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]

Javaによる大域的並列計算環境「Ninflet」を提案する．Ninfletの目標は，インターネット上に多数存在するとみられる遊休計算機のふんだんな計算資源を有効利用しつつ，ローカル及びワイドエリアネットワークをシームレスに統合する，新世代のオブジェクト指向並列計算環境を構築することにある．Ninfletは，大域的計算環境を実現するために重要な要素となる，ポータビリティ，セキュリティ，リソース割り当て，checkpointing，オブジェクトmigrationの機能の実現にJavaの特長を活用する．

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]

Ninflet is a Java-based global computing system that builds on our experiences in the research of the *Ninf* system which facilitated RPC-based global computing. The goal of Ninflet is to become a new generation of concurrent object-oriented system which harness abundant idle computing powers and seamlessly integrate global as well as local network parallel computing. Ninflet is designed to make use of Java features to implement important features in global computing, such as resource allocation, inter-Ninflet communication, security, checkpointing, object migration, and easy server management via HTTP.

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挙げ始めている．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など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というものであり，実際に広域ネットワークへ応用できるのはごく特殊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)2)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stub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他にNetsolve[4)], Legion[5)]などがある．しかし，これらのシステムでも，サーバ上で実行されるルーチンの作成者は一部の者に限られる．作成したルーチンはサーバ管理者によって登録してもらう必要があり，このとき，サーバに危害を与えるものでないかなどについての審査を受けなければならない．

真の広域分散計算を実現するには，不特定多者に，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と考える．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，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」システムを構築した．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，不特定多者に対して，自らが作成した任意のプログラムを任意の計算サーバに計算させる機会を与えるものである．広域分散計算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にあるが，Ninfletシステムでは，サーバの安全性はJavaのSecurityManagerによって保証されるため，誰でも自分の計算機を安心して提供することができる．

このシステムにより，例えば，夜間の遊休計算機を

---

†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

地球の裏側の昼間の地域に貸し出して有効利用することが可能となる．日本の大学と米国の研究所の協力によって，互いに相手の計算機をそれが空いている時間帯に借りて交換し合うといった応用が期待できる．また，誰でも安心して自分の計算機を提供できることから，個人の PC や Mac 等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．近年のインターネットの大衆化に伴い，世界には膨大な数の遊休計算機が存在しているはずである．1997 年 10 月，RSA Data Security の The RSA Data Security Secret-Key Challenge6) において，distributed.net のチーム7) が広域分散処理によって破ったことが話題となった．56bit RC5 が 250 日ほどで破られ，この間 50 万もの異る IP アドレスの計算機が参加したという．このようないわば「ボランティア計算」ための汎用のシステムとして Ninflet システムを利用することもできる．distributed.net では，この問題専用のプログラムが各プラットフォーム用のバイナリ形式で配布され，参加者はこのバイナリを (信頼して) 手動でインストールし，計算に参加しているが，Ninflet システムを利用すれば，参加者はプログラムの安全性について信頼すべきか迷うことなく計算機を提供でき，また同時に，Java のプラットフォーム非依存性から各プラットフォーム用のバイナリを用意する必要がない．

さらに，Ninflet システムを PVM や MPI のように，主に 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 は Java のオブジェクト指向システムを活用して，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サポートしている．また，利用可能な資源が事前に定まらないこのような環境に適した，様々なリソース割り当てマネージャを搭載可能にしている．これらの特徴を活かすことで，例えば，LAN 上のワークステーションクラスタ上で動作する並列プログラムを地球の裏側から利用するといったことも可能となる．このように Ninflet は，ローカル及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．

以下，2 章では，大域並列計算に求められる要件を再度整理し，Java がいかにこれに応えるものであるかについて述べ，Java の数値計算への応用の妥当性を検証する．次に 3 章で，Ninflet システムのアーキテクチャとして，概要と，インストールから，準備，実行までの具体的な手順を説明し，サーバの運営管理方法について，また，Ninflet migration と checkpointing の機能について説明する．そして 4 章では，並列処理への応用例として Master-Worker パターンに基づくプログラム例を紹介する．そして 6 章で関連研究について比較検討する．

2. 広域分散計算への 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採り入れる場合には，セキュリティ上の問題を解決する必要がある．例えば，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挙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，インタプリタ上で危険な命令および処理の実行を禁止する．

(b) 実行時コンパイル方式　計算ルーチンをソースコードでサーバへ送付し，これを受け取ったサーバホストがこれをコンパイルして実行する．このときコンパイル前のプリプロセッサによって，危険な処理部分をソースコードレベルで検出し，危険なルーチンの実行を禁止する．

