Javaによる大域的並列計算環境 Ninflet

高木浩光[†1] 松岡聡[†2] 中田秀基[†1]
関口智嗣[†1] 佐藤三久[†3] 長嶋雲兵[†4]

地球規模の広域分散計算システムを魅力的なものとするためには，不特定の者に対して，システムの利用だけでなく応用プログラムの作成をも解放する必要があると考える．その実現のためには，安全性を保証しながら任意のプログラムを実行できる仕組みが必要である．そこで，Javaのセキュリティ機構を活用してこれを実現した，大域的並列計算環境「Ninflet」を提案する．これを用いることで，任意の計算を他人が所有する計算機上でさせることが可能となる．このシステムは，夜間利用されていない計算機を地球の裏側の昼間の地域に貸し出すといった，地球規模の共同利用メタコンピュータシステムを実現するためや，また，ワークステーションクラスタ上に並列処理環境を構築するためにも利用することのできるものである．本論文では，Ninfletシステムのアーキテクチャを提案するとともに，並列処理環境として利用する場合の予備的な性能評価を行う．

Ninflet: A Java-based Global Parallel Computing Environment

HIROMITSU TAKAGI,[†1] SATOSHI MATSUOKA,[†2] HIDEMOTO NAKADA,[†1]
SATOSHI SEKIGUCHI,[†1] MITSUHISA SATO[†3] and UMPEI NAGASHIMA[†4]

To make global-wide distributed computing system attractive, the system should be open to an arbitrary individual not only for its usage but also for construction of wide variety of application programs. For this purpose, the system must supply a secure environment for safely executing arbitrary programs. Our proposed global computing environment "Ninflet" fulfills such a requirement by exploiting the security mechanism of the Java language, allowing computation to occur on machines not owned or administered by the individual invoking the computation. Ninflet realizes a globally-shared metacomputer which would allow "lending" of computing cycles of machines which would be otherwise unused at nights to the other side of the globe, or to simply build a parallel execution environment on a heterogeneous sets of workstation clusters. We present the system architecture of Ninflet and a preliminary performance evaluation when used as a parallel execution environment.

1. はじめに

ネットワーク技術の進歩は目覚しく，LAN上のワークステーション群で構築したクラスタ並列計算機が良い成果をあげ始めている．このようなシステムにおいてプログラミング環境として用いられる，PVMやMPI等のメッセージパッシングライブラリは，LAN上に限らず，広域ネットワーク上でも利用することができる．しかしこれらは，各計算機にアカウントを持つ者だけが利用できるものであり，実際に広域ネットワークで利用できるのは特別なケースに限られる．

これに対し，我々が提案してきたNinf（Network Based Information Library for High Performance Computing）[1)~3)]は，RPC（Remote Procedure Call）をベースとするが，RPCにおいて引数を転送する際に必要となるスタブ情報を実行時にサーバからロードする仕組みを持たせた「Ninf RPC」を採用することで，サーバにアカウントを持たない不特定多数の者からの利用を可能としている．これと同様の特徴を持つシステムには，ほかにNetSolve[4)]等がある．しかし，これらのシステムでも，サービスの利用は不特定の者に許せても，サーバ上で実行されるサービス（計算ルーチン）の作成は特定の者にしか許されない．なぜなら，作成されたルーチンはサーバ管理者による登録作業を経る必要があり，このときそのルーチンが

†1 電子技術総合研究所
Electrotechnical Laboratory
†2 東京工業大学
Tokyo Institute of Technology
†3 新情報処理開発機構
Real World Computing Partnership
†4 物質工学工業技術研究所
National Institute of Materials and Chemical Research

サーバに危害を与えるものでないかについての審査を受けることになるからである．

一般に，地球規模の広域分散計算システム，すなわち大域的計算（global computing）システムが成功する鍵は，計算サーバ提供者をどれだけ多く確保できるか，また，利用者にどれだけ魅力的なサービスを提供できるかにあると考える．

利用者にとって魅力あるものとするためには，利用者が自ら作成した任意のプログラムの実行をサービスとして提供する必要があると考える．すなわち，不特定の者に，サービスの利用だけでなくプログラムの作成をも解放する必要がある．このためには，安全性を保証しながら任意のプログラムを実行できる仕組みが必要である．

そこで本論文では，Java のセキュリティ機構を活用することでこれを実現する「Ninflet」システムを提案する．Ninflet システムは，Java の動的プログラムロード機構「ClassLoader」と，セキュリティ機構「SecurityManager」を活用することで，不特定の者が作成した任意のプログラムを，不特定の者によって，第三者が所有する計算機上で計算させることを可能にするものである．

また，計算サーバ提供者を十分に多く確保するためには，提供者の管理コストが十分に小さくなくてはならない．提案する Ninflet システムは，システムのインストールおよび管理が単純化されており，また，その安全性は Java のセキュリティ機構によって保証されるため，誰でも安心して気軽に自分の計算機を提供することができる．

本論文では，この Ninflet システムの提案とその並列処理システムとしての予備評価を示す．以下，2 章で，Ninflet システムが想定する運用形態を説明するとともに，その意義について議論し，3 章で，大域的計算に求められる要件に Java がいかに応えるものであるかについて整理し，Java の数値計算への応用の妥当性を検証する．次に 4 章で，Ninflet のシステムアーキテクチャの説明として，概要と，インストールから実行までの具体的手順，サーバの運営管理方法，プログラミング方法について説明する．そして 5 章で並列処理への応用例について紹介してその予備評価の結果を示し，最後に 6 章で関連研究についてまとめる．

2. 想定する Ninflet システムの運用形態

1997 年 10 月，米国 RSA Data Security 社の暗号解読コンテスト “The RSA Data Security Secret-Key Challenge”[5] において，distributed.net のチーム[6] が広域分散処理によってこれを解いたことが話題となった．56 bit RC5 が 250 日ほどで破られ，この間，世界中の数万台の計算機が参加したという．この事例は，不特定多数の者の協力によって 1 つの問題を解く，いわば「ボランティア計算」の有効性を示したといえよう．

しかし，distributed.net では，この問題専用のシステムが用いられている．この問題専用のプログラムコードが各 OS 用のバイナリ形式で配布され，参加者は，このバイナリを無害であると信頼したうえで手動でインストールして計算に参加している．

これを汎用のボランティア計算フレームワークとして一般化しようとするなら，次の点を解決するべきだろう．

(1) 信頼できない不特定の者が作成した応用プログラムも実行可能にしたい．
(2) 複数の問題の同時実行を可能にしたい．
(3) 計算資源提供者に手作業をともなわせないようにしたい．
(4) 応用プログラムの配布用コードの準備のコストを小さくしたい．

