ファイアウォールに対応したGlobusによる広域クラスタシステムの構築とその評価

田中良夫[†] 平野基孝[††] 佐藤三久[†††]
中田秀基[†] 関口智嗣[†]

Globusはグローバルコンピューティングのソフトウェアインフラストラクチャに必要とされる様々な要素技術を提供するツール群である．我々はファイアウォールを越えて計算資源を管理，利用するための機能をGlobusに組み込み，ファイアウォールを構築しているサイトにおけるGlobusの利用を可能とした．我々はファイアウォールに対応したGlobusを用いて広域クラスタシステムを構築し，いくつかのベンチマークを用いてその性能評価を行った．本稿では，Globusに新たに組み込んだ機能の設計および実装と，今回構築した広域クラスタシステムの性能に関して報告する．広域クラスタシステム上で木探索プログラムを実行した結果，通信量の抑制と負荷分散を効果的に行うことにより十分実用的な性能を得られることが分かった．

Performance Evaluation of a Firewall-compliant Globus-based Wide-area Cluster System

Yoshio Tanaka,[†] Motonori Hirano,[††] Mitsuhisa Sato,[†††]
Hidemoto Nakada[†] and Satoshi Sekiguchi[†]

In this paper, we present a performance evaluation of a wide-area cluster system based on a firewall-compliant Globus metacomputing toolkit. In order to utilize parallel systems and establish communication links beyond firewalls, we have built new mechanisms into the Globus system. We have built a firewall-compliant Globus based wide-area cluster system and run some benchmarks on it. In addition, we report on details of the firewall-compliant Globus system and performance evaluation of a wide-area cluster system. We have developed and run a tree-search problem using MPICH-G. The performance results indicate that the proposed system can achieve reasonable performance in the wide-area cluster system.

1. はじめに

近年，広い地域に配置された計算資源を用いて分散/並列計算を行うグローバルコンピューティングに関する研究がさかんに行われるようになってきた[1)]．グローバルコンピューティングは広域ネットワーク上に分散配置された計算資源を仮想的な高性能計算機（メタコンピュータ）とみたてて分散/並列計算を行う計算システムである．グローバルコンピューティングにおいては，ユーザ認証，通信，遠隔計算機上でのプロセス生成などの様々な要素技術が必要になる．Globus Metacomputing Toolkit（以下，Globus）[2)]は米国の大規模な研究チームによって開発されたそれらの要素技術を提供する低レベルなツールキットであり，Globusが提供するツールを用いて上位レベルにグローバルコンピューティングのソフトウェアを構築することができる．Globusはリリースされて間もないソフトウェアであるが現在非常に注目されており，グローバルコンピューティングのソフトウェアインフラストラクチャを構成する要素の事実上の標準になりつつある．

現在Grid Forum[3)]において世界中の研究者がグローバルコンピューティングの標準化について議論を進めている．Grid Forumにはセキュリティ，プログラミングモデル，情報サービス，スケジューリングなどの9つのワーキンググループが存在し，それぞれの分野における標準化に関して議論を行っているが，今後制定される標準案においてもGlobusの仕様，機能が大きな影響を与えることは間違いない．グローバルコ

† 電子技術総合研究所
Electrotechnical Laboratory
†† 株式会社SRA
Software Research Associates, Inc.
††† 新情報処理開発機構
Real World Computing Partnership

表1 Globusサービス
Table 1 Globus service.

Service	Name	Description
Resource Management	GRAM	リソースの割当ておよびプロセス生成
Communication	Nexus	Unicast/Multicast通信サービス
Information	MDS	システムの構造および状態に関する情報へのアクセス
Security	GSI	authenticationなどのセキュリティサービス
Helath and status	HBM	システムの状況サービス
Remote data access	GASS	データへのリモートアクセスサービス
Executable management	GEM	実行ファイルの構築，キャッシングおよび配置

ンピューティングでは国内のみならず世界中の計算資源の協調/相互利用が必要であり，現在Grid Forumで進めているような標準化は現存するシステムの互換性の提供などにおいて非常に重要な事項である．今後この標準化の動向をふまえながらグローバルコンピューティングの研究を行いつつ，標準化案に対して必要な機能を提案してゆくことが求められる．Globusを用いてシステムを構築し，その機能，利点，欠点や性能などを知り，その成果をGlobusに反映させることはその1つのアプローチとなる．

我々はGlobusを用いて構築したグローバルコンピューティング環境上で並列プログラムを実行し，その動作の仕組みや性能に関する知見を得た[4]．その際，クラスタシステムを計算資源として容易に利用する方法が提供されていないことと，ファイアウォールを構築しているサイトでは基本的にGlobusを利用することができないという2つの問題点が明らかになった．

我々はこれらの問題点を解決すべく，ファイアウォールを越えて複数のクラスタシステムや並列計算機の資源管理を行うリソースマネージャRMF（Resource Manager beyond Firewalls）を作成し，Globusに組み込んだ．また，計算プロセスどうしがファイアウォールを越えて通信することを可能とするため，Globusの通信レイヤであるNexus[5]のTCP通信を中継するNexus Proxyを作成した．RMFおよびNexus Proxyの機能により，Globusを用いたグローバルコンピューティング環境において，ファイアウォールを越えて計算資源を利用することが可能となる[6)~8)]．

本稿では，RMFおよびNexus Proxyの設計と実装を述べる．また，ファイアウォールに対応したGlobusを利用して広域クラスタシステムを構築し，その上で分枝限定法によるナップサック問題の並列解法を実行して得られた広域クラスタシステムの性能特性や経験，知見を報告する．次章ではGlobusの概要とその問題点について述べる．3章ではRMFの，4章ではNexus Proxyの概要および実装方法について述べる．5章では今回構築した広域クラスタシステムの性能について報告する．6章で関連研究を示し．最後にまとめを述べる．

2. Globusの概要と問題点

2.1 Globus Metacomputing Toolkit

Globusは米国の大規模なプロジェクトチームによって設計，実装されたツールキットであり，1998年10月にバージョン1.0.0がリリースされ，2000年4月の時点で世界中の136のサイトがGlobusを利用しており，日本でも6サイトがGlobusを導入，利用している．本節ではGlobusの概要と問題点について述べる．2000年7月の時点での最新バージョンは1.1.3であるが，本稿ではバージョン1.1.2を例に説明を行い，実験などもバージョン1.1.2を用いて行っている．

