修士論文

キャンパスグリッドの構築と
タスクスケジューリング法の研究

氏名　　：　林　雅樹
学籍番号　：　6124030181-3
指導教員　：　山崎　勝弘　教授

提出日　　：　2005 年 2 月 15 日

立命館大学　大学院　理工学研究科　情報システム学専攻

内容梗概

本研究では、大学内で限られたネットワークでのグリッドコンピューティングとグリッドに関する研究を行うキャンパスグリッドプロジェクトについて検討した。キャンパスグリッドは限定したネットワーク内で小規模なグリッド環境を構築し、誰でも簡単に参加でき、誰でもグリッドコンピューティングの研究、グリッドを用いた各種の大規模問題の高速化の研究を行えるような環境を提供すること、グリッドにおいて計算機資源を提供する最も重要な構成要素である PC クラスタを、グリッド環境に統合し親和性を高めることを目的としたグリッド環境である。本プロジェクトから Globus Toolkit を用いてキャンパスグリッドの計画における初期段階の研究室内でのグリッド環境構築を行った。本研究室で構築した PC クラスタ 2 組を統合し、グリッド環境を構築することができた。

また、タスクを実行する最適なホストの選択を行うタスクスケジューリング法の提案を行った。ホスト内でロードアベレージを取得し、ネットワークトラフィックを抑えるため、情報を各ホストに持たせ情報の分散管理を行う。また、ロードアベレージとネットワーク遅延を元にホストにランク付けし、最適なホストの探索を行う。以上の機能を持ったタスクスケジューラを実装した。研究室内のグリッド上でタスクスケジューラの評価を、Red-Black SOR プログラム、NAS Parallel Benchmark を用いて行った。ホストの選択では、ロードアベレージとレスポンスタイムを基にランクを付けることができた。また、並列プログラムの実行ではあるホスト 2 台に負荷をかけ、全ホスト 9 台のうちから 4 台を選択し、タスクスケジューラを用いて 4 台を選択する場合と、過負荷状態にある 2 台を含めた 4 台で実行する場合と比較した。タスクスケジューラを用いた場合の速度向上の最大は Red-Black SOR プログラムでは約 2.6 倍、EP ベンチマークでは約 4.2 倍、CG ベンチマークでは約 8.1 倍、MG ベンチマークでは約 2.9 倍、SP ベンチマークでは約 2.1 倍、BT ベンチマークでは約 3.8 倍の結果を得ることができた。

本論文では、まずキャンパスグリッドプロジェクトのついて述べ、キャンパスグリッドの構築、タスクスケジューリング法の提案、実装、評価を行う。

目次

図目次

表目次

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/並列計算を行うグリッド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。グリッドコンピューティングは広域ネットワーク上に分散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/並列計算を行うシステムである。グリッド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となる。Globus プロジェクトによる Globus Toolkit[1] はこのようなグリッド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 はグリッド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下位レベルの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リッドコンピューティングを構築することが可能となる。例えば、通信、ユーザ認証には Globus を用いて MPICH を実装した MPICH-G [7]などが存在する。Globus はグリッドコンピューティングシステムに必要とされる基本的サービスを提供しており。グリッドコンピューティングシステムの構築ミドルウェアとして事実上の標準になりつつある。

大阪大学でのバイオグリッドプロジェクトは、超高速ネットワークを用いて大学やバイオ系の研究機関の持つスーパコンピュータやデータベース等の研究資源を接続し、高性能なグリッド基盤環境を構築するプロジェクトで、その他にグリッドコンピューティングを用いた研究は、国内外で多く行われている。このようなグリッドプロジェクトは大規模なグリッド環境を構築することを目的としている[4]。しかし、逆に限定したネットワーク内で小規模なグリッド環境を構築し、誰でも簡単に参加でき、誰でもグリッドコンピューティングの研究、グリッドを用いた各種の大規模問題の高速化の研究を行えるような環境も必要であると考える。また、我々は PC クラスタを構築し、その応用を研究してきた。クラスタ技術はグリッドにおいても計算機資源を提供する最も重要な構成要素であり、グリッドと PC クラスタの親和性を高める研究が必要とされている。

グリッド研究で必要とされているのが、タスクに最適なホストの割り当てをする、スケジューリング法である[4]。また、Globus は、計算機資源とユーザ、アプリケーション間をつなぐ部分のみ提供しており、完全なグリッド環境を提供するわけではない。よって Globus Toolkit によって規定されるプロトコルを理解するアプリケーションの作成、及び協調動作の実現が必要である。Condor-G[8][9]と呼ばれるスケジューラが存在するが、Condor-G はハイスループットコンピューティング分野で研究開発された Condor を Globus Toolkit 用に開発されたものであり、グリッドに特化したスケジューラではないと言える。本研究で構築するグリッド環境に特化したタスクスケジューラの必要がある。

Globus Toolkit を用いて、大規模問題の Grid 環境での高速化、PC クラスタとグリッドコンピューティングの親和性を高めることを目的とした、大学という限られたネットワーク内でのグリッドコンピューティング、キャンパスグリッドの構築を行い、キャンパスグリッドの特性から、グリッド内の全ホストのロードアベレージによって決定されたランク

