Firewall に対応した Globus による
広域クラスタシステムの構築と性能評価

田 中 良 夫[†] 平 野 基 孝[†††] 佐 藤 三 久[††]
中 田 秀 基[†] 関 口 智 嗣[†]

我々はファイアウォールを利用しているサイトで Globus を利用することができるよう，Globus に新たな機能を組み込んだ．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，ファイアウォールに対応した Globus を用いて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な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. 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/並列計算を行なうグローバルコンピューティングに関する研究が盛んに行な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ツールキットである．Globus はグローバルコンピューティングのための資源管理機構，ユーザ認証システム，通信ライブラリなど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を構築することができる．例えば，通信やユーザ認証に Globus を用いて MPICH を実装した MPICH-G(MPICH Globus Device)[3)] などが存在する．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．

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システム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容易に利用する手段がないということと，ファイアウォールを構築しているサイトにおいては Globus を利用することができないという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，拒否ベースの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なうリソースマネージャ RMF(Resource manager beyond Firewalls) を作成し，Globus の GRAM として組み込んだ．また，計算プロセス同士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，TCP 通信を仲介する 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[5)～7)]．

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

G. 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[8)]．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．

本稿では，Nexus Proxy の設計と実装を述べる．RMF の詳細は文献[5)～7)] で述べている．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 Nexus Proxy の概要および実装方法について述べる．第 3章では実験による広域クラスタシステムの性能について報告し，最後にまとめを述べる．

2. Nexus Proxy の設計と実装

例えば 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なわれてしまうため，ファイアウォールがこれらのポートを通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 を介して行な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 プロトコル及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 プロトコル及び 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 で行なわれる「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なうのではなく，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，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 1に示す．図 1に Nexus Proxy を利用した場合の通信メカニズムを示す．*Process A (PA)* はファイアウォールを構築し

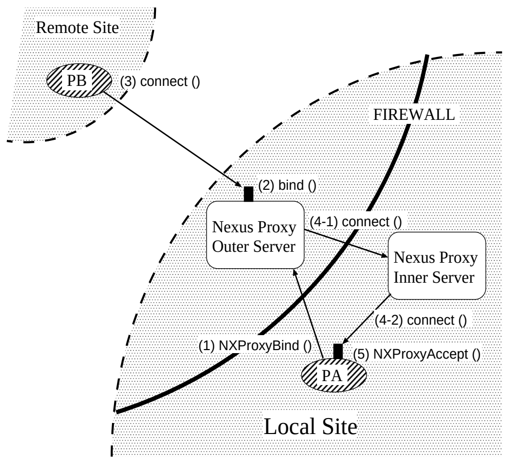


図 1　Nexus Proxy の通信メカニズム (passive connection)

ているローカルサイト内で，*Process B (PB)* 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，その上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 1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．

表 1　Nexus Proxy のライブラリ関数

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．

(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，その上で接続要求を待つ．
(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なう．
(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通信が許されること．

本論文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が，これは開けるポートが 1 つである上，通信相手も外部サーバに限定することができるため，セキュリティ上問題はない．

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なうような場合) には表 1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なうよう，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ない，そうでなければオリジナルの通信を行なうように修正を行なった．

3. 実験結果

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，その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．本節ではその結果を示す．

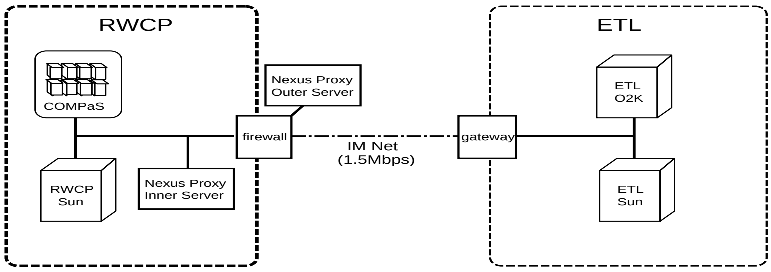
3.1 実験環境

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 2に実験環境を示す．COMPaS[9) は 4 つの Pentium Pro(200MHz) を搭載した SMP PC を 8 台 100Base-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 1.5Mbps の IM Net によって接続されている．