Globusはグローバルコンピューティングのためのサービスの集まり（ツールキット）を提供している．Globusが提供するサービスを**表1**に示す．Globusが提供するこれらのサービスは必要に応じて個別に利用することができるようになっており，既存アプリケーションへのインクリメンタルな導入が可能である．たとえばMPICHの広域環境での利用を可能とするMPICH-G[9]はGlobusを利用して通信やユーザ認証，ジョブの送信などの機能が実現されている．

Globusはこれらのサービスを利用するためのAPIとユーザコマンドを提供している．たとえば遠隔資源上でジョブを実行するためのコマンドとして`globusrun`が提供されている．ここで，`globusrun`コマンドを実行した場合にどのような流れで遠隔資源上でジョブが実行されるのかを例に，Globusの動作の仕組みを説明する．**図1**にローカルホスト（クライアント）からリモートホスト（gcitest.etl.go.jp）に送信されたジョブが実行される際の動作の仕組みを示す．

(1) `globusrun`コマンドは図1中に示されている形式で実行する．gcitest.etl.go.jp/jobmanager-forkというのはジョブを投げる相手（Globus Resrouce Allocation Manager, GRAM）の名前である．GRAMは計算資源（プロセッサ）

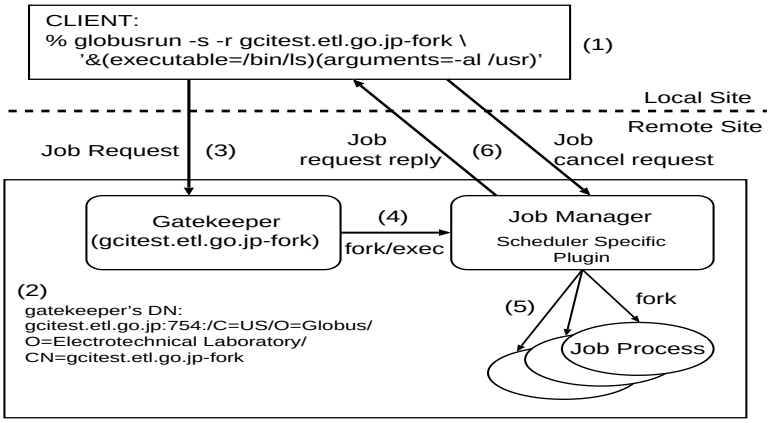


図 1 globusrun コマンドによるリモートサイトでのジョブ実行の様子
Fig. 1 Job execution mechanism using globusrun command.

管理のための構成要素を提供し，fork，LSF や Condor など，プロセスの生成・管理の方法に応じた型（manager type）を持つ．最後の引数は実行ファイルおよび引数を RSL（Resource Specification Language）と呼ばれる方法に従って記述したものである．

(2) `globusrun` コマンドが実行されると MDS のエントリを検索し，gcitest.etl.go.jp/jobmanager-fork という GRAM に対するサービス要求を受け付けるデーモン（gatekeeper）の DN（Directory Name）を獲得する．

(3) 獲得された DN を用いて gatekeeper にジョブの要求を送信すると，gatekeeper はパスワードの入力を求めてユーザ認証を行う．

(4) gatekeeper は実際にジョブを実行するためのプロセスを生成する **job manager** を起動する．

(5) job manager は各 GRAM に応じた方針でジョブを生成する．たとえば fork 型の GRAM の場合は *fork* 関数を使ってジョブプロセスを生成する．

(6) クライアントと job manager の間ではジョブ送信の成功/失敗の通知や，ジョブの取り消し要求などのやりとりが行われる．ジョブプロセスの出力等は GASS を介してクライアント側の出力に送られる．

gatekeeper は *execv* 関数によって job manager を起動する．fork 型の GRAM の場合は *fork* 関数を使ってジョブプロセスを生成するため，gatekeeper，job manager，およびジョブプロセスは同一計算機上でのみ動作する．gatekeeper や jobmanager が動作する計算機と異なる計算資源を計算サーバとして利用したい場合には LSF のようなジョブスケジューラを用いて資源管理を行う GRAM が必要である．

2.2 Globus の問題点

Globus を用いてシステムを構築した際，次の 2 つの問題点が明らかになった．

2.2.1 クラスタの利用方法の問題

Globus では複数の計算機を計算サーバとして利用する場合，それらすべてに Globus をインストールするか，あるいは LSF[10] などのような資源管理ソフトウェアを利用するリソースマネージャ（GRAM）を導入する必要がある．たとえば数十台，数百台規模のクラスタシステムを計算資源として利用する場合，それらの計算機すべてに Globus をインストールするのは管理上大きな負担となる．資源管理ソフトウェアを利用する場合，クラスタを 1 台の仮想的な並列計算機として扱うことができないことや，ライセンスの面において問題がある．また，今後グローバルコンピューティング環境におけるスケジューリングなどの研究を行う場合，製品として提供されているものを利用するのは難しい．

2.2.2 ファイアウォールの問題

ファイアウォールは最もよく利用されるセキュリティシステムの 1 つである．サイトの内部と外部（通常はインターネット）の間に配置され，内部と外部を区別するゲートウェイ計算機をファイアウォールと呼ぶ．ファイアウォールは通過する通信パケットを監視し，サイトのセキュリティポリシーに従って設定されているコンフィグレーションに応じて特定の通信パケットの通過を拒否することができる．具体的には通信パケットの送信元や送信先のホストとポートに応じて通過の許可，拒否を判断する．ファイアウォールのコン

フィグレーションは，基本的にすべての通信パケットの通過が許可される**許可ベース**とすべての通信パケットの通過が拒否される**拒否ベース**の 2 種類に大別できる．典型的なファイアウォールは，外部から内部への通信に対しては拒否ベース，内部から外部への通信に対しては許可ベースのコンフィグレーションを用いている．本稿においても，この典型的なコンフィグレーションを仮定する．