(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，実行不能にする．

これらそれぞれの方式の特質を比較すると，次のことが言える．

(1) (a) では一般に計算速度が遅い．

(2) (a), (b) では計算ルーチンを記述する言語が限定される．

(3) (c) 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) 方式によってこれを既に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，異なる ThreadGroup のスレッドに対する操作などの可否を制御する `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，そのひとつの応用である「Applet」のフレームワークにより既に実証されている．そしてこれらは，我々が目的とする広域分散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 (a) の方式には先に述べた 1, 2 の問題点があるが，Java を用いる場合には次のことがいえると考える．

(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．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[8)]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，安全性などから，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[9)]してきている．このことから，前述2.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，広域分散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一つであると考える．

3. Ninfletシステムのアーキテクチャ

3.1 daemonの稼働準備からNinfletの実行まで

Ninfletシステムの利用者には，計算機の貸し出しをする計算機提供者の立場と計算の依頼を発行するユーザプログラマの立場とがある．Ninfletシステムは，2つのdaemonプログラム「**Ninflet Server**」(以下単に「Server」)，「**Ninflet Dispatcher**」(以下単に「Dispatcher」)と，必要に応じて起動される**Client**プログラム，**Ninflet**プログラムから構成される．これにHTTPサーバ(ないしFTPサーバ)，WWWブラウザが併用される．

計算機提供者は，その計算機上でServerをdaemonプロセスとして稼動させておく．この起動の際に，少なくとも一つの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2(a))．

これとは別に，どこかのホスト上でDispatcher daemon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2(b))．

一方，ユーザプログラマは，実行させたい計算を二つのプログラムとして実装する．ひとつは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．もうひと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Web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プログラムの所在をURLで指定する(図2(c))．

```
::::::::::::::
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;
//...(abbreviated)
    NinfletStub master = session.newNinflet(
      MatMultMaster.class, new MatMultMasterArg(
        numOfWokers, m1, m2
      )
    );
    NinfletResult r = master.getResult();
    Matrix m3 = ((MatMultMasterResult)r).product
//...(abbreviated)
  }
}
```

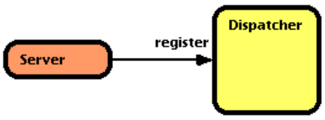
図1 Clientのプログラム例

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．このときどれかひとつの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2(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．

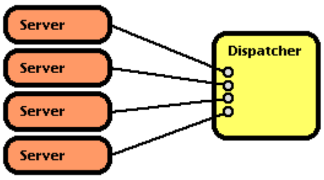
例えば，Clientプログラムは図1のように作成される．Codebasesクラスのインスタンスに，作成したNinfletの所在URLを設定している．具体的には，パッケージ`JP.ac.nitech.center.takagi.ninflet`以下のクラスは`http://www.center.nitech.ac.jp/~takagi/ninflet/`をcodebaseとするよう指定している．そして，newSessionメソッドで，指定したDispatcher上にSessionインスタンスを作成する際に，このcodebaseリストを渡している．

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2(d))．これにより，Dispatcherは，自分に登録されている利用可能なServerの中から適当なひとつの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2(e))．

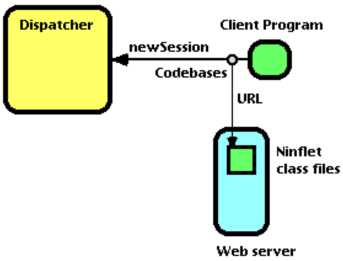
Serverは指定されたNinfletクラスのインスタンスを生成する際に，まず，そのインスタンス専用のクラスローダとしてNinfletClassLoaderのインスタンスを生成する．このとき，先ほど設定されたCodebases



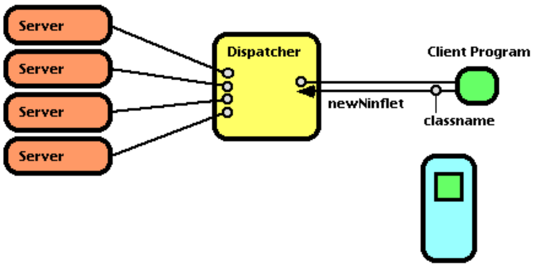
(a) Server は起動時にある Dispatcher に自分自身を登録する．



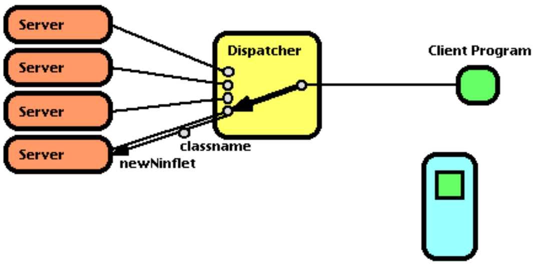
(b) 複数の Server が Dispatcher に登録される．



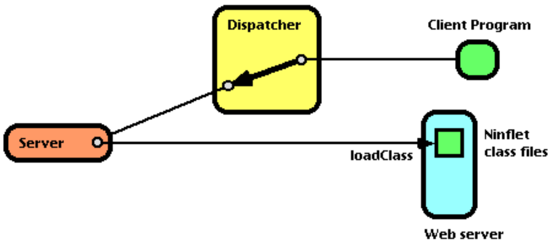
(c) Client は Ninflet の URL を指定して Session を作成する．