Ninflet システムは，これらの問題を次のように解決する．

(1) 任意のプログラムが与えられても，実行環境に致命的な害を与えない（ファイルシステムを破壊しない等），安全な実行を保証する．
(2) 発行されるジョブと提供される計算機資源の管理機構により，複数のジョブの複数の計算機資源への割当てを行う．
(3) ジョブ管理機構により，自動的にジョブの実行を開始させる．
(4) 実行プラットフォームに依存しない形式でプログラムコードを配布する．

このような特徴を持つ Ninflet システムを利用することで，参加者は，配布されるコードの安全性について信頼すべきか迷うことなく，任意の応用プログラム対して気軽に計算機を提供できるようになる．また，プログラム作成者は，応用プログラムの配布用コード生成のコストを気にすることなく，いくつものバージョンのプログラムを試すことができる．

このように，Ninflet システムは，社会的に解くことに意義の大きい大規模問題をボランティアを募って分散計算させるための，汎用システムとして利用できるものである．

またほかにも，夜間の利用されていない計算機を地球の裏側の昼間の地域に貸し出すといった，遊休計算

機の有効利用のためにも Ninflet システムを活用できる．一般に，計算の必要性は，突発的に，同時に多量に発生しがちである（たとえば締切りの直前等に）．そのような事態に備えてそれに十分なだけの計算機資源を日頃から準備しておくのはコストが高い．一方で，一般に多くの計算機は常時使われてはいない．ここで，所有者の異なる計算機を皆で互いに利用し合うことができれば，突発的に発生した計算要求を低いコストで捌くことができる．このような「高スループット計算」[7] のために Ninflet システムを活用することができる．

近年のインターネットの大衆化にともない，世界には膨大な数の遊休計算機がネットワーク上に存在しているはずである．Ninflet システムを利用すれば，誰でも安心して自分の計算機を提供できることから，個人が所有するパーソナルコンピュータを計算サーバとして提供してもらい，世界規模でこれを共有し有効利用するといったこともできるであろう．

さらに，Ninflet システムを，PVM や MPI のように，LAN 上のクラスタ計算機での並列処理のために利用することもできる．Ninflet では，Java 言語のオブジェクト指向メカニズムを活用して，いくつかの並列アルゴリズムを，オブジェクト間のパターンとして抽象化し，クラスライブラリの形で並列計算をサポートしている（詳細については 5 章で述べる）．そして，そのクラスタ計算機上で動作する並列プログラムを地球の裏側から実行させるといった利用も可能となる．

このように Ninflet システムは，ローカルおよびワイドエリアネットワークをシームレスに統合するオブジェクト指向大域並列計算環境を構築するものである．

3. 大域的計算への Java の適合性

任意のプログラムをサーバに送付して実行させる機構をとり入れる場合には，セキュリティ上の問題を解決する必要がある．たとえば，ユーザ定義の計算ルーチンによってサーバ上のファイルを消去するといったことができてはならない．一般にこの種の問題の解決には以下にあげる手法が知られている．

(a) インタプリタ方式 計算ルーチンの記述言語を特定のインタプリタ型言語に限定し，実行時にインタプリタ上で危険な命令および処理を検出し実行を禁止する．Java や Safe Tcl[8] 等がこれに含まれる．

(b) SFI（Software Fault Isolation）方式[9] 計算ルーチンをソースコードないしバイナリコードでサーバへ送付し，コンパイル時ないし実行時に安全性をチェックするコードをプログラムに挿入し，そのコードによってメモリアクセスおよびシステムコール等が安全であるかを実行時にチェックする．

(c) システムコールトラップ方式 計算ルーチンをバイナリコードで送付し，サーバはそれをそのまま実行するが，ファイル操作等のシステムコールをトラップして，条件によって実行を禁止する．

これらそれぞれの方式の特質を比較すると，次のことがいえる．

1. (a) ではインタプリタ方式のため一般には計算速度が遅い．
2. (b) ではメモリアクセスの安全性チェックのためのオーバヘッドが無視できない．
3. (a) および，(b) のソースコードを送付する場合では，計算ルーチンを記述する言語が限定される．
4. (c) および，(b) のバイナリコードを送付する場合では，サーバとして利用できる計算機のプロセッサおよび OS が限定される．

ここで，(a) 方式によってすでにこれを実現している言語システムとして Java に着目する．Java は，

- ファイルやネットワークへのアクセス，スレッド等に対する操作の可否を制御する `java.lang.SecurityManager` クラス
- 不正なメモリ番地へのアクセスを起こさないことを保証する，排除されたポインタ演算や添字チェックされる配列といった言語仕様
- 不正なコードが含まれていないかチェックする bytecode verifier
- 異なる ClassLoader からロードされた同名のクラスを別クラスとして扱う機構

といったセキュリティのための機構を持っており，これらに基づく Java の安全性は，その 1 つの応用である「Applet」のフレームワークにより，すでに実証されている．そしてこれらは，我々が目的とする大域的計算環境の構築に求められるセキュリティ上の要件を満たすに十分なものである．

一般には (a) の方式には先に述べた 1，3 の問題点があるが，Java を用いる場合には次のように反論できる．

(1) 高度な最適化を伴う JIT（Just-In-Time）コンパイラ（ネイティブコードに変換しながら実行するシステム）の開発が進みつつあり，前述 1 の速度が遅いという問題点は改善されつつある．特に，Ninflet システムが想定する運用形態では，数値計算主体のプログラムが主となるが，

そのようなプログラムには，非オブジェクト指向の従来型の最適化コンパイル技術の大部分が適用できるため，その技術的困難さは小さい☆．そして，今後も存続するであろうJavaのインターネットにおける市場価値から，この技術開発は，他の言語に比較して急速に進み続けるだろうと期待できる．

(2) 従来より，科学技術計算のプログラミングにはFORTRANが用いられてきており，他の言語へのシフトは起きにくいと考えられてきたが，並列プログラムを容易に構成するといった観点からオブジェクト指向技術を応用した科学技術計算の重要性[10]が認識されるようになってきており，そのようななかでJava言語は，単なるインターネット関連ソフトウェアの開発用としてだけでなく，その理解のしやすさ，安全性等から，汎用のプログラミング言語としても高い評価を受けており，科学技術計算にも利用しようという動きが活発化[11]してきている．このことから，前述3の言語が限定されるという問題点は，ある程度許容できる．

これらの点から，大域的計算環境の実現のためにJavaを採用することは，現時点において最適な選択の1つであると考える．

4. Ninfletシステムのアーキテクチャ

以下，本章で，Ninfletシステムのアーキテクチャを示す．まず4.1節で，システムの基本構造の説明として，その構成要素，システムが稼動を開始するまでの処理過程，ジョブの発行方法，発行されたジョブが実行開始に至るまでの処理過程について述べる．次に，4.2節で，計算機提供者の立場で見たときの特徴の説明として，システムのインストールに必要な作業と，その後の運用において留意すべきサーバの設定方法について述べる．次に，4.3節で，計算機提供者によって実行中のジョブを「退去」させる機能について説明する．そして，4.4節で，提供された計算機上でのセキュリティに関してどのように安全であるか，また，応用プログラムにどのような制限が課されるかについて説明する．