3.2 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 の間での MPICH-G を用いて通信遅延およびバンド幅を測定した．0 バイトメッセージのピンポン転送の片道分の時間を通信遅延とする．実験では Nexus Proxy を介した場合と，直接通信を行なった場合の 2 通りの通信遅延およびバンド幅を測定した☆．表 2に実験結果を示す．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なわれることによる．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した場合と比べて同じバンド幅が得られる．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

☆ 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．



site	nickname	system
RWCP	RWCP-Sun	Sun Enterprise 450 (4CPU)
RWCP	COMPaS	Pentium Pro SMP cluster (4CPU $\times$ 8nodes)
ETL	ETL-Sun	Sun Enterprise 3000 (6CPU)
ETL	ETL-O2K	SGI Origin 2000 (16CPU)
RWCP	外部サーバ	Sun Ultra Enterprise 450 (2CPU)
RWCP	内部サーバ	Sun Ultra 80 (2CPU)

図 2 実験環境

表 2 通信遅延とバンド幅

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

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ーヘッドが無視できるものとなる．

3.3 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並列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，木探索を行な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実装方法について簡単に説明する．

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なう．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なう (開いたノードをスタックに積む) ことによって行なう．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なう．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．

- 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なうようにするが，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なうことができる．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ーラップさせる．

上記 4 点を踏まえて以下のような実装を行なった．

- 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ない (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 (この時配るジョブの数を *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ない，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ーラップさせるため，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

表 3 実験に用いたテストベッド

Nickname	Description
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．

表 4 ナップサック問題の実行時間

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	8	3135	8.47
ETL-O2K	8	6849	3.88
Local-area Cluster	12	2936	9.04
Wide-area Cluster	20	2074	12.80

オーバー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ー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．

3.4 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 3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．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なわれるか) に極めて依存する．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，最適な組合わせを選びだした．表 3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 4に示す．ナップサック問題の逐次解法プログラムを RWCP-Sun で実行し，速度向上率はその実行時間を元に計算している．

COMPaS と RWCP-Sun の単体での性能はほぼ同じであることが分かっており，このことから COMPaS では十分高い速度向上率が得られている事が分かる．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 が LAN 内でいくつかのスイッチを介して接続されていることや，MPICH-G の通信性能が mpich ch_p4 device の通信性能に比べて低いことなどが原因と考えられる．一方で ETL-O2K 単体での性能が RWCP-Sun や COMPaS に比べると劣るため，Wide-area Cluster での並列化効率は約 64% となってしまう．しかし，いずれにおいてもネットワークや計算機の性能を考えれば妥当な効率が得られていると言える．

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 5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 6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 5より，Local-area Cluster および Wide-area Cluster において，RWCP-Sun と COMPaS は約 0.2 秒に 1 度，ETL-O2K 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わかる．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 6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

表 5　ジョブ要求の回数

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 6　走査したノード数 (単位は億)

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

散が行な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ーラップによる通信時間の隠蔽により，Nexus Proxy を介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．

4. まとめ

我々はクラスタや並列計算機など複数の計算資源をグローバルコンピューティング環境で利用することのできる仕組みを提供するため，RMF 型の GRAM を実装した．RMF では gatekeeper とは異なる計算機上で GRAM を動作させせることにより，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 の計算資源として利用する事が可能となる．RMF はジョブのキューイングなどを行なう Q システムと，計算資源の割り当てを決定する Resource Allocator によって構成される．また，ファイアウォールを越えて計算プロセスが行なう通信に対応するため，NexusProxy の設計，実装を行なった．今回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動かした結果，通信量の抑制や通信と計算のオーバーラップなどを意識してプログラミングすることにより，グローバルコンピューティング環境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 チームと連絡を取って何らかの形での組み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．

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) Foster, I. and Karonis, N. T.: A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, *Supercomputing '98* (1998).
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, 情報処理学会, pp. 71–76 (1999).
5) 田中良夫, 平野基孝, 佐藤三久, 中田秀基, 関口智嗣: Globus における Resource Manager の試作 – グローバルコンピューティング環境の構築に向けて–, ハイパフォーマンスコンピューティング研究会, Vol. 99, No. 66, 情報処理学会, pp. 191–196 (1999).
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) Mahinthakumar, G., Hoffman, F. M., Hargrove, W. W. and Karonis, N. T.: Multivariate Geographic Clustering in a Metacomputing Environment using Globus, *Supercomputing '99* (1999).
9) 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, Springer-Verlag, pp. 486–497 (1998).