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

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denyベースのfirewallを越えて計算資源を利用する機能をGlobusに組み込み，グローバルコンピューティング環境を構築した．今回追加した機能は，新しい型のGlobus Resource Allocation Manager(GRAM)であるRMF(Resource Manager beyond the Firewall)およびNexusのTCP通信を中継するNexusProxyにより実装される．また，globusを用いたMPICHの実装であるMPICH-Gを用いて探索問題の並列解法を実装し，グローバルコンピューティング環境でその実行性能を測定した．本稿では今回構築したグローバルコンピューティング環境の概要と実装，その上で並列プログラムを実行した際の性能，プログラミングの留意点，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．グローバルコンピューティング環境で並列計算を行なう場合，通信量の抑制と負荷分散が効率に大きな影響を与える．これらの点に留意してプログラミングを行なうことにより，十分実用的な性能が得られることがわかった．

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，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[12]．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

[†] 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

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，deny ベースの 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，deny ベースの firewall 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 the Firewall) 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 firewall を越えて通信することを可能とするため，Nexus の TCP 通信を仲介する NexusProxy を作成した．RMF および NexusProxy 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，firewall を越えて計算資源を利用することが可能となる[13]．

一方，グローバルコンピューティング環境に適したアプリケーションの探索も必要である．グローバルコンピューティングは大きく分けて以下の 2 つに分類することができる：

- 広域高性能計算システム
  広域ネットワーク上に配置された複数の計算資源を利用して分散/並列計算を行なうシステム．1 台の計算機で解くには大きすぎるような問題を，複数の高性能計算機を用いて分散/並列計算させることにより解く．例えば利用する計算機のアーキテクチャに応じて問題部分を切り分けることにより，効率の良い分散計算を行なうことが可能になる．
- **Datorr(Desktop Access to Remote Resource)**
  手元にある計算資源上でユーザインタフェースを提供し，遠隔の計算機を利用するシステム．手元にある PC や WS からネットワークで結合された高性能計算機を “シームレス” に利用することを目的とする．Desktop Supercomputing とも呼ばれる．

広域高性能計算に関する研究は，前述の GUSTO などのテストベッドを用いて行なわれている．Datorr システムとしては，Ninf[10]や NetSolve[11]などが挙げられる．Ninf は LAN あるいは広域ネットワーク上の数値計算ライブラリや科学技術計算に必要な数値情報データベースを通じて，主に科学技術計算分野の情報ならびに計算資源を提供・共有する仕組みを提供する．Ninf はグローバルコンピューティングのためのミドルウェアであり，RPC をベースに簡便なインタフェースを提供することにより容易に遠隔の計算資源を利用できるようにしている．このようなシステムは 1 つのまとまった「計算」を遠隔の計算資源に実行させ，その結果を得るようなシステムであり，発生する通信も「計算の依頼」と「結果の返却」であり，通信と計算をオーバーラップさせることなどにより広域ネットワーク上でも十分実用的な性能が得られることが報告されている．

本研究では広域高性能計算システム上での並列計算に注目する．広域ネットワーク上に分散配置された複数の計算機を仮想的なクラスタ (広域並列システム) とみなして並列計算を行なう．このような広域並列システム上で並列計算を行なう場合，問題になると考えられる点がいくつかある．特に以下の 2 点が重要であると考える：

- 通信性能
  Datorr と比較して計算中に通信が頻繁に発生し，通信性能が全体の性能に大きな影響を与えることが予想される．このようなシステムにおいては通信量や通信回数の抑制や計算と通信のオーバーラップなどの工夫が重要になる．
- 負荷分散
  広域並列システムは不均質システムである可能性が高く，また，動的に計算機やネットワークの性能 (負荷) が変動する可能性もある．従って，通常のローカルなクラスタシステムと比較して動的な負荷分散がより重要になると考えられる．しかし，通信性能が高くない場合には負荷分散処理によるオーバーヘッドを十分考慮する必要がある．

