社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－IOT－1（1）
2008/5/8

異機種混合並列計算ミドルウェア JSGrid

武田 和大† 小野 智司‡ 中山 茂‡

†‡鹿児島大学工学部 〒890-0065 鹿児島市郡元 1-21-40

E-mail: †takeda@aae.kagoshima-u.ac.jp, ‡{ono, shignaka}@ics.kagoshima-u.ac.jp

あらまし 本報では様々な OS で動作している計算機を，環境を変更することなく容易に並列計算に転用するための JSGrid を提案する．JSGrid ではソフトウェアインストール作業をすることなく遊休状態の計算資源を利用できる．そのため例えば教育機関の演習用計算機や企業や研究所の個人用・事務処理用の計算機，図書館の検索端末などをクラスタの一部として使用することができる．評価実験では Windows,Linux,MacOSX,が混在する計算機群において動的に参加台数を変化させて計算を行った結果，理論予測値に近い高いスループットを得られた．

キーワード 分散並列処理，異機種混合クラスタ，オブジェクト共有空間，Java

A Middleware for Parallel Computing with Heterogenous Cluster: JSGrid

Kazuhiro TAKEDA† Satoshi ONO‡ and Shigeru NAKAYAMA‡

†‡Faculty of Engineering, Kagoshima University 1-21-40, Korimoto, Kagoshima 890-0065, Japan

E-mail: †takeda@aae.kagoshima-u.ac.jp, ‡{ono, shignaka}@ics.kagoshima-u.ac.jp

Abstract This paper proposes JSGrid, which is a software environment to build up computer clusters running on various operating systems. JSGrid makes good use of idle computational resources without installing any software. JSGrid therefore enables various kinds of computers to be a part of a cluster, for instance, educational terminals at schools, computers for development, office computers at companies and laboratories, and information retrieval terminals at libraries. An Experimental result has shown that JSGrid could manage various operating systems such as Windows, Linux, and Mac OS, and keep high throughput almost equal to theoretical prediction even under a circumstance in which dynamic join and termination of computers occur.

Keyword Parallel and distributed processing, Heterogenous cluster, Object Shared Space, Java

1. はじめに

個人や研究室レベルで分散並列処理を行う場合，一般に PC クラスタが構築・利用されることが多い．PC クラスタの構築は Parallel Virtual Machine(PVM)[1]や Message Passing Interface(MPI)[2]等の並列計算用ライブラリやツールキットの導入が必要で，そのライブラリやツールキットの使用により使用できる OS が制限されてしまう．ネットワークやセキュリティ，ファイルシステム共有など様々な項目の設定も必要である．また PC クラスタは通常，並列計算専用として設置されるため，並列計算以外への転用が難しく，実際に計算する時間以外は休眠させてしまう．これは常に計算を行う場合を除いて効率が悪い．ある一時期にだけ大きな処理能力が必要であるといった状況もありえる．

一方で我々のまわりには，教育機関の演習室に設置してある共用計算機，事務用や開発用などの個人用計算機など，多数の計算機が存在しており，これらは夜間や休日には休眠している．この計算機を集結すれば大きな処理能力を得られ，用途によっては十分に実用的な並列計算環境を構築できるものの，計算終了後に元の環境に復帰する必要がある．光ディスクや，ネットワーク経由で配信されるオペレーティングシステムを利用することで，本来の用途の設定・環境を変更することなく，クラスタを構築することができるが，クラスタを構成する計算機が全て同一の機種である必要がある．実際には設置目的や設置時期により計算機のハードウェア構成やソフトウェア設定が異なり，さらにはすべての管理者権限を得るのはまず不可能なため，たとえ一時的にでもそれらを分散並列計算環境に仕立てるのは困難である．また，別の設置目的で利用される計算機をその休眠時間に限り利用するため，長時間連続で同じ計算機を利用できるとは限らない．連続して利用できる時間の長さは計算機によってまちまちで，かつ断続的である．このような状況下で，休眠中の計算資源を繰り返し集結して問題を解かせようとすると非常に煩雑な作業となる．

