修士学位論文

題目

Madoop: WebAssembly を用いた Web ブラウザベース分散処理フレームワーク

指導教員

楠本 真二 教授

報告者

松尾 裕幸

平成 31 年 2 月 6 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻

平成 30 年度 修士学位論文

Madoop: WebAssembly を用いた Web ブラウザベース分散処理フレームワーク

松尾 裕幸

内容梗概

自由参加型の分散処理パラダイムである従来の VC: Volunteer Computing を Web ブラウザ上に拡張した，BBVC: Browser-Based Volunteer Computing というアイディアが登場している．BBVC では，コードを埋め込んだページにブラウザからアクセスするだけで参加可能という手軽さに加え，世界中のインターネットユーザが対象になるという凄まじい潜在的処理能力が存在する．しかしながら，BBVC には幾つかの課題が存在する．課題の一つ目に分散処理プログラムの開発容易性の低さが挙げられる．分散処理の実現には，入力データの適切な配信やクライアント間の同期など様々な事柄を考慮しなければならず，開発容易性を下げる一因となっている．課題の二つ目に実行時性能の低さが挙げられる．BBVC ではその特性上 JavaScript で処理を行うが，JavaScript は実行時にソースコード解析を必要とするため，コンパイラ型言語と比べて性能が低く数値計算には向いていない．本稿では，これらの課題の解決を目的とした新しい BBVC フレームワーク Madoop を提案する．Madoop では MapReduce と WebAssembly を用いることで，開発容易性の確保と実行時性能の改善を実現する．また評価実験として，提案手法を用いて実際に分散処理を実行し，実行時間の長さを計測・比較することで，従来手法からの性能の改善度合いを確かめる．実験の結果，最も性能が向上した実験対象では，実行時間の長さが 60 % 以上改善した．さらに，Madoop のより実践的なユースケースを示すため，PDG（プログラム依存グラフ）を用いたコードクローンの検出を適用実験として行った．

主な用語

Volunteer computing，web browser，MapReduce，WebAssembly，JavaScript．

目次

図目次

表目次

1　まえがき

技術の発展に伴い，個人が所有する計算機の処理性能は飛躍的に向上した．また，世界中で稼働している計算機の数も非常に多く，控えめに見積もった場合でも現在 1 億台以上の計算機が使用されている [1]．世界中で稼働している全計算機を束ねた総処理能力は凄まじいが，一般的な計算機はその起動時間の多くがアイドル状態であり，その余剰処理能力の殆どが無駄になっている [1]．

科学技術の分野において，この余剰処理能力に着目した分散処理手法が提案されている．これを Volunteer Computing（以下，VC）という．近年，この VC を Web ブラウザ上で実行するように拡張した Browser-Based Volunteer Computing（以下，BBVC）という手法も登場している．従来，VC に参加するためには特別なクライアントアプリケーションのインストールが必要不可欠であった．一方，BBVC はユーザにクライアントアプリケーションのインストールを要求せず，多くのコンピュータに予めインストールされている Web ブラウザで特定の URL にアクセスするだけで動作させることができる．従って，BBVC は世界中のネットサーフィンユーザが対象であり，そこには莫大な潜在的処理能力が存在する [2]．

しかしながら，BBVC には幾つかの課題が存在する．一つ目の課題として，分散処理プログラムそのものの開発容易性の低さが挙げられる．一般に分散処理を行うためには，入力データの適切な配信や，クライアントノード間の同期，計算結果の統合などを行う仕組みが必要となる [1, 2]．開発者は実際に処理対象となるプログラムに加えてこれらの仕組みを用意する必要があり，その実装は開発者にとって非常に大きな負担となる．二つ目の課題に，処理言語である JavaScript の実行時性能の低さが挙げられる．JavaScript は実行時に動的なソースコードの解析を必要とするため，これがオーバヘッドとなりコンパイル型言語と比べてパフォーマンスに劣る．とりわけ，JavaScript は数値計算には向いていないことが指摘されている [3]．

本稿では，これらの課題を解決するための新しい BBVC フレームワーク Madoop を提案する．Madoop では，一つ目の課題の解決のために MapReduce を，二つ目の課題の解決のために WebAssembly をそれぞれ導入する．MapReduce [4] は分散処理パラダイムの一つであり，このパラダイムに沿ってプログラム開発を行うことで，開発者の負担が大きく削減される．WebAssembly [5] は Web ブラウザ上で実行可能なバイナリフォーマットであり，JavaScript が必要とした実行時解析によるオーバヘッドが大きく軽減されるため，パフォーマンスの向上が期待できる．

評価実験として，Madoop が従来手法と比べてどの程度パフォーマンスが向上しているのかを，計算量の大きさが異なる 2 つの実験対象を用いて，それぞれの実行時間の長さを計測することで確かめた．実験の結果，計算量が非常に大きい実験対象では，Madoop を用いた提案手法は従来手法よりも約 50 〜 64 % 以上パフォーマンスが向上した．一方，計算量が小さい実験対象では，提案手法が従来手法

よりも約 4 〜 8 % 遅くなった．この実験結果から，提案手法は計算量が大きな分散処理を行う際に非常に有効であることが分かった．

さらに，適用実験として，より実践的なアプリケーションを Madoop にどう適用させるかを示した．本稿では，大きな時間計算量が必要かつ大量のデータを処理する必要がある，PDG を用いたコードクローン検出を題材に選んだ．

また，Madoop はオープンソースソフトウェアとして公開しており，第三者が自由に利用できる．現在 BBVC の処理系として公開されているソフトウェアは殆ど存在しておらず，BBVC プログラムを開発したい開発者にとって非常に有用であると考える．

2 準備

本章では，本稿で一貫して言及している 2 つの重要なアイディア・技術と，それらが抱える現状の課題について述べる．

2.1 VC: Volunteer Computing

VC は分散処理を行うための手法の一つである．現在広く用いられている分散処理システムでは，同じスペックの計算機を複数台用意し，それらを LAN で相互接続した上でクラスタを組むのが一般的である．これに対して，VC では計算機やそれらの構成が大きく異なる．VC における計算機（ノード）はインターネットに接続された不特定多数の様々な計算機である．一般的な計算機はその起動時間の多くがアイドル状態であり，この際の余剰計算能力をユーザ（ボランティア）から提供してもらうことでリソースを確保する．また，VC では各ノードが物理的に離れており，LAN ではなくインターネット越しに相互接続してシステムを構成する．

VC のアイディアは古くから存在しており，実際に VC を大規模活用した有名な例として SETI@home [6] が挙げられる．VC を用いて宇宙から飛来した電波を解析し，地球外知的生命体が発信した信号を検出する試みである．SETI@home ではこれまでに 170 万人以上が計算に参加しており，2018 年 8 月現在でも常時 9 万人以上が参加している [7]．これは VC が数値計算プラットフォームとして有用であることを示した代表例である．

VC の問題点として，ユーザ層が限定的であることが挙げられる．ここ数年では，VC 全体のユーザ数は増加することなく約 50 万人を横這いで推移している．これはインターネットに接続された全デバイス数の 0.02 % を下回っており，ユーザ数の増加は頭打ちになっていると言える [2]．VC に参加するためには，ユーザは専用のクライアントアプリケーションをインストールする必要があり，ユーザに対するハードルとなっていると考えられる．また，複数のオペレーティングシステムの台頭やタブレット・スマートフォンの普及など，ユーザの計算環境はますます多様化している．VC では各ユーザの環境に合わせたクライアントアプリケーションを開発する必要があり，開発者に対するハードルとなっていると考えられる [8]．

2.2 BBVC: Browser-Based Volunteer Computing

VC を Web ブラウザ上に拡張したのが BBVC である．VC が専用のクライアントアプリケーション上で動作していたのに対して，BBVC ではユーザが Web ブラウザで特定の URL にアクセスしている間に，その Web ブラウザ上で処理が行われる．

一般的に，ユーザが利用する計算機には予め Web ブラウザがインストールされていることが多いた

め，BBVC に参加するために特別なアプリケーションをインストールする必要はない．よって，BBVC は世界中のネットサーフィンユーザが対象となり，その潜在的な計算能力は測り知れない．例えば，YouTube にアクセスしている各ユーザの Web ブラウザ上で，計算機の 25 % の処理能力を BBVC に利用できると仮定した場合，その総処理能力は 46.4 PFLOPS にも上ると試算されている [2]．2017 年 4 月時点での世界最速のスーパーコンピュータである Sunway TaihuLight の処理能力が 93 PFLOPS であることを考慮すると，YouTube への訪問者を束ねるだけでもスーパーコンピュータの約半分の処理能力に値すると言える [2].