☆ ただし，言語が提供する多次元配列が配列の配列として実現されている点と，配列インデックスの範囲チェックが言語仕様により必須となっている点が，従来の最適化技法の適用を難しくする場合がある．しかし，前者については，Javaの多次元配列を使用せずに，一次元配列をアドレス計算してアクセスすることで回避でき，後者については，範囲チェックを省略する最適化が可能な単純ループも多いと考えられる．

4.1 デーモンの稼働準備からNinfletの実行まで

4.1.1 システムの構成要素

Ninfletシステムの利用者には，計算機の貸出しをする**計算機提供者**の立場と，計算の実行を依頼する**ユーザプログラマ**の立場があり，Ninfletシステムは，以下のプログラム群によって構成される．

Ninflet Server（以下，単に「**Server**」）計算機提供者によって起動，運用されるデーモンプログラムで，発行されたジョブの実行受け入れと退去の機能を司る．

Ninflet Dispatcher（以下，単に「**Dispatcher**」）Ninfletシステム全体の管理者によって起動されるデーモンプログラムで，複数のServerを論理的に1つの計算機として扱うための機能を司る．

Ninfletプログラム ユーザプログラマによって書かれる応用プログラムのうち，Server上で実行される部分のコード．

Clientプログラム ユーザプログラマによって書かれる応用プログラムのうち，ジョブを発行する者のローカル計算機上で実行される部分のコードで，ジョブの発行と計算結果の取得の機能を司る．

HTTPサーバ（ないしFTPサーバ） Ninfletプログラムを置いておくための汎用Webサーバ．

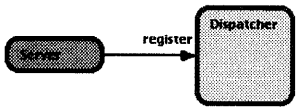
4.1.2 システムが稼動を開始するまでの処理過程

計算機提供者は，その計算機上でServerをデーモンプロセスとして稼動させておく．この起動の際に，1つのDispatcherをURLで指定する．これにより，このとき指定したDispatcherから，実行すべきジョブが指示されることになる（図1(a)）．

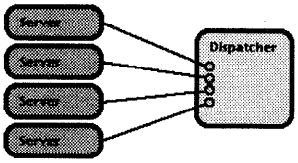
このとき，どこかのホスト上ですでにDispatcherデーモンが稼動している必要がある．あるServerが起動されると，そのServerの登録要求がDispatcherに対して発行され，これを受けたDispatcherは，そのServerを利用可能なServerとして自分自身に登録する（図1(b)）．

4.1.3 ジョブの発行方法

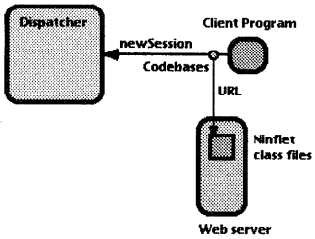
一方，ユーザプログラマは，実行させたい計算を2つのプログラムとして実装する．1つはServer上で実行されるべきNinfletプログラム（これを単に「**Ninflet**」と呼ぶ）で，`JP.go.etl.ninflet.Ninflet`のサブクラスとして作成する．もう1つはユーザプログラマが所有しているホスト上で実行させるClientプログラムである．ユーザプログラマは作成したNinflet（とそれが利用するクラス群）を，自分が利用できる任意のウェブサーバ(HTTPサーバないしFTPサーバ）上に置く．そしてClientプログラム内で，そのNinflet



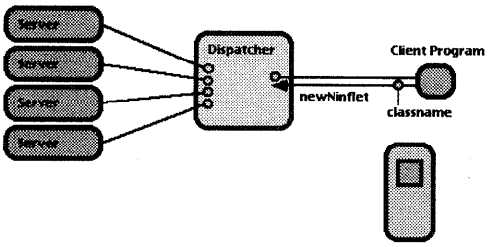
(a) Server は起動時にある Dispatcher に自分自身を登録する.



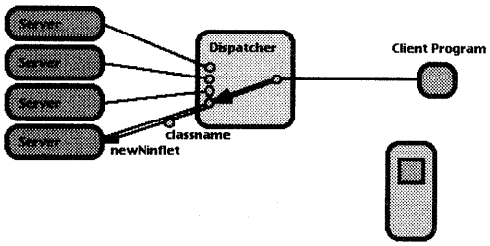
(b) 複数の Server が Dispatcher に登録される.



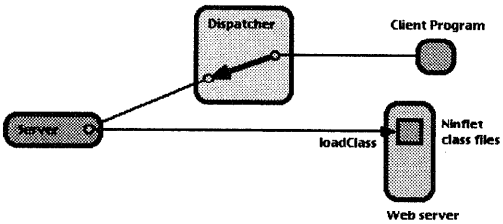
(c) Client は Ninflet の URL を指定して Session を作成する.



(d) Client は Session に対してクラス名を指定して newNinflet を呼ぶ.



(e) Dispatcher は利用可能な Server の中から 1 つを選ぶ.



(f) 選ばれた Server が目的のクラスをロードして インスタンス化する.

図 1 Ninflet の起動プロセス
Fig. 1 The launching process of Ninflet.

プログラムの所在を URL で指定する（図 1 (c)）.

4.1.4 ジョブが実行開始に至るまでの処理過程

ユーザプログラマは，計算を実行させたいとき，作成した Client プログラムを起動する．このときどれか 1 つの Dispatcher を指定する．Client プログラムは，起動されるとまず，指定された Dispatcher に対するリモート参照を取得する．次に，この Dispatcher に対して **Session** インスタンスの作成を依頼する．このとき，**Codebases** クラスのインスタンスに，サーバで実行されるクラスのパッケージ名とそのクラスファイルの所在 URL との対応表を渡しておく（図 1 (c)）．これは，後に目的の Ninflet がインスタンス化される際に，Server によって使用される．