分散処理を行なうことにより，広域高性能計算システム上においても十分高い効率を得ることのできるアプリケーションがいくつか報告されている．広域並列システムの性質を解析・理解することにより，広域並列システムに適したアプリケーションの性質やプログラミングの際の留意点を知ることができる．また，現在のネットワーク性能では広域並列システムが高い性能を示すことは難しいかも知れないが，今後ネットワーク性能は高いオーダーで向上することが予想され，近い将来広域並列システムが実用的になる事は間違いないと思われる．例えば大学などには複数のキャンパスがあることが多く，それらのキャンパスにある計算機をキャンパスを越えて 1 台の仮想的なクラスタシステ

ムとみなして並列計算させることが可能となる．このような点においても，広域並列システムの性質やその上でアプリケーションを動作させた場合の挙動に関して知見を得ることは重要である．

本稿では，firewall に対処するために今回 Globus に組み込んだ機能である RMF および NexusProxy の設計と実装を述べる．また，分枝限定法によるナップサック問題の並列解法をグローバルコンピューティング環境上で実行し，広域並列システムの性能特性を調査した結果および得られた経験，知見を報告する．次章では Globus および Globus を用いて実装された MPICH である MPICH-G を紹介する．第 3 章では RMF および NexusProxy の概要および実装方法について述べる．第 4 章ではナップサック問題および今回の実装方法について簡単に説明する．第 5 章では実験による広域並列システムの性能について報告し，第 6 章では今回得られた知見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議論を行ない，最後にまとめを述べる．

2. Globus と MPICH-G

2.1 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．LDAP は direcotry information tree と呼ばれる階層的な木構造の名前空間を定義する．MDS をサポートするために必要とされるローカルなサービスは LDAP サーバ (および他の LDAP サーバへのゲートウェイ) と，そのサイトの中の資源の状態に関する情報を更新しながらこのサーバを置くためのユーティリティだけである．Globus の MDS サービスは単にこれらのサーバの集まりである．

[Security]– セキュリティといっても authentication，authorization，privacy などの様々な要素があるが，Globus は authentication のためのツールとして **Globus Security Infrastructure(GSI)** を提供している．Globus ではユーザは計算に際し 1 度だけ authenticate を行ない，その時にプロセスがユーザに代わって資源を獲得できるように証明証が発行される．各サイトの様々な認証システムに対応できるよう，Globus ID とローカルユーザ ID との対応付けを行なうようになっている．

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

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

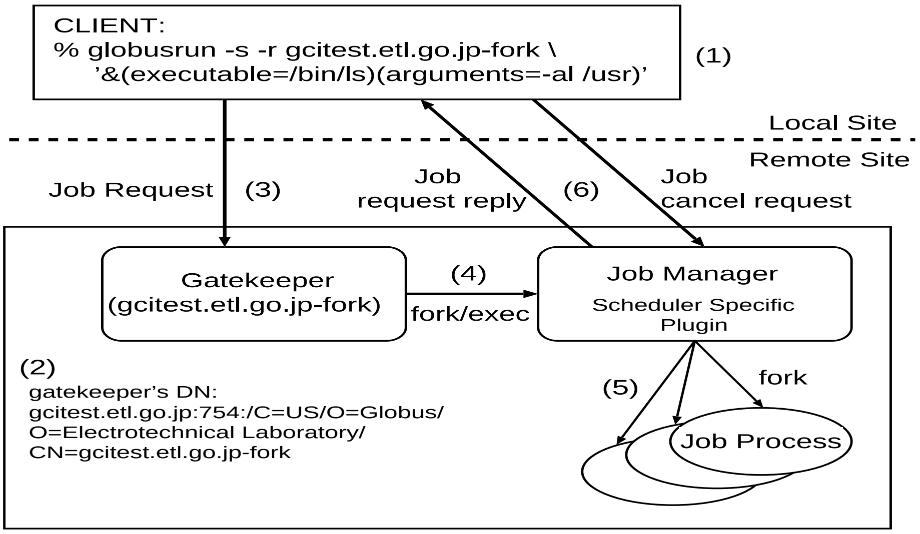

図 1　globusrun コマンドによるリモートサイトでのジョブ実行の様子

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．

2.2 MPICH-G

MPICH-G は globus device を用いて実装された MPICH である．MPICH-G は以下のように Globus

のサービスを利用している (一部は未実装) ．

- 利用したい計算機の獲得方法を MDS を介して決定する．
- authentication などに GSI を利用する．
- executable を配置するのに Globus executable management を利用する．
- 各計算機上でプロセスを生成するのに GRAM を使う．
- 通信には Nexus を使う．
- ファイルアクセスには GASS を使う．
- アプリケーションの監視，停止などに Globus のプロセス監視機構を使う．

