計算機資源の流通および集約のためのP2Pミドルウェア

首藤一幸[†] 大西丈治[†,☆]
田中良夫[†] 関口智嗣[†]

PCをはじめとする個人情報機器の計算能力やストレージといった資源を，個人間で融通，共有し，ひいては集約して分散処理を行うためのミドルウェアP3の設計と実装を述べる．自由な並列プログラミングを可能とするために，通信には，オーバレイネットワークを提供するP2P通信ライブラリ具体的にはJXTAを採用した．本論文では，計算機資源の授受や，集約しての分散処理という目的に対して，P2P通信ライブラリの提供するピアグループや発見などの諸概念をいかに適用するかを提案する．一方，P2P通信ライブラリの諸機能は通信性能を犠牲にしていることが予想される．そこで，基本性能とアプリケーションの実行性能を測定し，高スループット計算という目的への適合性と，本ミドルウェアで効率の向上を図ることが可能なアプリケーションの条件を調べた．現実装は，100×10^6bpsの帯域幅を十分活用できること，部分問題あたりの処理時間が1秒程度であっても32台で20倍以上の性能向上率は達成できることが判った．一方で，部分問題1つの配布と回収に10～30ミリ秒を要しており，単一のジョブに数万台規模の計算機を参加させるためには通信遅延の低減が不可避であることも判った．

P2P-based Middleware Enabling Transfer and Aggregation of Computational Resources

Kazuyuki Shudo,[†] Joji Onishi,[†,☆] Yoshio Tanaka[†]
and Satoshi Sekiguchi[†]

This paper describes the design and implementation of P2P-based middleware for transfer and aggregation of computational resources. We adopted a P2P communication software, JXTA, as the communication library in order to allow application programmers to write programs in any model of parallel programming. P2P communication presents an overlay network where any computer can communicate with each other directly. In this paper, we proposes an application method of useful concepts that JXTA provides to goals of our middleware. The applied concepts are "peer groups" and "discovery", and our initial goals include transfer of resources and distributed high-throughput computation by aggregating those resources. On the other hand, it is natural to be inferred that such a P2P communication software imposes certain amount of overhead on our middleware in communication performance. Then we measured communication performance of parallel programming library our middleware provides and throughput of an application program in different conditions. The results of these experiments outlined conformity of the implemented middleware with high-throughput computation and desirable conditions of applications for improvement in performance. The current implementation of this framework can fully utilize a bandwidth of 100×10^6 bps and shows a speed-up ratio of over 20 times with 32 computers even in case that the granularity of subproblems is adequately fine as less than a second.

1. はじめに

計算能力やストレージなど，自らが必要とするだけの計算機資源を，個人や組織で所有しておくのではなく，ネットワークを通して融通，流通させようという取り組みが活発になっている．ソフトウェアをサービスとして売ろうというアプリケーションサービスプロバイダ（ASP）の出現や，電気，水道のようにサービスの利用量に応じた課金を行おうというユーティリティコンピューティングの提唱は，その顕れである．

PCクラスタなどの高性能かつ稀少な機器については，Globus Toolkit[1)]などグリッドミドルウェアの認証，認可機能を利用しての組織間での融通が始まって

† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
☆ 現在，アライドテレシス株式会社
Presently with Allied Telesis K.K.

いる．融通を受けた計算機や実験機器を連係させての，大規模な計算や実験が試みられてきた．

一方，PC などの個人情報機器については，資源の融通と連係は，様々な点について限定された形で行われている．利用権（アカウント）の発行による資源の提供では，提供側の手間が大きいうえに，機器間の連係も難しい．SETI@home[2),3)] や distributed.net[4)]，GIMPS[5)] に代表されるインターネット上分散処理プロジェクトでは，アプリケーションプログラムを用意できるのはプロジェクトの運営者のみであり，参加者は一方的に計算能力を提供するだけである．

そこで我々は，PC をはじめとする個人情報機器の資源をネットワーク経由で容易に授受し，自由な並列プログラミングでそれら計算機資源を連係，集約して活用するためのミドルウェア P3（Personal Power Plant）を開発している．資源提供者側は，自らの意志で，どういったジョブに資源を提供するのかを決めることができる．資源利用者側が受け取った計算機資源の利用法としては，まずは並列処理，特に高スループット計算を想定して開発を進めている．数十万台規模の大規模並列処理も目標ではあるが，資源授受が目的であるので，互いに計算能力を融通する数台から数百台のグループを少ない手間でアドホックに構築できることが重要となる．

現在のインターネットでは，通信可能な相手や通信の方向に制約があることが一般的であり，このため，TCP/IP を直接利用したのでは，並列処理の通信パターン，プログラミングモデルが制限されてしまう．我々は，通信に P2P 通信ライブラリの提供するオーバレイネットワークを利用することで，この問題を解決した．オーバレイネットワーク上では，任意の計算機間での双方向通信が可能となる．

また，P2P 通信ライブラリの提供する各種の概念は，ジョブや計算機の発見や，特定のジョブに関係する計算機群への同報通信などに自然に適用でき，有効に活用できる．本論文では，発見機構およびピアグループといった概念を本ミドルウェアの目的に対していかに適用するかを提案する．

一方で，P2P 通信ライブラリを用いた場合，上述の諸機能のために通信性能が犠牲になることが予想できる．そこで，本ミドルウェアが提供する並列プログラミング支援ライブラリの基本性能と，アプリケーション実行性能を評価した．支援ライブラリが，目的とする高スループット計算に耐えるものであるかどうかを確認し，本ミドルウェアで処理性能の向上を図ることのできるアプリケーションの条件を考察した．

続く 2 章では，P2P 通信ライブラリの提供する諸概念，諸機能を本ミドルウェアの目的に対していかに適用するかを提案する．3 章と 4 章は，2 章で提案した適用法に基づいたミドルウェアの設計と実装の説明である．3 章では，計算機情報やジョブの管理，ジョブの実行制御を行うツールやデーモンであるジョブ管理ソフトウェアについて説明する．4 章では，アプリケーションプログラマが並列プログラミングに用いる支援ライブラリについて説明する．5 章では，性能測定の結果を示して本ミドルウェアの性能を評価する．最後に，6 章で本ミドルウェアを関連研究と比較し，7 章で本研究の成果をまとめる．

2. P2P 通信ライブラリの活用法

昨今の IPv4 インターネットでは，通信の相手や方向に大きな制約がある．多くの計算機にプライベートアドレス空間から IP アドレスが割り当てられており，その場合，異なる LAN に属する計算機間では直接通信を行うことができない．また，NAT，NAPT ゲートウェイがあっても，通信の開始はプライベートな計算機側からに限定されている場合が多い．

通信の相手や方向に制約があったのでは，分散処理の際の計算機間通信のパターン，ひいては並列プログラミングモデルが限定されてしまう．たとえば，マスタ・ワーカ型並列処理に限定すれば，ワーカ間で直接通信できる必要はない．しかしそれでも，マスタ側プログラムは全ワーカとの通信が可能な計算機に割り当てる必要がある．つまり，ジョブ割当てに制約が課せられる．メッセージパッシングモデルの並列プログラミングまで許したければ，結局，任意の計算機間での双方向通信が必要となる．

2.1 P2P 通信ライブラリ JXTA

本ミドルウェアは自由な並列プログラミングを目標としている．そのため，通信相手，方向の制約は許容できない．そこで，オーバレイネットワークを提供する P2P 通信ライブラリを採用した．オーバレイネットワークは，現 IPv4 インターネットや Bluetooth など，下位のネットワークに通信相手や方向の制約があっても，その上位層で，任意の計算機（ピア）間での双方向通信機能を提供する．

P2P 通信ライブラリとしては，利用者が多いことや，利用するためのプログラミングインタフェースがほぼ固定されていることから，JXTA[6)] を採用した．双方向通信などのオーバレイネットワークの恩恵をあらゆる局面で享受するために，本ミドルウェアが行うすべての通信に JXTA を用いている．実際に，ファイ