たとえば，Client プログラムは**図 2** のように作成される．Codebases クラスのインスタンスに，作成した Ninflet の所在 URL を設定している．具体的には，パッケージ `JP.ac.nitech.center.takagi.ninflet` 以下のクラスは `http://www.center.nitech.ac.jp/~takagi/ninflet/` を codebase とするよう指定している．そして，newSession メソッドで，指定した

```
::::::::::::::
Main.java
::::::::::::::
package JP.ac.nitech.center.takagi.ninflet.examples.matmult;
import JP.go.etl.ninflet.client.Client;
import JP.go.etl.ninflet.Codebase;
import JP.go.etl.ninflet.dispatcher.Dispatcher;
import JP.go.etl.ninflet.dispatcher.Session;
import JP.go.etl.ninflet.dispatcher.LANClusterResourceManager;
import java.net.*;
public class Main {
  public static void main(String[] args) {
    Dispatcher dispatcher = Client.getDispatcher(
      "rmi://ninf.etl.go.jp/ninflet/dispatcher"
    );
    Codebase codebase = new Codebase();
    codebase.add(
      "JP.ac.nitech.center.takagi.ninflet",
      "http://www.center.nitech.ac.jp/~takagi/ninflet/"
    );
    Session session = dispatcher.newSession(
      codebase, new LANClusterResourceManager()
    );
    Matrix m1;
    Matrix m2;
    int numOfWorkers;
//...
    NinfletStub master = session.newNinflet(
      MatMultMaster.class, new MatMultMasterArg(
        numOfWokers, m1, m2
      )
    );
    NinfletResult r = master.getResult();
    Matrix m3 = ((MatMultMasterResult)r).product
//...
  }
}
```

図 2 Client のプログラム例
Fig. 2 An example of Client programming.

Dispatcher 上に Session インスタンスを作成する際に，この codebase リストを渡している．

次に Client プログラムは，newNinflet メソッドにより，Ninflet のインスタンスを適当な Server 上に作成するよう Dispatcher に指示する（図 1 (d)）．これにより，Dispatcher は，自分に登録されている利用可能な Server の中から適当な 1 つの Server を選択し，これに対して，Client プログラムにより指定された Ninflet クラスのインスタンスを生成するよう指示する（図 1 (e)）．

Server は指定された Ninflet クラスのインスタンスを生成する際に，まず，そのインスタンス専用のクラスローダとして NinfletClassLoader のインスタンスを生成する．このとき，先ほど設定された Codebases がそのクラスローダに渡され，クラスローダはこの情報に基づいて，要求されている Ninflet クラスのクラスファイルを socket 経由でロードする（図 1 (f)）．また，同時に必要となる他のクラスもこのクラスローダによりオンデマンドにロードされる．

そしてロードされたクラスのインスタンスが 1 つ作られ，これへのリモートリファレンスが Dispatcher へ，そして Client へと返される．そしてその後は，Client プログラムから直接 Server 上の Ninflet インスタンスを呼び出す．

4.1.5 Dispatcher の運用

Dispatcher を誰がどこで稼働させておくかは，システムの運用形態によって異なる．LAN 上のワークステーションクラスタ用の並列実行環境として使用する場合には，システム管理者によって，クラスタ上の 1 台のワークステーション上で 1 つの Dispatcher を稼働させておき，クラスタを構成する各ワークステーションで，Dispatcher としてこれを指定して Server デーモンを稼働させておく．

地球規模の遊休計算機相互利用環境を構築する場合には，誰かが 1 つの Dispatcher をどこかで稼働させておくことになる．ただし，1 つの Dispatcher に対して膨大な数の Server が登録されればその Dispatcher が処理のボトルネックとなるので，これを避けるために，複数の Dispatcher が互いに接続しあって 1 つの論理的な Dispatcher を構成する機能も持たせている．

4.2 システムのインストールと運用管理

4.2.1 Server のインストール

Server のインストールは次の手順で行う．まずコアプログラムのパッケージを所定のサイトからダウンロードする．これには Server の立ち上げに必要な ClassLoader や SecurityManager 等の最小限の機能が含まれている．コアプログラムを Dispatcher を指定して起動すると，その Dispatcher のサイトから，Server 本体のパッケージがダウンロードされ，Server が自動的に起動される．これによって 1 つの論理 Dispatcher に登録される各 Server のシステムバージョンを一致させることができる．

4.2.2 Server 管理

Server では，Dispatcher からの Ninflet 発行要求の受入れの許可/拒否の条件を設定することができる．設定の変更は，Server デーモン稼働中であっても，コマンドラインから，あるいは，ウェブブラウザから操作することができる．このために Server は簡易 HTTP サーバの機能を持っており，HTML の FORM や Java Applet を用いてウェブブラウザから Server をコントロールできる．

最も基本的な設定項目は，Ninflet 受入れの許可/不許可の即時切替えである．計算機提供者は，自分がその計算機を使用していない間だけ Ninflet の受入れを許可することができる．不許可に設定されている場合，Dispatcher はその Server に Ninflet インスタンスの作成を依頼することをしない．また，タイマー機能により，指定した時間帯の間だけ許可状態に切り替えるよう設定することができる．これにより，不在にする夜間，休日だけ毎日許可するといったことが可能となる．

受入れ設定が不許可に切り替えられる際に，すでに実行中の Ninflet が存在した場合，その Ninflet に対して**退去命令**が発行される．退去命令を受けた Ninflet の動作については次節で述べるが，Ninflet の退去動作にある程度の時間がかかる（退去命令が発行されてもしばらくの間負荷を消費し続ける）場合がある．そこで計算機提供者は**退去猶予時間**を設定できるようにしている．退去命令を発行した後，退去猶予時間として設定された時間が過ぎても Ninflet が負荷を消費しているようであれば，強制退去命令が発行されて，Ninflet のすべての Thread が中止させられる．また強制退去ボタンも用意している．

これらの機能により，長時間にわたり実行を続ける性質を持つ Ninflet を受け入れても，計算機提供者に不利益のないよう配慮されており，安心して計算機を提供いただけるようになっている．

4.3 フォールトトレランス

十分な数の計算機提供者を確保するためは，計算機提供者に不利益が及ばないよう配慮する必要があり，Server により Ninflet の実行が途中で中止させられることは受け入れざるをえない．一方，特に科学技術計

算を目的とした利用においては，1 つの Ninflet の実行が長時間に及ぶ場合が多く，途中で中止させられてしまう可能性も高くなる．実行が長時間に及ぶものほど，中止された際の損失は大きい．

そこで，Ninflet には，退去命令が Server によって発行されたときに，計算の途中結果を退避させる機能を用意している．退避は，Dispatcher により指示された場所に対してなされる．退避される場所には，別の Server や一時保管用ファイルサーバが指定される．退避された Ninflet は，Dispatcher により再割当てされる別の Server 上で再実行を開始することができる．

このような，計算の途中結果の退避および実行の再開の仕組みは，従来より checkpointing として知られ，長時間の実行を要する科学技術計算のアプリケーション等において，しばしば，プログラマによって明示的にプログラムされている．コンパイラによる自動的な checkpointing の研究例[12] もあるが，現在の版の Ninflet では，ユーザプログラマによる明示的な checkpointing が必要となっている．