また，開発者はクライアントアプリケーションを Web アプリケーションとして開発すればよく，計算機のオペレーティングシステムやその他のアーキテクチャの多くは Web ブラウザが隠ぺいするため，VC に比べると多様化するユーザの計算環境に対応するコストも低くなる．このことから，BBVC では前節で挙げたような VC の課題を解決できると考えられる．

2.3 BBVC の課題

前節で述べたように，BBVC は VC の課題のいくつかを解決した一方で，以下に挙げる BBVC 固有の課題も存在する．

2.3.1 P_1: 分散処理プログラムの開発容易性の低さ

分散処理を実現するためには，処理対象データの適切な分割・配信や，クライアントノード間の同期，処理結果の統合，処理失敗時の復帰を始めとする，拡張性や可用性，耐障害性などを確保するための複雑な仕組みを要する [1, 2]．これらは分散処理そのものを実現するために必要な仕組みであるが，その実装は開発者にとって非常に大きな負担となる．分散処理の実装を支援する技術として，次章で述べる MapReduce パラダイムが有名だが，Web ブラウザ上で行う分散処理のために利用した例は少なく，またライブラリとして利用できる形に公開されているものも少ない．結果として，BBVC を行うためのプログラムを開発者が実装する際，現状では利用できる分散処理フレームワークの選択肢がないため，前述したような考慮事項を念頭に置きながら自分で開発しなければならない．これは開発者にとって大きな負担であり，BBVC の普及を妨げる要因となっていると言える．

2.3.2 P_2: JavaScript の実行時性能の低さ

BBVC では，Web ブラウザ上で処理を行うという性質上，これまで処理言語としては実質的に JavaScript 以外の選択肢が存在しなかった．しかしながら，JavaScript は動的型付けのスクリプト言語であり，静的型付けのコンパイル言語と比べると，実行時解析のオーバヘッドのためにパフォーマンスが劣る．2008 年に Google Chrome の JavaScript エンジンである V8 に Just-In-Time（JIT）コン

パイラが導入されてからパフォーマンスは改善されてきているものの，依然としてコンパイル言語と比べると処理速度は遅いと言える．例えば，数値計算タスクの実行速度を Java と JavaScript とで比べた場合，JavaScript の速度は Java よりも約 30 % 遅いことが実験から指摘されている [3].

3 提案手法: Madoop

3.1 概要

本稿では，前節で挙げた BBVC の課題の解決を目的とした新しい BBVC フレームワーク Madoop[*1] を提案する．本提案手法では，課題 P_1 の解決に MapReduce を，課題 P_2 の解決に WebAssembly をそれぞれ導入する．これにより，開発容易性を確保しつつ，クライアントノード上での実行時性能が向上することが期待できる．

3.2 採用技術

3.2.1 MapReduce

Madoop では，2.3.1 項で述べた課題 P_1: 分散処理プログラムの開発容易性の低さを解決するため，MapReduce を導入する．

MapReduce [9, 4] は Google により提起された分散処理パラダイムの一つである．MapReduce は分散処理における一連の操作を map と reduce の 2 つの関数に抽象化している．各ノードは入力データと map/reduce 関数を受け取り，それぞれ処理する．処理の概要は次の通りである．

1. MapReduce の処理系が入力データを適切な単位に分割する．
2. 各ノードが入力データと map 関数を受け取る．map 関数は中間値である (key, value) のペアの集合を出力する．
3. MapReduce の処理系が各 key に対して，同一の key を持つすべての value を集め，中間値である (key, value 列) のペアの集合を作成する．
4. 各ノードが (key, value 列) のペアと reduce 関数を受け取る．reduce 関数は value 列を集約し，その結果を出力する．

開発者が実際に開発するのは map/reduce 関数のみであり，2.3.1 項で挙げた分散処理に伴う諸考慮事項は処理系に任せればよい．これにより，MapReduce は分散処理プログラムの開発を容易にしたと言える．また MapReduce のアイディアは有名であるため，Madoop がこれに対応することで，これまで MapReduce を用いて分散処理プログラムを開発してきた開発者は，低い学習コストで Madoop を使い始めることができる．

MapReduce の実装としては Apache Hadoop [10] が有名だが，Hadoop は Java で記述されており，

[*1] 著者名の **M**atsuo と MapReduce の実装で有名な Apache H**adoop** から命名した．Madoop はオープンソースソフトウェアとして公開している．
`https://github.com/h-matsuo/madoop/`

表 1　WebAssembly と Java の比較.

項目	WebAssembly	Java
プログラミング言語	C/C++ など	Java
コンパイラ	Emscripten など	`javac`
コンパイラによる生成物	`.wasm` ファイル	`.class` ファイル
実行環境	Web ブラウザ	JVM

そのまま Web ブラウザ上で動作させることはできない．また JavaScript で記述された MapReduce 実装の例としては MRJS [11] や JSMapReduce [12] などが挙げられるが，いずれも実装を公開しておらず，第三者が利用することができない．Madoop はオープンソースソフトウェアとして公開しているため，第三者が利用することができ，BBVC を行いたい開発者にとって有用であると考える．

3.2.2　WebAssembly

2.3.2 項で述べた課題 P_2: JavaScript の実行時性能の低さを解決するため，WebAssembly を導入する．

WebAssembly [5] は Web ブラウザ上で実行可能なバイナリフォーマットである．厳密にはスタックベースの仮想マシン用バイナリフォーマットとして設計されており，この仮想マシンが各種 Web ブラウザに搭載されていると考えればよい．WebAssembly 形式へのコンパイルにはいくつかの言語が対応しており，主要なものでは C/C++ が挙げられる．2018 年 8 月現在，主要なモダンブラウザはすべて WebAssembly を実行可能であり[*2]，全ユーザが利用するブラウザの約 75 % がサポートしている [13] ため，多くのユーザの環境で利用できると考えてよい．

WebAssembly は Java に置き換えて考えると分かりやすい．表 1 は WebAssembly と Java を比較したものである．Java では，記述したプログラムを `javac` でコンパイルすると，中間言語で記述されたクラスファイル（`.class`）に変換される．コンパイル済みのクラスファイルを実行するのは仮想マシン JVM（Java Virtual Machine）であり，JVM を様々な環境に合わせて用意することで，同じ Java プログラムを多様な環境で動作させることができる．同様に，WebAssembly では，C/C++ などで記述されたプログラムを Emscripten [14] と呼ばれるツールなどを用いてコンパイルすることで，中間形式であるバイナリフォーマット（`.wasm`）が得られる．このバイナリフォーマットは Java のクラスファイルに相当する．変換後のバイナリフォーマットは各種 Web ブラウザが実行する．ここでは，Web ブラウザが Java の JVM に相当する．このように，バイナリフォーマットを Web ブラウザが直接解釈す

[*2] Microsoft Edge, Mozilla Firefox, Google Chrome, Safari, Opera のすべてがサポートしている．また，モバイルブラウザでも iOS Safari, Chrome Android の両方がサポートしている．

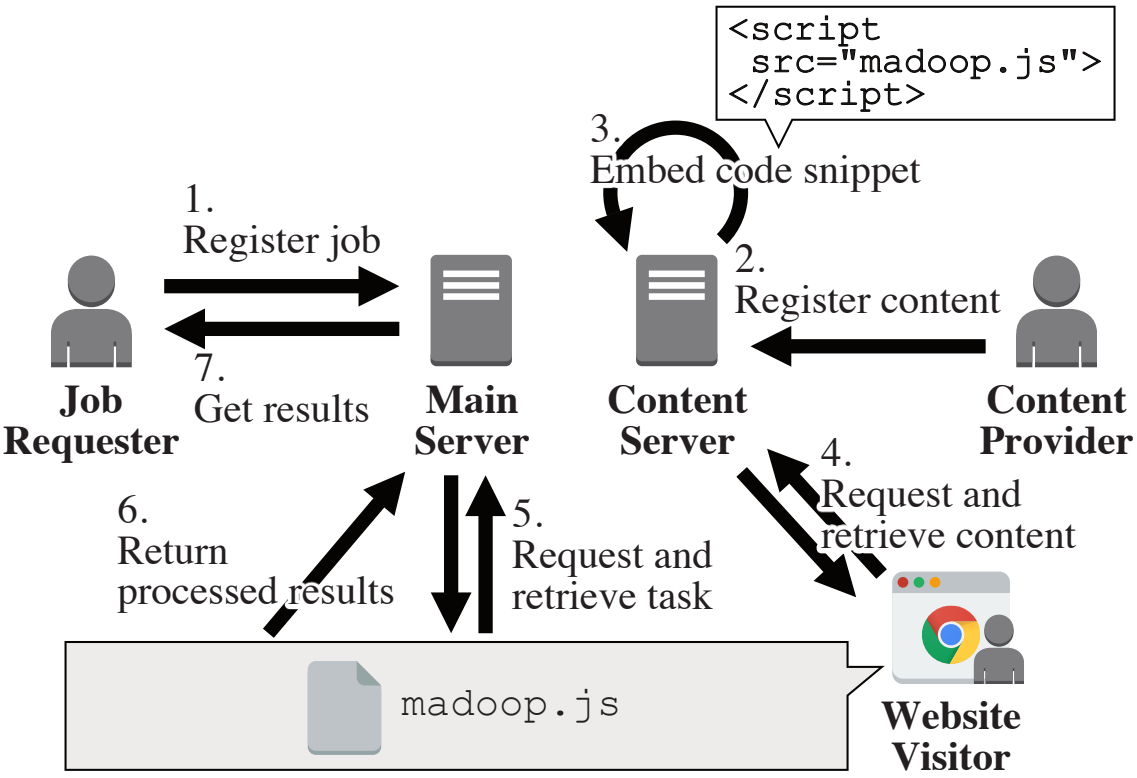

図 1　Madoop のアーキテクチャ．

るため，C/C++ プログラムを様々な環境に合わせてコンパイルし直す必要がない．WebAssembly 向けに一度コンパイルすれば，多様な環境の Web ブラウザ上で実行させることができる．