を基にしたタスクスケジューリング法の提案をし、その実装を行う。

2 章でキャンパスグリッドについて説明し、3 章でキャンパスグリッドの構築について述べ、4 章で空き計算資源とネットワーク遅延に基づいたスケジューリングついて説明し、5 章でスケジューラの評価を述べる。

2. キャンパスグリッドの構想

2.1 キャンパスグリッドとは

グリッドとは、電力の送電線網(電力網)を指す言葉である。これは我々が電力を使用する時に電源ケーブルをコンセントに差し込み、スイッチを ON にするだけで、発電機のことを考える必要がないように、グリッド技術を利用するときにも、コンピュータの構成を気にする必要がないことを表している。

グリッドの目的は、広域に分散した計算資源や情報資源をネットワークを介してまとめて管理し、ユーザが必要なときに必要なだけ利用できる環境を提供することである。これによってネットワークに接続された複数のスーパコンピュータを一つの巨大なコンピュータのように使用し、遊休状態になっている多数のパソコンを利用してスーパコンピュータと同様の計算環境を実現することである。実際に、SETI@Home や、遺伝子解析、たんぱく質解析など、科学技術計算分野で活躍している。[21]

遊休状態について言えば、キャンパス内に限っても研究室の PC とキャンパス内にある PC がアイドル状態になっていることが多い。我々はそれら PC に計算能力、計算時間メリット、ストレージの拡大等の手段として利用した場合に得られる利益の可能性があると考える。また PC クラスタという PC をベースにした並列計算機がある。PC クラスタは高い計算能力を発揮することができるが、遅延を最小限に抑えるため、小規模なネットワーク内でのハード構成に制約され、設定後のスケーラビリティに乏しい。

本研究では、既存の PC クラスタに用いられているハードウェア資源や、複数の PC を対象にしたグリッド環境を大学内で構築し、それらを大規模化する。このグリッド環境をキャンパスグリッドと称することにする。例えば、本研究室には既に 2 種類のスペックの異なるクラスタがあるので、これらを一元化することでより高性能な分散並列処理を可能にする。

キャンパスグリッドの目的として以下のことを挙げる。

(1) Globus Toolkit を導入することで、ヘテロジニアス環境を構築し、それを基盤とした並列分散処理システムを実現する。その後、設定面や特に研究室をまたがる形で将来利用した際の問題点、キャンパス内と限られたネットワーク内で発生する問題点を洗い出す。

(2) PC をベースにしたクラスタ技術はグリッドにおいて計算機技術を提供する最も重要な要素である。Globus Toolkit をベースにした、汎用タスクスケジューリングと資源管理の研究を行い、その PC クラスタとグリッドの親和性を高め、より高性能計算機として活躍できるグリッド環境を提供できるようにする。

(3) キャンパスグリッドを用いて、流体解析や画像処理、ハッシュ関数 MD5 など大規模問題を対象にした並列プログラムを実行させ、処理の高速化を図る。

(4) グリッド環境には、現在ジョブスケジューリングと資源管理というグリッドの特有の問題がある。この問題を題材としたタスクスケジューリング法や、リソースマネージメ

ントの研究を行う。

2.2　キャンパスグリッドの構成

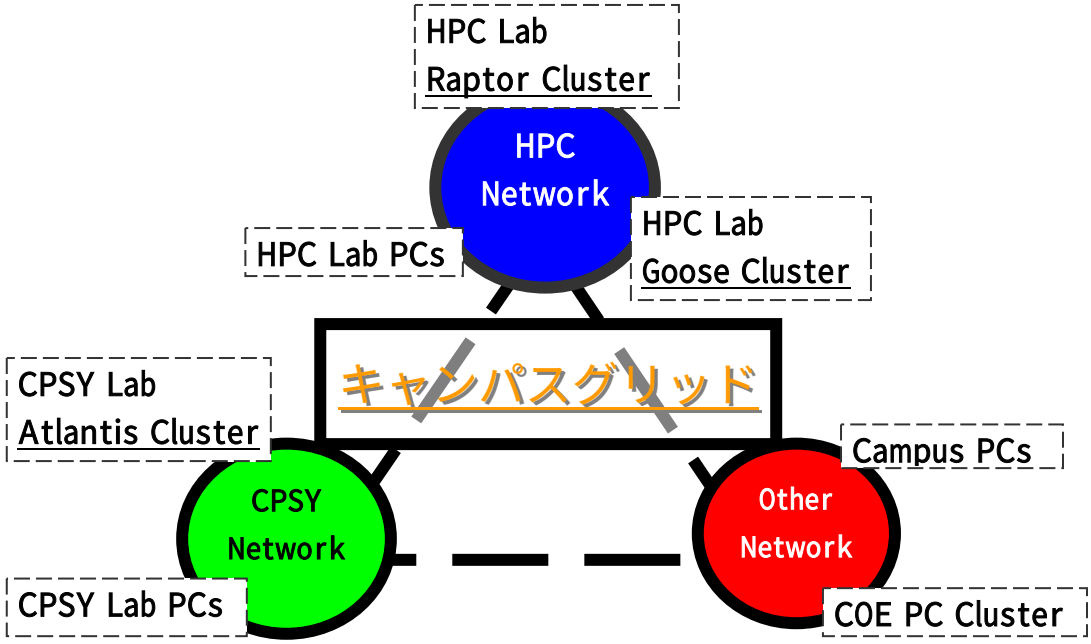

図 1：キャンパスグリッドの構成

キャンパスグリッドは、各研究室にある PC クラスタ、もしくは PC にグリッドミドルウェア Globus Toolkit を導入し、大学内のネットワークを利用した小規模ネットワーク上で、PC クラスタ、PC 群を 1 つの仮想大規模計算機として統合する。

