計算機アーキテクチャ 142−26
ハイパフォーマンス
コンピューティング 85−26
（2001．3．9）

# JavaによるPCクラスタソフトウェアの開発と評価

坂口聡 † 小畑正貴 †

本稿ではJavaによって作成されたPCクラスタ・ソフトウェアの開発とその簡単な評価について述べる．本ソフトウェアはJava実行環境のみで実行可能である．したがって，本PCクラスタ・ソフトウェアはすべてのコンピュータで運営することが可能で，他のPCクラスタと比較すると導入が単純である．本PCクラスタはMPICHと比較するとデータ転送速度が約1/10である．本ソフトウェアを使用したPCクラスタは，データ転送を頻繁に行わない計算でMPICHの0.2〜0.7倍の性能を発揮した．

# Implementation and Evaluation of PC cluster software written by Java

AKIRA SAKAGUCHI † and MASAKI KOHATA †

In this paper, we present an implementation and a simple evaluation of a PC cluster software written in Java alone. This software needs a Java runtime environment only. So that, this PC cluster software is possible to run under all computers and an installation is easyer than another PC clusters. This PC cluster shows 1/10 performance of a data transfer speed compared with the MPICH. The performance of a PC cluster with this software shows 0.2 to 0.7 times as much as the MPICH.

## 1. はじめに

新しくPCクラスタを構築する場合はすべて同じハードウェアで同じOSにすることが容易である．しかし，すでに導入されているPCをPCクラスタとして利用する場合は困難となる．計算機を複数のOSが起動できるように設定し，新たにOSをインストール，PCクラスタを構築することも可能であるが，アーキテクチュアの異なる計算機を混在させる場合，より実現が困難となる．既存のデータを誤って消去してしまうなどといったいくらかの危険もある．したがって，すでに他の目的で利用している計算機をPCクラスタとして利用することは手間のかかるものとなっている．

本稿では安全に既存の計算機をPCクラスタとして利用するためのPCクラスタ・ソフトウェアとその簡単な評価について述べる．本PCクラスタ・ソフトウェア（以下，本ソフトウェア）の作成と実行にはJava[3]を利用している．既に同様の研究[1]が成されているが，MPIタスクの実行などにネイティブ・メソッドの実行を行っている．本研究のPCクラスタ・ソフトウェアは各計算機上にあらかじめMPIタスクを実行するサーバを実行しておくことによって，ネイティブ・メソッドを呼び出すこと無くMPIタスクを実行する．

## 2. Javaについて

Javaは Sun Microsystems, Inc. が開発し，1995年に公開したプログラム言語であり，Java言語を中軸とした関連技術の総称でもある．Java言語の特徴には以下のようなものがある．

- C++に似た文法を持つ
- 国際化を考慮
- マルチスレッドのサポート
- ガーベジコレクションの自動化
- プラットフォーム独立

ここで注目したのが“プラットフォーム独立”という特徴である．これを利用すれば計算機のアーキテクチュアやOSに依存しないPCクラスタが構築できる．C言語でデータのビット数が計算機に依存することは有名である．しかし，Javaにおいてデータサイズは固定である．8bitの計算機であってもint型は32bitのデータサイズを持つ．この点もPCクラスタの構築を行う場合の利点である．

Java発表当時，Javaは低速であるとされていた．

† 岡山理科大学工学部
Okayama University of Science, Faculty of Engineering

比較的高速な実行環境も存在したがネイティブコードには遠く及ばない状況であった．しかしながら IBM Corporation の実行環境[2] や Sun Microsystems, Inc. の HotSpot[4] 等，Java の実行環境は確実に高速化してきている．

そこで整数，浮動小数それぞれについて加法計算を $10^8$ 回ずつ行ったところ，図 2 のような結果となった．早いものから IBM JDK 1.2.2, gcc 2.95.3 の最適化オプション付きネイティブコード，Blackdown JDK 1.2.2, Sun J2SDK 1.3.0, Sun J2SDK 1.3.0_01, gcc 2.95.3 ネイティブコード，Blackdown JDK 1.1.8_v3, Sun JDK 1.2.2 となっている．Java の実行速度はネイティブコードにかなり迫っていることがわかる．