WebAssembly を導入することの最大の利点は，パフォーマンスの改善にある．前述したように，WebAssembly を用いることで Web ブラウザはコンパイル済みのバイナリを解釈・実行すればよいため，コンパイラ型言語と同等の性能を期待することができる．実際，同じ内容の数値計算をネイティヴアプリケーションとして実装したものと，WebAssembly を用いて実装したものを実行時間の長さで比較した場合，WebAssembly はネイティヴアプリケーションと遜色ないパフォーマンスを発揮したというベンチマーク結果もある [15]．

3.3　アーキテクチャ

Madoop のアーキテクチャを図 1 に示す．図には，太字で示した 3 種類のアクタ，および 2 種類のサーバが登場する．

依頼者　Madoop 上で分散処理を行いたい者．

コンテンツ投稿者　ブログ記事などの Web コンテンツを投稿する者．

Web サイト訪問者　Web ブラウザを用いて，コンテンツ投稿者が投稿した Web コンテンツにアクセスする者．

メインサーバ　Madoop がインストール・セットアップされたサーバ．

コンテンツサーバ　コンテンツ投稿者が投稿した Web コンテンツを保持・配信するサーバ．

Madoop を用いて分散処理のジョブを登録してから処理結果が返却されるまでの一連の流れを説明する．なお，リストの番号は図中の番号と対応している．

1. 依頼者が Madoop にジョブ（map/reduce 関数および処理対象データ）を登録する．このとき，map/reduce 関数は C/C++ で記述し WebAssembly 形式にコンパイルしたものを登録することで，高パフォーマンスでの処理が期待できる[*3]．Madoop は登録されたジョブを適切な単位に分割し，タスクとして保持する．
2. コンテンツ投稿者がブログ記事などの Web コンテンツを作成し，コンテンツサーバに投稿する．
3. コンテンツサーバ上に保持されている Web コンテンツに，Madoop を用いた分散処理を行うためのコードスニペットが埋め込まれる．
4. Web サイト訪問者が Web ブラウザでコンテンツサーバにアクセスし，HTML ファイルを始めとするコンテンツの閲覧に必要な各種データをダウンロードする．
5. 3 で埋め込まれたコードスニペットにより分散処理が始まる．以降この Web ブラウザは，アクセスしたページに留まっている間はクライアントノードとして機能する．
6. クライアントノードがメインサーバから処理対象タスク（map/reduce 関数および入力データ）を取得する．
7. クライアントノード上でタスクを実行する．
8. タスクの実行結果をクライアントノードから Madoop のメインサーバへ返却する．ページから離れるか，処理すべきタスクがなくなるまで (5)〜(8) を繰り返す．
9. すべてのタスクが完了すると，依頼者は結果を取得できる．

上記 1 で登録される map/reduce 関数の例をそれぞれ図 2 及び図 3 に示す．ここでは，ジョブとしてワードカウントを想定し，それぞれの関数を C++ で記述している．図 2 の map 関数は入力として文字列を受け取り，文字列に含まれる単語とその出現回数を出力する．4 行目で与えられた文字列をスペースで区切り，6 行目で各単語の出現回数をカウントする．その後 14 行目で，各単語について $(\text{key}, \text{value}) = (\text{単語}, \text{出現回数})$ のペアを出力する．図 3 の reduce 関数では，入力として単語と出現回数のリストを受け取り，出現回数を足し合わせて単語と共に出力する．7 行目で与えられた出現回数のリストを足し合わせ，11 行目で $(\text{単語}, \text{出現回数})$ のペアを出力する．このように，開発者は

[*3] 従来手法通り，JavaScript で記述した map/reduce 関数も登録できるように設計している．

```
void map(const char* data) {
  // Count the number of occurrences of each word
  std::unordered_map<std::string, long> hashmap;
  const std::vector<std::string> &words =
    split(data, '␣');
  for (const auto &element: words) {
    hashmap[element]++;
  }
  // Emit key-value pairs
  for (const auto &element: hashmap) {
    const std::string &key = element.first;
    const std::string &value =
      std::to_string(element.second);
    emit(key, value);
  }
}
```

図 2　ワードカウント用 map 関数の例（C++）.

```
void reduce(const char* key,
            const char* values_list) {
  // Sum up values in the list
  const std::vector<long> &values =
    split(values_list, ',');
  long sum = 0;
  for (const auto value: values) {
    sum += value;
  }
  // Emit key-value pair
  emit(key, std::to_string(sum));
}
```

図 3　ワードカウント用 reduce 関数の例（C++）.

MapReduce パラダイムに沿って処理の本質となる部分のみを実装すればよく，それ以外の分散処理に必要な諸処理は Madoop が行うため，システム全体をフルスクラッチで実装する場合と比べて容易に開発することができる.

3.4 実装

Madoop のメインサーバは Node.js[*4] アプリケーションとして実装した．使用した言語は TypeScript[*5] である．また開発期間は約 8 ヶ月，コード行数は約 1,100 行である．

3.5 Madoop のシナリオ

Madoop を用いて計算量の大きい問題を処理するとき，鍵となるのは「大量のボランティア（ワーカー）をいかに確保するか」である．この節では，Madoop のワーカーを確保するための 3 つのシナリオを述べる．

3.5.1 ウェブサイトへの訪問者を使う

3.3 節でも述べたように，Madoop のワーカーとなるのに必要なのは，モダンな Web ブラウザだけである．従って，もし自分のウェブサイトを運営していれば，そのウェブサイトに Madoop の JavaScript スニペットを埋め込むことで，ウェブサイトへのすべての訪問者を Madoop のワーカーとして利用することができる．この可搬性は，Madoop が Google AdSense[*6] のようなオンライン広告プラットフォームの代替の成り得る可能性を秘めていることを示している．

今日，多くのウェブサイト運営者は，サイト上のコンテンツから利益を得るために広告を配置している．しかしながら，これらの広告はウェブサイトの訪問者を敬遠させてしまう可能性がある．Goldstein らは，動くアニメーション広告がユーザをいらいらさせていると指摘している [16]．Madoop や伝統的なオンライン広告には，バッテリー消費を含むウェブサイト訪問者の計算リソースを借りるか，ウェブサイト訪問者の UI/UX を犠牲にするかというトレードオフの関係がある．

また，ウェブサイト訪問者がワーカーになることをオプトアウトできる仕組みを提供する必要がある．この種の倫理的問題は BBVC の分野で長年議論されている．しかしながら，こういった倫理的なトピックを深く議論する前に，まず BBVC 関連の技術の潜在力を示すのが必要不可欠だとする意見も存在する [2]．

[*4] サーバサイド向けの JavaScript 実行環境．
`https://nodejs.org/`

[*5] Microsoft が開発したプログラミング言語．型の概念を追加した JavaScript のスーパセットであり，コンパイルすることで通常の JavaScript プログラムに変換できる．
`https://www.typescriptlang.org/`

[*6] `https://www.google.com/adsense`

3.5.2 クラウドインスタンスを利用する

Madoop のワーカーとして，Amazon Wev Services，Google Cloud Platform，Microsoft Azure といったクラウドサービスを利用することもできる．仮想マシンにモダンな Web ブラウザをインストールするだけで，高いスケーラビリティを持つ自分だけの分散処理環境を手に入れることができる．次の章で述べる実験では，このシナリオの詳細を説明している．

3.5.3 BBVC コミュニティを設立する

最後のシナリオとして，同じ目的を持つ沢山のメンバが参加する，特別な BBVC コミュニティを設立することが挙げられる．例えば，ソフトウェアリポジトリマイニングの研究者が特定の BBVC コミュニティを立ち上げた場合，専用の高パフォーマンスな計算環境を獲得することができる．常に Web ブラウザを立ち上げておくだけで，コミュニティメンバはマイニングの研究に貢献することができる．

4　実験

4.1　概要

本実験の目的は，Madoop で WebAssembly を導入したことにより，従来の JavaScript で記述した場合と比べてパフォーマンスがどの程度向上するのかを確かめることである．具体的には，Madoop をインストール・セットアップしたメインサーバ，および Web ブラウザがインストールされた複数台の計算機を用意し，分散処理ジョブを実行してから完了するまでにかかった時間を計測・比較する．

なお，実験対象となるジョブには，計算量が大きく入出力データは小さい「レインボーテーブルの生成」と，計算量が小さく入出力データは大きい「ワードカウント」の 2 種類を用意した．これらの詳細は 4.3 節で述べる．

4.2　環境

4.2.1　メインサーバ

表 2 にメインサーバの環境の詳細を示す．メインサーバの計算機には Amazon EC2[*7] のインスタンスを用いる．インスタンス上に Madoop をインストールし，外部からアクセスできるように設定・公開する．また，簡単のため，本実験ではメインサーバがコンテンツサーバも兼ねる．

表 2　メインサーバの環境.