(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 の中からひとつを選ぶ．



(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 2　Ninflet の起動プロセス

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 2(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ひとつ作られ，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．

3.2　Server の Ninflet 受け入れ条件の設定

Ninflet Server では，Dispatcher からの Ninflet 発行要求の受け入れの許可/拒否の条件を設定することができる．設定の変更は，Server daemon 稼働中であっても，Web ブラウザから操作することができる．このために Ninflet 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 Web ブラウザから Ninflet Server をコントロールできる．

最も基本的な設定項目は，Ninflet 受け入れの許可/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 の受け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．

受け入れ設定が不許可に切替えられる際に，既に実行中の 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

タイプの 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していただけるようになっている．

3.3 Checkpointing と Ninflet Migration

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得ない．一方，特に科学技術計算を目的とした利用においては，ひとつの 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，立ち退き命令が 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 により指示されたところに対してなされる．退避された Ninflet は，Dispatcher により再割当される別の 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という仕組みは，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などにおいて，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10) 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なう．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，計算ルーチン中の既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毎に状態を退避する．

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，前回作成したの複製を退避する．

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 のオーバーヘッドが大きく，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 のオーバーヘッドが少なく，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 の外にもう一段かぶせられた Ninflet stub としており，この Ninflet stub 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 migrate 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．

3.4 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．

- RMI 以外によるネットワーク接続を禁止．
- 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．

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 作成など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 の操作も禁止している点である．Ninflet では計算だけが目的であり，プログラマによってサーバ計算機上に余計なウィンドウを出されることは許さない．

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 のホストにしかソケット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．

4. 並列処理への応用

4.1 Master-Worker パターンによる並列実行

Ninflet システムでは，さまざまな並列処理を支援するクラスライブラリを併せて提供していく予定であるが，現時点ではまず，Master-Worker (あるいは Master-Slave) 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．

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 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.`

```
::::::::::::::
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;
  float subSize;
  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 = (int)(subSize * (float)workerNum);
    int nextRow = (int)(subSize * (float)(workerNum + 1));
    int rowLength = nextRow - row;
    Matrix m1Sub = m1.getSubmatrix(row, rowLength, 0, m1.cols);
    return new MatMultWorkerArg(m1Sub, m2);
  }
  protected void combineResultOf(
    NinfletResult r, int workerNum
  ) {
    int row = (int)(subSize * (float)workerNum);
    int nextRow = (int)(subSize * (float)(workerNum + 1));
    int rowLength = nextRow - row;
    m3.putSubmatrix(
      row, rowLength, 0, m3.cols,
      ((MatMultWorkerResult)r).product
    );
  }
  protected NinfletResult createMasterResult() {
    return new MatMultMasterResult(m3);
  }
}
```