また，近年，複数の遠隔地に存在する計算資源を 1 つの大きな並列処理環境として扱うグリッドコンピュ

ーティングが注目されている[3-7]. Globus などのミドルウェアが開発されており，安全性の高い分散並列環境で並列計算を行える．また，既存計算機を並列計算へ転用する観点として，事務処理，開発などの主に特定の個人が利用する計算機を，並列計算に参加させる試みも行われている[8,9]. BOINC などに代表されるこれらの並列計算環境は，様々な用途の PC を並列計算に参加させることができる．その一方で上記のグリッドコンピューティング環境は，OS 毎にコンパイルやインストールパッケージを製作する必要があることに加え，ソフトウェアのインストールが必要であるため管理者権限が必要，初心者にとって敷居が高い，参加登録を要するなどの欠点を持つ．また，計算プログラムの更新，変更が行われた場合に，繰り返しインストールが必要となってしまう．大学の図書館，演習室や共用空間などに設置された教育用，汎用の計算機は，ソフトウェアのインストールが禁止されている場合，インストールを伴う計算環境の導入は難しい．

本稿では，Web ブラウジングが可能な程度のリテラシを持つユーザであれば 3 回程度の簡単な操作で並列計算に参加でき，企業や大学などの組織の内部で容易に構築可能なクラスタシステム JSGrid を提案する．JSGrid は分散共有メモリを利用したマスタ- ワーカ型クラスタを構築するシステムであり，様々な OS を搭載し，計算性能が不均質な計算機を利用できる，計算機の動的な参加や離脱が可能である，障害の発生に対処できる，標準的な Java 実行環境以外のソフトウェアの導入が不要であるなどの特徴を持つ．また，JSGird は，参加している計算機群に任意の並列処理を任意のタイミングで委託できるシステムとなり，グリッドの一面を持つ[5].

2. 分散並列環境 JSGrid

2.1. 開発指針および特徴

JSGrid の開発指針を以下に示す．

1) 既存の技術を利用：実用性，信頼性が高い分散並列環境を構築するために，既に普及している技術や資源[10,11]を可能なかぎり利用する．ただし，開発者の労力削減，および既存のライブラリの更新に伴う並列プログラムの修正や再コンパイルを避けることを目的とした独自のライブラリを開発する．

2) OS に依存しないプログラミング言語を採用：本研究ではプログラムを Java 言語で記述し，機種依存性を排除する．従来，Java は C などと比較して実行速度が遅いとされていたが，HotSpot 技術により，計算を主とするプログラムであれば実行速度が大幅に改善されている．

3) 分散共有メモリを用いてマスタ- ワーカ型クラスタを構築：メッセージ送受信に基づく MPI や PVI などの分散メモリ型並列計算方式では，並列計算に参加している計算機の正確な管理が必要となり，並列計算実行中の動的な計算機の参加や離脱，障害に対処するモジュールを，並列プログラム作成者が記述する必要がある．本研究では，分散共有メモリを介して計算機間の通信を行うものとし，オブジェクトの共有に特化した Linda Tuple Space モデル[12]を採用する．Java によるオブジェクト共有空間である JavaSpaces[10]を利用し，JavaSpaces の内容のみを管理することで，並列計算に参加している計算機の記憶，管理が不要となり計算機構成の動的な変化や障害発生に容易に対応できる．

4)並列計算に参加する制限の緩和と手順の簡略化：共用や個人用など，並列計算に特化していない計算機をワーカ計算機として利用することを主たる目的としているため，管理者権限のないユーザであっても，計算機を並列計算に参加させられるようにする必要がある．また，より多くの計算機を並列計算に参加させるために，その作業が容易であることが重要である．本研究では Java Web Start(JWS)[11]によりプログラムの配信を行う．このため，標準的な Java 実行環境が導入されている全ての計算機において，ライブラリやソフトウェアを個別に導入することなく並列計算に参加することが可能となる．管理者権限は不要で，並列計算に参加させる作業は，Web ブラウザなどに URL を入力するだけでよく，簡単な操作で並列計算に参加させることができる．

5)プログラム作成者の労力を軽減：
並列プログラム開発の労力を最低限に抑えるべく，独自のライブラリを提供する．これにより，JavaSpaces やそれに関連する知識がなくとも，既存の Java プログラムを容易に並列化することが可能である．