項目	説明
インスタンスタイプ	t2.large[*8]
オペレーティングシステム	Ubuntu 16.04 LTS 64bit
Madoop のバージョン	0.1.6
Node.js のバージョン	8.11.3
TypeScript のバージョン	2.9.2
Emscripten のバージョン	1.38.8

[*7] Amazon Elastic Compute Cloud の略．Amazon.com, Inc. が提供する計算資源を利用して，クラウド上で仮想マシンを実行することができる．Amazon Web Services（AWS）のサービスの一つ．
`https://aws.amazon.com/ec2/`

[*8] CPU のコア数は 2，メモリサイズは 8 GiB である．

4.2.2　クライアントノード

クライアントノードの環境の詳細を表 3 に示す．クライアントノードも，メインサーバと同様に Amazon EC2 のインスタンスを用いる．複数のインスタンスを同時に立ち上げて Web ブラウザを起動し，それぞれメインサーバへ接続する．

なお，パフォーマンスの計測においては，いかに計測時の誤差を削減するかが重要となる．今回の実験では Web ブラウザの操作が必要となるが，これを GUI で操作することは大きな誤差に繋がる．最近の Web ブラウザには，人手を介さずに操作を自動化することを目的とした「ヘッドレスモード」と呼ばれる実行モードが存在する．ヘッドレスモードは Web ブラウザを GUI なしに実行するモードであり，一般的には Web アプリケーション開発におけるテストの自動化などの用途で利用される．例えば，Google Chrome を用いて次のコマンドを実行することで，指定した`<URL>`のスクリーンショットがカレントディレクトリに保存される．

```
$ google-chrome --headless --screenshot <URL>
```

このように，ヘッドレスモードで実行した Web ブラウザでも，DOM の構築や要素の更新は通常通りに行われる．Madoop の要件は`<URL>`のリソースに含まれるプログラムを Web ブラウザ上で実行することであり，Web ページを GUI で表示する必要はないため，ヘッドレスモードで問題なく実験が可能である．今回は，ヘッドレスモードに対応している Google Chrome および Mozilla Firefox の 2 種類の Web ブラウザで実験を行う．

表 3　クライアントノードの環境．

項目	説明
インスタンスタイプ	m3.medium[*9]
オペレーティングシステム	Ubuntu 16.04 LTS 64bit
Google Chrome のバージョン	67.0.3396.87
Mozilla Firefox のバージョン	60.0.2

[*9] CPU のコア数は 1，メモリサイズは 3.75 GiB である．

4.3 実験対象ジョブ

表 4 に各実験対象のパラメータを示す．以下では各実験対象の詳細について述べる．

4.3.1 E_1: レインボーテーブルの生成

レインボーテーブル [19] の生成は，Web サイトのログインパスワードに対する総当たり攻撃を効率良く行うための数値計算処理として知られる．セキュリティ上の理由から，Web サイト側のデータベースでは，ユーザのログインパスワードはハッシュ化した値を保存しておくことが多い．このハッシュ値が流出した場合，悪意のあるクラッカーが対応する元のパスワードを特定する方法として，パスワードとして有効な文字列を逐次ハッシュ化し，この値が入手したハッシュ値と一致するかどうかを総当たりで調べることが考えられる．レインボーテーブルは，このパスワード候補とそのハッシュ値の対応をメモリ効率が良くなるように計算して得られた表である．

map 関数は与えられた文字列（パスワード候補）をハッシュ化し，その値を還元関数と呼ばれる関数を用いることで，再びパスワードとして有効な文字列に戻す．さらにその文字列を再びハッシュ化する，といった処理を繰り返す．指定した回数だけこの処理を繰り返した後，最初の文字列と，一連の処理の結果として最終的に得られた文字列のペアを出力する．なお，reduce 関数の処理は特に存在せず，

表 4　実験対象ジョブのパラメータ．

共通パラメータ	
クライアントノードの数	1, 2, 3, 5, 10
試行回数	10 回
E_1: レインボーテーブルの生成	
ハッシュ値の総計算回数	10,000,000 回
1 タスクあたりのハッシュ値の計算回数	1,000,000 回
ハッシュアルゴリズム	MD5[*10]
E_2: ワードカウント	
テキストデータ[*11]の総容量	380 MB
1 タスクあたりのテキストデータの容量	44 MB

[*10] WebAssembly 形式の実装には，MD5 アルゴリズムの仕様が記述された RFC 1321 [17] の付録に記載されているプログラムを利用し，C++ プログラムとして記述した．比較対象となる JavaScript の実装には，以下で公開されている RFC 1321 の仕様に忠実に従ったものを用いた．
`http://rocketeer.dip.jp/sanaki/free/javascript/freejs17.htm`

[*11] Yelp Open Dataset [18] のデータを用いた．

受け取った (key, value) の組をそのまま出力するだけである．

このジョブでは，メインサーバが配信する各ノードの入力データとしては，パスワードとして有効なある文字列と，ハッシュ値を計算する回数の 2 つのみであり，データ容量としては無視できるほど非常に小さい．一方で，各ノードは毎回大量のハッシュ値を計算するため，1 回あたりの計算量が非常に大きい．このように，1 タスクあたりの入力データが小さく，計算量は大きいのがこのジョブの特徴である．

4.3.2 E_2: ワードカウント

ワードカウントでは，大容量のテキストファイルを解析し，テキスト内に存在する単語の一覧と，各単語の出現頻度を併せて出力する．MapReduce を用いたビッグデータ処理の題材として有名である．map/reduce 関数の内容は図 2，図 3 および 3.3 節で述べた内容と同一であるため，ここでは割愛する．

このジョブでは，各ノードの入力データは数 MB〜数十 MB と大きいが，実際に単語を数え上げる処理の計算量は少ないのが特徴であり，前項 E_1 のレインボーテーブル生成のジョブとは対照的である．

4.4 結果

4.4.1 E_1: レインボーテーブルの生成

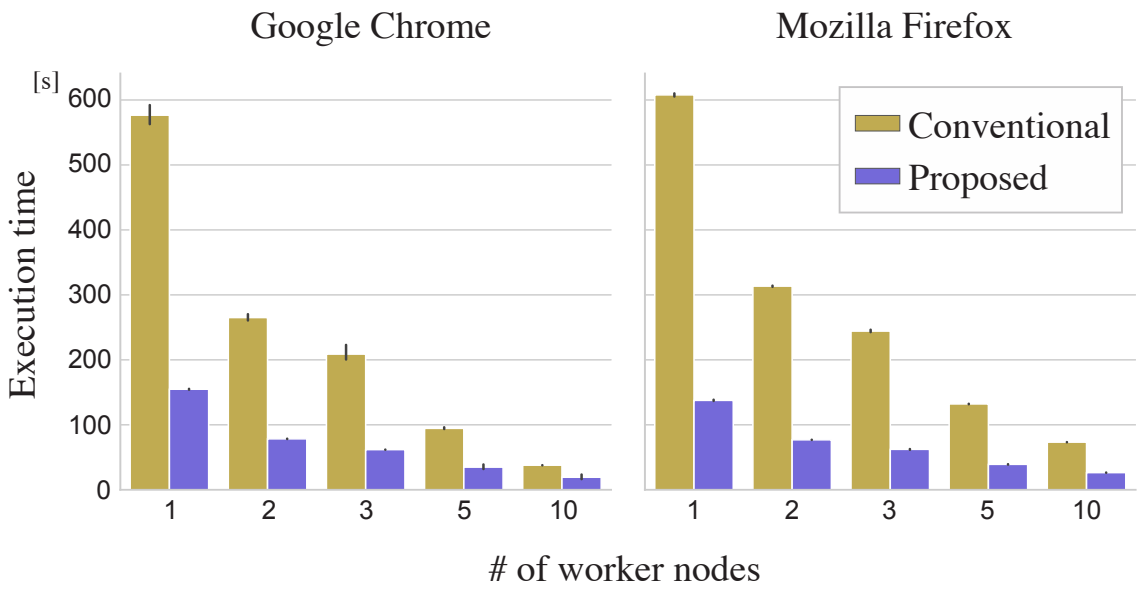

図 4　E_1: レインボーテーブル生成の結果．

表 5　E_1: レインボーテーブルの生成におけるノード数 10 の結果の平均値.