Globus が通信レイヤとして提供している Nexus ライブラリは通信の際に動的に TCP ポートを割り当てるため，拒否ベースのファイアウォールを構築しているサイトでは外部との通信リンクを張ることができず，グローバルコンピューティング環境を構築することができない．Globus に関する過去の報告において実験で利用されているテストベッドの場合，利用する計算資源どうしは直接通信可能となるようファイアウォールが設定されている．しかし任意の相手に対して計算サービスを提供するシステムなど不特定多数の計算資源の協調利用を考えた場合，ファイアウォールのコンフィグレーションの変更で対処するのは不特定な相手との任意の通信ポートを介しての通信を許可することとなり，実質ファイアウォールを利用しないのと同じになってしまう．より多くのサイトの計算資源の利用を考えた場合，ファイアウォールの内側の計算資源を利用できないということは大きな問題となる．また，LSF のようなスケジューラは通信の際に任意のポートを利用するものが多く，このような型の GRAM を利用する場合は計算サーバとして利用するすべての計算資源をファイアウォールの外側に配置する必要がある．

3. RMF の設計と実装

我々はサイト内（ファイアウォールの内側）にある複数のクラスタシステムや並列計算機をグローバルコンピューティングの計算資源として利用できる環境の構築に向けて新しい型の GRAM である RMF を作成した．本章では RMF の設計および実装について述べる．設計方針を以下に示す．

- 今後行う予定のグローバルコンピューティングにおけるスケジューリング（資源割当て）の研究の際に利用しやすいよう，柔軟性のある構造にする．
- ファイアウォールの外側で動作する job manager がファイアウォールの内側にある計算資源を管理，利用する仕組みを提供する．

RMF は Job Queueing System（Q システム）と計算資源の割当てを決定する Resource Allocator の 2 つのモジュールにより構成される．Q システムはサーバ（Q サーバ）とクライアント（Q クライアント）により構成される．Q システムはジョブの送受信，ディスパッチを行うシステムであり，Resource Allocator は資源割当てを決定するスケジューラである．**図 2** にこれらのモジュール間の関係と動作の仕組みを示す．gatekeeper に送られたジョブは次のような流れでファイアウォール内の計算機上で実行される．

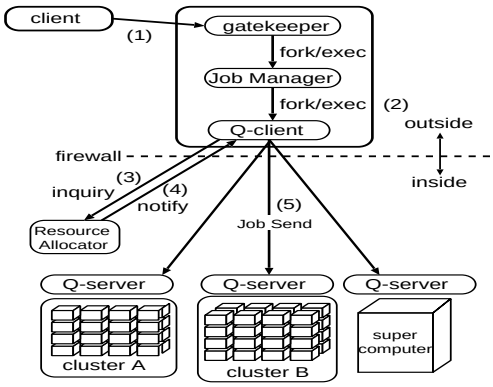


図 2 RMF の仕組みと動作の様子
Fig. 2 The architecture of RMF.

（0） RMF 型の gatekeeper をファイアウォールの外部の計算機上でデーモンプロセスとして起動しておく．Resource Allocator はファイアウォールの内部の計算機上で起動しておく．Q サーバはすべての計算サーバ上で起動しておく．

（1） クライアントから gatekeeper にジョブリクエストが送信される．

（2） gatekeeper により起動された job manager は Q クライアントを起動する．

（3） Q クライアントは Resource Allocator に対し，ジョブの送信先を問い合わせる．

（4） Resource Allocator は 1 つ以上の計算サーバを選択し，その名前を Q クライアントに通知する．

（5） Q クライアントは Resource Allocator に選ばれた Q サーバにジョブを送信する．

（6） Q サーバは Resource Allocator が選択した計算機上でジョブタイプに応じた方法でジョブプロセスを起動する．

RMF を利用するためには，Q クライアントと Resource Allocator および Q サーバの通信が許されるようにファイアウォールのコンフィグレーションを設定する必要がある．Q システムは job manager が異なる計算機上でジョブを実行する仕組みを提供しているほか，送信したジョブの監視，状態チェック，取消しなど

```
# Name      type procs nnodes clock   prefix
COMPaS        c     4      8   200    compas
COMPaS2       c     4      4   450 compas-2
SR2201        p     1    256   150    sr2201
```

図 3 計算資源コンフィグレーションファイルの例
Fig. 3 An sample configuration file for Resoruce Allocator.

のジョブ管理を行う機能を備えている．また，Globus は job manager が動作する計算機とジョブプロセスが動作する計算機がファイルシステムを共有していることを前提としており，入出力などはすべてファイルを介して行うようになっているため，Q システムは job manager 側のファイルに対する入出力とジョブプロセス側のファイルに対する入出力の仲介も行う．

Resource Allocator は起動時に管理対象とするクラスタシステムや並列計算機の情報をファイルから読み込み，資源割当て要求が来たらノード数や利用状況に応じて割り当てる資源を決定し，Q サーバに通知する．**図 3** に Resource Allocator が読み込むファイルの例を示す．

1 つの行が 1 つの計算資源のエントリを表す．“#” で始まる行はコメント行である．*Name* はクラスタや並列計算機の名前を表す．*type* はクラスタの場合は “c”，並列計算機の場合は “p” となる．*procs* は 1 ノードあたりのプロセッサ数，*nnodes* は全ノード数，*clock* はプロセッサのクロックスピードである．クラスタの各ノードは prefix に続いて 0 から始まる通し番号がつけられた名前がついているとする．たとえば図の例では COMPaS の場合は 8 台のノードには compas0 から compas7 までの名前が，COMPaS 2 には compas-20 から compas-23 までの名前がついていることになる．並列計算機の場合は prefix がそのまま計算機名となる．このように，RMF ではクラスタを 1 台の仮想的な並列計算機として扱うことができるうえ，クラスタの全ノードに Globus をインストールして gatekeeper を動かす必要がなく，RMF を Globus の GRAM として用いることによりクラスタを計算資源として容易に利用することができる．

Resource Allocator は試作段階であり，現在は各計算資源ごとに実行しているジョブの数と実行しているノードの情報を保持し，ジョブの割当ては指定されたプロセッサ数を保持している計算資源の中から，実行中のジョブが最も少ないものを求め，その中から CPU クロックが最も早いものを選択するという簡単な実装となっている．Q システムと Resource Allocator は独立している．Resource Allocator は適当な計算資源を選択し，その名前を Q システムと Resource Allocator の間のプロトコルに従って Q クライアントに通知すればよく，様々なポリシーで独自の Resource Allocator を実装，利用することが可能である．今後スケジューリングに関する研究を進めるうえで柔軟性の高い構造になっている．