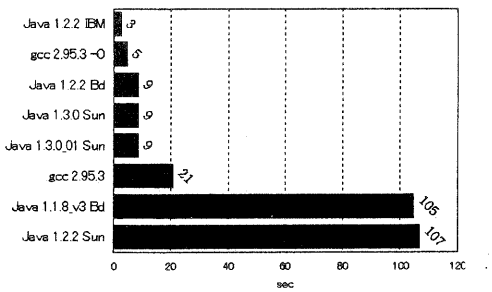


図 1　実行速度

## 3. Java による PC クラスタ・ソフトウェア

今回作成した Java による PC クラスタ・ソフトウェアは図 3 のような構成となっている．本ソフトウェアで構築される PC クラスタ (以下，本 PC クラスタ) は分散メモリ型の PC クラスタである．ユーザプログラムは CalculatorThread クラスを継承して作成される．ユーザプログラムは CalculatorClassLoader クラスによって CalculatorLoader プログラムにロードされる．引続き CalculatorLoader プログラムは各計算機が通信可能かつ計算中でないかを調べ，問題がなければユーザプログラムを各計算機へ転送する．ユーザプログラムは最終的に CalculatorServerImpl クラスが実行する．ユーザプログラムから各計算機へのデータ転送等は CalculatorServerImpl のメソッドを直接接続することによってなされる．いくつかの MPI ライクな命令を実装した MPI クラスのメソッドを利用することもできる．

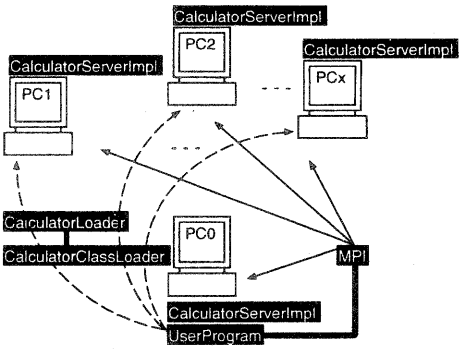


図 2　構成

### 3.1 CalculatorThread クラスと CalculatorThreadInterface インタフェース

CalculatorThread クラスとそのインターフェース CalculatorThreadInterface クラスは図 3.1 のような構造となっている．CalculatorThreadInterface クラスは計算の終了条件を監視するメソッド resultObserver() や初期データのロードを行うメソッド loadData()，スレッドの実行を開始するメソッド startThread()，スレッドの実行を終了するメソッド stopThread() を外部クラスから呼び出すためのインターフェースを提供する．このインターフェースは主に CalculatorLoader クラスや CalculatorServerImpl クラスが利用する．台形公式のサンプルプログラムを付録に示す．

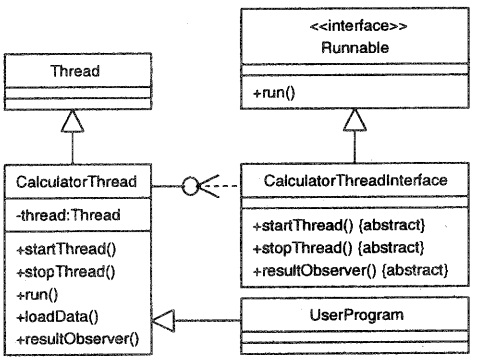


図 3　CalculatorThread クラスとその関連クラス

### 3.2 CalculatorClassLoader クラス

CalculatorClassLoader クラスは図 3.2 のような構造となっている．ユーザプログラムをバイト列としてロードする loadClassBytes() メソッド，必要なときに

クラスとして利用する defClass() メソッド，バイト列の取得・設定を行う setClassBytes()/getClassBytes() メソッドを提供している．

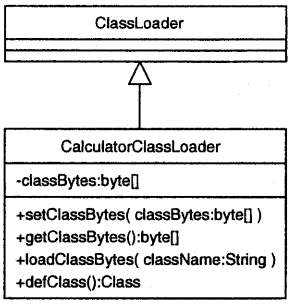


図 4 CalculatorClassLoader クラス

### 3.3 CalculatorServerImpl クラス，CalculatorServer インターフェース，MPI クラス

CalculatorServerImpl クラスと CalculatorServer インターフェースは本 PC クラスタの中核をなすものである．CalculatorServerImpl クラスと CalculatorServer インターフェースは図 3.3 のような構造となっている．CalculatorServerImpl クラスは分散メモリ型の PC クラスタを実現するクラスであり，rsh などのリモートシェルの代わりに MPI タスクとしてユーザプログラムを実行するクラスでもある．他の計算機の初期化を肩代わりして行う機能 (representationInit() メソッド) も有している．この機能は他の複数の計算機を自分と同じ設定で初期化する機能である．