2.2. JSGrid における分散並列処理

2.2.1. 構成

JSGrid における，オブジェクト共有空間 JavaSpaces を利用した分散並列処理の構成を図 1 に示す．JavaSpaces は，Linda モデルを参考として Java で実装されたオブジェクト共有空間(Object-Shared Space: OSS)であり，データやコードを含むエントリと呼ばれるオブジェクトを貯蔵・管理し，オブジェクトの流れを作り出すことができる．OSS を利用する分散並列処理は，処理を依頼するプログラム(Master)と，処理を引き受けて計算を行うプログラム(Worker)からなる．Master は処理対象の分割および処理結果の収集を行い，Worker は分割された処理対象を処理する．Master によって分割された個々の処理内容はタスクエントリ(Task)として OSS に書き込まれ，Worker によって処理した Task の結果は結果エントリ(Result)として OSS

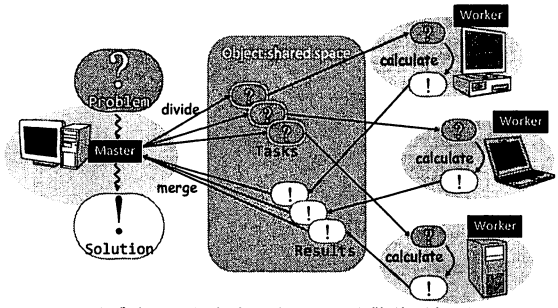

図 1.オブジェクト共有空間を用いた分散並列処理

に書き戻され，Master により回収される．OSS はサーバとして動作し，Master および Worker は，OSS サーバに対するクライアントとなる．Worker は JavaSpaces で共有されるエントリでもある．クライアントから OSS へエントリを転送する処理を write と呼び，OSS からクライアントへエントリを転送する処理のうち，転送後に OSS からエントリを削除する場合の処理を take，削除しない場合を read と呼ぶ．

2.2.2. 並列計算環境の構築手順

JSGrid を用いた分散並列計算環境を構築する手順を以下に示す．

まず，Java の実行環境がインストールされて，Jini ライブラリをコピーした OSS サーバを準備する．Jini ライブラリは OSS の機能を利用・提供するライブラリである[13]．また，JSGrid 実行環境およびデーモンの配布に用いる Web サーバを準備する．Web サーバには，JSGrid 実行環境と JWS 定義ファイルを保存しておく．JSGrid 実行環境には，Jini および JSGrid 独自のライブラリが含まれる．並列計算を依頼するユーザ(依頼ユーザ)は，Master および Worker を作成する．また，Master を実行する計算機に，Java 実行環境をインストールし，JSGrid 実行環境をコピーする．Worker を実行する計算機は，Java の実行環境があればよい．JSGrid 実行環境，およびデーモンは JWS によって自動的にダウンロードされるため，インストールは不要である．

2.2.3. 並列計算の実行手順および処理の流れ

JSGrid における分散並列処理の流れを図 2 に示す．

step 1: 依頼ユーザは，Master を実行する．

step 2: 並列計算を引き受ける計算機のユーザ(提供ユーザ)は，Web サーバから JSGrid 実行環境を読込む．JSGrid 実行環境がダウンロードされると，JSGrid 実行環境に含まれるデーモンが，JWS によって自動的に起動する．デーモンは OSS に Worker を見つけると Worker をダウンロード(read)して，起動する．

step 3: Master は，処理対象を Task に分割し，Task を OSS に書き込む(write)．

step 4: Worker は OSS を監視し，Task が存在する場合はダウンロード(take)し，計算を行う．Task を処理することで得られた Result は，Worker によって OSS へ書き戻される(write)．

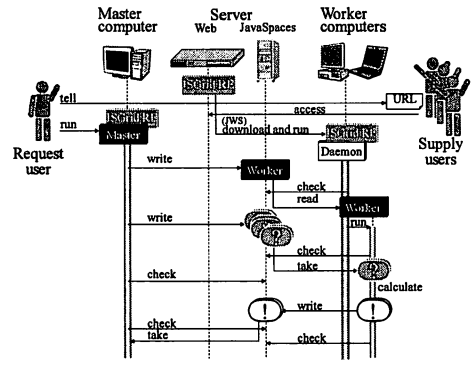

図 2.JSGrid における処理の流れ

step 5: Master は OSS を監視し，Result が存在する場合はダウンロード(take)する．

step 6: Master が，すべての Task に対する Result を回収し終えるまで，step 4～6 を繰り返す．すべての Result を回収し終えた場合は，OSS 上の Worker を削除し，計算完了を表す Task を OSS に書き込む．

step 7: Worker は計算完了の Task を得ると自身を破棄．

提供ユーザは，並列計算に参加させる計算機上で，依頼ユーザが指定する URL にアクセスすることで，JSGrid 実行環境のデーモンを起動できる．途中で並列計算から離脱する場合は，デーモンのインタフェースウィンドウ内に表示されている中止ボタンを押すことで，デーモンおよび Worker を停止できる．デーモンが停止すると計算機は並列計算参加前の状態に復帰する．