手法	Google Chrome	Mozilla Firefox
提案手法	19.2 s	26.3 s
既存手法	37.8 s	73.2 s
実行時間の削減比率	49.2 %	64.1 %

実験対象 E_1 の結果を図 4 に示す．この箱ひげ図では，縦軸がジョブ全体の実行時間の長さ，横軸がクライアントノードの数をそれぞれ表す．黄色のデータが従来手法である JavaScript を用いた場合の結果を，紫色のデータが提案手法（Madoop）である WebAssembly を用いた場合の結果をそれぞれ示す．左側の図は Google Chrome での結果を，右側の図は Mozilla Firefox での結果をそれぞれ表す．図 4 より，クライアントノードの数 n が増えるに従って，実行時間の長さはほぼ反比例して短くなっていることが分かる．また，いずれの結果においても，提案手法の方が従来手法よりも実行時間の長さが短くなっていることが見て取れる．さらに特筆すべき点として，$n = 5$ および $n = 10$ の結果に着目すると，従来手法における $n = 10$ のときよりも，提案手法における $n = 5$ のときの方が，実行時間の長さの方が短くなっていることが挙げられる．

表 5 に，$n = 10$ の場合におけるジョブの実行時間の長さの平均値を示す．このように，提案手法と従来手法の実行時間の長さを比較した場合，Google Chrome では約 49 %，Firefox では約 64 % 短くなった．

4.4.2 E_2: ワードカウント

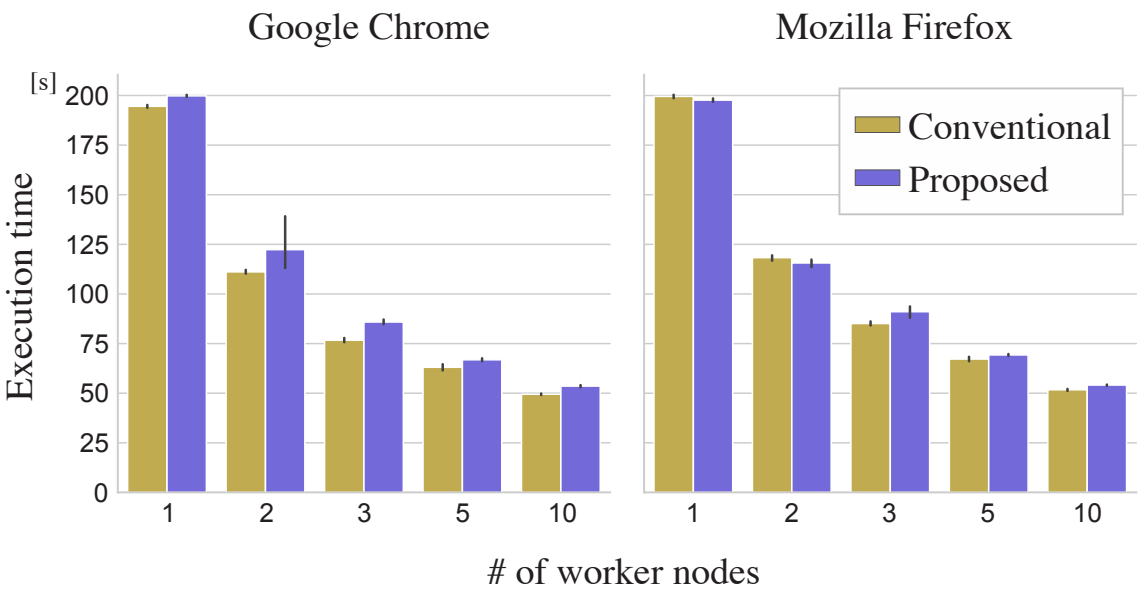

図 5　E_2: ワードカウントの結果.

表 6　E_2: ワードカウントにおけるノード数 10 の結果の平均値.

手法	Google Chrome	Mozilla Firefox
提案手法	53.6 s	54.0 s
既存手法	49.5 s	51.7 s
実行時間の削減比率	−8.3 %	−4.4 %

実験対象 E_2 の結果を図 5 に示す．今回も同様に，ノードの数 n に反比例して実行時間の長さは短くなっている．しかしながら，E_1 の結果とは異なり，今回の実験対象では，紫色の提案手法の方が黄色の従来手法よりも実行時間が長くなっている結果が多い．

表 6 に，$n = 10$ の場合におけるジョブの実行時間の長さの平均値を示す．提案手法の方が従来手法よりも，実行時間の長さは Google Chrome で約 8 %，Mozilla Firefox で約 4 % 増加した．

4.5 考察

実験対象 E_1: レインボーテーブルの生成において，提案手法を用いた場合 Google Chrome で約 49 %，Mozilla Firefox で約 64 % 実行時性能が改善した．このことから，E_1 のような計算量が大きくデータ量が小さいジョブにおいて，提案手法が明らかに有利であることが分かる．

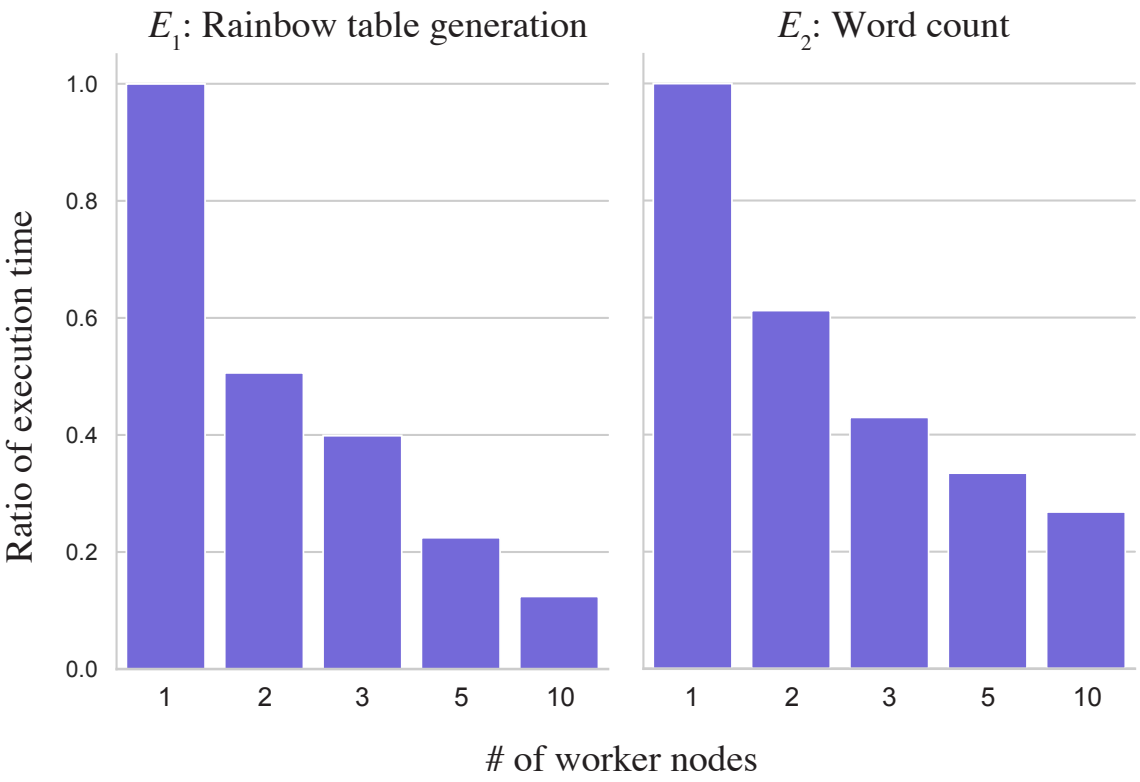

図 6　提案手法におけるクライアントノード数を増やした場合の実行時間の長さの比率.

図 6 は，提案手法において，クライアントノード数 $n = 1$ を基準とした場合における，n を増加させたときの実行時間の長さの比率を表している．理論的には，$n = 5$ のときの比率は 1/5（0.2）となる．クライアントノードの数に応じて性能もスケールしており，特に実験対象 E_1 においては理論値に近い性能が出ていることが分かる．このことから，Madoop が分散処理システムのフレームワークとして正しく機能していることが見て取れる．

なお，実験対象 E_2: ワードカウントの結果の多くでは，提案手法を用いた場合の方が従来手法を用いた場合よりも実行時間の長さが増加するという結果になった．その増加比率は Google Chrome で約 8 %，Mozilla Firefox で約 4 % であった．この原因として，Web ブラウザが WebAssembly バイナリを解釈・実行する際に発生するオーバヘッドが考えられる．Web ブラウザが WebAssembly バイナリを実行する手順の概要を以下に示す．

1. バイナリデータ（`.wasm` ファイル）をサーバからダウンロードする．
2. バイナリデータを Web ブラウザ内で再度コンパイルする．これにより，多様なアーキテクチャに合わせた最適化が可能となる．
3. バイナリを実行するためのメモリ領域を確保する．
4. バイナリを実行する．

5. 3 で確保したメモリ領域を解放する．

上記の手順の中で，WebAssembly バイナリが実際に実行されているのは 4 のステップだけであり，それ以外のステップはバイナリを実行するための前後の処理である．これらは JavaScript を実行する場合から見てオーバヘッドとなる．よって，計算量が小さいタスクを実行した場合，WebAssembly を導入することにより削減された時間よりも，このオーバヘッドにより増加した時間の方が長くなる場合があると考えられる．4.3.2 項でも述べたように，実験対象 E_2 のワードカウントは 1 タスクあたりの計算量が小さいジョブであるため，WebAssembly を用いた方が遅くなったと考えられる．

5 適用実験

5.1 概要

前章では小さなタスクに対して実験を行った．本章では，以下の観点からさらに適用実験を行う．

- Madoop をより実践的で複雑な課題にどう適用するか？
- そのような課題に対して，Madoop はパフォーマンスをどの程度改善させるか？

このケーススタディとして，Madoop をプログラム依存グラフ（Program Dependence Graph，以降 PDG）を用いたコードクローンの検出に用いる．