4. Nexus Proxy の設計と実装

RMF はファイアウォール内にある計算資源を外部から利用する枠組みを提供しているが，このままでは各計算サーバで起動されたプログラムが計算過程においてファイアウォール外部の計算機と通信を行うような場合には対応できていない．たとえば MPICH-G で記述されたプログラムを複数のサイトに配置された計算機群で実行する場合，MPI の通信はこれらの計算機上で動作するプロセス間で動的に割り当てられるポートを用いて直接行われてしまうため，ファイアウォールがこれらのポートに対する通信を許さないような設定になっている場合は通信することができない．このように RMF はすべてのプロセスがファイアウォール内部の計算機群で動作するような場合にはファイアウォールに対応できているといえるが，複数のサイトに配置された計算機群を利用するような場合には RMF だけでは対応できない．

この問題を解決するためには SOCKS サーバのような Nexus の通信を中継する proxy をたててやり，TCP の通信はすべてこの proxy を介して行うように修正する必要がある．当初，ファイアウォール対応システムとして一般的に使われているプロキシメカニズムである SOCKS プロトコルおよびその実装システムである `socks` を用いることを検討したが，Globus が必要としている initial passive socket open（TCP，UDP の被接続側ポイントを最初に確立し，以降その被接続ポイントへの発呼側からの接続を待ち続ける手法）が SOCKS プロトコルおよび `socks` ではサポートされていない☆．つまり，SOCKS プロトコルの場合，最初にどこかにクライアント側から connection を張ったあと「その相手からだけ」クライアント側への接続を許すというモデルしか使えない．したがって，Globus で行われる「最初に被接続ポイントを確立して，任意の通信相手からの接続を待つ」ような仕掛けには対応できない．

そこで我々は initial passive socket open が可能であるプロキシメカニズムとして新たに NXProxy プ

☆ SOCKS の次期バージョンでは initial passive socket open が検討課題となっている．

表 2 Nexus Proxy のライブラリ関数
Table 2 Nexus Proxy library functions.

Function	Description
NXProxyConnect()	外部サーバに対して接続要求を出す．返り値は接続相手のプロセスと通信するためのファイルデスクリプタ．
NXProxyBind()	外部サーバに対してバインド要求を出す．返り値は接続要求を待ち受けるためのファイルデスクリプタ．
NXProxyAccept()	*NXProxyBind()* でバインドされたポート上で接続要求をアクセプトするための関数．
	接続が確立した場合，通信するための新しいファイルデスクリプタが返される．

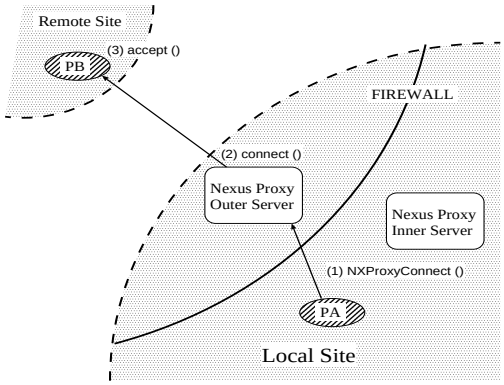


図 4 Nexus Proxy の通信メカニズム（active connection）
Fig. 4 Communication mechanism via the Nexus Proxy (active connection).

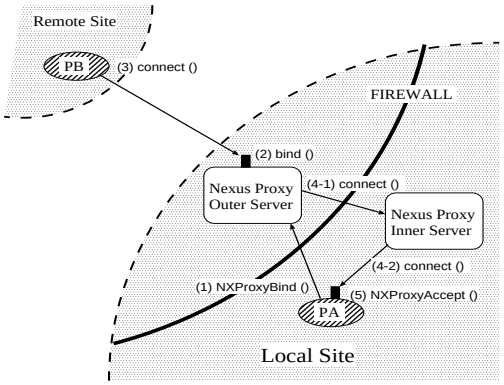


図 5 Nexus Proxy の通信メカニズム（passive connection）
Fig. 5 Communication mechanism via the Nexus Proxy (passive connection).

ロトコルを設計し，その実装システムとして Nexus Proxy を開発した．本節では NXProxy プロトコルおよび Nexus Proxy の設計と実装を述べる．Nexus Proxy はファイアウォール内外とのプロキシ通信を行う外部サーバと内部サーバの 2 つのプロキシサーバと NXProxy プロトコルを用いて通信を行うプログラムのためのクライアントライブラリで構成される．外部サーバはファイアウォールの外側で，内部サーバは内側でそれぞれデーモンプロセスとして稼働させる．ファイアウォールの内側で動作するクライアントプロセスが外部に対して接続をする場合や，外部からの接続要求を待ち受けるためのポートをバインドする場合，クライアントは *connect()* や *bind()* などの関数を呼んでポートへの接続やポートのバインドを直接行うのではなく，外部サーバに対してポートの接続/バインド要求を送る．要求を受け取った外部サーバはクライアントの代わりに相手のプロセスに対して接続要求を出したりポートをバインドしたりし，接続が確立されたら以後の通信を中継する．

Nexus Proxy の機能を利用するために提供されているライブラリ関数の一部を**表 2** に示す．**図 4** および **図 5** に Nexus Proxy を利用した場合の通信メカニズムを示す．図 4 と図 5 のいずれにおいても，*Process A* (*PA*) はファイアウォールを構築していないリモートサイトで動くプロセスである．内部サーバと外部サーバはいずれも *nxport* と呼ばれるポートをバインドし，そのうえでクライアントからの要求を待つ．外部から内部への接続に関しては，*nxport* を介した外部サーバから内部サーバへの接続が許可される必要がある．図 4 は *PA* が *PB* に対して接続を試みる場合の様子を示している．

(1) *PA* は *connect()* ではなく *NXProxyConnect()* を呼ぶ．すると接続要求が外部サーバに対して送信される．
(2) 外部サーバは *PA* からの接続要求を受信すると，*PA* と外部サーバとの間の通信路が確立される．外部サーバは，要求に従って *PB* に対して *connect()* を利用して接続を試みる．
(3) *PB* が外部サーバからの接続要求を受け付けると，*PA* と *PB* との間の通信路が外部サーバを介して確立される．これ以降外部サーバは *PA* と *PB* の間の通信を中継する．