JSGrid は，同時に複数の並列計算を受け付ける．すなわち，複数の開発ユーザが複数の Master を立ち上げ，同時に並列計算を行うことができる．

2.2.4. フォルトトレランス

JSGrid は簡単なフォルトトレランス機能を有する．すなわち Master は，Worker が take した Task のうち，一定時間が経過しても Result が OSS に書き戻されない Task を発見した場合，該当する Task をもう一度 OSS に書き込む動作をする．

2.3. JSGrid で動作する並列プログラム

既存のプログラムから JSGrid で動作するプログラムを作成する際には以下の手順を踏む

1.既存のプログラムの並列可能な処理の部分を Worker プログラムとする

2.Task と Result を定義する

3.既存のプログラムの並列可能部分を，Task 作成と Result 処理とに置き換え，Master プログラムとする

JSGrid は Master,Worker を作成するためのクラスを用意しており，そのクラスをユーザが継承することで並列プログラムを容易に作成できる．以降に Master プログラムと Worker プログラムの作成について示す．

2.3.1. Masterプログラムの作成

Masterの動作のうち，ユーザが直接意識しなければならない2つの流れを図3に示す．図中，灰色で示した動作は自動で処理され，白色背景の動作がユーザが記述可能な処理ある．このうち，ユーザは最低限「配布するTaskの作成処理」と「回収したResultの処理」の2つの動作を記述すれば，分散並列処理が可能となる．

1．配布するTaskの作成処理(図4)

配布するTaskの作成は，createEntry()またはcreateEntries()メソッドをオーバライドして記述する．このメソッドは，共有空間のTaskの数が少なくなると呼び出され，戻り値であるエントリがOSSに書き込まれる．メソッドの引数は毎回変化する．

2．回収したResultの処理(図5)

回収したエントリの処理は，processEntry()メソッドをオーバライドして記述する．このメソッドは，ワーカによってOSSに書きこまれたResultを得る毎に呼び出される．

2.3.2. Workerプログラムの作成

Workerの動作のうち，ユーザが直接意識しなければならない1つの流れを図3に示す．図中，灰色で示した動作は自動で処理され，明るい背景の動作がユーザが記述可能な処理である．このうちユーザは「受け取ったTaskの処理」の動作を記述すれば，分散並列処理が可能となる．

1．Masterから受け取ったTaskの処理(図6)

Masterから受け取ったTaskの処理は，processEntry()メソッドをオーバライドして記述する．メソッドは，OSSに書きこまれたTaskを得る毎に呼び出される．

2.3.3. エントリへの情報の記憶

エントリ(Task,Result)は，Masterがwriteした時のstage(図3,4)を継続して記憶している．String, int, long, double, boolean, byte ,Object型の記憶領域を1つずつ持ち，それぞれの型をそれぞれの記憶領域に，単体または配列(1次,2次)のいずれか1ずつを記憶可能である．また，ファイルも1つ記憶でき，zip,jar形式の圧縮展開に対応する．

2.4. プログラムの例

「0以上1000未満の整数から7の倍数を見つけて表示する」動作をするMasterプログラムの例を図7に，Workerプログラムの例を図8に示す．

3. 評価実験

本稿で提案する分散並列環境JSGridの有効性を検証するため，評価実験を行った．なお，すべての計算機にはJavaの実行環境が備わっているものとする．

3.1. 準備

実験では，特定のユーザが使用する個人用の計算機

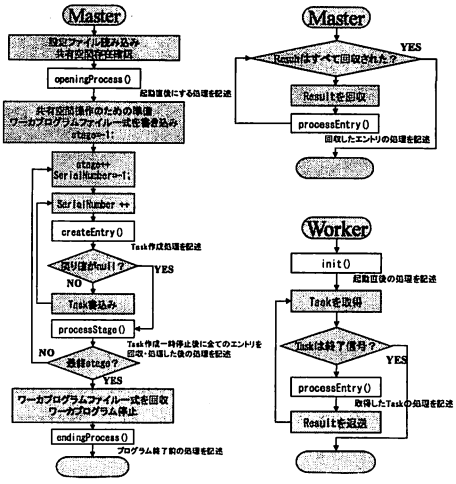

図 3. Master(左,右上)とWorker(右下)の動作(一部)