アウォール上の NAPT ゲートウェイをまたいだ双方向通信と，それによる並列処理が可能であることを確認できている．

JXTA は 2001 年に Sun Microsystems 社が立ち上げたプロジェクトであり，P2P ソフトウェアに共通する機能を提供するプロトコルを規定している．現在，Java および C 言語の参照実装が配布されている．参照実装自体は，台数のスケーラビリティや処理効率の向上を目指して改良が続けられているが，JXTA の API は，2001 年末の時点でほぼ固定されている．JXTA を用いるプログラムの初回起動時を除いて，JXTA は集中サーバをまったく必要としない．初回起動時には，seed rendezvous と呼ばれるピアのネットワーク上位置を HTTP サーバから取得する．

JXTA が提供するオーバレイネットワーク上では，計算機はピア ID で識別され，通信メッセージの送信先はピア ID で指示される．ピアはピアグループを作成でき，複数のピアグループに参加できる．ピアグループ内では，他のピアやピアグループ，また，通信手段であるパイプを発見することができる．JXTA の実装自体は下位の通信プロトコルとして TCP や HTTP，IP マルチキャストなどを利用するが，JXTA の利用者はそれを意識する必要はなく，任意のピアを相手に reliable，unreliable 通信を行える．

このように，JXTA が提供する機能は様々な P2P ソフトウェアに共通するものなので，その API はおのずと低レベルな，たとえば BSD ソケット API 程度の層のものとなっている．これを直接用いて並列処理を記述することは大きな労力を要する．そこで本ミドルウェアは，JXTA API の上に，マスタ・ワーカ等の並列処理用 API を提供する．また，ジョブの投入，起動といった管理を行うためのソフトウェアも提供する．

2.2 JXTA の諸概念の適用法

オーバレイネットワーク上の双方向通信といった恩恵があるだけでなく，JXTA の提供する諸概念は，本ミドルウェアの要件に対して自然に適用でき，有効に活用できる．本節では，その適用法を提案する．

2.2.1 ピアグループ

並列処理では，複数の計算機に対する実行制御や，全計算機へのブロードキャストなど，同報通信がしばしば必要となる．これを 1 対 1 通信だけで行うと，台数に比例した多数回の通信が発生して効率が悪い．そこで，ジョブ（投入されたアプリケーションとデータの組）ごとにピアグループを作成し，そのジョブピアグループ中で，JXTA のピアグループ内同報通信機能を用いて同報通信を行う．

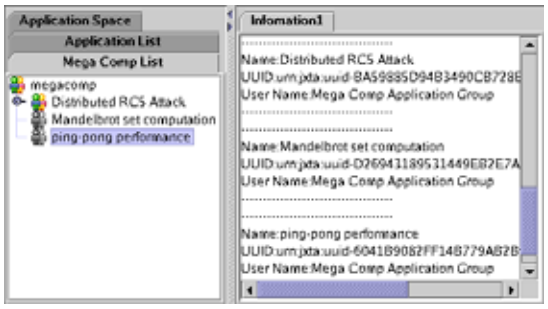


図 1 Host のシェルを用いた受け入れジョブの選択
Fig. 1 Job selection using Host's shell.

また，アプリケーション自体や入力ファイルの配布も，ジョブピアグループ内で行う．これらファイルの配布は，JXTA のファイル共有サービス CMS を用いて行う（3.1 節）．共有を始める際に行われる広報の到達範囲は，ジョブピアグループ内であり，ちょうど当該ファイルを必要とする計算機群と一致させることができる．

2.2.2 発　　見

資源提供者は，資源利用者が作成，広報したジョブピアグループを発見する．ジョブグループの広報，発見は，JXTA の発見機構を用いて行う．

また逆に，資源利用者も資源提供者を発見し，発見で得られた提供者リストを，ジョブ起動時に提供者に識別子 rank を割り振る際に利用する（4.2 節）．

発見機構によって，ジョブや，通信相手のネットワーク上位置（IP アドレス等）を実行時に発見できるので，これらの情報を人手で設定しておく必要はない．

3. ジョブ管理機構の設計と実装

本ミドルウェアは，ジョブ管理のためのデーモンおよびツールと，起動されたアプリケーションの並列処理を支援する各種のライブラリからなる．前者のソースコードが 9524 行，後者が 9620 行，合わせて約 2 万行のソフトウェアである．本章では前者を，次章では後者を説明する．

3.1 ジョブ管理ソフトウェア

アプリケーションプログラムの配布や，ジョブの起動，停止といったジョブ管理は，次のプログラム群を用いて行う．

Host　資源提供者が起動しておくデーモンプログラムである．ジョブのピアグループを発見し，資源提供者に提示する．提供者は，シェル（**図 1**）を用いて，実行を受け入れるジョブを選択できる．

Controller　資源利用者が使用するツールである．こ

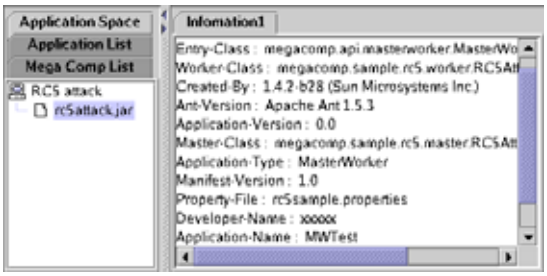


図 2 Controller のシェルを用いたジョブの投入
Fig. 2 Job submission using Controller's shell.

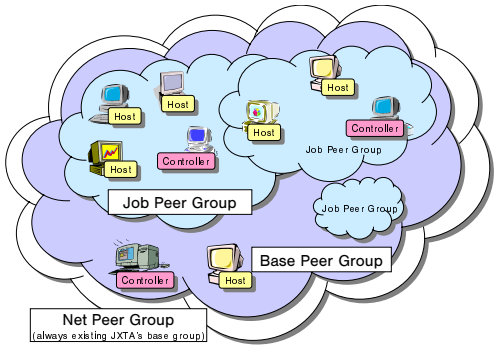


図 3 ジョブ管理ソフトウェアと関係するピアグループの構成
Fig. 3 Organization of job management software and related peer groups.

れを用いて，アプリケーションプログラムを指定し，ジョブとして投入する（**図 2**）．

Host と Controller が参加するピアグループを**図 3** に示す．図中のベースピアグループは本ミドルウェアの既定のピアグループであり，すべての Host，Controller が起動直後に参加する．

以下，アプリケーションの開発からジョブの実行までの過程を述べる．

3.1.1 アプリケーション開発

アプリケーションプログラムは Java 言語で開発し，コンパイルして JAR 形式のアーカイブに格納しておく．この JAR ファイルと入力データファイルを，Controller を用いてジョブとして投入することになる．

アプリケーションは，単に各 Host 上で起動だけされればよいのであれば，通常のスタンドアロン Java プログラムと同様に，`main` メソッドさえ持っていればよい．並列処理を行うのであれば，Host 間通信のために，4 章で述べる何種類かのプログラミングインタフェースを利用できる．

3.1.2 ジョブの投入

資源利用者が Controller を用いてジョブ，すなわちアプリケーションプログラムと入力データファイルを投入した際の Controller の処理を**図 4** に示す．ま

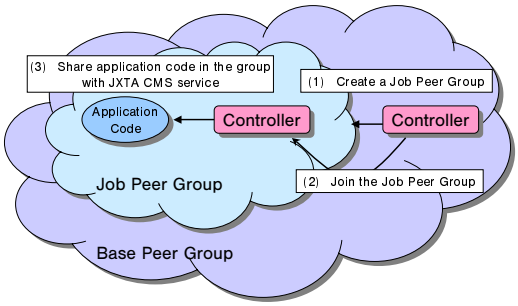


図 4 ジョブ投入時の Controller の処理
Fig. 4 Job submission process by Controller.