キャンパスグリッドの構成を図 1 に示す。主にグリッド環境での計算は PC クラスタ群を用いる。

キャンパスグリッドでは、グリッド上で Globus サービスを提供する Globus サーバを設置する。Globus サーバが提供するサービスは、計算資源管理 GRAM(Globus Resource Allocation Manager)、グリッド上での FTP である GridFTP、マシン、ネットワーク等の情報管理の MDS、グリッド認証局 Globus SimpleCA[2]となる。

クライアントは計算ホストを兼ねており、クライアントには Globus Toolkit のクライアント用のミドルウェアをインストールする。

サーバを含めたすべてのホストは、同一ネットワーク上に位置していなければならない。つまり、Globus サーバはプライベートネットワークに属しているため、計算ホストは Rainbow ネットワークに接続しなければならない。よって、同一プライベートネットワークにルータを介して属しているホストの場合は、グリッド環境に入ることができない。

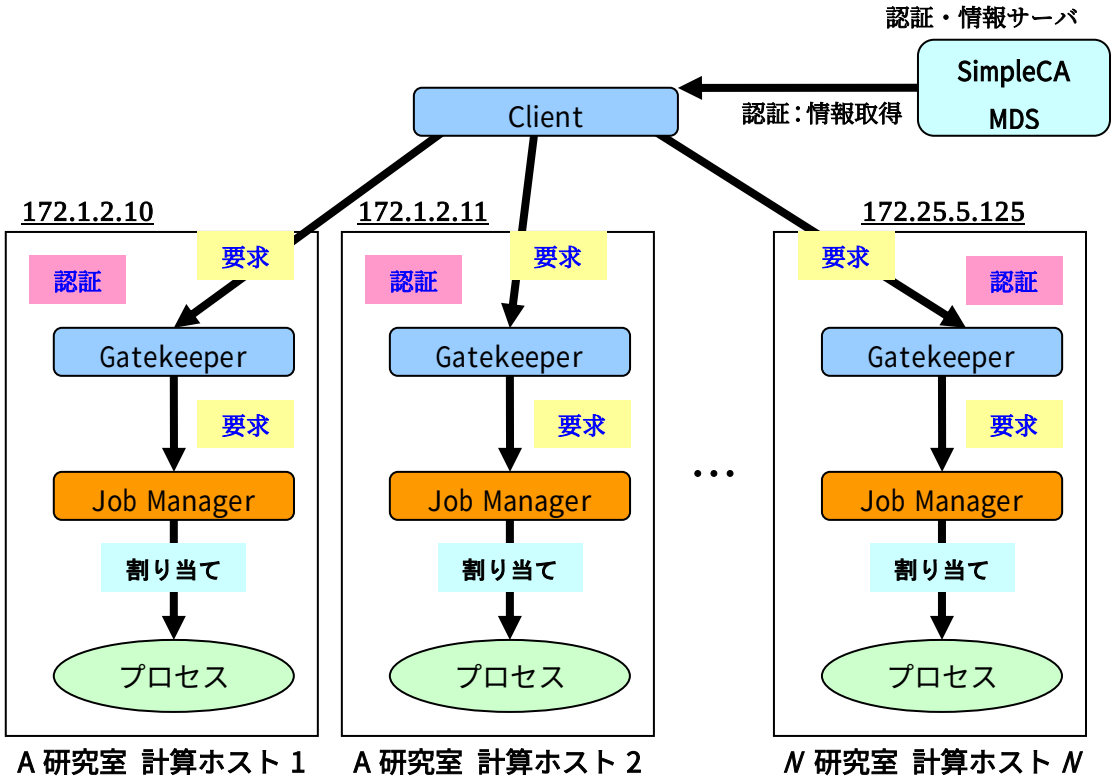

図 2：キャンパスグリッドサービス

2.3 キャンパスグリッドプロジェクト

2.3.1 キャンパスグリッドの実現

キャンパスグリッドの構築を以下の手順で行う。

(1) Globus Toolkit によるグリッド環境を山崎研内で構築する。対象とするホストはクラスタ 2 号機『Raptor』の構成ホストであり、それらに限定する形で MPICH-G2 の動作環境の構築を行う。