メソッドの種類

配布するエントリの作成メソッドは2種類あり，どちらかのメソッドを実装すればよい．

・常に1つのエントリを作成する
public JSGridEntry createEntry(int stage, int serialNum){}
・複数のエントリを作成する
public JSGridEntry[] createEntries(int stage, int serialNum){}

引数

serialNum:

エントリ作成メソッドは繰り返し呼び出される．呼び出される度にserialNumは+1増える．最初は「0」．値がInteger.MAX_VALUEに達すると次は「-1」となり，再び+1ずつ増える．
また，もう一つの引数"stage"の値が変化するとserialNumは再び「0」から開始する．

stage:

過去にnullで終了した回数．最初は「0」．「stage>最終stage(デフォルトは0)」となった時点でエントリ作成を完全に終了する．stageがInteger.MAX_VALUEに達すると次は「-1」となり，再び+1ずつ増える．
最終stageはsetFinalStage(int stage)メソッドで設定する．

戻り値

Workerに配布する(共有空間に書きこむ)エントリまたはその配列

null値を戻すとマスタプログラムは「いったん」書き込みを終了する(エントリ作成は一時停止される)．デフォルトの設定では一時停止すると，それまで作成・配布したエントリがすべて回収されるまでエントリ作成は再開されない．

図 4.配布するTaskエントリの作成処理メソッド

メソッドの種類

・Workerから戻ってきたエントリを処理する
public void processEntry(JSGridEntry ent){}

引数

ent:

Workerから戻ってきたエントリ

戻り値

なし

図 5.回収したTaskエントリの処理メソッド

メソッドの種類

・共有空間から持ってきたエントリを処理する．
public abstract boolean processEntry(JSGridEntry ent){}

引数

ent:

共有空間から持ってきたエントリ

戻り値

trueのとき，引数であるentをResultとしてMasterに返す．

図 6. Masterから受け取ったTaskエントリの処理メソッド

からなるクラスタ環境Eを用意した．クラスタを構成する計算機の一覧を表1に示す．Eの計算機は，複数の研究室に設置され，様々なLANに接続された計算機群であり，管理者権限を持たない計算機も含まれる．E内では並列計算以外の通信も行われている．表1で，S_1,S_2,S_3はそれぞれ独立したプライベートLAN，S_4はインターネットに直接接続されたLANで，OSSはS_4に配置した．並列計算を請け負う計算機は動的に参加および離脱を行うものとし，各計算機の参加スケジュールを表2に示す．約3日間実験を行い，毎日9時30分，13時30分，18時30分に計算機が増減する．

この実験では，計算機の参加台数による処理能力の変化を観測するため，不定方程式[14]の解探索プログラムP_{DE}を用いて実験を行った．P_{DE}はTaskごとの計算時間はほぼ一定となる．実験では，1時間あたりに処理するエントリの数をスループットと定義して，環境の変化に対するクラスタ全体の処理能力の変化を観察する．表1にある実験に用いる個々の計算機のスループットは，予備実験により求めた．

PDEは，n = 788，671，148について，以下の式

$n = x^3 + y^3 + 2z^3$　　(1)

を満たすx，y，zの整数解を，以下の手順で探索するプログラムである．まず，|x|，|y|，|z|の中で最大となる値a(a = max{|x|, |y|, |z|})を1つ定め，x，y，zを変化させて式(1)を満たす解を探索する．見つからない場合はaを変化させて再度探索を行う．aの値は7,798,336から1ずつ増加させ，1つのaの値についての探索を1つのTaskとした．

3.2. 実験結果

表2に示す計算機の参加離脱スケジュールに従って環境を変化させたときのクラスタ全体のスループットの変化を図4に示す．灰色の線が予備実験の結果をもとに算出したスループットの予測値であり，黒色の線が実際のスループットである．計算機が増加すると，予測値よりも性能が若干低下するものの，全体を通して，予測値に近い結果が得られている．よって，JSGridは，各計算機の処理能力に応じてTaskの分配を行えており，計算機の増減にも適切に対応できていることがわかる．t_2からt_3への移行の際に，観測値が予測値を上回っているが，これは，計算機毎に参加や離脱の時刻に最大で10分程度の差があったためと考える．