5.1.1 PDG を用いたコードクローン検出

PDG はプログラム内の要素間に存在する依存関係を表す有向グラフである．PDG における頂点はプログラムの要素を，辺は 2 頂点間に依存関係が存在することをそれぞれ表す．

PDG には制御依存データ依存の 2 つの依存が存在する [20]．次のすべての条件を満たすとき，文 s_1 から文 s_2 に制御依存が存在する．

- 文 s_1 は条件付き述語（条件文または繰返し文の条件式）である
- 文 s_1 の評価値（`true` または `false`）によって文 s_2 が実行されるか否かが決まる

一方で，次の条件をすべて満たすとき，文 s_3 から文 s_4 にデータ依存が存在する．

- 文 s_3 が変数 v を定義している
- 文 s_4 が変数 v を参照している
- 文 s_3 から文 s_4 への実行経路のうち，変数 v を再定義しないものが一つ以上存在する

図 7 はあるメソッドと，それに対応する PDG の例である．グラフ中の頂点の数字は，対応するプログラムの行番号を示している．また，制御依存辺を青色の線で，データ依存辺を赤色の線でそれぞれ描いている．以下では，頂点 a から頂点 b へ引かれた辺を (a, b) と表す．PDG ではその定義上，メソッドの宣言部から直属の子要素に制御依存が引かれる．$(1, 2)$ に引かれた辺はこれを表す．2 行目で `if` 文が宣言されており，この評価値によって以降の文が実行されるかが決まる．したがって，$(2, 3)$，$(2, 4)$，$(2, 6)$ にそれぞれ制御依存が引かれる．さらに，3 行目で変数 `proj` が定義され，この変数が途中で上書きされることなく 4 行目で参照されている．したがって，$(3, 4)$ にはデータ依存が引かれる．

PDG は与えられたソースコード中に含まれる各関数またはメソッドから生成される．PDG を用いたコードクローン検出では，生成された PDG 間に部分グラフ同型（isomorphic subgraph）を検出する

```
1 void Sample() {
2   if (this.trueOrFalse()) {
3     Project proj = this.getProject();
4     this.setPath(proj.getBaseDir());
5   } else {
6     this.setPath(this.DEFAULT_DIR);
7   }
8 }
```

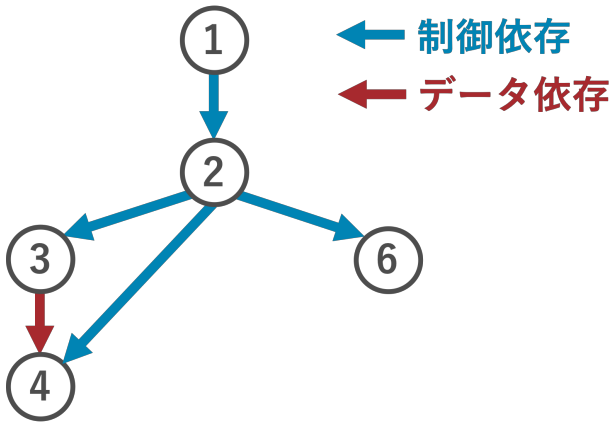

図 7　PDG の例.

ことで行うため，ソースコードから直接クローンを検出することはない．言い換えると，Bellon ら [21] により分類された以下のコードクローンのうち，PDG を用いたコードクローン検出では Type 1 および Type 2 のクローンを検出することができる．

Type 1　空白やタブを除き，表面上まったく同一であるコードクローンのペア．

Type 2　変数名やメソッド名などのユーザ定義の識別子だけが異なり，他は同一であるコードクローンのペア．

Type 3　Type 1 や Type 2 が許容する違いに加えて，文・式単位で異なる要素が存在するコードクローンのペア．

Type 1 のクローンに比べて，Type 2 のクローンは同じ振る舞いをするコードを互いに含む可能性が高いため，リファクタリング対象として適している [22, 23].

しかしながら，部分グラフ同型の検出は NP 完全な問題である [24] ため，PDG を用いたコードクローン検出は大きな計算コストを必要とする．以降では，Madoop を用いてこれを高速化をすることを

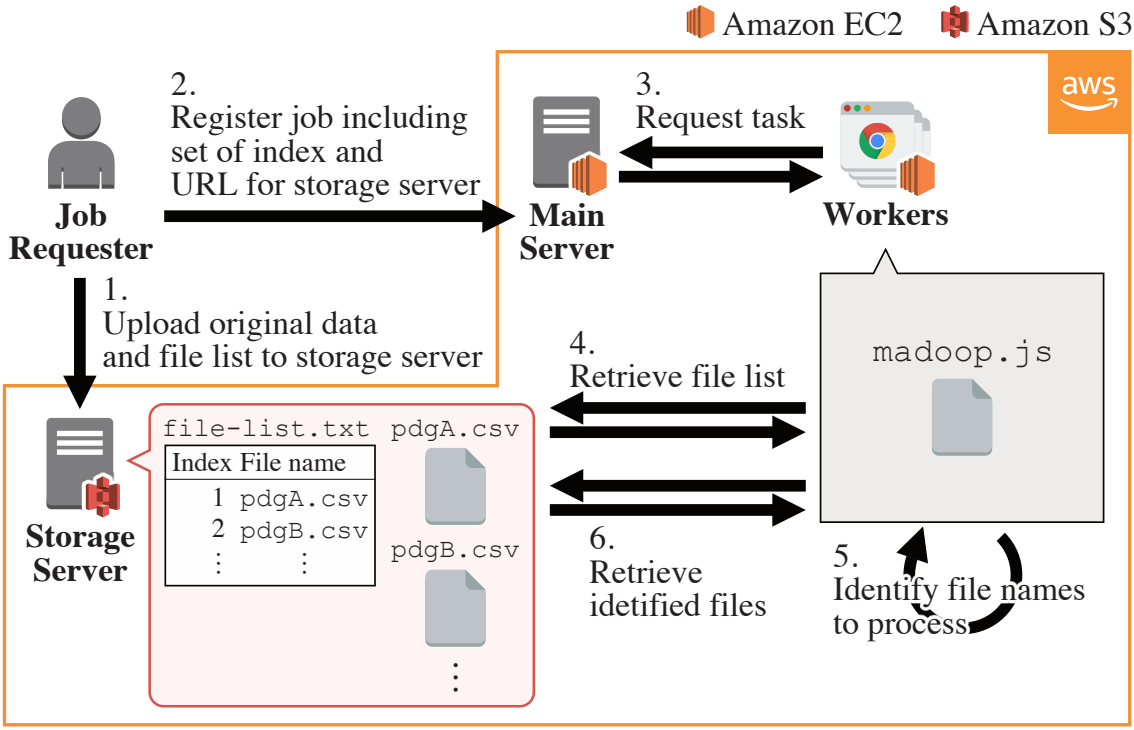

図 8　PDG を用いたコードクローン検出を Madoop で行う際の戦略.

試みる.

5.2　Madoop 環境のセットアップ

各ワーカーノードに入力データを配布する方法は様々存在する．一つの方法として，メインサーバのメモリ上にすべての入力データを格納しておき，ワーカーからのリクエスト時にそのデータを配布することが考えられる．しかしながら，コードクローン検出では膨大な数のファイルを処理する必要があるため，この方法ではメモリ領域を使い果たしてしまう可能性がある．

この問題を解決するため，本ケーススタディでは，メインサーバが入力データのインデックスのみを保持しているような Madoop 環境を構築する．図 8 にこの戦略を示す．簡単のため，図 8 からはコンテンツサーバ及びコンテンツ提供者を省略している．また，Web サイトの訪問者が BBVC のワーカーとなる過程もここでは言及していない．これらの手順は図 1 で既に説明している．

本戦略は以下の手順で実行される．

1. ジョブ登録者が元のデータ（PDG）及びそのリストをストレージサーバにアップロードする．ここでは，ストレージサーバとして Amazon S3[*12] を用いた．

[*12] Amazon Simple Storage Service．AWS により提供されているサービスの一つ.