すなわち，ユーザプログラマは，まず，Ninflet の計算ルーチン中のループ内の適切な位置に checkpoint() メソッドの呼び出しを挿入する．これにより，スーパークラスである NinfletImpl クラスがオブジェクトの退避を行う．そして，ユーザプログラマは，この Ninflet が再実行させられた際に，計算ルーチン中のすでに済んでいる部分をスキップするように，アルゴリズムを工夫しておく．

checkpointing には以下の 3 レベルを用意した．

レベル 1 checkpoint() メソッドが呼ばれるごとに状態を退避（Dispatcher から指示された場所へ退避）する．

レベル 2 checkpoint では現在の状態の複製を作成して内部に保持しておき，Server によって退去命令が発行されると即座に，前回作成した複製を退避する．

レベル 3 通常 checkpoint では何もしないでおき，Server によって退去命令が発行されたら，次の checkpoint で状態を退避する．

これらは次のように使い分けることができる．レベル 1 の利用は，Server の稼働しているホスト自体がクラッシュあるいはネットワーク的に unreachable になる場合にも備えることになるが，checkpointing のオーバヘッドが大きく，checkpoint の挿入位置に十分に配慮せねばならない．レベル 3 の利用は，最も checkpointing のオーバヘッドが少なく，無駄になる計算の量も少ないが，Server 管理者が，退去猶予時間を短く設定している場合に強制的に中止されてしまう可能性がある．レベル 2 の利用は，比較的小さなオーバヘッドである程度の安全性を確保できるが，計算が少し無駄になることがある．これらは，プログラムのアルゴリズムの特性に応じて選択する．

退避された Ninflet が別の Server 上で再実行される場合，Client プログラムのインスタンスが保持している Ninflet オブジェクトへのリモートリファレンスも，これに応じて指す場所を変更する必要がある．これがユーザプログラマが意識する必要なく実現されるために，newNinflet メソッドで返されるオブジェクトは，RMI の stub の外にもう一皮かぶせた NinfletStub クラスのインスタンスとしており，この NinfletStub が参照先の自動更新の役割を担っている．具体的には，Dispatcher により，このオブジェクトが別の Server に移動したことが通知されると，ここで指定される新しい Ninflet リモートリファレンスに切替えるという動作をする．またこの機能により同時に，Server が応答しなくなって RemoteException が発生したような場合にも，新たな Ninflet オブジェクトを別の Server に作成して再実行するよう Dispatcher に指示することも実現している．

4.4 セキュリティモデル

Ninflet のセキュリティモデルは `JP.go.etl.ninflet.server.NinfletSecurityManager` によって定義され，以下のポリシーを持つ．

- AWT によるウィンドウの作成を禁止．
- スレッドの操作を禁止．
- RMI 以外によるネットワーク接続を禁止．

その他，AppletSecurityManager と同様に，ローカルファイルへのアクセス，System.exit()，新たな ClassLoader 作成等の操作を禁止をしている．

Applet のセキュリティモデルと大きく異なる点は，AWT の操作やスレッドの操作も禁止している点である．

AWT 操作の禁止によって，計算機提供者が望まないウィンドウがユーザプログラマの意図によって Server 計算機上に出されてしまうといった事態が起きないことが保証される．計算機提供者は，CPU サイクルとメモリだけを貸し出すのであり，それ以外は何ら影響を受けない．Ninflet では計算だけを目的しており，Ninflet プログラミングにおけるこの制約は許容されるだろう．計算結果の可視化やパラメータ入力の GUI が必要であれば，クライアントプログラム上で行う．

スレッド操作を禁止しているのは，1 個の Ninflet インスタンスが消費する負荷の大きさを固定化するこ

とで，Dispatcher によるスケジューリングをやりやすくするためである．応用プログラムの実行時並列性は，複数の Ninflet の同時実行によってのみ提供される．通信と計算をオーバラップさせる並列性は，スレッドによって実現しているが，これは NinfletStub 内で実装しており，セキュリティチェックの対象から除外している．

また，標準の RMISecutiryManager のモデルと異なり，クラスファイルを取ってきたホスト以外に対する RMI 接続を許可している．これは Applet の，codebase のホストにしか接続を許可しないというセキュリティポリシーよりも緩くなっており，Ninflet 間での直接呼び出しを可能にしている．

5. 並列処理への応用

5.1 Master-Worker パターンによる並列実行

Ninflet では，Java のオブジェクト指向言語としてのメカニズムを活用して，並列アルゴリズムを，オブジェクト間のパターンとして抽象化し，クラスライブラリの形で並列計算をサポートする．今後，様々な並列処理を支援するクラスライブラリをあわせて提供していく計画であるが，現時点ではまず，Master-Worker（あるいは Master-Slave とも呼ばれる）パターンに基づいた並列実行を支援するライブラリを用意している（図 3）．

Master-Worker パターンは，問題が多数の独立な小問題に分割できる場合に適用できるもので，分割された小問題を Worker としてプログラムする．Master

```
::::::::::::::
Master.java
::::::::::::::
package JP.go.etl.ninflet.lib.masterworker;
import JP.go.etl.ninflet.Ninflet;
import JP.go.etl.ninflet.NinfletImpl;
import JP.go.etl.ninflet.NinfletStub;
import JP.go.etl.ninflet.NinfletArg;
import JP.go.etl.ninflet.NinfletResult;
import java.rmi.RemoteException;
public abstract class Master extends NinfletImpl {
  protected int numOfWorkers;
  public Master() throws RemoteException {}
  protected void init(NinfletArg arg) {
    numOfWorkers = ((MasterArg)arg).numOfWorkers;
  }
  public void run() {
    NinfletStub[] workers = new NinfletStub[numOfWorkers];
    Class workerClass = getWorkerClass();
    for (int i = 0; i < workers.length; i++) {
      NinfletArg arg = getWorkerArg(i);
      workers[i] = this.session.newNinflet(workerClass, arg);
    }
    for (int i = 0; i < workers.length; i++) {
      combineResultOf(workers[i].getResult(), i);
    }
    result = createMasterResult();
  }
  protected abstract Class getWorkerClass();
  protected abstract NinfletArg getWorkerArg(int workerNum);
  protected abstract void combineResultOf(
    NinfletResult result, int workerNum
  );
  protected abstract NinfletResult createMasterResult();
}
::::::::::::::
Worker.java
::::::::::::::
package JP.go.etl.ninflet.lib.masterworker;
import JP.go.etl.ninflet.NinfletImpl;
import java.rmi.RemoteException;
public abstract class Worker extends NinfletImpl {
    public Worker() throws RemoteException {}
}
```