ず，Controller は，(1) ジョブのためのピアグループを作成し，(2) そのピアグループに参加する．続いて，当該ジョブピアグループ内で，JXTA のファイル共有サービス CMS（Content Management Service）を用いて，アプリケーションと入力ファイルをジョブピアグループ内で共有する．この処理によって，これらのファイルは，ジョブピアグループ内で発見および取得されうる状態になる．

次に資源利用者が Controller を用いて行う操作は，ジョブの起動である．本ミドルウェアで動作するアプリケーションプログラムには，起動時にジョブピアグループに参加していた Host 群のみで実行を行うものと，起動された後も Host のジョブ参加を受け入れうるものと，2 通りがある．後者のジョブであれば，投入した直後に起動してかまわない．しかし前者では，いくつかの Host がジョブピアグループに参加したことを確認してから起動の指示を出す必要がある．メッセージ・パッシング API（4.2 節）を用いるアプリケーションが前者であり，マスタ・ワーカ API（4.3 節）を用いるアプリケーションが後者である．

Host 群の参加確認は，Controller のシェルを目視して行う．シェル上に Host 群の名前が表示されている様子を**図 5** に示す．

3.1.3 ジョブへの参加

資源提供者がデーモンプログラム Host のシェルを用いてジョブの実行を受け入れる際の，Host の処理を**図 6** に示す．まず，Host は，(1) ジョブピアグループを発見する．発見されたジョブピアグループが Host のシェル（図 1）に表示される．資源提供者は，ジョブの名前，投入者といった属性を見て，(2) そのジョブの実行を受け入れるか否かを判断できる．受け入れるのであれば，シェルを用いて，その旨の指示をする．指示を受けた Host は，(3) ジョブピアグループに参加し，当該ピアグループ内にてアプリケーションプログラムを (4) 発見し，(5) 取得する．

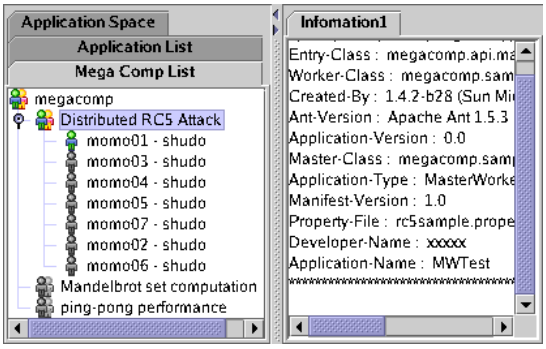


図 5 Controller のシェル上に表示された Host 群
Fig. 5 Job submission using Controller's shell.

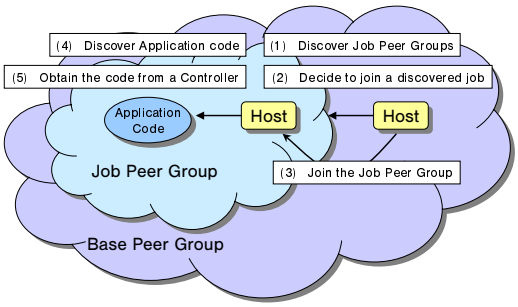


図 6 ジョブ参加時の Host の処理
Fig. 6 Job participation process by Host.

資源提供者にしてみると，ジョブが確かに属性にあるとおりの投入者によって投入されたものだと信じるためには，認証が必要である．このために，ジョブ投入者によるジョブへの署名，および，Host による署名検証機能を開発している．

このように，資源提供者が自身の資源を提供する先を決められることは，計算機資源の授受を目的とする本ミドルウェアにとって重要な要件である（6 章）．しかし，人手で選択しない限り Host がジョブに参加しないのでは，資源提供者の労力が大きく，資源の提供はなかなか進まないだろう．特に，本ミドルウェアを，インターネット上での資源授受ではなく組織内計算機群の集中制御に利用する場合には，Host の自動運転機能は必須である．そこで，Host に自動運転機能を実装した．自動運転機能を有効にして起動した Host は，ジョブを発見すると，人手での指示なしに，発見したジョブに参加する．

3.2 ジョブの起動と停止

続いて，資源利用者は，Controller を用いて，起動の指示を行う．この指示は，ジョブピアグループ内同報通信で各 Host に伝達される．また，実行停止の指示も任意の時点で行える．

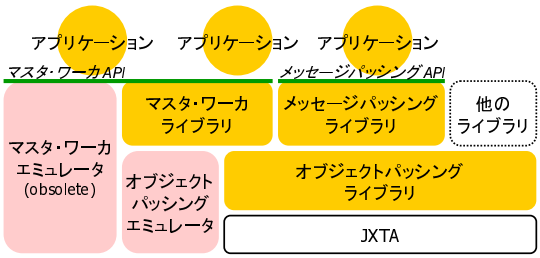


図 7 並列プログラミング支援ライブラリ
Fig. 7 Libraries supporting parallel programming.

4. 並列プログラミング支援ライブラリ

本章では，本ミドルウェア向けにアプリケーションを開発する際に用いる API と，その API を実装している並列プログラミング支援ライブラリについて説明する．現実装は，アプリケーションプログラマ向けに，メッセージパッシングとマスタ・ワーカという 2 種類の API を提供している．これら支援ライブラリの目的は，アプリケーションプログラマが，通信に必要な手続きや実際の実行環境の詳細を気にすることなく，アルゴリズムの記述に専念できるようにすることである．

現実装が提供している支援ライブラリの構成を**図 7** に示す．図中で長方形や円が上下に接している箇所は，上のソフトウェアが下位のソフトウェアを直接利用することを示している．

4.1 オブジェクトパッシングライブラリ

図 7 中のライブラリのうち，アプリケーションプログラマが直接利用するのは，マスタ・ワーカおよびメッセージパッシングである．両ライブラリとも，オブジェクトパッシングライブラリを用いて実装されている．これは，通信相手を JXTA のピア ID で識別して，Java のオブジェクトを送受信するライブラリである．**図 8** にその API を示す．オブジェクトはシリアライザを用いてバイト列に変換され，JXTA のメッセージに埋め込まれる．オブジェクトパッシングライブラリのインスタンスは，特定のピアグループ内で動作するように初期化される．`broadcast` メソッドを呼ぶことで，そのピアグループ内の全ピアに対する一斉送信を行うことが可能である．

オブジェクトパッシング API は，メッセージの送受信という多くのプログラマにとってごく一般的な概念さえ理解していれば容易に利用できる．一方，JXTA の通信機構であるパイプを直接利用するためには，JXTA 特有のいくつかの概念を理解し，広告の作成，広報，発見などを行わなければならない．オブジェクトパッシング API が JXTA の通信 API の繁雑さを吸収し

```
interface ObjectPassing {
  void send(PeerID receiver, Serializable obj);
  void broadcast(Serializable obj);
  Envelope recv(PeerID sender);
  Envelope recv(PeerID sender, long timeout);

  // receive a message from anyone
  Envelope recv();
  Envelope recv(long timeout);

  // check if a message is available
  boolean available();
  boolean available(PeerID sender);

  // register and unregister a message listener
  void addObjectPassingListener(
         ObjectPassingListener listener);
  void removeObjectPassingListener(
         ObjectPassingListener listener);
}
```

図 8 オブジェクトパッシング API
Fig. 8 API of object passing library.