2. ジョブ登録者が入力データを含むジョブをメインサーバに登録する．この入力データはインデックスと URL から形成され，インデックスは各ワーカーがどの PDG を処理するのかを，URL は各ワーカーがファイルリストや PDG をどのストレージサーバからダウンロードしてくるのかをそれぞれ示している．
3. 各ワーカーはメインサーバからタスクを取得する．
4. ワーカーはまず，ストレージサーバからファイルリストを取得する．
5. ワーカーは次に，インデックスおよびファイルリストを用いて，どのファイルを処理するのかを特定する．
6. ワーカーは特定されたファイルをストレージサーバからダウンロードし，map 関数を実行して PDG を処理する．処理結果はメインサーバへと返却する．

表 7 に本実験におけるパラメータを示す．

5.3 MapReduce プログラムと入力データの準備

本ユースケースでは，map 関数はある 2 グラフ間に部分グラフ同型が存在するかどうかを判定するアルゴリズムを実行する必要がある．3.2.2 項でも述べたように，Madoop の map/reduce 関数は純粋な C/C++ プログラムとして実装できるため，サードーパーティ製のライブラリである Boost[*13] を利用することができる．今回は，Boost の `vf2_subgraph_iso`[*14] をアルゴリズムの実装として用いた．

この実験では，コードクローンの検出対象として，オープンソースプロジェクトである Apache ACE[*15] を用いた．Apaceh ACE の対象バージョンでは，自動生成されたファイルやテスト用のファイルを除いて，441 個の Java ファイルが含まれている．ソースコードの総行数は約 72,000 である．

PDG の生成には TinyPDG[*16] を用いた．TinePDG は肥後らによる PDG を用いたコードクローン検出関連の研究 [25] で開発されたツールである．TinePDG には，与えられた Java ソースファイル中の各メソッドに対応する CSV ファイルを出力する機能がある．この CSV ファイルは 3 カラムで構成されており，有向辺の始頂点，終頂点，依存関係の種類がそれぞれ記載されている．言い換えると，

表 7　PDG を用いたコードクローン検出におけるパラメータ．

説明	値
ワーカーノード数	1, 5, 10 or 20
PDG の総数	3,391

[*13] 最も有名なオープンソースの C++ ライブラリの一つ．`https://www.boost.org/`
[*14] `https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/vf2_sub_graph_iso.html`
[*15] `https://ace.apache.org/`
[*16] `https://github.com/YoshikiHigo/TinyPDG`

CSV ファイルの各行には，ある 2 ノード間の制御またはデータ依存が記載されている．ここで，始頂点および終頂点は，それぞれ頂点の文字列表現から計算したハッシュ値で示される．なお，Type 2 クローンの検出のため，ハッシュを計算する前にユーザが定義した識別子は正規化する必要がある．例えば，図 7 の PDG に対応する CSV ファイルの内容は，表 8 のようになる．

5.4 パフォーマンス比較

表 9，図 9 はワーカーの数を増やした際にパフォーマンスがスケールする様子を示している．1 ノードのみを用いた場合，ジョブが完了するまでに約 230 秒経過している．しかし，処理を 20 個のワーカーに並列化すると，実行時間は約 90% 削減され，約 25 秒となった．

表 8　図 7 の PDG に対応する CSV ファイル.

始頂点	終頂点	依存関係の種類
1	2	control
2	3	control
2	4	control
2	6	control
3	4	data

表 9　PDG を用いたコードクローン検出の実行時間の一覧.

ワーカーノードの数	実行時間
1	231.060 [s]
5	57.934 [s]
10	36.903 [s]
20	25.173 [s]

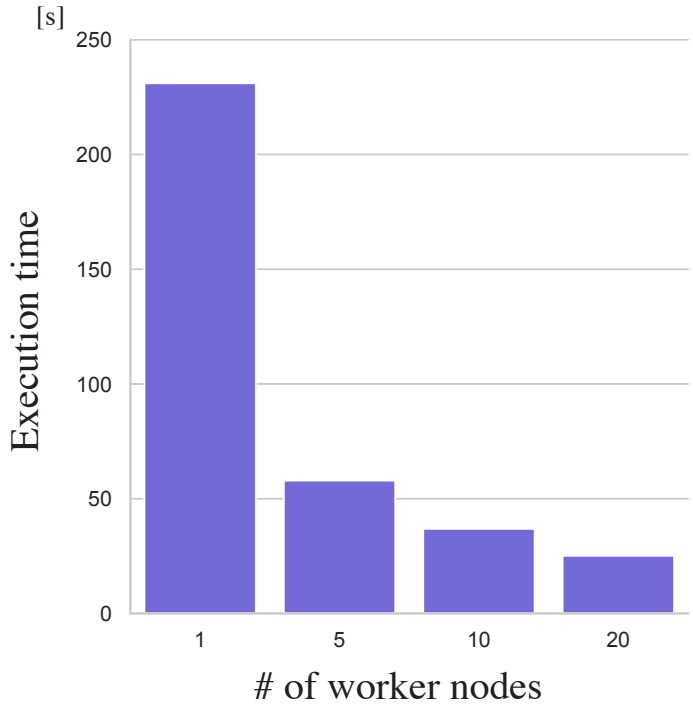

図 9　PDG を用いたコードクローン検出におけるパフォーマンス比較.

6　妥当性への脅威

本研究では，メインサーバ・ワーカーノード共に AWS EC2 上のインスタンスで行った．AWS EC2 は仮想化技術を用いたサービスの一つであり，実験中，実際には同じ物理マシン上で第三者の仮想マシンが動いている．従って，共存している第三者の仮想マシン上の処理によっては実験で用いた仮想マシンのリソースが制限を受けている可能性があり，異なる時間帯や異なるリージョンで再実験した際に，結果が少し変わる可能性がある．

今回用いた WebAssembly はバージョン 1.0 の正式リリースを終えているものの，まだまだ発展中の技術であり，将来のバージョンで仕様が変わった際に実験結果も変わる可能性がある．

今回の実験では計算機として AWS EC2 の仮想マシンを，サーバサイドの実行環境として Node.js を，クライアントサイドの実行環境として Google Chrome や Mozilla Firefox をそれぞれ用いている．しかしながら，Web アプリケーションにおいては，とりわけユーザ側の実行環境は多種多様であり，今回と異なる環境だと結果が終わる可能性がある．

今回は C/C++ 言語で map/reduce 関数を実装し，Emscripten を用いて WebAssembly 形式へとコンパイルを行った．このほか，C#，Go，Java，Python，Ruby など様々なプログラミング言語が WebAssembly 形式へのコンパイルをサポートもしくは開発中の段階であり [26]，プログラミング言語やコンパイラが変われればパフォーマンス性能に差が生まれる可能性がある．

今回の実験では，対象ジョブとしてレインボーテーブルの生成，ワードカウント，PDG を用いたコードクローン検出の 3 種類を行った．Madoop のパフォーマンスはジョブの特徴に強く左右されるため，これ以外のジョブや，とりわけ計算量が少ないタスクでは実験結果が変わる可能性がある．

7 関連研究

BBVC 関連の研究には様々なものが存在する [1, 27, 2, 28, 29, 30, 31]．Charlotte [31] は最も初期の BBVC 実装であり，クライアントサイドのプログラムとして Java Applet を用いている．BBVC の潜在能力は，Ajax，HTML5，Web Workers [32] や WebAssembly [15, 5] といった新しい Web スタンダード技術の到来により急速に発展した．2015 年に提案された Gray Computing [27, 2] は，いくつかのモダンな Web 技術を用いて良くデザインされた BBVC フレームワークである．Gray Computing の根本的なリサーチクエスチョンは，Web サイトの訪問者を用いたデータ処理における実現可能性及びコスト効率の確認である．Gray Computing と対象的に，Madoop は BBVC の重要な 2 つの課題である，実行パフォーマンスの低さと開発容易性の低さに着目している．それゆえに，本研究成果は Gray Computing と併用して適用することができる．

MapReduce パラダイムを JavaScript に適用するため，様々な研究が行われている [11, 12]．Langhans らは JavaScript で MapReduce を実装した，JSMapReduce を提案している [12]．JSMapReduce では，MapReduce ジョブの処理に Web Worker [32] を用いることで，Web サイト訪問者の UX に影響を与えることなく，バックグラウンドで処理を行っている．MRJS [11] も同様に，JavaScript ベースの MapReduce フレームワークである．Madoop と同様に，MRJS の主要な目的は分散システムにおける複雑さを抽象化することである．MRJS，Madoop 共にワードカウントといった小さなタスクで評価を行っている一方，Madoop では実践的な研究課題であるコードクローン検出を行い，適用実験も行っている．

Google Chrome，Mozilla Firefox，Safari といった人気のブラウザベンダーは，JavaScript エンジンのパフォーマンスを改善するため様々な試みを行ってきた．Native Client (NaCl) [33]，Portable Native Client (PNaCl) [34]，Emscripten [14]，asm.js [35]，WebAssembly [15, 5] などが代表的な試みである．これらの試みはすべて，Just-In-Time (JIT) コンパイルや実行時型情報の利用を避け，Web ブラウザ上で低レベルなコードを実行するという共通のコンセプトを持つ．本稿で提案する Madoop では，Web ブラウザ上でネイティヴに近いパフォーマンスを発揮できる最新の Web 標準技術である，WebAssembly を採用している．

高木らも本研究と同様に，Web ブラウザベースで VC を行うプラットフォームの提案を行っている [8]．高木らの手法では PNaCl を用いて高速化を図るとともに，Web ブラウザ上でタスクをバックグラウンドで行うために Web Worker [32] を，Web ブラウザ上のローカルストレージにデータを保存し通信量を削減するために WebStrage [36] や IndexedDB [37] を，ユーザが指定したファイルの読み込みを行うために FileAPI [38] をそれぞれ用いており，モダンな Web 技術を多く用いていることが分かる．高木らの手法と本研究との大きな違いとして，研究の主目的と使用技術が挙げられる．高木らは主

目的として Web アプリケーションの高速化を掲げているが，本研究ではそれに加え，MapReduce パラダイムの導入による開発者の負担軽減を掲げている．また，高木らはパフォーマンス向上のため PNaCl を用いているが，本研究ではそこからさらに発展した WebAssembly という最新の技術を用いている．