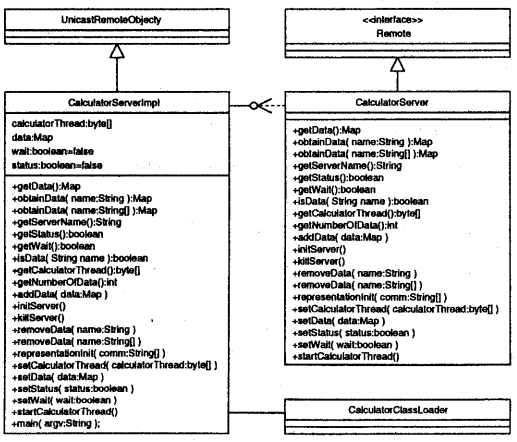


図 5 CalculatorServerImpl クラスとその関連クラス

データはすべてキーをその値にマッピングするオブジェクトに変換して扱われる．これによってすべてのデータを同じ手法で扱うことが可能になる．データの操作は操作対象となるデータのキー名を指定することによって行われる．この操作を MPI ライブラリライクにするために MPI クラス (図 3.3) を用意してある．MPI クラスは Static に呼び出されて使用される．このクラスは開発中であり頻繁に変更が加えられている．

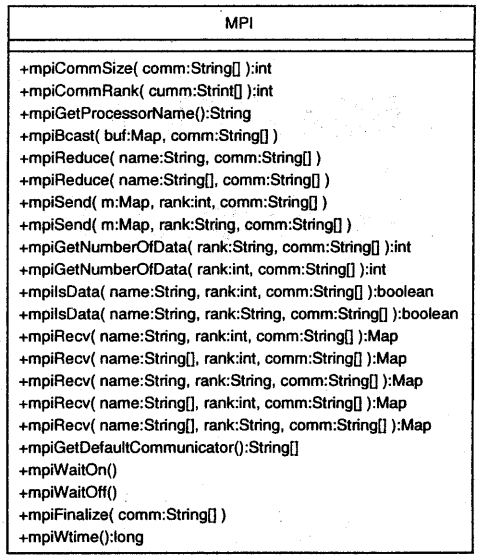


図 6 MPI クラス

データの送受信は実行されているサーバのデータに対して行われ，ユーザプログラムが送信されたデータを直接受信することは無い．ユーザプログラム同士で通信を行いたい場合は，送信側のユーザプログラムからいったんどこかのサーバにデータを送り，そのデータを受信側のユーザプログラムで受信する．その際，受信側のユーザプログラムがサーバへデータの存在を確認する必要があるが，その機能は CalculatorServerImpl クラスの isData() メソッドで提供されている．

CalculatorServerImpl クラスは RMI[5](Remote Method Invocation) を利用して通信を行う．そのため，CalculatorServerImpl クラスの実行前に RMI サービスを管理する rmiregistry を起動しておく必要がある．RMI を利用すると遠隔地のクラスのメソッドを直接に実行することができる．本ソフトウェアでは計算機同士の通信に RMI を利用することでコーディングの簡略化を行った．より高速な分散オブジェクトを提供する HORB[6] を利用することでデータ転送の

高速化[7] が望める．しかし，HORB は Java 実行環境とは別にインストールが必要なため，今回は使用していない．1000byte のデータを 1000 回転送するのに要した時間は図 3.3 のようになった．ローカルホストとリモートホスト共に MPICH に遠く及ばない結果となっている．この速度差の理由は RMI の使用やデータの管理の際にオブジェクトの生成が大量に発生するためと思われる．