実行時には通常の `mpirun` コマンドを使ってプログラムを起動する．machine ファイルには図 2のように resource manager と生成するプロセス数を記述する． MPICH-G は MDS を介して指定された resource manager にアクセスするためのポート番号を取得するなどの処理を行なう．

```
donner.mcs.anl.gov-fork 8
bonny.isi.edu-fork 8
moti4.ncsa.uiuc.edu-lsf 64
```

図 2　machine ファイルの例

`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 と NexusProxy の設計と実装

3.1 RMF の基本モジュール

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 3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．

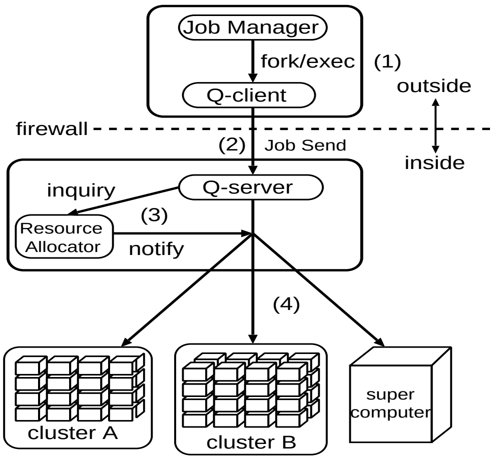

図 3　RMF の仕組みと動作の様子

(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．
(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 4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．

```
# 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 4　計算資源コンフィグレーションファイルの例

3.2 NexusProxy

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，firewall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NexusProxyを開発した．NexusProxyはfirewall内外とのプロキシ通信を行うプロキシサーバと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．図 5にNexusProxyの動作概要を示す．このクライアントライブラリをGlobusの通信層であるNexusで使用する事で、Globusの並列プログラミング機構であるMPICH-GがNexusProxyを使用できるようになる．具体的にはプログラム中の*bind()*や*connect()*システムコールなどをNexusProxy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を提供するライブラリ関数(*NxProxyBind()*や*NxProxyConnect()*)に置き換えれば良い．

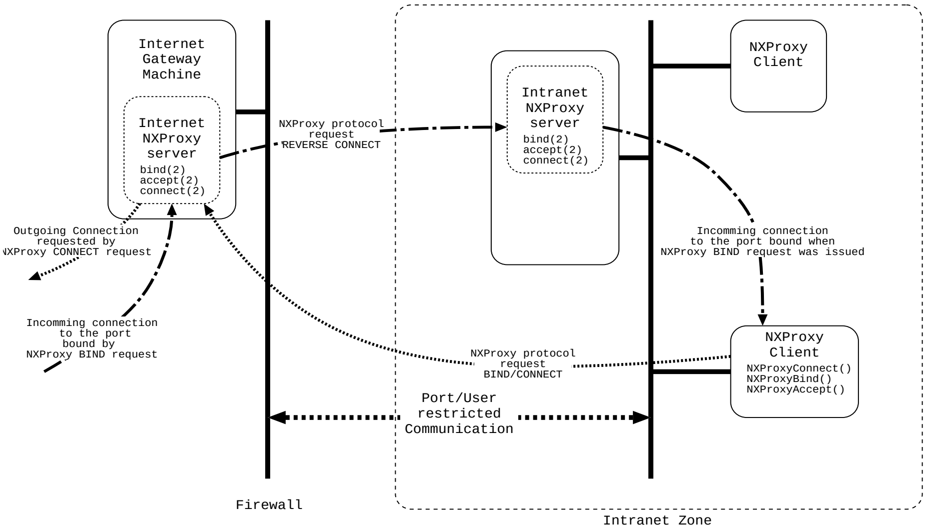

図 5　NexusProxy の動作概要

4. 分枝限定法によるナップサック問題の解法

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．本節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．

4.1 ナップサック問題

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 を越えない範囲で，p_i の合計が最大となるような荷物の組合わせを選びだ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．

4.2 実装方法

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 (Global Low, 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．
- プロセッサ間で何らかのタイミングで 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．

上記 3 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．

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．

5. 実 験

5.1 実験環境と基本性能

まずはじめに，今回実験に使用したテストベッドおよびその基本性能について述べる．表 2に実験環境を示す．次に，MPICH-G の *MPI_Send()* および *MPI_Recv()* を用いて測定した上記 2 つの計算機間の point-to-point のバンド幅を図 6 に示す．バンド幅は 1 時間おきに 24 度測定したが，結果には大きな変動は見受けられなかったのでグラフには平均値を示す．レイテンシは約 5msec であった．

表 2 実験環境

ETL 側マシン	:	Sun SPARC Station 5 85MHz × 1CPU, 32MB memory