(2) クラスタ 1 号機『Goose』を(1)と統合し、近距離ネットワーク環境の研究室内でのグリッド環境の構築を行う。

(3) 将来的には異なるネットワーク上に存在する小柳研のクラスタ、複数 PC などと統合する。

2.3.2 キャンパスグリッドにおける研究

従来のクラスタ環境と異なり、グリッド環境では主に次の特性がある。

- 構成ホストの性能が多様
- 構成ホスト間の通信コストが多様

また、キャンパスグリッドの特徴として

・ プライベートネットワーク上での高度なセキュリティ
・ 近距離ネットワーク環境

このため、これまでのクラスタ上での並列化手法とは異なった手法を用いる必要が出てくる。グリッドにおける問題として、タスクの分配方法や、リソースの統合などがあげられる。グリッドにおける問題を、キャンパスグリッドの特長を生かしながら解決していきたいので、我々は Globus Toolkit をベースとしたタスクスケジューリング、資源管理の研究を進める。

本研究で実現するタスクスケジューラ、リソースマネージャのソフトウェアは、Globus Toolkit を利用する上位ミドルウェアとして位置づけられる。

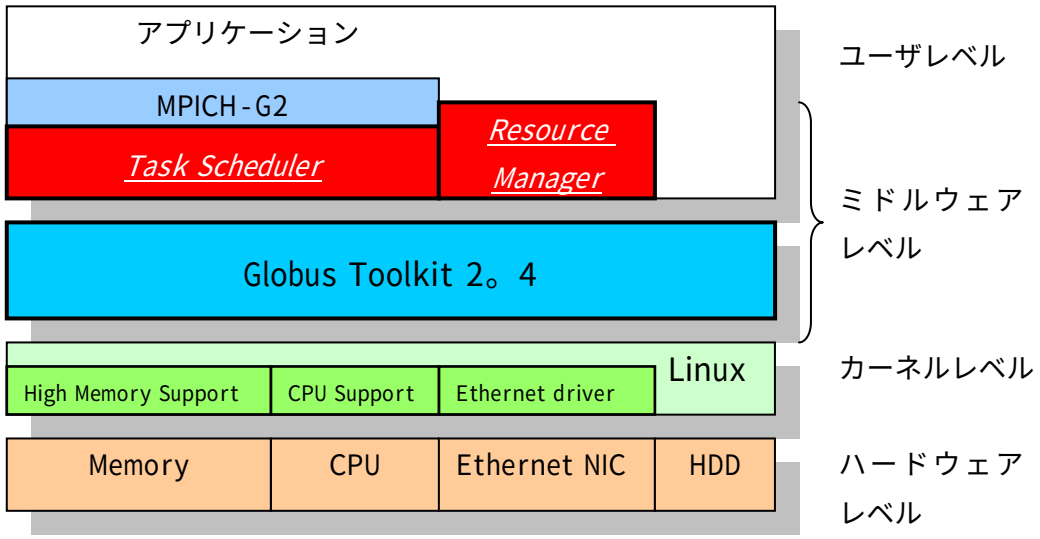

図 3：ソフトウェア階層

[1] タスクスケジューリング

ホストの性能差が出ることで処理時間の見積もりはより困難になり、ネットワークコストのばらつきも新たなスケジューリング問題を引き起こす。このため最適なタスクスケジューリング法の模索はこれまで以上に困難になることが予測される上、動的に状況が変動し得るため複雑にもなりかねない。このため、シンプルで汎用的な動的タスクスケジューリング法を研究する。ユーザがこれを利用することによりベストではないにせよ現実的な速度向上を容易に得ることができるようにしたい。

[2] 資源管理(Resource Management)

グリッドによってキャパシティを増強した場合、安直に資源管理をすると局所的に高負荷を引き起こしかねない。例えば MPI プログラムで次々に求まる結果をランク 0 で管理する場合、それを非常に大規模なホスト数の上で実行すると、ランク 0 相当のホストには主

にメモリに集中的な負荷が掛かる上、ランク 0 周辺のネットワークにも高負荷がかかる可能性がある。このため、P2P をベースにそういった負荷の一極化を避ける方法を研究する。

3. キャンパスグリッドの構築

3.1 グリッドミドルウェア Globus Toolkit の概要 [4][5][19][21]

Globus Toolkit は米アルゴンヌ研究所と南カルフォルニア大学情報科学研究所を中心に構成されている The Globus Alliance が開発しているグリッド環境構築のためのミドルウェアである。Globus Toolkit はグリッド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下位レベルのツールキットであり、Globus が提供するツールを用いて、上位レベルにグリッド コンピューティングシステムを構築することができる。

The Globus Alliance の目的は、共同作業を行う組織間でのコンピュータ、ストレージデータ、アプリケーションの共有を可能にすることによって、コンピュータの新しい利用方法を生み出すことにある。Globus Toolkit はオープンソースのプロトコル、サービス及びツールセットにより構成されるており、透過的で階層的分散マルチベンダーグリッドコンピューティング環境を実現できる。Globus Toolkit ディレクトリサービスはグリッド上でのリソースを特定し、それらのリソースの管理、選択確保を容易とし、リソースの機能を提供する。また、グローバル・グリッド・フォーラム(GGF)におけるグリッドコンピューティングのプロトコルと API の標準化を行っている。