ているため，マスタ・ワーカといった，並列処理のパターンに応じた各種支援ライブラリの開発は容易なものとなっている．

4.2 メッセージパッシングライブラリ

メッセージパッシング API も，オブジェクトパッシング API と同じく Host 間でオブジェクトを送受信するための API である．オブジェクトパッシング API との唯一の違いは，通信相手の識別方法である．JXTA のピア ID ではなく，MPI[7] と同様に，各 Host に割り当てられた非負整数 rank で識別する．メッセージパッシングライブラリは rank とピア ID の対応表を管理し，アプリケーションプログラムとオブジェクトパッシングライブラリの間で，rank とピア ID の相互変換を行う．

各 Host の rank は Controller が割り当て，rank とピア ID の対応表をジョブピアグループ内の全 Host に配布する．これにより，全 Host が同一の対応表を保持できる．

4.3 マスタ・ワーカライブラリ

マスタ・ワーカ API は，マスタ・ワーカ型の並列処理の記述を支援する API である．アプリケーションプログラマは，ワーカが処理する部分問題を表すクラスと，その処理結果を表すクラスを定義する．続いて，ワーカ側プログラムと，問題全体から部分問題を作成してワーカ群に向けて投入するマスタ側プログラムを記述する．本 API に沿って記述したプログラムの構造を，**図 9**，**図 10** に示す．図 9 がマスタ側プログラム，図 10 がワーカ側プログラムであり，図中の下線で示した箇所は，本 API が規定または提供するクラス，インタフェース，メソッドを示している．

```
class AMaster implements Master {
  Serializable getWorkerInitData() {
    return <data to initialize a worker>;
  }

  start() {
    while (...)  {
      WorkUnit wu = <a subproblem>;
      // submit a subproblem
      submitWorkUnit(wu);
    }
  }
}
```

図 9 マスタ側アプリケーションプログラムの構造
Fig. 9 Master side of an application program complying with master-worker API.

```
class AWorker implements Worker {
  void init(Serializable initData) {
    // initialize this worker instance
  }

  WorkResult process(WorkUnit wu) {
    // process a subproblem and
    // return the result
  }
}
```

図 10 ワーカ側アプリケーションプログラムの構造
Fig. 10 Worker side of an application program complying with master-worker API.

本ライブラリは，ワーカ側がマスタ側に対して部分問題を要求し，それに対してマスタ側が部分問題を配布するというプロトコルを採用している．そのため，ジョブの起動後にジョブに参加した Host であっても，処理に参加できる．また，ジョブ実行中に Host が離脱しても，ジョブ全体の実行は完遂される．

4.3.1 マスタを動作させる Host の選択

すべての Host は，JXTA の提供するオーバレイネットワーク上で双方向に通信が可能である．ゆえに，どの Host でマスタ側プログラムを起動してもアプリケーションは動作する．

しかし，通信のコスト，すなわち遅延の大きさや帯域幅は，送信元と送信先ピアの組や通信方向によってまちまちである．マスタ・ワーカ型並列処理では，マスタ側にトラフィックが集中するため，どの Host でマスタ側プログラムを起動するかが，全体の処理効率に影響を与える．マスタ側プログラムを起動する Host は，通信コストを考慮して選ばれることが望ましい．

現実装でのマスタ起動先 Host の選択法は次のとおりである．Host デーモンに対して，起動する際に，率先してマスタを受け入れるか否かを指示できる．Con-

troller は，マスタ・ワーカ型アプリケーションを起動する際に，その種の Host に対してマスタを割り当てる．その種の Host が複数見つかった場合は，そのうちの 1 台にマスタを割り当てる．見つからなかった場合も，Host 群のうちどれか 1 台を選ぶ．この場合の選択方法は研究課題であり，現在は，Controller が持っている Host 群管理表から先頭の Host を選んでいる．

4.3.2 障害と妨害への対応

ワーカに割り当てた部分問題に対して，処理結果が必ず返されることは期待できない．本ミドルウェアはインターネット上の個人情報機器を対象とするため，Host の故障や電源断，ネットワーク障害，通信メッセージの喪失を前提とする必要がある．また，Host 提供者が悪意を持って，部分問題の割当てを受けておきながら処理結果をマスタに返さないことも想定される．こういった障害や妨害があった場合でも，すべての部分問題が処理されて，ジョブ全体としては実行が完了されることが望ましい．

ジョブの実行完了を保証するために，本マスタ・ワーカライブラリは部分問題の再割当て機能を備えている．ワーカへいったん割り当てた部分問題に対して一定時間処理結果の返戻がない場合，その部分問題を，再び，ワーカへの割当て対象とする．

悪意ある Host 提供者の存在を前提とする以上，処理結果の返戻がない場合だけでなく，虚偽の処理結果がマスタに返されることも想定する必要がある．虚偽の返戻があってもジョブ全体として正しい実行結果を得るためには，虚偽を検出する必要がある．このためには，投票（voting）や抜き取り検査（spot-checking）が一般的に行われている[8]．たとえば，United Devices 社のインターネット上分散処理プロジェクトは，同一の部分問題を 5 回配布し，最初に返戻された 2 つの処理結果が一致すれば，その結果を正しい結果として採用するという投票方式を採っている[9]．

本マスタ・ワーカライブラリは，投票方式の虚偽検出機能を備えている．同一の部分問題を m 個配布して最初に返戻された n 個の処理結果を照合し，それらが一致すれば正しい結果として採用，一致しなければ再び配布されている個数が m となるようにする．ここで，m と n は調節可能である．$m = n = 1$ と設定することで，虚偽検出機能を無効化できる．

この虚偽検出機能は，マスタ側プログラムが悪意ある Host で実行された場合は想定していない．マスタ側プログラムだけは，信頼できる Host，たとえば Controller 使用者が提供する Host で起動する，といった運用での対策を検討している．

マスタ・ワーカライブラリが備える再割当て機能および虚偽検出機能は，アプリケーションプログラムとは完全に独立している．すなわち，プログラマがアプリケーションを開発する際は，これらの機能について意識する必要はまったくない．

4.4 エミュレータ

並列処理プログラムのデバッグは難しい．プログラムが複数の計算機上で動作することに起因する，挙動の監視，把握の難しさが一因である．

そこで，本ミドルウェア向けの並列処理プログラムの計算機 1 台上での動作確認，デバッグを可能とするために，オブジェクトパッシングライブラリのエミュレータを用意した．単一の Java 仮想マシン上のスレッド間で，オブジェクトパッシング API を利用した通信を可能とする．オブジェクトパッシング API 上に実装された支援ライブラリ（マスタ・ワーカとメッセージパッシング）をこのエミュレータと組み合わせて利用することで，支援ライブラリの API を利用しているアプリケーションプログラムを，計算機 1 台上で動作させることができる．

エミュレータは，アプリケーションプログラムの開発だけでなく，支援ライブラリ自体の開発においても活用できる．実際に，マスタ・ワーカライブラリの動作確認とデバッグは，オブジェクトパッシングライブラリのエミュレータ上で行った．エミュレータ上で動作を確認できたマスタ・ワーカライブラリが，そのまま JXTA を用いるオブジェクトパッシングライブラリ上で動作し，分散環境でのデバッグを完全に回避できた．

5. 支援ライブラリの性能評価

本ミドルウェアは，P2P 通信ライブラリを利用して，その提供する機能の様々な恩恵を受けている．たとえば，オーバレイネットワーク上での任意計算機間の双方向通信や，同一ジョブを実行する Host 群への同報通信，ジョブ情報と Host 情報の発見による集中管理回避などがその代表である．

一方で，これらの機能のためには，通信性能が犠牲となっていることが予想できる．そこで，本ミドルウェアの提供する支援ライブラリの通信性能が，目的とする高スループット計算に耐えるものであるかどうか，また，どういったアプリケーションプログラムであれば本ミドルウェア上で効率良い処理が可能であるのかを見積もるために，支援ライブラリの通信性能を測定した．

実験は，Intel 社 2.4 GHz Xeon プロセッサを 2 つ

表 1 通信遅延
Table 1 Communication latency.