図 5 は *PA* がポートをバインドして他のプロセスからの接続要求を待ち，*PB* が *PA* に対して接続要求を出した場合の様子を示している．

(1) *PA* は *bind()* はなく *NXProxyBind()* を呼ぶ．するとバインド要求が外部サーバに対して送信される．*NXProxyBind()* は *PA* が間接的にクライアントからの接続要求を受け付けるための

ファイルデスクリプタを返す．

(2) 外部サーバは *PA* からのバインド要求を受け取ると，ポートを 1 つバインドし，そのうえで接続要求を待つ．

(3) *PB* が *PA* に対して接続を試みる場合，*PB* は *PA* ではなく外部サーバに対して接続を行う．

(4) 外部サーバは *PB* からの接続要求を受け付けると，内部サーバに対して接続する．内部サーバは外部サーバからの接続を受け取ると，*PA* に対して接続する．

(5) *NXProxyBind()* が返したファイルデスクリプタ上で接続要求を受け付けるため，*PA* は *accept()* ではなく *NXProxyAccept()* を呼ぶ．*PA* が内部サーバからの接続要求を受け取ると *PA* と *PB* の間の通信路が内部サーバと外部サーバを介して確立する．以後内部サーバと外部サーバは *PA* と *PB* の間の通信を中継する．

Nexus Proxy が動作するための条件をファイアウォールのコンフィグレーションの観点でまとめると次の 2 点になる．

- 内部から外部への接続に関しては任意のポート（特権ポートは必要ない）での接続が許されること．
- 外部から内部への接続に関しては外部サーバから内部サーバへの *nxport* を介した通信が許されること．

本稿で仮定しているファイアウォールの場合，外部サーバから内部サーバへの通信が許されるようにファイアウォールのコンフィグレーションを変更する必要がある．Nexus Proxy を介してファイアウォールを越えた通信を行う場合，ファイアウォールのコンフィグレーションを変更してファイアウォールを越えた任意のサイト/ポート間での直接通信を許可する場合に比べ，開けるポートが 1 つだけであるうえに通信が必ずそのポートを介して行われるようになるため，セキュリティも向上する．

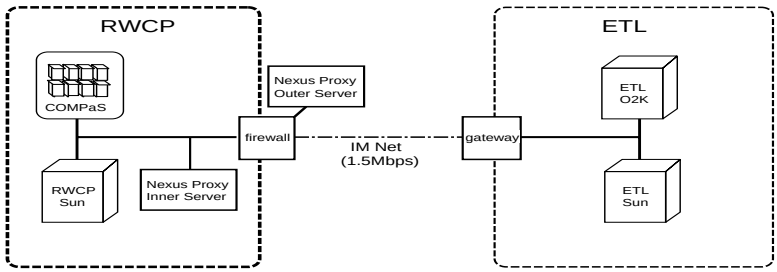
我々は Globus に Nexus Proxy を組み込んだ．具体的には，必要な場合（ファイアウォールを越えた通信を行うような場合）には表 2 に示すライブラリ関数を用いて Nexus Proxy を利用した通信を行うよう，Globus のソースコードを修正した．具体的には環境変数 *NEXUS_PROXY_OUTER_SERVER* や *NEXUS_PROXY_INNER_SERVER* が定義されている場合には Nexus Proxy を利用した通信を行い，そうでなければオリジナルの通信を行うように修正を行った．

5. 実　　験

我々はファイアウォールに対応した Globus を用いて広域クラスタシステムを構築し，いくつかのベンチマークプログラムを動かして性能を測定した．本章では行った実験およびその結果を示す．

5.1 実験環境

我々は新情報処理開発機構つくば研究センタ（RWCP）と電子技術総合研究所（ETL）にファイアウォールに対応した Globus をインストールし，2 つのサイトの計算機を利用して広域クラスタシステムを構築した．**図 6** に実験環境を示す．COMPaS[11] は 4 つの Pentium Pro（200 MHz）を搭載した SMP PC



site	nickname	system	OS
RWCP	RWCP-Sun	Sun Enterprise 450 (UltraSPARC 300 MHz, 4 CPU)	Solaris 2.6
RWCP	COMPaS	PC-based SMP cluster (Pentium Pro 200 MHz, 4 CPU × 8 nodes)	Solaris 2.5.1
ETL	ETL-Sun	Sun Enterprise 3000 (UltraSPARC 333 MHz, 6 CPU)	Solaris2.6
ETL	ETL-O2K	SGI Origin 2000 (MIPS 195 MHz, 16 CPU)	IRIX6.5
RWCP	内部サーバ	Sun Ultra Enterprise 450 (UltraSPARC 300 MHz, 2 CPU)	Solaris2.6
RWCP	外部サーバ	Sun Ultra 80 (UltraSPARC 450 MHz, 2 CPU)	Solaris2.7
RWCP	RMF-gatekeeper	Sun SS20 (SuperSPARC 60 MHz, 1 CPU)	Solaris2.6

図 6 実験環境
Fig. 6 Experimental environment.

表 3 通信遅延とバンド幅
Table 3 Communication latency and bandwidth.

	遅延	バンド幅（4 KB message）	バンド幅（1 MB message）
RWCP-Sun ↔ ETL-Sun (direct)	3.9 msec	112.0 KB/sec	174.4 KB/sec
RWCP-Sun ↔ ETL-Sun (indirect)	25.1 msec	109.5 KB/sec	176.1 KB/sec

を 8 台 100 Base-T Ethernet で接続した SMP クラスタである．RWCP は拒否ベースのファイアウォールを構築しており，インターネットから直接 COMPaS や RWCP-Sun にアクセスすることはできない．ETL もファイアウォールを構築しているが，ETL-Sun や ETL-O2K は RWCP から直接アクセスすることができるように設定されている．RWCP と ETL の間は 1.5 Mbps の IM Net によって接続されている．

5.2 Nexus Proxy の性能

Nexus Proxy を介して通信した場合にどの程度の性能が得られるのかを調べるため，RWCP-Sun と ETL-Sun の間で MPICH-G を用いて通信遅延およびバンド幅を測定した．利用したプログラムは MPI で書かれたピンポン転送プログラムであり，通信遅延は 0 バイトメッセージの片道分の転送時間（片道分の転送時間は往復にかかった時間を 2 で割って求めている），バンド幅は転送メッセージサイズを片道分の転送時間で割ることにより求めている．実験では Nexus Proxy を介した場合と，直接通信を行った場合の 2 通りの通信遅延およびバンド幅を測定した☆．実験は 3 回の計測を 1 時間ごとに 24 回，計 72 回行った．実験中計算機やネットワークには外乱が発生するが，実験誤差は最大で 10%程度であった．**表 3** に実験結果を示す．示されているデータは最善値である．