図 3 Master-Worker ライブラリの実装
Fig. 3 Implementation of the Master-Worker library.

```
::::::::::::::
MatMultWorker.java
::::::::::::::
package JP.ac.nitech.center.takagi.ninflet.examples.matmult;
import JP.go.etl.ninflet.Ninflet;
import JP.go.etl.ninflet.NinfletArg;
import JP.go.etl.ninflet.lib.masterworker.Worker;
import java.rmi.RemoteException;
public class MatMultWorker extends Worker implements Ninflet {
  public MatMultWorker() throws RemoteException {}
  Matrix m1, m2, m3;
  protected void init(NinfletArg arg) {
    super.init(arg);
    m1 = ((MatMultWorkerArg)arg).m1;
    m2 = ((MatMultWorkerArg)arg).m2;
    m3 = new Matrix(m1.rows, m2.cols);
  }
  public void run() {
    for (int r = 0; r < m1.rows; r++) {
      for (int c = 0; c < m2.cols; c++) {
        double sum = 0;
        for (int k = 0; k < m1.cols; k++) {
          sum += m1.value[r][k] * m2.value[k][c];
        }
        m3.value[r][c] = sum;
      }
    }
    result = new MatMultWorkerResult(m3);
  }
}
::::::::::::::
MatMultMaster.java
::::::::::::::
package JP.ac.nitech.center.takagi.ninflet.examples.matmult;
import JP.go.etl.ninflet.Ninflet;
import JP.go.etl.ninflet.NinfletArg;
import JP.go.etl.ninflet.NinfletResult;
import JP.go.etl.ninflet.lib.masterworker.Master;
import JP.go.etl.ninflet.lib.masterworker.Worker;
import java.rmi.RemoteException;
public class MatMultMaster extends Master implements Ninflet {
  public MatMultMaster() throws RemoteException {}
  Matrix m1, m2, m3;
  protected void init(NinfletArg arg) {
    super.init(arg);
    m1 = ((MatMultMasterArg)arg).m1;
    m2 = ((MatMultMasterArg)arg).m2;
    m3 = new Matrix(m1.rows, m2.cols);
    subSize = (float)m1.rows / (float)this.numOfWorkers;
  }
  protected Class getWorkerClass() {
    return MatMultWorker.class;
  }
  protected NinfletArg getWorkerArg(int workerNum) {
    int row = rowStart(workerNum);
    int rowLength = rowStart(workerNum + 1) - row;
    Matrix m1Sub = m1.getSubmatrix(row, rowLength, 0, m1.cols);
    return new MatMultWorkerArg(m1Sub, m2);
  }
  protected void combineResultOf(
    NinfletResult r, int workerNum
  ) {
    int row = rowStart(workerNum);
    int rowLength = rowStart(workerNum + 1) - row;
    m3.putSubmatrix(
      row, rowLength, 0, m3.cols,
      ((MatMultWorkerResult)r).product
    );
  }
  private int rowStart(int workerNum) {
//...
  }
  protected NinfletResult createMasterResult() {
    return new MatMultMasterResult(m3);
  }
}
```

図 4 Master-Worker パターンを使った行列積プログラム
Fig. 4 Matrix multiplication using the Master-Worker pattern.

は問題を分割して Worker を生成する役割と，Worker の結果を結合して最終的な解を構成する役割を担う．

Ninflet システムでは，`JP.go.etl.ninflet.lib.masterworker` パッケージの Master クラスおよび Worker クラスをサブクラス化することで，アプリケーションプログラムを作成する．**図 4** に行列積のプログラム例を示す．Worker のサブクラスとして作成した MatMultWorker では，run() メソッドに計算のメインループを記述し，init() メソッドに部分問題のパラメータに従った初期設定を記述している．そして，Master のサブクラスとして作成した MatMultMaster では，getWorkerClass メソッドで Worker のクラスを指定し，getWorkerArg メソッドで Worker の初期値，すなわち問題の部分問題への分割方法を記述し，combineResultOf メソッドで Worker の結果を結合して最終解を構築する方法を記述している．MatMultMasetr は Client プログラムの一部として実行され，MatMultWorker は Ninflet として Ninflet Server 上で実行される．

Ninflet 間の通信や，Worker のスケジューリングのためのコードは，ライブラリとして用意されているスーパークラスの Master クラス内に書かれている．そのため一般ユーザプログラマはこれらを意識する必要がなく，対象問題固有のコードだけを記述すればよいようになっている．

5.2 実 行 例

予備評価として，LAN 上のワークステーションクラスタで MatMult を実行させた場合の性能評価を行った．ワークステーションクラスタとして，電子技術総合研究所の ETL-Wiz（100Base/TX Ethernet スイッチで結ばれた 32 台の Alpha ワークステーションから構成される）を使用した．Java 実行系には DIGITAL JDK V1.1.5 を使用し，JIT コンパイラおよび native threads を使用した．

図 5 に，その並列実行による速度向上率のグラフを示す．横軸は Worker の数を表しており，各線は行列のサイズ（$n \times n$ の正方行列の n）を表している．n は 200 から 1000 へと変化させた．

このグラフから，n が小さいと，Worker の実行時間が無視できるほどに小さくなるため，Ninflet のインスタンス化時間が性能に対して支配的になり，Worker 数が大きくなると，データ転送オーバヘッドが支配的になることが読みとられる．この結果は，Master-Worker パターンに基づいた並列処理の典型的な性能特性を表しているわけであるが，粒度の小さい問題においても実用的な並列処理を実現するためには，Ninflet のインスタンス化のコストや，Master-Worker 間のデータ転送のコストを小さくする必要があり，さらに詳しい分析が必要である．

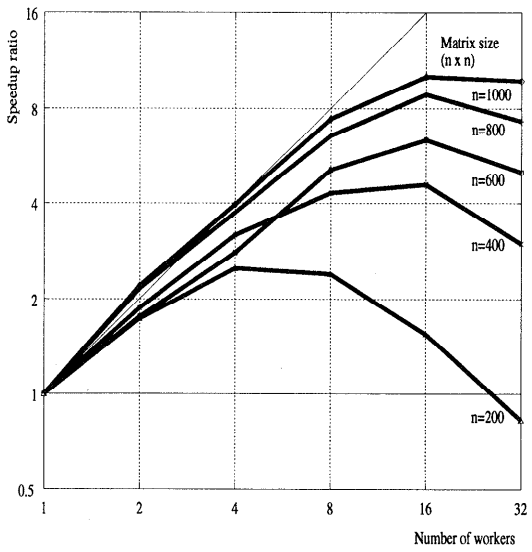


図 5 ETL-Wiz における MatMult の並列実行による速度向上率
Fig. 5 Speedup ratio of MatMult execution.