TCP（C プログラム）	0.062
TCP（Java プログラム）	0.064
メッセージパッシング	4.5
	（ミリ秒）

搭載した PC を計算ノードとし，それを 32 台を備えた PC クラスタ上で行った．PC 群は 1000BASE-T イーサネットスイッチで結合されている．OS は Linux 2.4.19，Java 実行系は Sun Microsystems 社が配布している Java 2 SDK 1.4.2 を用いた．以下の実験では，特に断わりのない限り HotSpot Client VM を使用している．JXTA の実装としては，Java 2 SE 用参照実装のバージョン 2.1 を用いた．JXTA には，1 対 1 通信には必ず TCP が使われるよう，HTTP の使用禁止という設定を施した．

ギガビットイーサネット 1000BASE-T の LAN で得られる通信性能は，現在のインターネット上で通常得られる性能よりはるかに高い．本ミドルウェアの主要ターゲットであるインターネット環境とは大きな差がある．しかし，むしろ，十分に高い性能を持つネットワーク上で実験を行うことによって，本ミドルウェアや JXTA 自体の性能やオーバヘッドを明確に測定できる．

5.1 遅延とスループット

メッセージパッシングライブラリの通信遅延とスループットを測定し，TCP のそれと比較した．メッセージパッシングライブラリ，ひいては JXTA による通信は実験環境では TCP で行われるので，この比較によって，JXTA と支援ライブラリのオーバヘッドを見積もることができる．

1 バイトを 2 台の計算機間で 1000 回往復させ，要した時間を 2000 で割ることで片道の通信遅延を算出した．結果を**表 1** に示す．この実験で用いている程度の性能を持つ計算機，OS，Java 実行系では，メッセージパッシングライブラリを用いた通信には 4 ～ 5 ミリ秒を要することが判った．この比較的大きな通信遅延は，5.2 節で評価するマスタの部分問題処理スループットに大きな影響を与えている．

スループットは，1 回に送受信するメッセージに含めるデータのサイズを変化させて，2 台の Host 間で 100 回往復させて測定した．結果を**図 11** に示す．以下では，すべて，キロ（K）は 1000 ではなく 1024，メガ（M）は 1024^2，以下同様である．データサイズが 128 KB のときに最もスループットが高くなっており，データサイズを大きくすると，再びスループットは低下する．JXTA 参照実装が数百 KB 以上の大きなメッセージを扱う際に，何らかのオーバヘッドがあると考えられる．この原因は，今後，より詳細に調査する．メッセージパッシング，または，オブジェクトパッシングライブラリ内で，メッセージを 128 KB 程度に分割して送受信することにより，512 KB 以上の大きなメッセージも，より高いスループットで送受信できると考えられる．

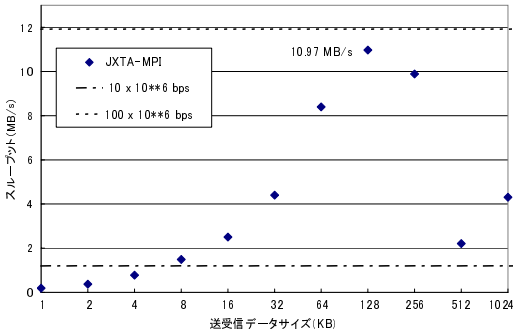


図 11 メッセージパッシングライブラリのスループット
Fig. 11 Throughput of message passing library on JXTA.

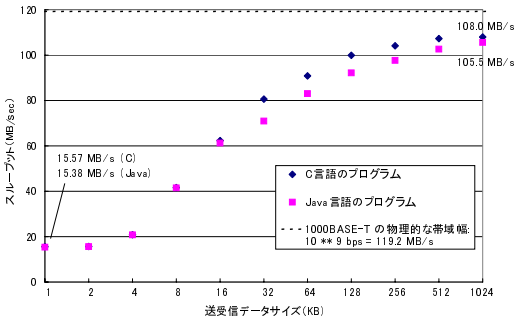


図 12 TCP のスループット
Fig. 12 TCP throughput.

比較対象として，**図 12** に，TCP を BSD ソケットインタフェースで直接利用した場合のスループットを示す．TCP は，1 回の送受信データサイズを 1 MB にした場合，物理的な帯域幅の約 9 割のスループットを達成している．それに対し，JXTA を利用するメッセージパッシングライブラリのスループットは，物理的な帯域幅の 1 割弱にとどまっている．とはいえ，それでも，100BASE イーサネットの物理的な帯域幅である 100×10^6bps をほぼ埋められる程度のスループットは達成している．10×10^6bps であれば，8 KB という比較的小さい単位での送受信で達成できている．家庭や小規模オフィスのインターネット接続線（1 ～ 100×10^6bps）や 100×10^6bps 程度までの低速な LAN の帯域幅なら，JXTA および支援ライブラリ

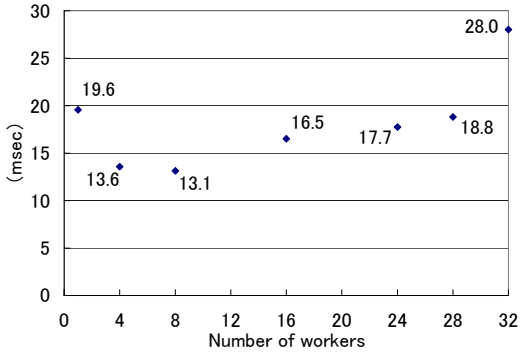


図 13 部分問題 1 つの配布と結果の回収に要する時間（HotSpot Client VM 使用）
Fig. 13 Handling time per subproblem with HotSpot Client VM.

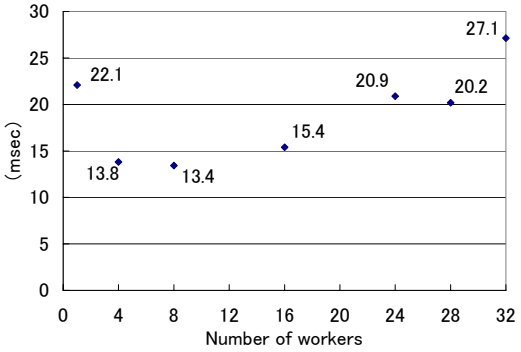


図 14 部分問題 1 つの配布と結果の回収に要する時間（HotSpot Server VM 使用）
Fig. 14 Handling time per subproblem with HotSpot Server VM.

の現実装でも充分に活用できることが分かった．

5.2 マスタの部分問題処理スループット

マスタ・ワーカ型の並列処理では，ワーカ群の処理能力を無駄なく発揮させて効率の良い並列処理を行うためには，マスタに集中する，部分問題の配布および結果回収処理が律速とならないようにすることが重要である．そのため，単位時間あたりに処理できる部分問題の数，すなわち部分問題処理のスループットが，マスタ側プログラムの性能を表す重要な指標となる．マスタ・ワーカライブラリについて，このスループットを求めた．

測定には，共通鍵暗号方式 RC5 の鍵探索アプリケーションを用いた．これは，distributed.net の Project RC5[10] で行われている処理を，本ミドルウェアのマスタ・ワーカ API を用いて実装したものである．問題として暗号文が 1 つ与えられ，その暗号化に使われた鍵を求めるというアプリケーションであり，RC5 が数学的に破られていない以上は，膨大な数の鍵候補を総当たりで試すことが最も効率の良い解法である．マスタは，適当な数の鍵候補を部分問題としてワーカに割り当て，ワーカは与えられたすべての鍵候補についてそれが求める鍵であるかどうかを検証する．

ここでは，マスタ・ワーカライブラリの性能測定が目的なので，部分問題に含める鍵候補の数を 0 とすることで，アプリケーション固有の処理にかかる時間を可能な限り小さくした．これによって，マスタ・ワーカライブラリが部分問題の配布と結果回収処理に要した時間だけを計測できる．ここでは，部分問題の総数を 200 として，配布と回収に要した時間を 200 で割り，部分問題 1 つあたりの配布，回収時間を求めた．HotSpot Client VM での結果を**図 13** に，HotSpot Server VM での結果を**図 14** に示す．

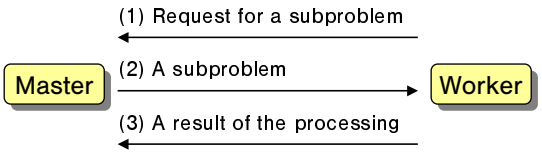


図 15 マスタとワーカの間の対話手順
Fig. 15 Communication protocol between master and worker.