なお，本実験を含め5,000,000から10,387,947までのaの値について計算を行ったが，この範囲には条件を満たす解が存在しなかった．

3.3. 考察

3.3.1. 並列化効率

実験結果から，JSGridは，OSやネットワークの構成に依存せずに大規模計算の並列化を行えることがわ

```
import takeda.jsgrid.JSGridEntry;
import takeda.jsgrid.master.JSGridMaster;
public class SampleMaster extends JSGridMaster{
  public JSGridEntry[]
            createEntries(int stage,int serialNum){
    if ( serialNum < 10 ){
      JSGridEntry[] ret;
      int[] vals = new int[100];
      int count = serialNum*100;
      for(int ii = 0; ii < 100; ii++){
        vals[ii] = count;
        count = count+1;
      }
      //値の配列から一気にエントリを作成
      ret = getEntries(vals, "SampleProgram");
      return ret;
    }else{
      return null;
    }
  }
  public void processEntry(JSGridEntry ent) {
    if( ent.getString.equals("bingo!") )
      System.out.println( ent.getInt() );
  }
  public static void main(String[] args){
    (new SampleMaster()).start();
  }
}
```

図 7.Masterプログラムの例

```
import takeda.jsgrid.JSGridEntry;
import takeda.jsgrid.master. JSGridWorker;
public class SampleWorker extends JSGridWorker{
  public boolean processEntry(JSGridEntry gottenEnt){
    if( judge(gottenEnt.getInt()) )
      gottenEnt.setString("bingo!");
    else
      gottenEnt.setString("");
    return true;
  }
  Public boolean judge(int val){
    return(val % 7 == 0);
  }
}
```

図 8.Workerプログラムの例

表 1. 環境Eを構成する算機

ID	CPU type	Freq. [GHz]	Mem. [GB]	OS	Throughput [Entries/h]	Network Segment
C_1	Celeron	0.63	0.26	Windows	7	S_1
C_2	AthlonXP	1.8	1.02	Windows	28	S_2
C_3	AthlonXP	1.8	1.02	Linux	28	S_3
C_4	PowerPC G4	0.45	0.13	MacOS X	5	S_1
C_5	Pentium4	2.5	0.51	Windows	28	S_1
C_6	Pentium4	3.0	1.02	Windows	16	S_4

表 2.時間毎の各計算機の参加台数

ID	Number of Computers									
	13:30… t_1	18:30… t_2	9:30… t_3	13:30… t_4	18:30… t_5	9:30… t_6	13:30… t_7	18:30… t_8	9:30… t_9	13:30… t_{10}
C_1	5	3	10	5	5	0	0	10	7	0
C_2	2	4	0	2	4	2	1	4	4	2
C_3	0	0	5	0	8	8	3	3	0	0
C_4	0	0	0	0	1	1	1	0	0	0
C_5	2	5	3	0	5	2	2	4	4	1
C_6	0	0	0	0	1	0	0	0	0	0

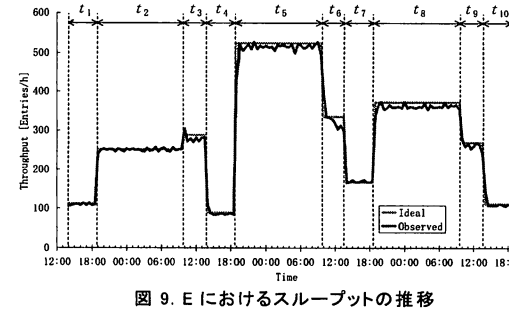

図 9. Eにおけるスループットの推移

かる．マスタ-ワーカ型の並列計算環境においては，各タスクの計算時間が通信時間よりも十分大きいことが並列化効率を維持する条件の 1 つだが，この条件を満たす場合であっても，Worker を実行する計算機を増加させた場合に，OSS がボトルネックとなって並列化効率が頭打ちとなる可能性がある．JSGrid では，OSS として JavaSpaces を用いているが，より多くの Worker を利用する場合は，JavaSpaces を複数設置して連携する，負荷分散機能を持つ他の OSS に置き換えるなどすることで，OSS の負荷分散を容易に実現できると考える．