図 3　Master-Worker パターンを使った行列積プログラム

```
::::::::::::::
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 4　Master-Worker ライブラリの実装

`master_worker` 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，応用プログラムを作成する．図 3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したが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．

4.2 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なった．ワークステーションクラスタとして，電総研の 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成る) 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，データ転送オーバーヘッドが支配的になることが読みとられる．この結果から，粒度の小さい問題においては Ninflet のインスタンス化のコストがオーバーヘッドとなっており，この高速化の改良の必要があることが結論付けられる．

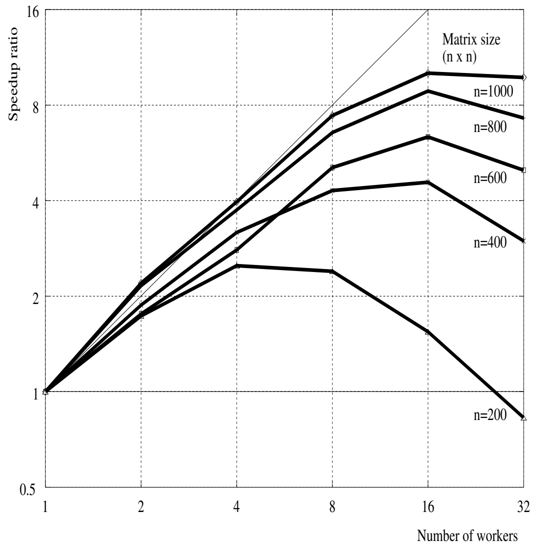


図 5　ETL-Wiz における MatMult の並列実行による速度向上率．

5. 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[11]，Javelin[12]，Charlotte[13] などがある．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 Web 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くものである．

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，別途 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 キューからひとつの Task を，また Host プールからひとつの 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，常に 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クラスタに仮想計算機を構成するために利用するということを考えると，ワークステーション毎に Web ブラウザを起動して特定のページを開かせておくのは繁雑である．Javelin のような Applet によるシステムは，RSA Factoring[14] や一部の scientific grand challenge 問題など，一時期に集中して大衆が興味を持って協力してくれるような問題には適しているかもしれないが，不特定多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できないため，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，将来 signed Applet を利用することによりセキュリティの制約をなくすと主張しているが，不特定多者が自由に利用するシステムを実現するためには signed Applet 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 standalone application 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[15] や POPCORN[16] などがある．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，オブジェクト指向設計には程遠いものとなっている．

一方，科学技術計算目的以外のシステムに目を向けると，ObjectSpace の Voyager[17] や IBM 東京基礎研の Aglets[18]，富士通研の Kafka[19] などの mobile agent 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り付け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，これらを主に科学技術計算をターゲットとして特化したものとみることもできる．

6. おわりに

不特定のユーザ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 システムについて述べた．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．この種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 の安全性が一般にも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，Ninflet を LAN 上にクラスタ並列計算機を構築するための応用として，Master-Worker パターンによる並列プログラミングを紹介した．またこのようなサーバの安定性が保証されない環境で重要となる fault tolerance のために，object migration と 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した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り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20)．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8)されているが，これに checkpointing や migration 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．
- LAN 上のワークステーションクラスタでの並列実行性能を定量的評価する．
- 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) M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture, *Lecture Note in Computer Science, High-Performance Computing and Networking*, pages 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, 並列処理シンポジウム JSPP'97 論文集, pp.281-288, 1997.

3) A. Takefusa, S. Matsuoka, H. Ogawa, H. Nakada, H. Takagi, M. Sato, S. Sekiguchi, and U. Nagashima. Multi-client LAN/WAN Performance Analysis of Ninf: a High-Performance Global Computing System, in *Proceedings of Supercomputing'97*, 1997.

4) H. Casanova, and J. Dongarra. Netsolve: A Network Server for Solving Computational Science Problems. in *Proceedings of Supercomputing'96*, 1996.

5) A. S. Grimshaw and W. A. Wulf. Legion: Flexible Support for Wide-Area Computing, in *Proceedings of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, 1996.

6) The RSA Data Security Secret-Key Challenge, `http://www.rsa.com/rsalabs/97challenge/`.

7) DISTRIBUTED.NET – Fastest Computer on Earth, `http://www.distributed.net/`.

8) Y. Ishikawa. Towards a Parallel C++ Programming Language Based on Commodity Object-Oriented Programming, in *1997 International Scientific Computing in Object-Oriented Parallel Environments Conference* (ISCOPE'97), 1997.

9) ACM 1997 Workshop on Java for Science and Engineering Computation, `http://www.cs.rochester.edu/u/wei/javaworkshop.html`.

10) M. Beck, J. S. Plank, and G. Kingsley. Compiler-Assisted Checkpointing, in *25th Annual International Symposium on Fault-Tolerant Computing*, 1995.

11) L. M. Silva, H. Pedroso, J. G. Silva. The Design of JET: A Java Library for Embarrassingly Parallel Applications, in *Proceedings WOTUG'20 – Parallel Programming and Java Conference*, IOS Press, pages 210–228, 1997.

12) B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu, Javelin: Internet-Based Parallel Computing Using Java, *ACM Workshop on Java for Science and Engineering Computation*, 1997.

13) A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the Web, in *Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems*, 1996.

14) RSA Factoring-By-Web project!, `http://www.npac.syr.edu/factoring.html`.

15) P. A. Gray, and V. S. Sunderam, The IceT Environment for Parallel and Distributed Computing, in *Proceedings of 1997 International Scientific Computing in Object-Oriented Parallel Environments Conference*, 1997.

16) The POPCORN Project, `http://www.cs.huji.ac.il/~popcorn/`

17) ObjectSpace Voyager – The Agent ORB for Java, `http://www.objectspace.com/voyager/`

18) Programming Mobile Agents in Java, `http://www.trl.ibm.co.jp/aglets/`.

19) Kafka: Yet Another Multi-Agent Library for Java, `http://www.fujitsu.co.jp/hypertext/free/kafka/`.

20) 竹房, 合田, 小川, 中田, 松岡, 佐藤, 関口, 長嶋. 広域計算システムのシミュレーションによる評価 — Ninf システムの広域分散環境でのジョブスケジューリング実現に向けて —, 並列処理シンポジウム JSPP'98.