実験結果のうち，最も悪い場合で，部分問題 1 つあたり 28 ミリ秒，最も良い場合には 13 ミリ秒を要している．悪い場合で 1 秒あたり約 35 の部分問題，良い場合で 76 の部分問題を処理したことになる．これは十分なスループットだろうか．たとえば，本ミドルウェアをインターネット上分散処理に適用する場合を想定する．United Devices 社のあるプロジェクトでは，部分問題配布サーバが取扱う部分問題の数を，1 秒あたり 120 程度に調整している[11]．このスループットが何秒，何分あたりの平均値かは不明であるが，部分問題配布サーバの性能は，少なくとも，この平均スループットの数倍，数百個/秒には達していると考えることが自然である．単一のジョブに数万台規模のワーカを参加させようとするなら，本ミドルウェアには，現実装と比較しておよそ $10^1 \sim 10^2$ 倍のスループットが望まれる．

では，ミドルウェアのどの部分が律速なのかを考察する．マスタ，ワーカ間での部分問題および処理結果の授受は，次の手順で行われる（**図 15**）．

(1) ワーカがマスタに対して，部分問題を要求する．
(2) マスタがワーカに対して，部分問題を送付する．
(3) ワーカがマスタに対して，処理結果を送付する．
(4) （以下，繰返し）

1 つの部分問題について，処理結果がマスタに返戻さ

れるまでに (1)〜(3) の 3 回のメッセージ送受信が行われる．このメッセージ送受信は，マスタ・ワーカライブラリがオブジェクトパッシングライブラリを用いて行う．ここで，オブジェクトパッシングライブラリの通信遅延はメッセージパッシングライブラリのそれとほぼ同じ，約 4.5 ミリ秒（5.1 節）であると考えられる．なぜなら，メッセージパッシングライブラリが行う処理は整数値 rank と JXTA のピア ID 間の相互変換だけであり，これはハッシュ表を索くという，4.5 ミリ秒と比較すると無視し得るくらいのごく軽い処理だからである．

仮に (3) の送信と次の (1) の送信はオーバーラップさせることができたとしても，部分問題 1 つあたり，メッセージ送受信に要する時間だけで，約 4.5 ミリ秒 × 2 = 9.0 ミリ秒は要すると見積もることができる．これでは，1 秒あたりに処理できる部分問題は，最大でも 1000 ミリ秒 ÷ 9.0 ミリ秒 = 約 111 個 に制限されてしまう．

マスタが処理できる部分問題数を向上させるには，オブジェクトパッシング，および，それが利用する JXTA 自体の性能向上が必要であることが判った．

また，ワーカとの通信がマスタ側のボトルネックであるなら，ワーカとの通信を複数のマスタで分担するという方法もある．将来はマスタ群のフェデレーションを構成できるように，マスタの設計においては，部分問題をマスタ外部の DBMS（データベース管理システム）に格納しやすい構造を採っている．

5.3 粒度の細かい部分問題への耐性

ワーカ側プログラムでの部分問題の処理時間が短いほど，つまり，部分問題の粒度が細かくなるほど，ワーカの台数に応じた性能向上は得られなくなる．理由の 1 つは，ワーカ側で，部分問題を処理している時間に対する配布を待っている時間の比が大きくなり，この配布待ち時間が顕在化してくることである．また，粒度が細かくなると，マスタが部分問題を配布，回収する頻度が高くなる．この頻度がマスタの部分問題処理スループット（5.2 節）を超えてしまうと，ワーカが部分問題に飢えて，ワーカの台数を増やしても性能は向上しなくなる．

マスタ・ワーカライブラリがどの程度まで細かい粒度の部分問題に耐えられるかを見積もるために，いくつかの異なる粒度について並列化効率を観察した．

アプリケーションプログラムとしては，5.2 節と同じ RC5 の鍵探索アプリケーションを用いた．部分問題の粒度の調整は，部分問題 1 つあたりに含める鍵候補の数を 0x8000，0x4000，0x2000（16 進）と変化させることで行った．部分問題の総数は 200 である．それぞれの粒度について，ワーカの台数を 1 から 32 台まで変化させて実行に要した時間を計測し，単位時間あたりに処理できた鍵候補の数を求めた．HotSpot Client VM での結果を**図 16** に，HotSpot Server VM での結果を**図 17** に示す．また，両グラフについて，鍵処理のスループットを並列化効率に置き換えたものを**図 18**，**図 19** に示す．**表 2** には，それぞれの粒度について，部分問題 1 つあたりの計算時間を示す．

部分問題あたりの鍵候補数が 0x4000，0x8000 の場合は 32 台まで性能が向上しているのに対し，0x2000 の場合，性能向上はワーカ数 24 台で頭打ちとなっている．これは，台数が増えて実行時間が短くなった結果，ワーカ群の負荷分散が均等ではなくなったためであると考えられる．Host の台数が増えるに従い，Controller からのジョブ起動指示（3.2 節）が Host に到達するまでの時間が Host ごとに 1 秒以内から数秒まで様々となる傾向が見られている．0x2000，ワーカ 24 台という条件では，実行時間は 6.13 秒と短くなっており，ジョブ起動指示の到達が 1，2 秒違っても，各ワーカが処理する部分問題数に大きな影響を与える．

この実験では，部分問題あたりの処理時間が 1 秒程度まで短くなっても，32 台で 20 倍以上の性能向上率は達成できることが確認できた．この程度の台数，部分問題の粒度であれば，本ミドルウェアが原因で性能向上が大きく制限することはないといえる．台数をさらに増やしての実験は今後の課題である．また，インターネット上など通信遅延がより大きい環境では，台数効果を得るためには部分問題の粒度はもっと粗い必要があるだろう．

6. 関 連 研 究

我々は，個人間での計算機資源の授受という目的のためには，以下が重要な要件であると考えている．

- 誰でも，自由なプログラミングでアプリケーションプログラムの開発を行えて，他者の計算機に対してジョブとして投入できること．
- 資源提供者が，ジョブを単位として，受け入れるか否かを決められること．

本ミドルウェアはこれらの機能を備えている．**表 3** は，これらの観点について，本ミドルウェア P3 と，インターネット上での分散処理，ひいては，計算機資源の募集や授受を目的とするソフトウェアを比較したものである．

後者については，資源提供者がジョブの配布元計算機を選択でき，それによって配布元計算機や配布元組

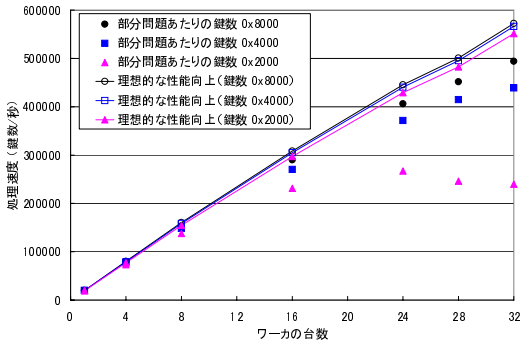


図 16 部分問題の粒度に応じたスループットの変化（HotSpot Client VM 使用）
Fig. 16 Variation in key crunching throughput according to granularities of subproblems (with HotSpot Client VM).

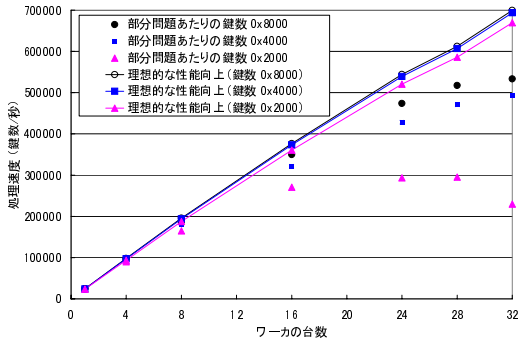


図 17 部分問題の粒度に応じたスループットの変化（HotSpot Server VM 使用）
Fig. 17 Variation in key crunching throughput according to granularities of subproblems (with HotSpot Server VM).

表 2 部分問題 1 つあたりの計算時間
Table 2 Calculation time per subproblem.