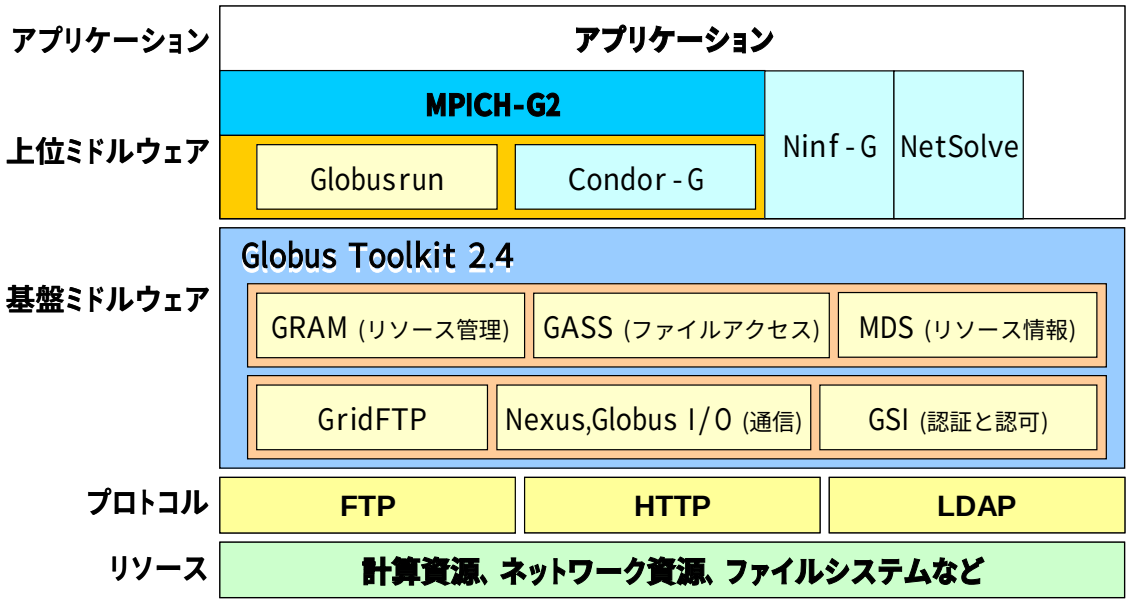

図 4：Globus のアーキテクチャ

Globus Toolkit は現在、Version2.4 と Version3.2 がリリースされている。Version2.4 は、科学技術計算分野で利用されており、実績が高いグリッドミドルウェアであり、また、Version3.2 では、OGSI (Open Grid Service Infrastructure)[3]という考えから Web サービスの利用を目的としたグリッドミドルウェアに仕上げられている。キャンパスグリッドの目的は科学技術計算分野、また並列処理分野における研究を対象としたグリッド環境を実現するため、我々は Globus Toolkit 2.4 を採用する。以下、Globus Toolkit 2.4 について説明する。Globus のアーキテクチャを図 4 に示す。Globus のアーキテクチャは次の４つで構成される。

- アプリケーション層
- 上位ミドルウェア層
- 基礎ミドルウェア層（Globus Toolkit）
- プロトコル、リソース層

3.1.1 Globus Toolkit の機能 [5]

Globus Toolkit はグリッド環境において計算機のための 3 つの要素を提供する。まず、ジョブのリモート実行を可能にするリソース管理を行う GRAM(Globus Resource Allocation Manager)である。GRAM はグリッドを構成する各ノードが提供するリソースの割り当て機能も含む。第二は情報サービス MDS(Monitoring and Discovery Service)で、グリッド構成に関する動的・静的な情報を提供する。そして最後はデータ管理サービス DMS(Data Management Service)あり、グリッド環境にあるデータへのリモートアクセス管理を行う。さらにこれらの要素をつなぎあわせるのがセキュリティ基盤 GSI(Grid Security Infrastructure)である。

Globus Toolkit のコアサービスは以上の 3 つの要素以外に表 1 に示すように、Globus のホスト間通信の「Nexsus」、及びデータの管理を担当する「GridFTP」から構成される。

表 1：Globus Toolkit のコアサービス

サービス	名前	概要
資源管理	GRAM	リソースの割り当ておよびプロセス生成
遠隔データアクセス	GASS	データへのリモートアクセス
情報	MDS	システム構造および状態に関する情報へのアクセス
セキュリティ	GSI	Authentication などのセキュリティサービス
通信	Nexsus	Unicast/Multicast 通信サービス
データ管理	GridFTP	各種転送方式を利用した FTP によるデータ管理
実行ファイル管理	GEM	実行ファイルの構築、キャッシングおよび配置

次項以降では Globus の上でアプリケーションが実行される際に使われる機能と、その状

態遷移を順をおって説明する。

3.1.2　セキュリティーGSI

グリッド環境というオープンなネットワーク上での認証と許可、また通信内容の保護を実現するために「GSI」を使用する。GSI は Globus Toolkit を用いて構築したグリッド環境での安全な通信やシングルサインオン、グリッド環境の様々な資源への権限の委譲といった、分散環境であるグリッドにおいて必要なセキュリティの機能を提供する。