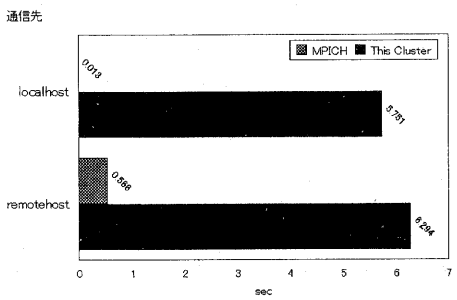


図 7　転送速度

### 3.4　CalculatorLoader クラスと計算機の初期化

CalculatorLoader クラスは指定されたユーザプログラムと計算機のリストを元に各計算機を初期化する．各計算機で実行されている CalculatorServerImpl の初期化手順 (図 3.4) を知っているならば，CalculatorLoader クラス以外の独自に作成した初期化プログラムを使用することもできる．初期化手順は先ず利用する計算機のリストを取得，次に計算機の状態を調べる．そして，ユーザプログラムと初期データを各計算機へ転送，各計算機はユーザプログラムを実行する．最後に CalculatorLoader クラスは計算終了を待つ．

## 4.　性 能 評 価

MPICH との性能比較図だが，図 3.3 の通り通信速度に問題があるため通信をあまり行わない台形公式の計算で行った (図 4)．実験環境は CPU が PentiumIII 677MHz の計算機 4 台で，100Base-T のイーサネットで接続されている．OS として Linux(Kernel 2.2.16)，Java 実行環境として Sun J2SDK 1.3.0_01 をインストールして使用した．MPICH のとき，4 台目は他の 3 台と同様の Linux マシンを利用した．本 PC クラスタのとき，4 台目は Windows98 と IBM JDK 1.2.2 をインストールした計算機を使用した．本 PC クラスタは動作に大きなムラがあり，最大で MPICH の約 0.7 倍，最小で約 0.2 倍の計算速度を記録した．この動作速度のムラの原因は各計算機で実行されている

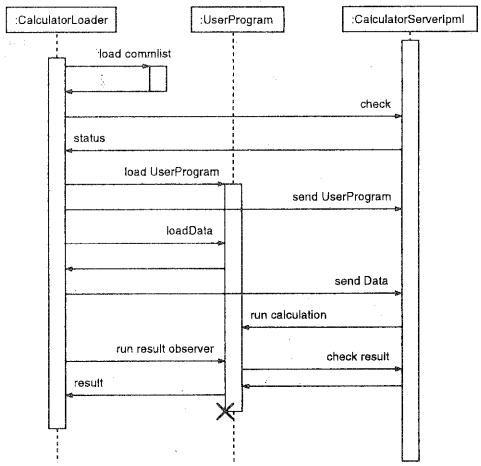


図 8　初期化手順

CalculatorServerImpl クラスの実行状況に依るものと思われる．

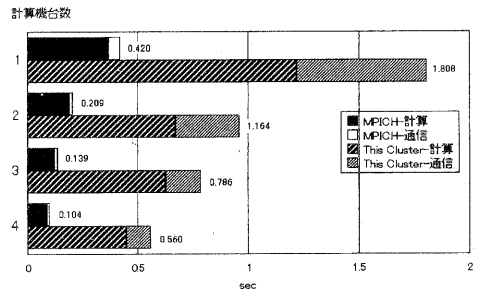


図 9　台形公式

## 5.　お わ り に

本稿では Java による PC クラスタ・ソフトウェアとその簡単な評価について述べた．

本 PC クラスタの今後の課題として通信速度の改善とそのサポートクラスである MPI クラスの充実があげられる．TCP 通信や HORB を使用した場合等の性能を比較する必要がある．RMI は通信をするためだけの技術ではないため通信速度は遅い．そこで java.net パッケージ[8] のクラスで TCP 通信を行うことにより，動作の高速化が期待できる．あるいは RMI より高速である HORB の利用も検討したい．

更に今後の拡張としては携帯電話をはじめとする携帯情報端末を使用したクラスタの構築を考えている．家電製品・組み込みデバイス向けの Java 実行環境の規格，特に携帯電話を主体にした J2ME[9] MIDP(Mobile

Information Device Profile) では浮動小数点計算をサポートしていない．ハードウェアも最小限しか利用できない．これを本ソフトウェアの拡張で補うことができれば，一台が携帯電話程度の安価な並列計算機が実現されるだろう．