6. 関 連 研 究

Java による分散計算環境のうち，Java Applet を利用したものとして，JET[13]，Javelin[14]，Charlotte[15] 等がある．これらは，目的の計算ルーチンを Java Applet として実装し，計算機提供者にその Applet の張り付けられたウェブページを開いてもらうことによって計算を実行してもらうというアイデアに基づいたものである．

JET では，単に作成した Applet を Web ページに置くだけのものであり，実際にそのページを計算機提供者に開いてもらうためには何らかの別の手段が必要となっているが，Javelin の場合には，計算 Applet とは別の Broker Applet がこの役割を果たしている．

Javelin では，計算機提供者は，まず Web ブラウザを起動し Javelin の Broker Applet のページを開いておく（これを Host と呼んでいる）．一方，ユーザプログラマは，Applet として作成したプログラムとそれを張り付けた HTML（これを Task と呼んでいる）を自分の Web サーバ上に置いたうえで，Broker に対してその URL を通知しておく．Broker は Task キューから 1 つの Task を，また Host プールから 1 つの Host を選びだし，Host に対してその Task のページを開くように指示する（これは Broker Applet が AppletContext クラスの showDocument メソッ

ドを呼ぶことにより実現されている).

Javelin の Broker, Task, Host はそれぞれ, Ninflet の Dispatcher, Ninflet, Server に対応するといえる. Javelin は Ninflet システムに比較して, Web ブラウザという既存のインフラを使用してるため, Server のインストールが不要であるという特長がある. Javelin のグループは, これによって誰でも気軽に計算に参加できるのだと主張している. しかし, 継続的に自分の計算機を提供するには, つねに Web ブラウザを動かしてそのページを開いておく必要があり, 長期にわたって協力してくれる計算資源提供者がどれだけ得られるかにはやや疑問がある. また, LAN 上のワークステーションをクラスタ計算機として利用するためにこれを使用することを考えると, ワークステーションごとにウェブブラウザを起動してつねに特定のページを開かせておくのは繁雑であり, 現実的でない. Javelin のような Applet によるシステムは, RSA Factoring[16] や一部の scientific grand challenge 問題等, 一時期に集中して大衆が興味を持ち, ボランティアの協力が得られる問題には適しているかもしれないが, 不特定の者による永続的利用を目的とするには Ninflet システムの方が向いていると考える.

また, Applet を利用した方法では, Applet のセキュリティモデルの制約により, その Applet の置かれていたホストとしか通信が許されないため, 複数の Applet 間の通信を利用した並列処理を実現しようとすると, その Applet の置かれていたホストで通信を仲介する必要があり, 台数が多くなるとこれがボトルネックとなるという問題もある. Javelin では, デジタル署名付き Applet を利用することによりセキュリティ上の制約をなくすと主張しているが, 一般に, 署名付きアプレットの実行を承認できるのは, 署名者を信頼できる場合に限られるのであり, 不特定の者が作成したプログラムの実行を可能にするシステムを実現するうえでは, この方法は向いていない. Ninflet は独自のセキュリティモデルを持つためこのような問題がない.

次に, Applet ではなくスタンドアロンアプリケーションとして Java で専用のシステムを構築しているものとしては, IceT[17] や POPCORN[18] 等がある. IceT はクラスの「uploading 機能」を持っており, これにより他の IceT ホストに対して目的のクラスをインスタンス化させることを可能としている. しかし IceT は PVM のインターフェイスをそのまま取り入れた設計となっており, Java のオブジェクト指向メカニズムが有効に活用されていない.

一方, 数値計算以外を目的としたシステムに目を向けると, Java をベースとした, ObjectSpace の Voyager[19] や IBM 東京基礎研の Aglets[20], 富士通研の Kafka[21] 等のモバイルエージェントシステムが, ユーザが記述したプログラムを他のホストに送付して実行させる機能を持つという点で Ninflet と共通している. Ninflet は, これらを数値計算をターゲットとして特化したものと見ることもできる. また, 使用言語を特定せずバイナリコードでプログラムを送付することを実現し, OS レベルで安全性を保証するシステムとして, Planet[22] 等がある. これらを数値計算目的に応用するアプローチもありうるだろう.

7. おわりに

不特定の利用者が記述したプログラムをインターネット上の遊休計算機に実行させることで, 世界規模の共同利用計算機環境を実現する,「Ninflet」システムを提案した. Ninflet は Java のインフラを利用することで安全性を保証している. このシステムが普及する鍵は, Server を立ち上げて Dispatcher に登録してくれる計算機提供者がどれだけ現れてくれるかにある. Java の安全性が広く一般に十分に知れ渡っていることが, ある程度協力を得やすくしていると期待する.

また, ワークステーションクラスタ上に並列計算環境を構築するために Ninflet システムを用いる場合の応用例として, Master-Worker パターンによる並列プログラミングの例を紹介した. このときプログラムの作成は, オブジェクト指向デザインパターンに基づいた差分プログラミングで行われることを示した. さらに, サーバの安定性が保証されない環境で重要となるフォールトトレランスのために, checkpointing の機能を用意した. このように Ninflet システムは LAN から WAN までをシームレスに統合するオブジェクト指向大域並列計算環境である.

今後の課題として以下を計画している.

- 効率的な実行を支援する, 広域分散計算のためのリソース割当てアルゴリズムの検討. 我々は Ninf の枠組の上で, 広域ネットワークにおけるジョブスケジューリングの研究をシミュレーションベースで進めている[23]. この結果に基づいたアルゴリズムを Ninflet 用のスケジューラとしても実装する.
- Ninflet のための様々なプログラミングスタイルを確立し, クラスライブラリを提供する. これまでに, C++ 上において, ライブラリベースの並列プログラミングスタイルが提案[10] されている

が，これに checkpointing を加味した新たなスタイルを模索する．

- 補助的な機能として，いわゆるスクリーンセーバのように一定時間以上キーボードやマウスを操作していない状態が続くと，Server の Ninflet 受け入れ許可モードに移行し，再びキーボードやマウスを操作すると受け入れ拒否モードに移行するといった仕組みを実現する．これにより Server を利用できる時間をより多く確保できるであろう．
- ユーザプログラマの利用機会に関する公平さを保つための仕組みが必要となるであろう．ユーザの管理を行い，使用頻度の低いユーザを優先するといったスケジューリングを実現する．
- 広域ネットワーク上で実際に運用した場合の，各 Server の稼働率と，Ninflet システムの効率（非 Java 環境でローカルに実行した場合に対する実行時間比）との関係を定量的に評価する．
- 故意に誤った結果を返してくる不当なサーバからの防衛手段を提供する．