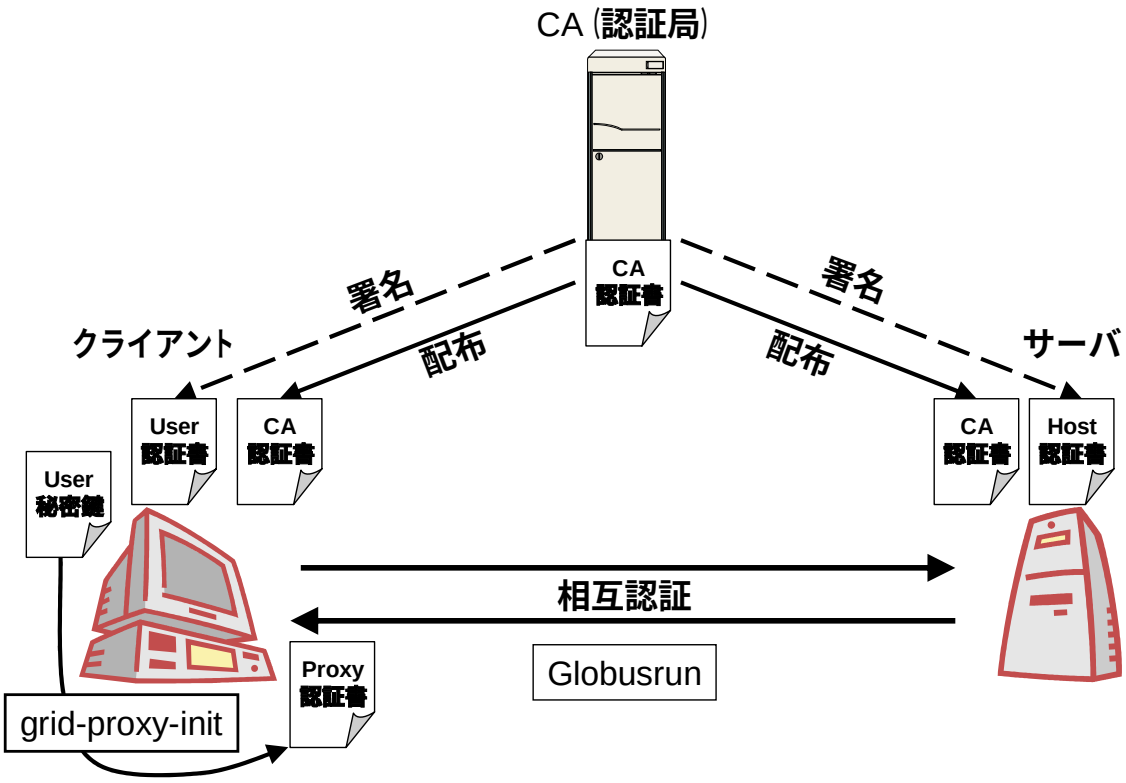

図 5：認証の流れ

ジョブ実行時の認証の流れを図 5 に示す。この図では、クライアントはサーバに対して何らかのジョブ（プログラム等）を投入しようとしている。ここで必要となる認証は「リクエストは正規のクライアントであるか」というのをサーバが知ることと、「リクエストしてサーバは正規のサーバであるか」というのをユーザが知ることの 2 つになる。つまり、リクエストを投入するクライアントとそのリクエストを受け取るサーバ間で相互認証が必要である。この相互認証が完了すれば、お互いに正しい相手であるということで、ジョブの投入を開始する。この認証で必要な条件は、クライアントが自分を証明する認証書を持っていること、サーバが自分を証明する証明書を持っていること、また、その認証書を署名した CA（認証局）の証明書をクライアントとサーバがそれぞれ持っていることである。

(1) GSI の認証書

GSI では特に認証の機能が充実しており、Globus Toolkit を用いて構築したグリッド環境で強力な認証を可能にする。GSI における認証には、ID とパスワードによる単純な認証方式ではなく、X.509 認証書を用いた認証方式を採用している。この認証書による認証には PKI（公開鍵暗号基盤）が前提となる。グリッド環境のすべてのユーザとサービス（リソース、アプリケーションプログラム）は、証明書を使用して本物であるかどうか識別される。

GSI で使用する証明書を**エラー! 参照元が見つかりません。**に示す。

表 2：GSI で使用する証明書

名前	証明
ユーザ証明書 (usercert.pem)	クライアント用の証明書。この証明書（の公開鍵）と対となる秘密鍵からプロキシ証明書を作成する
ホスト証明書 (hostcert.pem)	GRAM、GridFTP の認証にサーバ側で必要となる証明書
LDAP 証明書 (ldapcert.pem)	MDS の認証にサーバ側で必要となる証明書
CA 証明書 (<ハッシュ値>.o)	上記の証明書を保証するために署名を行った CA の証明書
プロキシ証明書	クライアントが実際に使用する証明書。この証明書によりグリッド環境における相互認証や権限委譲、シングルサインオンを実現する