Nexus Proxy を介した場合，直接通信した場合と比較して通信遅延は約 6 倍になる．これは，外部サーバおよび内部サーバがメッセージを中継する際に *read/write* による TCP ストリームのコピー処理がユーザプロセスにより行われることによる．メッセージサイズが大きくなるとネットワークを介するデータ転送時間が増加し，外部/内部サーバのコピー処理によるオーバヘッドが相対的に無視できるものとなるため，Nexus Proxy を介した場合でも直接通信した場合と比べてほぼ同じバンド幅が得られる．Nexus Proxy を介した方が直接通信した場合より若干高いバンド幅が得られているのは実験誤差によるものである．このようにメッセージサイズがある程度大きい場合や，広域環境のように通信回線のバンド幅が細いような場合にはデータの転送にかかる時間が増加し，外部/内部サーバによる中継処理のオーバヘッドが隠蔽され，Nexus Proxy システムを利用することによるオーバヘッドが無視できるものとなる．

5.3 分枝限定法によるナップサック問題の実装

広域並列システムの特徴を考慮した場合，次のような性質を持つアプリケーションが広域クラスタシステム上でも高い効率を得られる可能性があると考えられる．

- 各プロセッサが非同期に計算を進められる．
- データの独立性が高い（プロセッサ間でのデータの交換が少ない）．
- 計算量が多く，高い並列性を持つ．

以上の特徴を持つアプリケーションの 1 つに探索問題がある．そこで，今回はナップサック問題を分枝限定法により並列に解くプログラムを MPICH-G を用いて実装した．ナップサック問題は整数計画問題であり，木探索を行う応用問題の典型例である．ここではナップサック問題の概要および実装方法について簡単に説明する．

ナップサック問題（0-1 ナップサック問題）は，それぞれに重さ（w_i）と価値（p_i）を持つ N 個の荷物が与えられたとき，収容力 C のナップサックに対して，w_i の合計が C を超えない範囲で，p_i の合計が最大となるような荷物の組合せを選び出す最適化問題である．N 個の荷物が与えられた場合には探査空間は 2^N と膨大な大きさになり，分枝限定法などの探索空間を狭める手法が用いられる．

分枝限定法によるナップサック問題の解法を実装した．これまでに算出された下界値の最大のもの（GLow）を用いて枝刈りを行う．基本的なデータ構造は探索木のノードを表す「現在注目している荷物のインデックス，今まで詰めた荷物の価値の合計，残りの容量」の三つ組みであり，探索はスタックから取り出した三つ組みに対して分枝操作を行う（開いたノードをスタックに積む）ことによって行う．分枝操作の際に上界値と下界値を求め，GLow を用いて枝刈りを行う．より良い GLow が見つかれば GLow を更新する．今回 MPICH を用いて並列解法を実装したが，このプログラムを並列化する際のポイントを以下に述べる．

☆ 実験のため，RWCP-Sun と ETL-Sun が直接接続できるよう RWCP のファイアウォールのコンフィグレーションを一時的に変更した．

表 4 実験に用いたテストベッド
Table 4 Experimental testbed.

名称	概要
COMPaS	COMPaS の **8 プロセッサ**．全部で 8 ノード，1 ノードにつき 1 プロセッサを利用． mpich **ch_p4 device** を利用
ETL-O2K	ETL-O2K の **8 プロセッサ**．SGI が提供している mpi を利用．
Local-area Cluster	**RWCP-Sun + COMPaS**．全部で **12 プロセッサ**． RWCP-Sun の 4 プロセッサ，COMPaS の 8 プロセッサを利用． mpich **Globus device** を利用．
Wide-area Cluster	**RWCP-Sun + COMPaS + ETL-O2K**．全部で **20 プロセッサ**． RWCP-Sun の 4 プロセッサ，COMPaS の 8 プロセッサ，ETL-O2K の 8 プロセッサを利用． mpich **Globus device** を利用．

- ナップサック問題の探索空間は木構造で表すことができるが，枝刈りにより探索空間が一様にならないことが予想されるため，動的にジョブを分散する．
- 動的に負荷分散を行うようにするが，負荷分散のために発生する通信回数をなるべく抑えるようにする．
- プロセッサ間で GLow を通知しあい，より良い GLow を枝刈りに用いるようにした方が効率良く枝刈りを行うことができる．しかし，GLow の通知は通信が発生するため，その頻度や方法を考慮する必要がある．
- できる限り通信と計算をオーバラップさせる．

上記 4 点をふまえて以下のような実装を行った．

- rank 0 のノードをマスタ，それ以外のノードをスレーブとする．
- マスタは何回か分枝処理を行い（この回数を *interval* と呼ぶ），スレーブからジョブ要求が来ていればスレーブにジョブを配る（このとき配るジョブの数を *stealunit* と呼ぶ）．
- スレーブはスタックが空になるまで分枝操作を行い，スタックが空になったらマスタからジョブをもらう．
- スレーブからのジョブ要求やマスタからのジョブ配送に GLow の更新処理を含めてしまう．
- 計算と通信をオーバラップさせるため，可能な場所では *MPI_Isend()* や *MPI_Irecv()* などの非同期送受信関数を用いた．

我々のアルゴリズムはセルフスケジューリングアルゴリズムである．セルフスケジューリングアルゴリズムは低オーバヘッドで動的な負荷分散を可能とするため広域クラスタシステムなどの非均質な分散環境に適している．基本的な戦略は通信をなるべく減らすことにあり，以下の 2 点が実装の特徴である．

- スレーブは自分のスタックが空になったときにマスタのジョブを盗みにゆく．負荷分散のためのオーバヘッドが少なくなり，負荷が均一になりやすい．
- GLow を更新するための通信はジョブのやりとりに含めてしまう．