RWC 側マシン	:	Sun Enterprise 450 300MHz × 4CPU, 640MB memory
ネットワーク	:	1.5Mbps (IMnet)

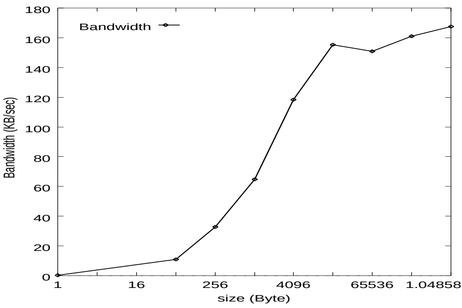

図 6 ETL⟷RWC 間のバンド幅 (MPICH-G)

5.2 ナップサック問題による実験

ナップサック問題の並列解法プログラムを実行し，その性能を測定した．実験に用いたデータは Parallel Software Contest(PSC) で用いられたデータの中から，枝刈りが発生しないデータ (PSC6) と枝刈りが発生するデータ (PSC7) の 2 種類を採用した．参考のため，本プログラムを PC クラスタ上で実行した際の並列化効率を表 3に示す．使用した PC クラスタは，ノードプロセッサが Pentium Pro 200MHz，ネットワークは 100Base-T Ethernet である．このように，PC クラスタ上では高い効率を得ていることがわかる．表 4に SS5 および Enterprise 上で逐次プログラムにより上記問題を解いた時間を示す．

表 3 PC クラスタ上での並列化効率

ノード数	1	2	4	6	8
効率	1	0.93	0.90	0.90	0.89

表 4 逐次実行時間

データ	SS5	Enterprise
PSC6	333.4 秒	70.8 秒
PSC7	125.7 秒	26.0 秒

実験は SS5 をマスタ，Enterprise をスレーブとして実行した．実験では起動するプロセスの数，*interval*，*stealunit* を変化させて実行時間を測定した．*interval* はどの程度の頻度でスレーブからのジョブ要求をチェックするかを，*stealunit* は 1 度にどの程度の量のジョブを配るのかを決定するパラメータである．*interval* が小さいと頻繁にジョブ要求をチェックするためマスタのオーバーヘッドが増加するが，大きすぎるとスレーブがジョブ待ちのためアイドル状態になる時間が増加してしまう．*stealunit* が小さすぎるとジョブ要求が頻繁に発生してしまう可能性が高くなるが，大きすぎるとジョブの転送に時間かかったり，負荷の不均一が生じる可能性がある．表 5に実行時間を示す．

表 5 実行時間 (単位は秒)

	PSC6			**PSC7**		
stealunit	*interval*			*interval*		
	10	100	1000	10	100	1000
10	82.6	73.9	76.0	51.9	42.9	39.5
100	×	116.7	104.1	×	209.9	177.5
1000	×	×	137.0	×	×	160.9

PSC6 は枝刈りが発生しないために並列度も高く，いずれの場合も SS5 単独で実行した場合に比べて大きく時間が短縮されている．PSC7 は枝刈りが行われ，ノードの開きかたと枝の刈り方によって実行時間が変わってしまうが，*stealunit* が 10 の場合には実行時間が大きく短縮されている．それぞれの実行時間についてブレークダウンした結果を図 7と図 8に示す．SS5 と Enterprise の実行時間を分枝操作にかかる時間，スレーブからジョブ要求が来ているかどうか調べる時間 (マスタのみ)，マスタにジョブ要求を出す時間とジョブを受け取る時間 (スレーブのみ)，その他の時間にブレークダウンした．これらの結果を見ると，*interval* や *stealunit* などのパラメータによって負荷分散処理のオーバヘッド (マスタのジョブ要求チェックの時間)，

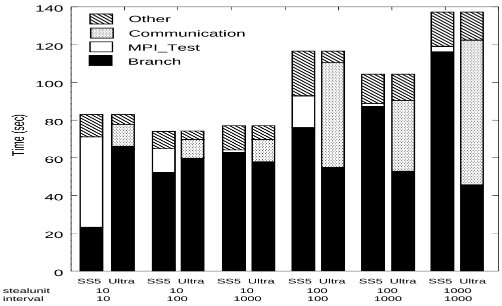

図 **7**　PSC6 の実行時間のブレークダウン

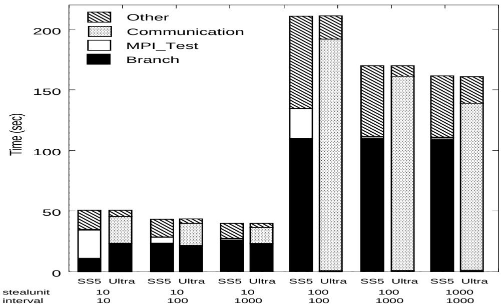

図 **8**　PSC7 の実行時間のブレークダウン

マスタとスレーブの分枝処理の量，データの送受信にかかる時間などがトレードオフの関係にある事が分かる．今回のデータではいずれも *stealunit* は10程度で良いことが分かる．

6. 考　　察

今回 RWC と ETL にある2台のワークステーションを使ったグローバルコンピューティング環境でナップサック問題の並列解法プログラムを実行し，その性能を調査した．今回このようなグローバルコンピューティング環境においてアプリケーションを開発する際に特に注意した点は，

- 動的負荷分散を行なう
- 通信量を抑え，なるべく通信と計算をオーバーラップさせる

の2点である．今回のプログラムはデータの独立性が高く，「ジョブがなくなったらもらいにいく」という方針により，負荷分散処理にあまりコストをかけなくても負荷が均一になり，ある程度の結果を得ることができた．例えば ETL 側の計算機が SS5 ではなく Enterprise と同程度の性能の計算機であれば，並列化による高い効率を得ることができると予測できる．今回題材として選んだ探索問題のような問題は，広域並列システムにおいても十分高い性能を得ることができると思われる．