部分問題あたりの鍵候補の数（16 進）	HotSpot Client VM	HotSpot Server VM
0x8000	1.7	1.4
0x4000	0.84	0.69
0x2000	0.42	0.36

（秒）

織を単位とした選択が可能なソフトウェアもある．しかしこの粒度での選択では，たとえば，A さんの暗号研究ジョブには資源を提供するが，娯楽目的の音声データ変換には提供しない，といった要求には対応できない．

JNGI[12] は大規模分散処理向けフレームワークである．本ミドルウェアと同様に，通信プロトコルに JXTA を採用している．しかし，本ミドルウェアとはピアグループの利用法が大きく異なっている．本ミドルウェアはジョブごとにピアグループを用意することで，そのピアグループを，ジョブに関与する全計算機への同報通信とアプリケーションプログラム配布のための領域として活用している（2.2 節）．一方，JNGI の設計者は，ピアグループを，一定数までの計算機群を効率良く集中管理するための領域として活用することを提案している[12]．あるピアグループ内の計算機が増えすぎないように，複数のピアグループを階層的に管理する方式も提案している．この方式では，本ミドルウェアとは異なり，資源提供者がジョブを選ぶことはできない．また，2003 年 9 月末時点の実装では，すべてのピアが単一のピアグループに参加するようになっており，ピアグループ群を用いた計算機群の効率的な管理というアイディアは未実装である．

論文 12) では JNGI を用いた分散処理で得られた

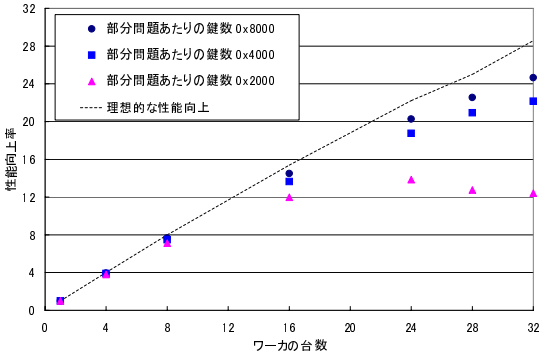


図 18 部分問題の粒度に応じた並列化効率の変化（HotSpot Client VM 使用）
Fig. 18 Variation in efficiency of parallel key crunching according to granularities of subproblems (with HotSpot Client VM).

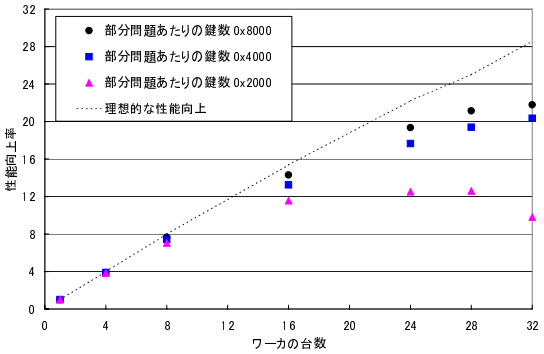


図 19 部分問題の粒度に応じた並列化効率の変化（HotSpot Server VM 使用）
Fig. 19 Variation in efficiency of parallel key crunching according to granularities of subproblems (with HotSpot Server VM).

表 3 インターネット上での分散処理を目的とするソフトウェアと P3 の比較
Table 3 Comparison between other Internet-wide distributed computing software and P3.

	一般利用者による プログラミングと ジョブ投入	資源提供者 による ジョブ選択	通信 プロトコル	アプリケーション 開発言語	並列 プログラミング モデル	虚偽の 処理結果 検出
P3	**yes**	**yes**	JXTA	Java	マスタ・ワーカ, メッセージパッシング	**yes**
JNGI	yes	no	JXTA	Java	マスタ・ワーカ	no
XtremWeb	yes	no	選択可能 （RMI 等）	ネイティブ （C 等）	分割統治	言及なし
GreenTea	yes	no	RMI	Java	マスタ・ワーカ	no
Javelin	yes	no	RMI	Java	分枝限定モデル	言及なし
Bayanihan	yes	no	HORB	Java	マスタ・ワーカ, BSP	不明
Ninflet	yes	no	RMI	Java	マスタ・ワーカ	no
distributed .net	no	yes	HTTP	C，C++, アセンブリコード	分割統治	yes
SETI@home	no	no	HTTP	不明	分割統治	yes

性能も報告されている．実験に用いられているアプリケーションプログラムは，1 つの部分問題の処理に 130 秒という長い時間を要するものであり，通信性能に対する要求は非常に低い．この実験では，JXTA が分散処理に耐える性能を達成していたのか否かは評価できていない．また，Solaris，Windows 2000，RedHat Linux を合わせて最大 150 ワーカ用いるという非均質環境での実験にもかかわらず，結果として示されている性能向上率の基準が述べられていない．

XtremWeb[13] も，インターネット上での大規模分散処理を目的としたソフトウェアである．複数のアプリケーションに対応することと高い性能を発揮することがその設計目標である．投入可能なアプリケーションプログラムはコンパイル済みのネイティブコードである．並列処理の方式は，あらかじめ用意しておいた多数の入力ファイルを多数の計算機に配布して処理させるという，単純な分割統治である．計算機間の通信プロトコルは差し替えが可能であり，現在，TCP，Java RMI，XML-RPC の実装が用意されている．XtremWeb 自体が提供しているアプリケーションのプログラミングモデルは以上のとおりであるが，耐故障性を備えたメッセージパッシングライブラリ MPICH-V[14] を用いるアプリケーションプログラムを，XtremWeb を用いて多数の計算機上で起動するという実験が行われている．

GreenTea（GT）Distributed Network Computing Platform[15] は，米 GreenTea Technologies 社が開発しているソフトウェアである．通信には Java RMI を用いており，IP アドレスなどの計算機情報の管理，部分問題の配布などは，GTServer と呼ばれるサーバプログラムが行う．部分問題をどのワーカに割り当てるかは，GTServer 内のスケジューラに決めさせることも，アプリケーションプログラムが自身で指定することも可能となっている．

Javelin 2.0[16] は，多数の部分問題を計算機群に配布するという点では他のソフトウェアと同じであるが，分枝限定計算モデルを採っている点に特徴がある．プログラミングモデルも計算モデルを反映したものとなっている．枝刈りをまったく行わなければすべての部分問題が処理されるので，単純な分割統治型の並列処理を記述できる．

Bayanihan[17] は，プログラミングモデルとして，マスタ・ワーカだけでなく，BSP（Bulk Synchronous Parallel）[18] をサポートしている．これは，計算，大域通信，バリア同期の 3 ステップを繰り返して計算を進めるというモデルであり，任意のワーカ間での直接通信を記述できるが，通信結果がローカルメモリに反映されるのはバリア同期後になるというモデルである．実際のワーカ間通信はマスタが中継しており，ワーカ間で直接通信を行っているわけではない．

distributed.net[4] は，共通鍵方式 RC5 の鍵探索など，いくつかのアプリケーションプログラムについてインターネット上分散処理プロジェクトを主宰している．SETI@home[2),3)] とは異なり，資源提供者は，ソフトウェアを起動する際に，実行を行うアプリケーションプログラムを指定できる．

インターネット上での大規模分散処理において虚偽の処理結果を検出する方法として，投票（voting）と抜き取り検査（spot-checking）が提案されている[8]．また，部分問題の割当て方法，処理結果の正しさを確認する方法，部分問題の処理に対する報酬の支払い方を調節することで，虚偽の処理結果を返戻することで

得られる利得を，虚偽発覚のリスクよりも小さくするという方法も検討されている[19]．

CPM（Compute Power Market）[20] は，計算機資源の提供者と利用者それぞれが条件を提示してお互いを見つけるための市場の枠組みである．その目的は需要と供給を仲介することであり，CPM 自体は並列処理を支援する機能は用意していない．

Phoenix[21),22)] は，メッセージパッシング型の並列処理において，処理の途中での計算機の参加，離脱を可能とする並列プログラミングモデルである．Phoenix では，たとえば 0 から $2^{32}-1$ といった膨大な数の通信エンドポイントを，各プロセスで分担して保持する．新たなプロセスが計算に参加してきた場合は，エンドポイントを分け与え，離脱する際は他のプロセスに対して返却することで，参加と離脱を可能としている．本ミドルウェアでは，マスタ・ワーカ型並列処理であればジョブ実行中の Host の参加，離脱が可能だが，メッセージパッシング型での実行中の参加，離脱は考慮していない．本ミドルウェア用の支援ライブラリとして Phoenix のプログラミングモデルを設計することは興味深い課題である．