8　あとがき

本研究では，Web ブラウザ上で行う分散処理手法 BBVC のまったく新しいフレームワークである Madoop を提案した．Madoop では，BBVC における重要な課題である (1) 分散処理プログラムの開発容易性の低さ，および (2) JavaScript による実行時性能の低さを解決するため，MapReduce と WebAssembly をそれぞれ導入した．また，BBVC の従来手法と比べて提案手法（Madoop）が実行時パフォーマンスを改善するのかを実験で評価した．実験の結果，時間計算量の大きな問題である，レインボーテーブル生成のジョブでは実行時間が 50 〜 64% 改善した．しかしながら，データサイズの大きな問題である，ワードカウントジョブでは 4 〜 8% 増加した．これらの結果から，Madoop は時間計算量の大きな問題により適しているといえる．

本研究ではまた，ソフトウェアエンジニアリングの分野でより実践的な問題である，PDG を用いたコードクローン検出に Madoop を用いるデモンストレーションとして適用実験を行った．実験の結果，20 台のワーカーで処理を並列化すると，1 台の場合と比べて実行時間が 90% 削減された．

今後の課題として，各ワーカーノードの特性を考慮した，より効率的なスケジューリングアルゴリズムの設計が挙げられる．現在 Madoop ではノードの通信状況や処理性能，滞在時間といった特性を考慮せず，すべてのノードに一律して同じサイズのタスクを配布している．BBVC では Web ブラウザ上で処理をするという特性上，VC に比べてノードの特性の影響を非常に受けやすいため，さらなる性能の改善にはスケジューリングアルゴリズムの改善が必要である．

さらに，通信量の削減も課題として挙げられる．7 章でも述べたように，高木らの手法 [8] では，WebStrage や IndexedDB といった技術を用いて通信量の削減を行っている．通信量を削減するためには，このほか WebRTC API [39, 40, 41] を用いてサーバを経由せずクライアント（Web ブラウザ）間同士で直接 P2P 通信を行ったり，WebSocket プロトコル [42, 43, 44] を用いて双方向のソケット通信を行ったりすることも有効である．現状の Madoop はサーバを介して HTTP プロトコルによる通信を行うため，ネットワークを流れるパケット中のヘッダが増大し，ペイロードは相対的に小さくなる．このため，これらの技術を用いる場合と比べてオーバヘッドが大きいといえる．

謝辞

本研究を行うにあたり，理解あるご指導を賜り，常に励ましていただきました楠本真二教授に，心より感謝申し上げます．

本研究に関して，議論の中で貴重なご助言をいただきました肥後芳樹准教授に，深く感謝申し上げます．

本研究の全過程を通して，研究に対する考え方や方向性など，丁寧かつ熱心なご指導を賜りました柗本真佑助教に，心より感謝申し上げます．

本研究を進めるにあたり，様々な形で励まし，ご助言を頂きましたその他の楠本研究室の皆様のご協力に深く感謝致します．

最後に，本研究に至るまでに，講義，演習，実験等でお世話になりました大阪大学大学院情報科学研究科の諸先生方に，この場を借りて心から御礼申し上げます．

参考文献

[1] Tomasz Fabisiak and Arkadiusz Danilecki. Browser-based harnessing of voluntary computational power. *Foundations of Computing and Decision Sciences*, Vol. 42, No. 1, pp. 3–42, 2017.

[2] Yao Pan, Jules White, Yu Sun, and Jeff Gray. Gray computing: A framework for computing with background javascript tasks. *IEEE Transactions on Software Engineering*, 2017.

[3] Juan Julián Merelo Guervós, Mario García Valdez, Pedro Ángel Castillo Valdivieso, Pablo García-Sánchez, Paloma de las Cuevas, and Nuria Rico. Nodio, a javascript framework for volunteer-based evolutionary algorithms: first results. *CoRR*, Vol. abs/1601.01607, , 2016.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pp. 137–150, 2004.

[5] WebAssembly. `https://webassembly.org/`.

[6] SETI@home. `https://setiathome.berkeley.edu/`.

[7] SETI@Home - Detailed stats | BOINCstats/BAM! `https://boincstats.com/en/stats/0/project/detail/`.

[8] Shogo Takaki, Ken Watanabe, Masaru Fukushi, Noriki Amano, Nobuo Funabiki, and Toru Nakanishi. A proposal of web browser-based volunteer computing platform. Technical Report 29, 2014.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 20–43, 2003.

[10] Apache Hadoop. `http://hadoop.apache.org/`.

[11] Sandy Ryza and Tom Wall. Mrjs: A javascript mapreduce framework for web browsers, 2010. `http://www.cs.brown.edu/courses/csci2950-u/f11/papers/mrjs.pdf`.

[12] Philipp Langhans, Christoph Wieser, and François Bry. Crowdsourcing mapreduce: Jsmapreduce. In *Proceedings of the 22nd International Conference on World Wide Web*, pp. 253–256. ACM, 2013.

[13] Can I use... Support tables for HTML5, CSS3, etc. `https://caniuse.com/#search=wasm`.

[14] Alon Zakai. Emscripten: An llvm-to-javascript compiler. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, pp. 301–312. ACM, 2011.

[15] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 185–200. ACM, 2017.

[16] Daniel G. Goldstein, R. Preston McAfee, and Siddharth Suri. The cost of annoying ads. In *Proceedings of the International Conference on World Wide Web*, pp. 459–470, 2013.

[17] The MD5 Message-Digest Algorithm. `https://www.ietf.org/rfc/rfc1321.txt`.

[18] Yelp Open Dataset. `https://www.yelp.com/dataset`.

[19] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Annual International Cryptology Conference*, pp. 617–630. Springer, 2003.

[20] Mark Weiser. Program Slicing. In *Proceedings of the 5th International Conference on Software Engineering*, pp. 439–449, 1981.

[21] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions on software engineering*, Vol. 33, No. 9, 2007.

[22] Keisuke Hotta, Yoshiki Higo, and Shinji Kusumoto. Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*, pp. 53–62, 2012.

[23] Davood Mazinanian, Nikolaos Tsantalis, Raphael Stein, and Zackary Valenta. JDeodorant: Clone Refactoring. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 613–616, 2016.

[24] Arvind Gupta and Naomi Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoretical Computer Science*, Vol. 164, No. 1-2, pp. 287–298, 1996.

[25] Yoshiki Higo and Shinji Kusumoto. Code Clone Detection on Specialized PDGs with Heuristics. In *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*, pp. 75–84, 2011.

[26] Awesome WebAssembly Languages. `https://github.com/appcypher/awesome-wasm-langs`.

[27] Yao Pan, Jules White, Yu Sun, and Jeff Gray. Gray computing: an analysis of computing with background javascript tasks. In *Proceedings of the 37th International Conference on Software Engineering*, pp. 167–177, 2015.

[28] E. Lavoie, L. Hendren, and F. Desprez. Pando: A volunteer computing platform for the web. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems*, pp. 387–388, 2017.

[29] F. Boldrin, C. Taddia, and G. Mazzini. Distributed computing through web browser. In *2007 IEEE 66th Vehicular Technology Conference*, pp. 2020–2024, 2007.

[30] Sachith Gullapalli. Atlas: An intelligent, performant framework for web-based grid computing. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 1154–1156. ACM, 2016.

[31] Arash Baratloo, Mehmet Karaul, Zvi M Kedem, and Peter Wijckoff. Charlotte: Metacomputing on the web. *Future Generation Computer Systems*, Vol. 15, No. 5-6, pp. 559–570, 1999.

[32] W3C: Web Workers. `https://w3c.github.io/workers/`.

[33] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. In *2009 30th IEEE Symposium on Security and Privacy*, pp. 79–93, 2009.

[34] Alan Donovan, Robert Muth, Brad Chen, and David Sehr. Pnacl: Portable native client executables. *Google White Paper*, 2010.

[35] asm.js. `http://asmjs.org`.

[36] William West and S. Monisha Pulimood. Analysis of privacy and security in html5 web storage. *J. Comput. Sci. Coll.*, Vol. 27, No. 3, pp. 80–87, 1 2012.

[37] S. Kimak and J. Ellman. The role of html5 indexeddb, the past, present and future. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 379–383, Dec 2015.

[38] File API. `https://www.w3.org/TR/FileAPI/`.

[39] Alan B. Johnston and Daniel C. Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, 2012.

[40] J. K. Nurminen, A. J. R. Meyn, E. Jalonen, Y. Raivio, and R. Garcıa Marrero. P2p media streaming with html5 and webrtc. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 63–64, April 2013.

[41] Antony J. R. Meyn. Browser to browser media streaming with html5. Master's thesis, 2012.

[42] V. Pimentel and B. G. Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, Vol. 16, No. 4, pp. 45–53, July 2012.

[43] Y Furukawa. Web-based control application using websocket. *ICALEPCS2011*, pp. 673–675, 2011.

[44] Yan Zhangling and Dai Mao. A real-time group communication architecture based on web-socket. *International journal of computer and communication engineering*, Vol. 1, No. 4, p. 408, 2012.