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 –

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 firewall 内部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するため，Globus の Resource Manager(GRAM) として RMF(Resource Manager beyond the Firewall) を試作した．本実装は Globus の gatekeeper と GRAM を異なる計算機上で動作させることにより，firewall に対応可能となっている．本稿では今回構築した計算環境の概要と実装，およびグローバルコンピューティング環境の構築に向けての指針を述べる．

An Experimental Resource Manager for Globus
– Toward Building Global Computing Environment –

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

Globus provides a tookit for building software infrastructure for global computing environment. In order to utilize cluster systems and parallel systems inside the firewall for global computing, we developed an experimental Globus resource manager (GRAM) called RMF (Resource Manager beyond the Firewall). Since Globus gatekeeper and GRAM are deployed on individual systems, our system enables to utilize computing resources which are placed inside the firewall for global computing. In this paper, we describe and discuss the design and implementation of our global computing environment.

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/並列計算を行なう計算システムである．グローバルコンピューティングシステムにおいては，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[3),4)]はこのようなグローバルコンピューティングに必要とされる基本的なサービスを提供する．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システム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)[5)]などが存在する．Globus はグローバルコンピューティングシステムに必要とされる様々な基本的サービスを提供しており，グローバルコンピューティングシステム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．

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[2)]や GUSTO[3)]などのテストベッドを用いた実験が行なわれている．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，

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

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．

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[7]．その際，以下の 2 つの点が問題として明らかになった．

(1) 多数の計算資源の利用
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 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．例えば数 10 台，数 100 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．また，資源管理ソフトウェアも高価なものが多く，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なう場合，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) **firewall** の問題
Globus は通信レイヤとして Nexus[6] を用いているが，Nexus は通信の際に動的に TCP ポートを割り当てるため，firewall 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．報告されているテストベッドの場合，利用する計算資源はすべて firewall の外側に配置されていると考えられるが，今後より多くのサイトの計算資源の利用を考えた場合，firewall の内側に存在する計算資源を利用できないという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 firewall の外側に配置する必要がある．

我々はこれらの問題点を解決すべく，firewall の内側にある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 the Firewall) を作成し，Globus の GRAM として組み込んだ．本稿では，今回構築したグローバルコンピューティングシステムの設計と実装を述べ，今後の方針に関して議論を行なう．次章では Globus の概要を詳しく述べる．第 3 章では今回実装したシステムの概要および実装方法について述べる．第 4 章ではグローバルコンピューティング環境の構築に向けての議論を行ない，最後にまとめを述べる．

2. Globus の概要

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．Globus は階層的な構造を持ち，高レベルの Globus サービスはローカルサービスの上に構築され，ローカルサービスの種類に依存しない均一なインタフェースを提供している．これらのサービスのいくつかについて簡単に説明する．

[**Resource Management**]– **Globus Resouce Allocation Manager(GRAM)** は計算資源 (プロセッサ) 管理のための構成要素を提供する．GRAM は 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．計算サーバは 1 つ以上の GRAM を提供し，各 GRAM ごとにクライアントからの計算要求を受け付けるサーバプロセスである **gatekeeper** をデーモンとして稼働させる．例えば fork 型と LSF 型の 2 つの GRAM を提供する計算サーバの場合，fork gatekeeper と LSF gatekeeper の 2 つの gatekeeper がサーバプロセスとして動作することになる．fork gatekeeper にクライアントからジョブ要求が届くと *fork* 関数を使って gatekeeper が動作するホスト上でプロセスが生成されてジョブが実行され，LSF gatekeeper にクライアントからジョブ要求が届くと LSF の *bsub* コマンドを使ってジョブをキューに投入し，LSF のスケジューリングに応じていずれかのホスト上でジョブが実行される．

[**Communication**]– Globus ツールキットの通信サービスは **Nexus** 通信ライブラリによって提供される．Nexus は低レベルな通信 API を定義し，メッセージパッシングやリモートプロシージャコールなどの高レベルなプログラミングモデルをサポートするために用いられる．Nexus はメタコンピューティングのための通信として，様々な通信プロトコルや通信方法をサポートする．Nexus の通信は communication link と remote service request により構成される．通信方法は通信プロトコルだけではなく，セキュリティ，信頼性，質や圧縮などの情報も含んでいる．属性を特定の start point や end point に結び付けることにより，アプリケーションは各リンクごとの通信方法を制御することができる．

[**Information**]– 情報サービス部分は **Globus Metacomputing Directory Service(MDS)** が担当する．MDS はアーキテクチャタイプ，OS，メモリ，ネットワークバンド幅およびレイテンシ，利用可