また，今回試作した RMF および NexusProxy 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．

6.1　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$^{14)}$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．

6.2　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．

6.3 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[9]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[7),8)]，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．

7. 結論

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．また，firewall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．今後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 and Carl Kesselman, "The GRID: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers (1998).

2) I. Foster, Jonathan Geisler, Bill Nickless, Warren Smith, and Steven Tuecke, "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 and Carl Kesselman, "The Globus Project: A status report", Proc. Heterogeneous Computing Workshop, pp. 4–18 (1998).

4) http://www.globus.org/

5) I. Foster and Nicholas T. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems", Proc. Supercomputing (1998).

6) I. Foster, Carl Kesselman, and Steven Tuecke, "The Nexus approach to integrating multithreading and communication", Journal of Parallel and Distributed Computing, Vol. 37, pp. 70–82 (1996).

7) http://www.cs.vu.nl/albatross/.

8) H. E. Bal, Aske Plaat, Thilo Kielmann, Jason Maassen, Rob van Nieuwpoort, and Ronald Veldema, "Parallel Computing on Wide-Area Clusters: the Albatross Project", Proc. Extreme Linux Workshop, pp. 20–24 (1999).

9) T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang, "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).

10) M. Sato and S. Sekiguchi, "Ninf: A Network based Information Library for Global World-Wide Computing Infrastructure", Proc. HPCN, pp. 491–502 (1997).

11) H. Casanova and J. Dongarra, "Netsolve: A Network Server for Solving Computational Science Problems", Proc. Supercomputing (1996).

12) 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", 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).

13) 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– ", 情報処理学会システム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, pp. 191–196 (1999).

14) 中田秀基, 竹房あつ子, 松岡聡, 佐藤三久, 関口智嗣, "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).