5 テスト結果のテキストデータ
Fig. 5 text data of test result

スクリプトを起動すると，JUnit テストの並列分散実行が開始する．フレームワークは指定されたディレクトリからファイル名の探索・フィルタリングを行い，テキストデータを生成して MapReduce への入力とする．テキストデータはスレーブへ分散され，ファイル名を受け取ったスレーブはテストを実行する Java コマンドを生成・実行する．

4.2.3 テスト結果の取得

クライアントはテスト結果を HDFS に出力されたテキストデータで確認することができる．テキストデータには失敗したテストに関する情報が含まれる．テスト結果の例を**図 5** に示す．

Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

5. 性能評価

本章では，JUnit に同梱されているサンプルを 4.2 節で実装したフレームワークへ適用し，性能評価を行う．

5.1 評価環境

評価環境として 10 台のマシンを用意した．そのうち 2 台をマスタノードとしそれぞれで NameNode と JobTracker を動作させ，残り 8 台をスレーブノードとして DataNode と TaskTracker を動作させた．各ノードのスペックを**表 1** に示す．この Hadoop クラスタ上でサンプルを並列分散実行し，MapReduce プログラムの開始から終了までの時間を計測する．サンプルを複製し合計 8 パターンのターゲットを用意し，それぞれフレームワークへ適用する（**表 2**）．

5.2 実行時間

実行時間の結果を**図 6** および**表 3** に示す．また，台数効果の一つの指標として，

（単一マシン上での実行時間）/（Hadoop クラスタ上での実行時間） (1)

という値を算出し，**図 7**，**表 4** に示す．有効数字は小数点以下三桁までを採用する．以後この値を Hadoop クラスタ全体の処理能力として考える．値が 1 より大きければ，単一マシン上よりも高速にテストを行っているということであり，より大きいほどクラスタの能力が高いことを意味する．

表 1 Hadoop クラスタを構成するマシンのスペック
Table 1 node spec

	namenode	jobtracker	スレーブ
CPU⋆1	XeonE5420	XeonE5420	XeonX3320
Memory	3.2GB	8.0GB	3.2GB
Disk	2TB	1TB	140GB
NIC⋆2	1Gbps	1Gbps	1Gbps

表 2 実験対象となる JUnit テストターゲット
Table 2 test targets

Target ID	1	2	3	4	5	6	7	8
サイズ [MB]	0.3	3.3	6.5	12.8	25.5	48.3	93	154.5
テストファイル数	37	592	1184	2368	4736	8991	17316	28736

サイズ：テストデータを含むディレクトリの合計サイズ
テストファイル数：フレームワークが実行可能と判断したテストファイルの数

⋆1 Intel(R) Xeon(R) CPU E5420 @ 2.50GHz Quad Core
Intel(R) Xeon(R) CPU X3320 @ 2.50GHz Quad Core
⋆2 すべて BroadcomBCM95721 1000Base-T Ethernet

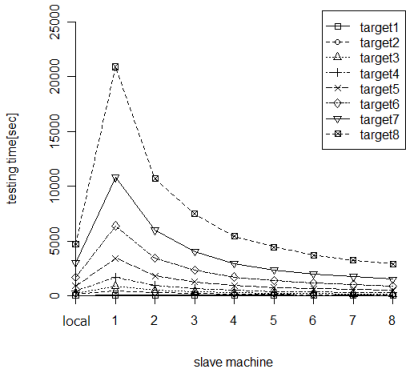

図 6 実行時間
Fig. 6 test executing time
横軸：スレーブ数，縦軸：テストの実行時間 [秒]

表 3 実行時間 [秒]
Table 3 test executing time

	0	1	2	3	4	5	6	7	8
target1	6	47	34	32	32	32	29	33	24
target2	110	446	263	178	140	116	99	93	81
target3	220	868	469	356	249	227	177	155	143
target4	445	1697	913	651	468	385	329	290	249
target5	890	3394	1795	1266	929	748	630	549	491
target6	1689	6388	3447	2303	1708	1384	1158	1020	896
target7	3008	10802	5981	4040	2933	2366	1999	1737	1497
target8	4719	20899	10691	7457	5432	4436	3700	3234	2893

Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

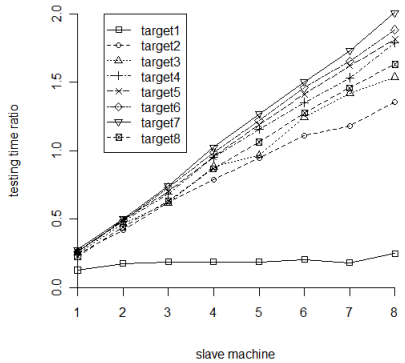

図 7 実行時間の比
Fig. 7 test executing time speedup
横軸：スレーブ数，縦軸：実行時間の比（式 1）

表 4 実行時間の比
Table 4 test executing time speedup

	1	2	3	4	5	6	7	8
target1	0.128	0.176	0.188	0.188	0.188	0.207	0.182	0.250
target2	0.247	0.418	0.618	0.786	0.948	1.111	1.183	1.358
target3	0.253	0.469	0.618	0.884	0.969	1.243	1.419	1.538
target4	0.262	0.487	0.684	0.951	1.156	1.353	1.534	1.787
target5	0.262	0.496	0.703	0.958	1.190	1.413	1.621	1.813
target6	0.264	0.490	0.733	0.989	1.220	1.459	1.656	1.885
target7	0.278	0.503	0.745	1.026	1.271	1.505	1.732	2.009
target8	0.226	0.441	0.633	0.869	1.064	1.275	1.459	1.631

5.3 考　　察

5.3.1 台 数 効 果

スレーブの台数に関する値の違いに注目すると，明らかにスレーブ数が多いほど高速にテストを実行していることがわかる（図 7）．よって，TaskTracker が動作するスレーブの数を増やせば，テストの実行も高速になるということがいえる．

5.3.2 スケールアウト

Hadoop クラスタのスレーブ 1 台の環境では単一マシン上よりも JUnit テストの実行が遅い（図 7）．理由として，MapReduce フレームワークを通してテストを実行する場合，単一マシン上でテストを実行するときに比べテスト実行そのもの以外の処理を行うことが挙げられる．テスト実行そのもの以外の処理とは，JobTracker がファイル名のデータから KeyValue ペアを生成する処理や，マスタとスレーブの通信処理などである．．ただし，スレーブの数を増やすとその分並列分散実行能力が上昇するので，HDFS や MapReduce のオーバーヘッドが影響しても単一マシン上より高速に実行する可能性が高くなる．今回の実験ではスレーブ数 4,5 付近で Hadoop クラスタが単一マシンの能力を超えはじめ，スレーブ数 8 では target1 を除くすべてのテストに対し単一マシンよりも高速に実行していることがわかる（図 7）．

5.3.3 テスト数に依存するオーバーヘッド

対象とするテストの数をを target7 から target8（表 2）へ増やしたとき，クラスタの能力が減少していることがわかる（図 7，表 4）．すべてのスレーブ数のときに同じことが言えるため，テスト数に依存した何らかのオーバーヘッドが存在すると考える．よりテスト数を増やすなどして，原因を特定する必要がある．

6. お わ り に

6.1 ま　と　め

本研究では JUnit 向け単体テストを効率的に実行するための並列分散実行フレームワークを提案した．提案したフレームワークでは HDFS 上に保存されたテストデータから MapReduce への入力データを生成し，Map 関数でテストを実行させるようにした．また，実際に JUnit に同梱されているサンプルを用いての動作確認および性能評価を行った．サイズの異なる 8 つのテストを用意し，それぞれについて最大 8 台のスレーブを用意して実行時間の計測を行った．その結果，マシンの数を増やせばテスト高速に実行できること，単一マシンでの実行よりも高速に実行できる可能性が高くなることを示した．

6.2 今後の課題および方針

JUnit 向け単体テストを並列分散実行することによって台数効果およびスケールアウトの可能性を示した一方で，テスト数の増加に起因するオーバーヘッドの存在が認められた．また，フレームワークの実装と事例の適用は JUnit のみにとどまっており，単体テストとしての一般性に欠ける．さらに，Hadoop のコンフィグレーションについて，テスト対象に最適

Vol.2010-SE-168 No.5
Vol.2010-EMB-17 No.5
2010/6/1

なパラメータをチューニングしているとは言えない．今後はまず，テスト数およびスレーブ数を増やして評価を行いオーバーヘッドの原因を特定する．また，JUnit 以外の xUnit に対してフレームワークを実装し，提案する設計手法が xUnit に対して一般的となるよう拡張する．さらに，並列分散処理の対象に応じて適切なパラメータのチューニングを行えるような手法の確率も目指す．

参 考 文 献

1) ：テスティングフレームワーク　JUnit, http://www.alles.or.jp/˜torutk/oojava/maneuver/2000/6-3.html.
2) Duate, A., Cirne, W., Brasileiro, F. and Machado, P.: GridUnit: Software Testing on the Grid, *Proceedings of the 28th international conference on Software engineering*, pp.779–782 (2006).
3) Oram, A. and Wilson, G.(eds.): MapReduce での分散プログラミング，ビューティフルコード，Vol.1, No.1, chapter23, pp.389–403, オライリー・ジャパン (2008).
4) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *ACM SIGOPS Operating Systems Review*, pp.29–43 (2003).
5) 藤田昭人（編）：*Hadoop/MapReduce*, Unix Magazine, pp.58–65, 角川グループパブリッシング (2009).