表 1　Globus サービス

Service	Name	Description
Resource Management	GRAM	リソースの割り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

能な通信プロトコル，IP アドレスとネットワークテクノロジとのマッピングなどの情報を保持している．MDS は計算資源の構造や状態に関する情報を発見，公開したりそれらの情報にアクセスするためのツールや API を提供する．標準的には LDAP(Lightweight Directory Access Protocol) という情報データベース (ディレクトリ) にアクセスするための標準プロトコルによって定義されるデータ表現やアプリケーションプログラミングインタフェースを用いる．

Globus はこれらのツールを利用するための API とユーザコマンドを提供している．例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-fork というのはジョブを投げる相手 (GRAM) の名前であり，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-fork という GRAM に対するサービス要求を受け付ける gatekeeper の DN(Directory Name) を獲得する．獲得される DN は図に示されているようなものであり，ホスト名やポート番号などが含まれている．

(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なう．

(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 GRAM ごとに設定された configuration file を読み込んで，各 GRAM に応じた方針でジョブを生成する．例えば 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し，LSF 型の GRAM の場合は LSF の *bsub* コマンドを呼び出す．

(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なわれる．job manager は生成したプロセスの実行が終了するまで，クライアントとジョブプロセスとの間の仲介を行なう．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なう GRAM が必要である．

また，MPICH-G はこれらの Globus のサービスを利用して実装されている．`mpirun` は内部で `globusrun` を呼んでジョブの送信および実行を行なう．それらの上でプロセスを生成したり互いにリンクを張る作業や，立ち上げ時のエラー検出などの処理は GRAM や co-allocator ライブラリ (Dynamically-Updated Request Online Co-allocator, DUROC) が行なう．なお，現時点ではすべての計算機上の同じ場所 (ディレクトリ) に executable があらかじめ置かれている必要がある．

3. RMF の設計と実装

3.1 基本モジュール

Globus を用いた場合，前述のように計算サーバとして利用する計算機すべてに Globus の GRAM をインストールするか，あるいは LSF 型の GRAM のように複数の計算資源を管理するスケジューラの機能を利用する GRAM を提供する必要がある．また，例えば LSF は通信に任意のポートを利用するため firewall を挟んでクラスタを構成することができず，LSF 型の GRAM を用いる場合は計算サーバとして利用する計算機はすべて firewall の外側に出す必要がある．我々はサイト内 (firewall 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 を試作した．RMF の設

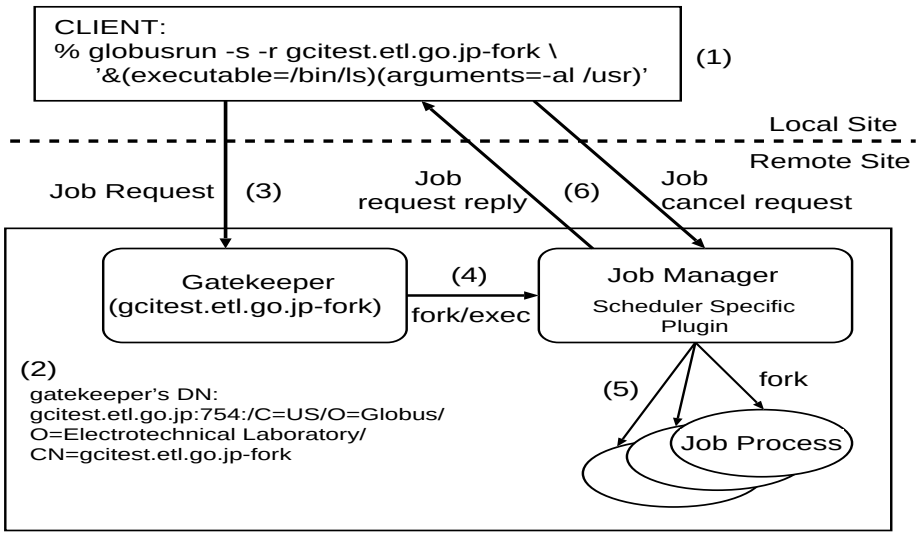

図 1　globusrun コマンドによるリモートサイトでのジョブ実行の様子

計方針を以下に示す．

- 今後行な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 (資源割り当て) 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．
- firewall の外側で gatekeeper(job manager) を動作させ，firewall の内側にある計算資源を利用する仕組みを提供する．