クライアントユーザの認証に使用されるのがユーザ認証書と、それから作成されるプロキシ証明書である。後述する GRAM、GridFTP の認証に使用される認証書がホスト認証書で、MDS の認証に使用される証明書が LDAP 証明書である。これらの証明書にはすべての対になる秘密鍵が存在する。また、これらの証明書が本当に正しいものであるかを確認するためには、署名した CA（認証局）から配布される CA 証明書が必要となる。

GSI のユーザ・ホスト・LDAP 証明書は以下の４つの情報を含む。

i. ユーザあるいはオブジェクトを識別するサブジェクト名。これはグリッド環境における身元証明になる DN（識別子）である。
ii. サブジェクトの属する公開鍵。認証時にはこの公開鍵を認証相手に渡す。またこの公開鍵と対になる秘密鍵が別に存在する。
iii. 証明書に署名した CA の DN。この証明書がどの CA で署名されたものかは、この DN を確認することにより判断される。
iv. CA のデジタル署名。CA の正当性を証明するために使用される。証明書のハッ

シュ値を秘密鍵で暗号化する。

(2) プロキシ証明書

クライアントはサーバにリクエストする前に、グリッド環境へログインする必要がある。コマンドを実行しプロキシ証明書を発行することが、Globus Toolkit におけるグリッド環境へのログインに相当する。一度ログインしてしまえば、同じドメイン（グリッド環境）に属しているマシンであれば再度ログインする必要は無い。すなわち、シングルサインオンを実現している。プロキシ証明書は権限委譲を行うためにも使われ、グリッド環境でのパスポートような役割をしている。通常、クライアントの認証で使用するユーザ秘密鍵は、万が一奪われた時に悪用されるのを防ぐため、パスワードで保護されている。グリッド環境では多くのマシンが存在するので、ジョブの実行時に認証のために秘密鍵にアクセスし、何度もパスワードを入力するのは大変で、権限委譲時にパスワードを入力するのは困難である。そのために GSI ではプロキシ証明書を作成し、認証に利用する。プロキシ証明書は作成時に秘密鍵のパスワードを一度入力するだけでよい仕組みになっており、その後の認証には秘密鍵のパスワードを入力する必要は無い。プロキシ証明書作成時には、ユーザ秘密鍵で署名された新証明書（新公開鍵を含む）とパスワード無しの秘密鍵のペアが作成される。プロキシ証明書はこの鍵ペアとユーザ証明書（公開鍵を含む）で構成され、これを利用してクライアントの認証が行われる。なお、プロキシ証明書を盗まれた場合、クライアントに許可されている範囲のことはパスワードなしで何でもできてしまうことになりかねないため、有効期限（デフォルトで 12 時間）が設けられている。

3.1.3 リソースとジョブの管理－GRAM

「リソースとジョブの管理」を行っているのが「GRAM」コンポーネントである。GRAM はジョブを実行するためのリクエストとリモートシステムのリソースを利用するための標準的なインタフェースを提供する。GRAM を利用することにより、上位のサービスと下位のサービスを結びつけるのに必要とされるインタフェースや API、プロトコルの数を減らすことができる。

GRAM によるジョブの実行時には、リモートの GRAM サーバの処理が終わるまでコネクションを維持させる同期処理と、リモートの GRAM サーバの処理をリクエストしたらコネクションを切断する非同期処理の 2 種類を指定することができる。同期処理を例にした GRAM を使用したジョブの実行の仕組みを図 6 に示す。

Gatekeeper は、ジョブを遠隔から実行するために、リモートホスト上でルート権限で走っているプロセスである。Gatekeeper はジョブ要求が投入される前にリモートコンピュータの上に存在しなければならない、Gatekeeper はクライアントノードからアロケーションリクエストを受けとると以下の処理を行う。

- クライアントノードとの間で相互認証を実施

- リモートのリクエストをローカルユーザにマッピングする
- ローカルホスト上で、ローカルユーザに対して Job Manager を起動
- 新しく生成された Job Manager に対してリソースの割り当てを依頼

各種アプリケーションはリモートマシン上でコンパイルされていてもよい。

Job Manager は Gatekeeper に投入された要求を実行するために Gatekeeper によって生成される。ローカルシステムの上でジョブが起動し、それ以降のクライアントノードとすべての通信を処理する。