5.4 ナップサック問題の実行性能

実装したナップサック問題の並列解法を**表 4** に示す 4 種類のローカル/広域クラスタシステム上で実行した．Wide-area Cluster 上では，Nexus Proxy を介して通信を行う場合と，ファイアウォールのコンフィグレーションを一時的に変更して直接通信を行う場合の 2 通りの実験を行った．これらの結果を比較することにより，Nexus Proxy のオーバヘッドが明らかになる．また，ナップサック問題の逐次解法プログラムを RWCP-Sun および COMPaS の 1 ノードで実行した．実験は様々な時間帯に数回行った．実験中は外乱が発生している．すべての計算機は実験中でも他のユーザがログインして利用しており，ネットワークもクリーンな状態ではない．実験誤差は最大で 10%程度におさまっており，実験結果としては最善値を示している．

ナップサック問題を分枝限定法により並列に解く場合，実行時間は入力データ（つまり枝刈りがどのように行われるか）にきわめて依存する．ここでは広域クラスタシステムの性能特性を明確に評価することが目的であるため，問題を正規化して枝刈りが発生せず探索空間全体を走査する必要があるようなデータを実験に用いた．荷物の数は 50 とし，*stealunit* や *interval* を様々な値に設定してプログラムを実行し，最適な組合せを選びだした．表 4 に示す 4 つのテストベッド上で実行した際の実行時間および速度向上率を**表 5** に示す．速度向上率は RWCP-Sun 上での逐次プログラムの実行時間を基に計算している．

COMPaS における逐次プログラムの実行性能は RWCP-Sun における実行性能の 1.14 倍であり，RWCP-Sun と COMPaS の処理能力はほぼ同等と見なすことができる．このことから COMPaS では十分

表 5 ナップサック問題の実行時間
Table 5 Execution time of the 0-1 knapsack problem.

システム	プロセッサ数	実行時間 (sec)	速度向上率
RWCP-Sun	1	26547	1
COMPaS	1	23211	1.14
COMPaS	8	3135	8.47
ETL-O2K	8	6849	3.88
Local-area Cluster	12	2936	9.04
Wide-area Cluster (use Nexus Proxy)	20	2074	12.80
Wide-area Cluster (direct communication)	20	2003	13.25

表 6 ジョブ要求の回数
Table 6 Number of steals.

System	Master	RWCP-Sun			COMPaS			ETL-O2K		
		Max	Min	Average	Max	Min	Average	Max	Min	Average
Local-area Cluster	160459	13869	15649	14981	17219	11385	14436			
Wide-area Cluster	217330	11603	8394	10563	13289	8007	11465	8508	2105	5693

表 7 走査したノード数（単位は億）
Table 7 Number of traversed nodes (in billions).

System	Master	RWCP-Sun			COMPaS			ETL-O2K		
		Max	Min	Average	Max	Min	Average	Max	Min	Average
Local-area Cluster	26.6	45.6	44.4	44.8	47.0	43.4	45.0			
Wide-area Cluster	14.7	34.3	31.7	32.7	34.9	31.0	32.5	20.3	17.4	18.5

高い速度向上率が得られていることが分かる．また，Local-area Cluster では 12 プロセッサで 9 倍と，約 75%の並列化効率が得られているが，COMPaS 上で動かした場合と比べてあまり実行時間が短縮されていない．これは，RWCP-Sun と COMPaS の間の通信にも Nexus Proxy を介した通信を行っているために，COMPaS 単体での実行に用いられる mpich ch_p4 device の通信性能に比べると通信遅延やバンド幅が低下していることが原因である．現在この問題を解決すべく，ファイアウォール内部の計算機どうしは Nexus Proxy を介さずに直接通信できるように Nexus Proxy に改良を加えているところである．

Wide-area Cluster での並列化効率は Nexus Proxy を介した通信を行った場合で約 64%，直接通信を行った場合で約 66%となるが，これは ETL-O2K 単体での性能が RWCP-Sun や COMPaS に比べると劣ることや，RWCP と ETL の間のネットワーク性能が LAN の性能に比べて低いことによる．しかし，これら 2 つの結果より，広域環境では Proxy を介した通信を行うことによるオーバヘッドは直接通信を行った場合に比べても無視できる程度のものであり，Globus のファイアウォール対応として提供している Nexus Proxy の有効性を確認することができる．

今回の実装では，通信の発生はスレーブからマスタへのジョブ要求と，マスタからスレーブへのジョブの送信の際にのみ発生する．それらの回数を見れば，通信の頻度が分かる．**表 6** にマスタが処理したジョブ要求の数と，クライアントがマスタにジョブ要求を出した回数の最大値，最小値，平均値をシステムごとに示す．ジョブ要求の回数から並列処理の粒度を知ることができる．もしジョブが粗粒度であればジョブ要求の回数は減り，通信量も減る．一方でジョブの粒度が小さくなるとジョブ要求の回数が増え，通信量も増えるが，より効率の良い負荷分散が期待できる．**表 7** にマスターおよび各クライアントが走査したノードの数を示す．ただしクライアントの場合は各システムごとの最大値，最小値，平均値を示す．表の数値の単位は億である．

表 6 より，Local-area Cluster および Wide-area Cluster において，RWCP-Sun と COMPaS 上のスレーブは約 0.2 秒に 1 度，ETL-O2K 上のスレーブは 0.36 秒に 1 度の割合でジョブ要求を出していることが分かる．粒度としては細粒度であり通信量は増加してしまうが，そのかわり表 7 に示すように非均質な環境でもその計算機の性能に応じて効果的な負荷分散が行われていることが分かる．効果的な負荷分散と，通信と計算のオーバラップによる通信時間の隠蔽により，Nexus Proxy を利用した広域クラスタシステムにおいても十分実用的な性能が得られることが分かる．

6. 関連研究

Globusを用いたグローバルコンピューティングに関する研究として，I-WAY [12]やGUSTO [13]などのテストベッドを用いた実験が行われている．I-WAYは北米の17のサイトに存在するスーパーコンピュータ，記憶装置や視覚化のための装置などをATMネットワークにより接続したテストベッドである．I-WAY上では大規模シミュレーションなどのアプリケーションを用いて広域高性能計算の有効性を示している．また，GUSTOは北米，ハワイ，スウェーデン，ドイツに存在する全17サイトに置かれた330台の計算機（3600台のプロセッサ）を専用のOC3および共用のインターネットにより接続したテストベッドである．GUSTO上ではSF-Expressという分散シミュレーションシステムなどを用いて性能を測定している．これらのテストベッドを用いた実験のほとんどが，遠隔地にある計算サーバで計算した結果を手元のシステムで視覚化するといった分散計算を行うものであり，本稿で報告しているような広域に分散配置された計算機全体を1台の仮想的な並列計算機と見立て，その上でSPMDスタイルのプログラムを実行して並列計算を行うといったものはあまりない．