RMF は Job Queueing System(Tiny Remote Job Queueing System for RMF, Q システム) と計算資源の割り当てを決定する 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 サーバは firewall 内の計算機上で稼働し，クライアントからの要求を受け付ける．Q クライアントは job manager によって firewall の外側に配置された計算機上で起動され，Q サーバに対してジョブを送信する．Q クライアントと Q サーバが通信に利用するポートだけ開いている必要がある．Q サーバは受け取ったジョブをキューに格納し，今後のジョブの状態問い合わせや取り消し要求に備える．また，Resource Allocator にどの計算機上でジョブを実行させれば良いかを問い合わせ，指定された計算機上でジョブを実行する．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 firewall 内の計算機上で実行される．

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

(2) Q クライアントは Q サーバにジョブを送信する．クライアントからサーバに送信する情報は，ユーザ情報，ジョブタイプ (Globus が利用するジョブタイプと同じ物)，ノード数，実行ファイルのパスおよび引数，(shell の) 環境である．

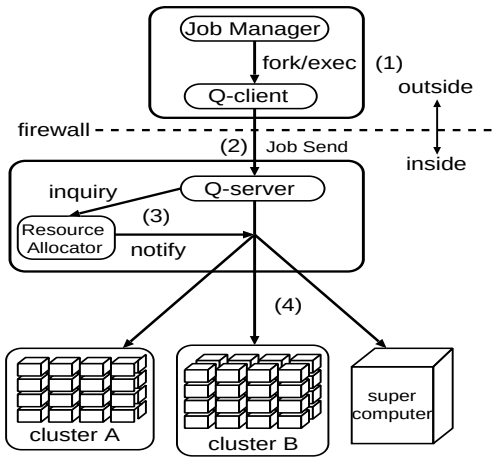

図 2　RMF の仕組みと動作の様子

(3) Q サーバは Resource Allocator にジョブを送信する計算機を問い合わせる．その際，必要とするノード数を Resource Allocator に通知する．

(4) 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．ジョブタイプが MPI の場合，MPICH-G で記述されたプログラムを実行することを意味する．この場合は Resource Allocator によって指定された各計算機上で直接プログラムを実行する．ジョブタイプが MPI でない場合は mpirun コマンドを使ってプログラムを実行するように解釈する．この場合は Resource Allocator によって指定された各計算機を machine ファイルに記述して mpirun コマンドの引数として与え，プロセスを起動する．指

定された計算機上でのプロセスの起動はいずれの場合も rsh コマンドを用いて行なう．

Q システムは job manager が異なる計算機上のリソースマネージャを呼び出す仕組みを提供している他，送信したジョブの監視，状態チェック，取り消しなどのジョブ管理を行なう仕組みを備えている．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な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なう．