3.3.2. さまざまな実行環境への対応

E における計算機群 C_3 は，MPI を利用可能な既存の計算機クラスタである．JSGrid は既存の計算機クラスタにおいて，MPI プログラムと同様の手順で並列プログラムを実行することも可能である．すなわち，ファイルシステムを共有した状態であれば，JSGrid 実行環境の配布が不要であり，簡単なスクリプトを記述することで，1 台の計算機のコマンドラインから，複数の計算機でデーモンを一斉起動することが可能である．

また，一度 Java Web Start によって Worker を起動すると，JSGrid の実行環境がキャッシュに保持されるとともにショートカットが作成され，二度目以降は 1 回の操作でデーモンを起動できる．ショートカットから JSGrid を起動する場合は，JSGrid 実行環境が最新であるかが自動的にチェックされる．

4. おわりに

オブジェクト共有空間を分散共有メモリとして用いる分散並列処理環境 JSGrid を提案した．JSGrid は，開発用，事務処理用などの特定のユーザが使用する計算機，教育用などの不特定多数のユーザが使用する共用の計算機を，容易により多く並列計算に参加させることを主眼とする．JSGrid の特徴を以下にまとめる．

- Java が動作する全ての計算機を並列計算に参加させることができ，OS に依存した事前準備は不要である．
- Worker を実行する計算機において，実行環境のインストールが不要であり，管理者権限を持たないユーザであっても，計算機を並列計算に参加させることができる．
- 「Web ブラウザの起動」「URL の入力」「セキュリティ確認」の高々3 回の操作で計算機を並列計算に参加させることができる．
- Worker を実行している計算機において，デーモンを停止することで，並列計算に参加する前の環境に即座に復帰する．
- 並列計算中に，計算機の追加，離脱を容易に行うことができる．
- JavaSpaces や JavaSpaces 操作に関連するライブラリの知識がなくとも容易に並列プログラムを記述でき，ライブラリが更新された場合にも並列プログラムの変更が不要である．

複数の異なる環境の計算機を用いて評価実験を行い，JSGrid は並列化効率の高い並列計算を実行できること，および，計算機の動的な追加，離脱にも対応できることを確認した．

今後，OSS の負荷分散に関する検討，分散並列言語 Espace[15]による JSGrid 用ソースコードの自動生成，などについて検討を行う．また，通信経路のセキュリティ，アクセス権制御などを付加することで，地理的に分散したより多数の計算機からなる環境でも利用可能なグリッド環境構築ツールキットとして発展させる

文　献

[1] PVM: Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/pvm_home.html

[2] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Technical Report UTCS-94-230, 1994.

[3] Foster, I., Kesselman, C., and Tuecke, S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of High Performance Computing Applications, Vol. 15, pp. 200-222, 2001.

[4] 谷村勇輔，廣安知之，三木光範，グリッド計算環境でのマスタ- ワーカシステムの構築，情報処理学会論文誌: コンピューティングシステム，Vol.45, No.SIG6, pp.197-207, 2004.

[5] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit, The International Journal of Supercomputer Applications and High Performance Computing, Vol. 11, No. 2, pp. 115-128, 1997.

[6] Grid MP Overview, http://www.univaud.com/products/grid-mp/

[7] Condor Project Homepage, http://www.cs.wisc.edu/condor/

[8] FightAIDS@home, http://fightaidsathome.scripps.edu/

[9] BOINC, http://boinc.berkeley.edu/

[10] JavaSpaces Service Specification, http://java.sun.com/products/jini/2.1/doc/specs/html/js-spec.html

[11] Java Web Start Technology, http://java.sun.com/products/javawebstart/, 2005.

[12] Gelernter, D., Generative Communication in Linda, ACM Trans. Program. Lang. Syst., Vol. 7, No. 1,pp. 80–112, 1985.

[13] Jini Network Technology, http://www.sun.com/jini/

[14] Koyama, K., On searching for solutions of the Diophantine equation $x^3 + y^3 + 2z^3 = n$, Mathematics of Computation, Vol. 69, pp. 1735-1742, 2000.

[15] 岩川建彦，小野智司，中山茂，"分散並列処理プログラミング言語 Espace の開発"，システム制御情報学会論文誌，Vol.19, No.7, pp.296-298,2006

[16] 武田和大,小野智司,中山茂,異機種混合並列計算ミドルウェア JSGrid の開発と評価,日本計算工学会論文集, Vol.8, No.20060005, pp.211-221,2006