MahinthakumarらはGlobusを用いたメタコンピューティング環境上におけるパラメータサーチ問題の実行性能について報告している[14]．実験環境は米国の2つのサイトに分散配置されたIBMのSP2とSP3およびSGIのOrigin 2000によって構成され，セルフスケジューリングアルゴリズムによって負荷の均衡をはかっている．しかしこれらの計算機は直接通信することが可能であり，ファイアウォールがグローバルコンピューティングシステムの性能に対してどのように影響するかを知ることはできない．また，上記いずれのテストベッドにおいても利用されている計算資源は並列計算機が中心であり，クラスタを利用したものに関する報告はあまりない．これはクラスタを容易に利用するための仕組みが提供されていないことによるが，現在クラスタシステムは高性能計算の分野における重要なプラットフォームとなっており，グローバルコンピューティングシステムにおいてもクラスタを容易に利用する仕組みの提供や，利用した場合の性能を知ることが必要とされる．

7. まとめ

我々はグローバルコンピューティング環境においてファイアウォールを越えてクラスタや並列計算機など複数の計算資源を利用することのできる仕組みを提供するため，新しい型のGRAMであるRMFおよびTCP通信を中継するNexus Proxyを設計，実装しGlobusに組み込んだ．これによりGlobusはファイアウォールへの対応が可能となり，ファイアウォール内に存在するクラスタシステムや並列計算機をGlobusを用いたグローバルコンピューティング環境の計算資源として利用することが可能となった．また，ファイアウォールに対応したGlobusを用いて広域クラスタシステムを構築し，その性能を調べた．今回構築した広域クラスタシステム上で並列木探索プログラムを動かした結果，効果的な負荷分散，通信量の抑制や通信と計算のオーバラップなどを意識してプログラミングすることにより，広域クラスタシステム上でも十分受け入れられる性能が得られることが分かった．今回Globusに組み込んだファイアウォール対応機能はGlobusチームと連絡をとって何らかの形での組込みを依頼する予定である．今後広域クラスタシステムにおけるスケジューリング，開発/実行環境，プログラミング技法，性能評価などに関する研究を進める予定である．

謝辞 本研究のサポートをしていただいた新情報処理開発機構島田潤一所長と電子技術総合研究所大蒔和仁情報アーキテクチャ部長に感謝いたします．

参考文献

1) Foster, I. and Kesselman, C.: *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers (1998).
2) Team, T.G.: The Globus Project. http://www.globus.org/.
3) Member, G.F.: The Grid Forum. http://www.gridforum.org/.
4) 田中良夫，佐藤三久，中田秀基，関口智嗣：globusを用いたグローバルコンピューティングの性能評価，システムソフトウェアとオペレーティングシステム研究会報告，Vol.99, No.32, pp.71–76, 情報処理学会 (1999).
5) Foster, I., Kesselman, C. and Tuecke, S.: The Nexus approach to integrating multithreading and communication, *Parallel and Distributed Computing*, Vol.37, pp.70–82 (1996).
6) Tanaka, Y., Hirano, M., Sato, M., Nakada, H. and Sekiguchi, S.: Resource Manager for Globus-based Wide-area Cluster Computing, *IEEE International Workshop on Cluster Computing*, pp.237–244 (1999).
7) 田中良夫，平野基孝，佐藤三久，中田秀基，関口智嗣：Globusを用いたグローバルコンピューティ

ング環境の構築とその評価，インターネットカンファレンス’99, pp.97–106 (1999).
8) Tanaka, Y., Hirano, M., Sato, M., Nakada, H. and Sekiguchi, S.: Performance Evaluation of a Firewall-compliant Globus-based Wide-area Cluster System, *IEEE Symposium on High Performance Distributed Computing*, pp.121–128 (2000).
9) Foster, I. and Karonis, N.T.: A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, *Supercomputing ’98* (1998).
10) Zhou, S.: LSF: Load sharing in large-scale heterogeneous distributed systems, *Workshop on Cluster Computing* (1992).
11) Tanaka, Y., Matsuda, M., Ando, M., Kubota, K. and Sato, M.: COMPaS: A Pentium Pro PC-based SMP Cluster and its Experience, *IPPS ’98 Workshop on Personal Computer Based Networks of Workstations*, Lecture Notes in Computer Science, Vol.1388, pp.486–497, Springer-Verlag (1998).
12) Foster, I., Geisler, J., Nickless, B., Smith, W. and Tuecke, S.: Software Infrastructure for the I-WAY high performance distributed computing experiment, *IEEE Symposium on High Performance Distributed Computing*, pp.562–572 (1996).
13) Foster, I. and Kesselman, C.: The Globus Project: A status report, *Heterogeneous Computing Workshop*, pp.4–18 (1998).
14) Mahinthakumar, G., Hoffman, F.M., Hargrove, W.W. and Karonis, N.T.: Multivariate Geographic Clustering in a Metacomputing Environment using Globus, *Supercomputing ’99* (1999).

(平成 12 年 5 月 8 日受付)
(平成 12 年 8 月 30 日採録)

田中 良夫（正会員）

昭和 40 年生．平成 7 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．平成 8 年技術研究組合新情報処理開発機構入所．平成 12 年通産省電子技術総合研究所入所．現在に至る．博士（工学）．クラスタを中心とした並列システム上での高性能計算，広域/分散計算，Lisp 処理系（主に並列ガーベッジコレクション）に関する研究に従事．IC ’99 論文賞．ACM 会員．

平野 基孝

昭和 40 年生．昭和 63 年図書館情報大学図書館情報学部卒業．同年シャープ（株）入社．平成 2 年（株）SRA 入社．主に DTP ソフトウェアの日本語化，GUI アプリケーションの開発，移植の業務に従事．平成 6 年より，新情報処理開発機構つくば研究センターにソフトウェア受託開発者として常駐．同センターにおいて，超並列計算機用低レベルソフトウェア開発環境，計算機性能評価ツール，コンパイラ，ネットワーク通信ライブラリ等の開発に従事．

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．

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．現在同所主任研究官．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ーコンピュータ 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．