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，資源割り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．'#' で始まる行はコメント行である．Name

```
# Name      type procs nnodes clock  epithet
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

はクラスタや並列計算機の名前を表わす．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 epithet 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．例えば図の例では 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 epithet がそのまま計算機名となる．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，ジョブの割り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．

3.2 firewall への対応

RMF は firewall 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 firewall 外部の計算機と通信を行なうような場合には対応できていない．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，firewall 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．このように RMF はすべてのプロセスが firewall 内部の計算機群で動作するような場合には firewall に対応出来ていると言えるが，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 の通信を仲介する proxy をたててやり，TCP/UDP 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．具体的には，firewall gateway 上に proxy サーバを立て，firewall 内のプロセスと firewall 外のプロセスの通信はこの proxy サーバを経由して行なうようにする．このためには Globus のソースプログラムを修正する必要がある．プログラム中の *bind()* や *connect()* システムコールなどを proxy サーバに対してコネクションを張り，以後自分に対する通信の仲介を依頼するような機能に置き換えれば良い．現在この部分に関しては開発中であるが，Globus のソースプログラムで *bind()* や *connect()* が呼び出されている部分は数 ヵ所であり，修正量はそれほど大きくならないと思われる．

4. 考　　察

今回試作した RMF は firewall 内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ための枠組みを提供している．今後この環境をより充実したものとするために，現時点で明らかになっている問題点および今後の方針について議論する．

4.1 スケジューリング (資源割り当て)

現在の Resource Allocator は実装中であり，現在はきわめて単純な方法でジョブを実行する計算機を選択している．今後 Resource Allocator により高度な判断技術を導入してゆく予定であるが，その際の問題点および方針を以下に示す．

- 各計算資源の負荷情報の管理
  現在は実行している各計算資源上で実行しているジョブの数だけを管理しているが，ロードアベレージのような数値によってクラスタシステムや並列計算機などの各計算資源の負荷情報を管理する必要がある．Globus の gram reporter のように，定期的に各計算資源が負荷情報を Resource Allocator に通知する方法が考えられる．
- ユーザによる資源の指定
  例えば特定のアーキテクチャを持つ計算資源上でジョブを実行したいなど，ユーザが計算資源を指定したい場合に備えてユーザがジョブを実行する計算資源を指定する方法を提供する．これは Globus と同様に RSL により指定する方法が考えられる．

Resource Allocator はサイト内のスケジューリングを行なうモジュールであるが，複数のサイトにある計

算資源のなかからジョブの実行に最適な計算資源を自動的に選択するようなサイト間のスケジューリング機能も望まれる．Ninf のメタサーバアーキテクチャ[8]はこのような仕組みを提供しているが，Globus が提供している MDS サービスを利用すれば同じような機能を提供することが可能であると考えられる．

4.2 実行ファイルの自動生成

現在はジョブが実行される計算資源上にあらかじめ実行ファイルが存在していなければならない．従って，現状では前もってリモートシステムにログインして実行ファイルを作成したり，あるいは FTP などによってファイルを転送しておく必要がある．リモートサイトに実行ファイルを自動生成する仕組みを提供することが望まれる．実行ファイルの自動生成は OS やアーキテクチャの違いを吸収する必要があり簡単な仕事ではないが，グローバルコンピューティング環境の整備の上で非常に重要であると考えられる．これができれば，ユーザは並列プログラムを記述すればあとはシステムが自動的に最適な計算資源やノード数を決定して実行ファイルを自動生成し，遠隔資源上で実行するという理想的なグローバルコンピューティング環境の提供に向けて大きく前進できる．

4.3 グローバルコンピューティング環境でのプログラミング

グローバルコンピューティング環境におけるプログラミングを考えた場合，通常のクラスタシステムや並列計算機で動作するプログラムがそのままグローバルコンピューティング環境で動作することが望ましい．例えば，MPI で書かれたプログラムなどがそのまま効率良く動けば良い．MPICH-G は MPI で書かれたプログラムをそのままグローバルコンピューティング環境で動作させる仕組みを提供しているが，性能面では問題がある．MagPIe[11]のように collective 通信をグローバルコンピューティング環境に応じて最適化するなどの工夫を行なう必要がある．現在広域環境に配置された複数のクラスタを 1 台の仮想的なクラスタシステムとみなして並列計算を行なう Wide-Area Cluster System に関する研究が盛んに行なわれるようになり[9),10)]，我々も高性能計算のプラットフォームとして注目し，研究を進める予定である．

5. ま と め

我々は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 を動作させることにより，firewall 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．firewall を越えて計算プロセスが行なう通信に対応するため，現在 SOCKS サーバと同じような仕組みを導入すべく，Globus のソースを修正中である．今後 firewall 問題の完全解決および Resource Allocator の高度化などを行い，Wide-Area Cluster System における性能評価，性能解析，プログラミング技法などに関する研究を進める予定である．

参 考 文 献

1) I. Foster et.al., “The GRID: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann Publishers (1998).
2) I. Foster et.al., “Software Infrastructure for the I-WAY high performance distributed computing experiment”, Proc. 5th IEEE Symp. on High Performance Distributed Computing, pp. 562–572 (1996).
3) I. Foster et.al., “The Globus Project: A status report”, Proc. Heterogeneous Computing Workshop, pp. 4–18 (1998).
4) I. Foster et.al., “Globus: A metacomputing infrastructure toolkit”, International Journal of Supercomputer Applications, Vol. 11, No. 2, pp. 115–128 (1997).
5) I. Foster et.al., “A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems”, Proc. Supercomputing (1998).
6) I. Foster et.al., “The Nexus approach to integrating multithreading and communication”, Journal of Parallel and Distributed Computing, Vol. 37, pp. 70–82 (1996).
7) 田中良夫 他, “globus を用いたグローバルコンピューティングの性能評価”, 情報処理学会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 (1999).
8) 中田秀基 他, “グローバルコンピューティングのためのスケジューリングフレームワーク”, JSPP’99, pp. 277–284 (1999).
9) http://www.cs.vu.nl/albatross/.
10) H. E. Bal et.al., “Parallel Computing on Wide-Area Clusters: the Albatross Project”, Proc. Extreme Linux Workshop, pp. 20–24 (1999).
11) T. Kielmann et.al., “MagPIe: MPI’s Collective Communication Operations for Clustered Wide Area Systems”, Proc. Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 131–140 (1999).