## 参 考 文 献


1) 日下部明, 廣安知之, 三木光範:“Java による MPI の実装と評価”,2000 年記念並列処理シンポジウム JSPP2000 論文集,pp.269-276(2000)
2) IBM Corporation,“developerWorks: a world-wide resource for developers”, http://www-106.ibm.com/developerworks/
3) Sun Microsystems, Inc.,“java.sun.com - The Source for Java Technology”, http://java.sun.com/
4) Sun Microsystems, Inc.,“Java HotSpot Technology”, http://java.sun.com/products/hotspot/
5) Sun Microsystems, Inc.,“Java Remote Method Invocation”, http://java.sun.com/products/jdk/rmi/
6) ETL,“HORB Home Page”, http://www.horb.org/
7) ETL,“HORBって何?”, http://www.horb.org/horb-j/doc/what.htm
8) Sun Microsystems, Inc.,“Java 2 Platform SE v1.3: Package java.net”, http://java.sun.com/j2se/1.3/docs/api/java/net/package-summary.html
9) Sun Microsystems, Inc.,“J2ME - JavaTM 2 Platform, Micro Edition”, http://java.sun.com/j2me/


## 付　　録

### A.1 台形公式サンプルプログラム

```
//使用パッケージを import
import java.rmi.Naming;
import java.util.HashMap;
import java.util.Map;
import jp.gr.java_conf.TMA.dunwich.*;

//台形公式サンプルプログラム
public class Daikei extends CalculatorThread
        implements CalculatorThreadInterface {

    Thread thread = null;

    //コンストラクタ.
    public Daikei() {
        super();
    }

    //コンストラクタ.
    public Daikei( Runnable target ) {
        super( target );
    }

    //スレッドを開始する.
    public void startThread() {

        thread = new Thread( this );
        thread.start();

    }

    //計算内容を記述する.
    public void run() {

        try {

            Map m = new HashMap();
            double ave,x,y,h,a=0.0,b=1.0,
                I=0.0,J,e,N=1000000;
            double  start, end;
            double stock = 0;

            //デフォルトの通信空間を取得する.
            String[] comm
              = MPI.mpiGetDefaultCommunicator();

            //計算機のランクを取得する.
            int rank = MPI.mpiCommRank( comm );

            //通信空間のサイズを取得する.
            int size = MPI.mpiCommSize( comm );

            //台形公式の計算ここより
            h = ( b - a ) / N;
            ave = N / size;

            if( rank == ( size - 1 ) ) {
                stock = ( double )( N % size );
            }

            start = ave * rank;
            end = ave * ( rank + 1 );

            for( double i = start + 1;
                i <= end + stock; i++ ) {

                x = a + ( double ) i * h;
                y = ( double )Math.exp( x );

                if( ( i == 0 ) || ( i == N ) ) {
                    I += y * h / 2;
                }
                else {
                    I += y * h;
                }

            }
            //台形公式の計算ここまで

            //Map にデータを格納
            m.put( "data" + rank, "" + I );
```

```
            //データを送信
            MPI.mpiSend( m, comm[ 0 ], comm );

        }
        catch( Exception e ) {
            System.err.println( e );
        }

    }

    //データの初期化コードを記述する.
    public Map loadData() {

        //空のデータを作成する.
        Map data = new HashMap();

        //初期データを返す.
        return data;

    }

    //計算の終了を監視するコードを記述する.
    public void resultObserver() {

        try {

            //デフォルトの通信空間を取得する.
            String[] comm
              = MPI.mpiGetDefaultCommunicator();

            //計算機のランクを取得する.
            int size = MPI.mpiCommSize( comm );

            double a = 0.0;

            //データの数を確認するまで待つ
            while( MPI.mpiGetNumberOfData(
              comm[ 0 ], comm ) != size + 1 ) {
            }

            //データを受信し, 合計する.
            for( int c = 0; c < size; c++ ) {
                a += Double.valueOf(
                    ( String )MPI.mpiRecv(
                        "data" + c, comm[ 0 ],
                        comm ).get( "data" + c )
                        ).doubleValue();
            }

            //結果の出力.
            System.out.println( a );

        }
        catch( Exception e ) {
            System.err.println( e );
        }

    }

}
```