7. ま と め

PC に代表される個人情報機器の計算機資源を個人間で授受し，ひいては集約して分散処理を行うためのミドルウェア P3 の設計と実装を述べた．P3 は，自由な並列プログラミングを可能とするために，P2P 通信ライブラリ JXTA を採用している．本論文では，その JXTA が P2P ソフトウェア向けに提供している諸概念を本ミドルウェアの目的に対していかに適用するかを提案した．提案した適用法に基づいて実装した分散処理ミドルウェアを用いて，並列計算を行う計算機グループをアドホックに構成できることが確認できた．また，実装したミドルウェアについて，基本的な通信性能やマスタ・ワーカ型並列処理用ライブラリのスループット，実際のアプリケーションでの性能向上率を測定し，報告した．スループットは 100BASE イーサネットの帯域幅を埋められる程度であること，マスタ・ワーカライブラリは，部分問題 1 つの配布，結果回収に 10 ～ 30 ミリ秒を要しており，部分問題の粒度が細かい場合にはこのスループットが律速となると予想されること，また，粒度によっては良好な性能向上が得られることを確認できた．

本ミドルウェア開発の目標は，アイディアの検証や試作ではなく，実用的な有用性，品質の達成である．そのため，本論文では言及していないものも含め，数多くの課題が残されている．台数をさらに増やしてのスケーラビリティ試験，参加ジョブ選択ポリシ（3.1 節）の記述方法や，ジョブのチェックポインティングを支援する方法，ジョブが使用する計算機資源を制限する方法といった研究課題だけでなく，容易なインストールや，ミドルウェア自体の自動更新など，実用上は欠かせない要件の達成にも取り組んでいく．

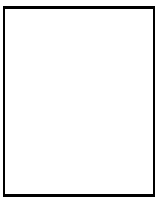
謝辞 本研究の一部は，情報処理推進機構（旧情報処理振興事業協会）による次世代ソフトウェア開発事業の委託業務として実施された．

参 考 文 献

1) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *Int'l Journal on Supercomputer Applications*, Vol. 11, No. 2, pp. 115–128 (1997).
2) Korpela, E., Werthimer, D., Anderson, D., Cobb, J. and Lebofsky, M.: SETI@home: Massively Distributed Computing for SETI, *Computing in Science and Engineering*, Vol.3, No.1, pp. 78–83 (2001).
3) SETI@home: SETI@home: Search for Extraterrestrial Intelligence at Home. http://setiathome.ssl.berkeley.edu/.
4) distributed.net: distributed.net: Node Zero. http://www.distributed.net/.
5) GIMPS: The Great Internet Mersenne Prime Search. http://www.mersenne.org/.
6) Project JXTA: jxta.org. http://www.jxta.org/.
7) Message Passing Interface Forum: *MPI-2: Extensions to the Message-Passing Interface* (1997).
8) Sarmenta, L. F.G.: Sabotage-Tolerance Mechanisms for Volunteer Computing Systems, *Future Generation Computer Systems (FGCS)*, Vol. 18, No. 4, pp. 561–572 (2002).
9) Patel, P.: Personal communication (2002).
10) distributed.net: Project RC5. http://www.distributed.net/rc5/.
11) Ueno, K.: Personal communication (2003).
12) Verbeke, J., Nadgir, N., Ruetsch, G. and Sharapov, I.: Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment, *Proc. of Third International Workshop on Grid Computing (GRID 2002)*, pp. 1–12 (2002).
13) Fedak, G., Germain, C., Néri, V. and Cappello, F.: XtremWeb: A Generic Global Computing System, *CCGrid2001 Special Session Global Computing on Personal Devices* (2001).
14) Bosilca, G., Bouteiller, A., Cappello, F., Dji-

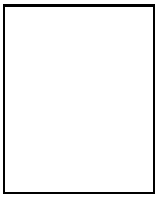
lali, S., Fedak, G., Germain, C., Herault, T., Lemarinier, P., Lodygensky, O., Magniette, F., Neri, V. and Selikhov, A.: MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes, *Proc. SC2002: International Conference for High Performance Networking and Computing* (2002).
15) GreenTea Technologies, Inc.: GreenTea Platform Whitepaper (2002).
16) Neary, M. O., Phipps, A., Richman, S. and Cappello, P.: Javelin 2.0: Java-Based Parallel Computing on the Internet, *Lecture Notes in Computer Science (LNCS) for 6th Int'l Euro-Par Conference (Euro-Par 2000)*, Vol. 1900, Springer Verlag, pp. 1231–1238 (2000).
17) Sarmenta, L. F. G. and Hirano, S.: Bayanihan: Building and Studying Web-based Volunteer Computing Systems using Java, *Future Generation Computer Systems (FGCS)*, Vol. 15, No. 5-6, pp. 675–686 (1999).
18) Sarmenta, L. F. G.: An Adaptive, Fault-tolerant Implementation of BSP for Java-based Volunteer Computing Systems, *Lecture Notes on Computer Science (LNCS) for IPPS'99 Workshop on Java for Parallel and Distributed Computing*, Vol.1586, Springer Verlag, pp.763–780 (1999).
19) Golle, P. and Stubblebine, S.: Distributed Computing with Payout: Task Assignment for Financial- and Strong- Security, *Proc. Financial Cryptography '01 (FC01)* (2001).
20) Buyya, R. and Vazhkudai, S.: Compute Power Market: Towards a Market-Oriented Grid, *Proc. 1st International Conference on Cluster Computing and the Grid (CCGrid2001)* (2001).
21) Taura, K., Kaneda, K., Endo, T. and Yonezawa, A.: Phoenix : a Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources, *Proc. of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03)* (2003).
22) 田浦健次朗: Phoenix: 動的な資源の増減をサポートする並列計算プラットフォーム, 情報処理学会研究報告, 2001-HPC-87-24, pp. 135–140 (2001).

(平成 15 年 10 月 5 日受付)
(平成 16 年 2 月 19 日採録)



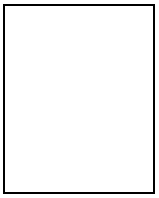
首藤 一幸（正会員）

1996 年早稲田大学理工学部情報学科卒業．1998 年同大学メディアネットワークセンター助手．2001 年同大学大学院理工学研究科情報科学専攻博士後期課程修了．同年産業技術総合研究所入所．現在に至る．博士（情報科学）．分散処理方式，プログラミング言語処理系，情報セキュリティ等に興味を持つ．IEEE-CS，ACM 各会員．



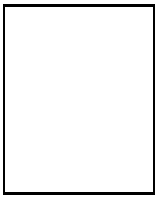
大西 丈治（正会員）

1984 年ジャストシステム株式会社入社．パソコン用アプリケーションおよびフレームワークの設計開発に従事．1997 年セイコーエプソン株式会社入社．組込機器のネットワーク対応と分散処理システムの研究開発に従事．2002 年産業技術総合研究所入所．2004 年アライドテレシス株式会社入社．無線アドホックネットワークの研究開発に従事．



田中 良夫（正会員）

1965 年生．1995 年慶応義塾大学大学院理工学研究科後期博士課程単位取得退学．1996 年技術研究組合新情報処理開発機構入所．2000 年通産省電子技術総合研究所入所．2001 年 4 月より独立行政法人産業技術総合研究所．現在同所グリッド研究センター基盤ソフトチーム長．博士（工学）．グリッドにおけるプログラミングミドルウェア，計算ポータル，およびテストベッド構築に関する研究に従事．IC'99 論文賞．ACM 会員．



関口　智嗣（正会員）

1959 年生．1982 年東京大学理学部情報科学科卒業．1984 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．以来，データ駆動型スーパーコンピュータ SIGMA-1 の開発等の研究に従事．2001 年独立行政法人産業技術総合研究所に改組．2002 年 1 月より同所グリッド研究センターセンター長．並列数値アルゴリズム，計算機性能評価技術，グリッドコンピューティングに興味を持つ．市村賞，情報処理学会論文賞受賞．グリッド協議会会長．日本応用数理学会，ソフトウェア科学会，SIAM，IEEE，つくばサイエンスアカデミー各会員．