参考文献

1) Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U. and Takagi, H.: Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture, *Proc. High-Performance Computing and Networking*, Lecture Notes in Computer Science, Vol.1225, pp.491–502 (1997).
2) 中田，高木，松岡，長嶋，佐藤，関口：Ninfによる広域分散並列計算，情報処理学会論文誌，Vol.39, No.6, pp.1818–1826 (1998).
3) Takefusa, A., Matsuoka, S., Ogawa, H., Nakada, H., Takagi, H., Sato, M., Sekiguchi, S. and Nagashima, U.: Multi-client LAN/WAN Performance Analysis of Ninf: A High-Performance Global Computing System, *Proc. Supercomputing'97* (1997).
4) Casanova, H. and Dongarra, J.: Netsolve: A Network Server for Solving Computational Science Problems, *Proc. Supercomputing'96* (1996).
5) The RSA Data Security Secret-Key Challenge, `http://www.rsa.com/rsalabs/97challenge/`.
6) DISTRIBUTED.NET – The Fastest Computer on Earth, `http://www.distributed.net/`.
7) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proc. 7th IEEE International Symposium on High Performance Distributed Computing*, pp.140–146 (1998).
8) Safe-Tcl, `http://sunscript.sun.com/plugin/safetcl.html`.
9) Wahbe, R., Lucco, S., Anderson, T.E. and Graham, S.L.: Efficient software-based fault isolation, *14th Symposium on Operating Systems Principles*, pp.203–216 (1993).
10) Ishikawa, Y.: Towards a Parallel C++ Programming Language Based on Commodity Object-Oriented Programming, *1997 International Scientific Computing in Object-Oriented Parallel Environments Conference* (*ISCOPE'97*) (1997).
11) ACM 1997 Workshop on Java for Science and Engineering Computation, `http://www.cs.rochester.edu/u/wei/javaworkshop.html`.
12) Beck, M., Plank, J.S. and Kingsley, G.: Compiler-Assisted Checkpointing, *25th Annual International Symposium on Fault-Tolerant Computing* (1995).
13) Silva, L.M., Pedroso, H., Silva, J.G.: The Design of JET: A Java Library for Embarrassingly Parallel Applications, *Proc. WOTUG'20 – Parallel Programming and Java Conference*, pp.210–228, IOS Press (1997).
14) Christiansen, B.O., Cappello, P., Ionescu, M.F., Neary, M.O., Schauser, K.E. and Wu, D.: Javelin: Internet-Based Parallel Computing Using Java, *ACM Workshop on Java for Science and Engineering Computation* (1997).
15) Baratloo, A., Karaul, M., Kedem, Z. and Wyckoff, P.: Charlotte: Metacomputing on the Web, *Proc. 9th International Conference on Parallel and Distributed Computing Systems* (1996).
16) RSA Factoring-By-Web project!, `http://www.npac.syr.edu/factoring.html`.
17) Gray, P.A. and Sunderam, V.S.: The IceT Environment for Parallel and Distributed Computing, *Proc. 1997 International Scientific Computing in Object-Oriented Parallel Environments Conference* (1997).
18) The POPCORN Project, `http://www.cs.huji.ac.il/~popcorn/`.
19) ObjectSpace Voyager – The Agent ORB for Java, `http://www.objectspace.com/voyager/`.
20) Programming Mobile Agents in Java, `http://www.trl.ibm.co.jp/aglets/`.
21) Kafka: Yet Another Multi-Agent Library for Java, `http://www.fujitsu.co.jp/hypertext/free/kafka/`.
22) 松原，加藤：分散永続性を提供するモーバイルオブジェクト・システムの実現，情報処理学会論

文誌, Vol.39, No.8, pp.2494–2508 (1998).
23) 竹房, 合田, 小川, 中田, 松岡, 佐藤, 関口, 長嶋：グローバルコンピューティングシステムのシミュレーションによる評価, 情報処理学会論文誌, Vol.40, No.5, pp.2192–2202 (1999).

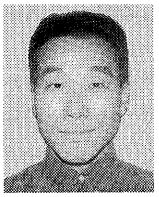
(平成10年9月1日受付)
(平成11年3月5日採録)

高木 浩光（正会員）

昭和42年生．平成元年名古屋工業大学工学部電気情報工学科卒業．平成6年同大学大学院工学研究科博士課程修了．博士（工学）．同年名古屋工業大学工学部付属情報処理教育センター助手．この間，VLIW型プロセッサのための同期機構とスケジューリング方式に関する研究等に従事．平成10年電子技術総合研究所入所．情報アーキテクチャ部主任研究官．グローバルコンピューティング，オブジェクト指向言語の最適化コンパイル方式等に興味を持つ．電子情報通信学会，ソフトウェア科学会各会員．

松岡 聡（正会員）

昭和38年生．昭和61年東京大学理学部情報科学科卒業，平成元年同大学大学院博士課程中退．同大学情報科学科助手，情報工学専攻講師を経て，平成8年より東京工業大学情報理工学研究科数理・計算科学専攻助教授．理学博士．オブジェクト指向言語，並列システム，リフレクティブ言語，制約言語，ユーザ・インタフェースソフトウェア等の研究に従事．現在進行中の代表的プロジェクトは，世界規模の高性能計算環境を構築するNinfプロジェクト，計算環境に適合・最適化を目指すJava言語の開放型Just-In-Timeコンパイラ OpenJIT，種々のユーザインタフェースソフトウェア等．並列自己反映型オブジェクト指向言語ABCL/R2の研究で1996年度情報処理学会論文賞受賞．1999年情報処理学会第7回坂井記念特別賞受賞．オブジェクト指向の国際学会ISOTAS'96，ECOOP'97，並列オブジェクトの国際会議ISCOPE'99のプログラム委員長等を務める．IEEE Concurrency edtior，ソフトウェア科学会，ACM，IEEE-CS各会員．

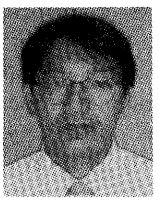


中田 秀基（正会員）

昭和42年生．平成2年東京大学工学部精密機械工学科卒業．平成7年同大学大学院工学系研究科情報工学専攻博士課程修了．博士（工学）．同年電子技術総合研究所研究官．並列プログラミング言語，オブジェクト指向言語，分散計算システムに関する研究に従事．

関口 智嗣（正会員）

昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和59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．情報アーキテクチャ部主任研究官．以来，データ駆動型スーパコンピュータSIGMA-1の開発等の研究に従事．並列数値アルゴリズム，計算機性能評価技術，ネットワークコンピューティングに興味を持つ．市村賞受賞．日本応用数理学会，ソフトウェア科学会，SIAM，IEEE各会員．



佐藤 三久（正会員）

昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和61年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成3年通産省電子技術総合研究所入所．平成8年より，新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術等の研究に従事．日本応用数理学会会員．

長嶋 雲兵（正会員）

昭和30年生．昭和58年北海道大学大学院博士後期課程修了．理学博士．同年岡崎国立共同研究機構分子科学研究所助手．平成4年お茶の水女子大学理学部情報科学科助教授．平成8年同教授．平成10年工業技術院物質工学工業技術研究所理論化学研究室長．理論化学，並列分散処理，性能評価の研究に従事．日本化学会，日本応用数理学会，IEEE，ACM各会員．