GASS は http プロトコルに GSI 認証を加えた通信プロトコルで、Job Manager からセキュアに出力された結果をデータ転送する。

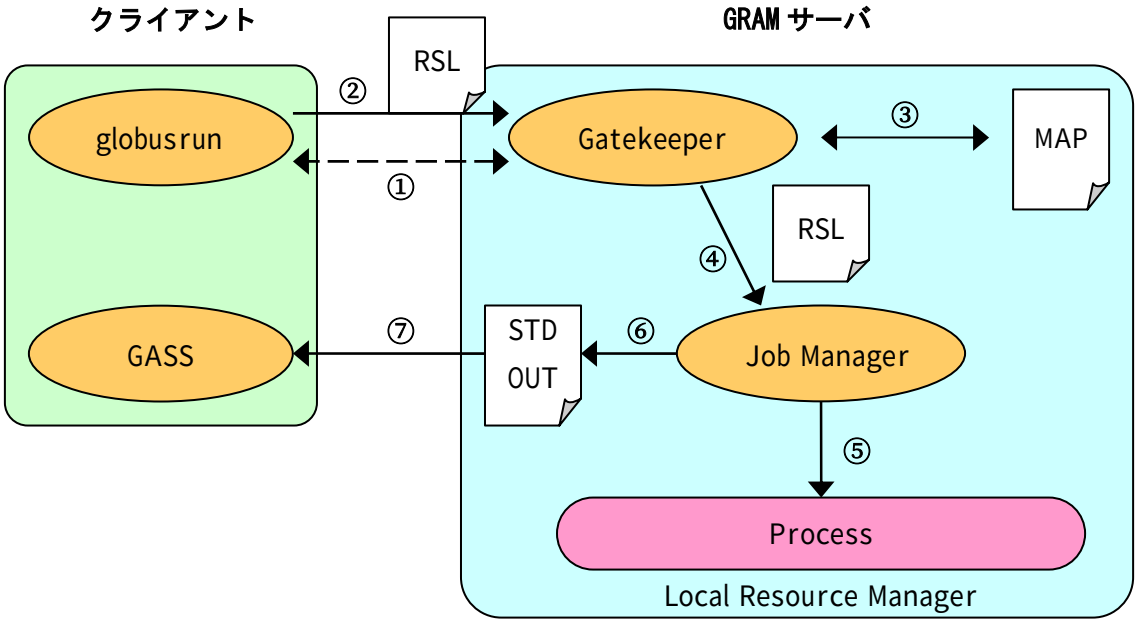

図 6：GRAM による同期処理の流れ

① 最初に、GSI に基づいた相互認証を実行する。この認証にクライアントはユーザ証明書を、GRAM サーバ側はホスト証明書を使用する。

② 相互認証に成功すると、実際に処理をする内容をクライアントから受け取る。このリモートシステムに実行させる処理の内容は Resource Specification Language（以下 RSL）という Globus Toolkit 特有の言語を使用する。

③ 次にマッピングファイル（grid-mapping）を確認し、リクエストしたユーザの DN を GRAM サーバ上のローカルユーザにマッピングする。

④ Gatekeeper はマッピングしたローカルユーザで、要求された Job Manager を起動

し、RSL で書かれたジョブを渡す。

⑤ Job Manager は要求されたジョブを実行する。Job Manager は RSL ファイルを読み込んでジョブの内容をそれぞれの Local Resource Manger 用のジョブの書式に変換して、ローカルジョブやスケジューラジョブとして処理を実行する。

⑥ GRAM サーバもしくはその他のリモートの実行ホストでジョブが終了すると、GRAM サーバのキャッシュディレクトリ（$HOME/.gass_cache ディレクトリ）に処理結果である標準出力と標準エラー出力がファイルとして出力される。

⑦ Job Manager は GASS の API を使用して、GASS サーバ経由で処理結果をクライアントに戻す。その後、キャッシュディレクトリに出力された処理結果のファイルは削除される。

このジョブの実行処理のなかで GRAM は次の役割を果たす。

- ジョブの要求の概要を記述する言語 RSL(Resource Specification Language)の解釈と処理、この RSL によりリソースの選択、ジョブプロセスの生成、そしてジョブの制御が記述される
- 遠隔監視とすでに生成されたジョブの管理
- ローカルホストの状態を監視するための情報を持つ MDS(3.1.4 章参考)を更新

RSL はリソースを記述するための共通交換言語である。GRAM の様々なコンポーネントがシステム内の他のコンポーネントと協力してマネージメント機能を実行するための RSL ストリングを操作する。RSL には複雑なリソースを記述するためにスケルトンシンタックスが用意されており様々なソース管理コンポーネントが、共通に属性値というペアの使用を導入している。RSL によってユーザはリソース要求やパラメータを指定することができる。

3.1.4 リソース情報の収集－MDS

「リソース情報の収集」を行っているのが「Monitoring and Discovery Servise(以下 MDS)」コンポーネントである。MDS は Lightweight Directory Access Protocol(LDAP)によって実装されており、標準的なインタフェースとグリッド環境で使用されるリソース情報の定義を提供する。その情報には、OS 名やディスクサイズといった静的なデータ、またはメモリや CPU の使用状況といった動的なデータがある。MDS の利用方法としては、たとえば、グリッド環境に属している複数のホストのリソース情報を検索して、あるリモートジョブの必要要件にあった割り振り先ホストを決定するといったものがある。

MDS コンポーネントは、さらに Grid Resource Information Service(GRIS)と Grid Index Information Service(GIIS)という２つのコンポーネントに分かれる(図 7)。GRIS はローカルのリソース情報のみを扱うコンポーネントである。GIIS では、Information Provider と

呼ばれる機能がローカルマシンのリソース情報を調べて、キャッシュしている。キャッシュは決められた一定間隔ごとに GRIS 本体に報告され、その結果 GRIS はローカルのリソース情報全体を保持し、外部に対して公開することが可能になる。その情報内容を一定間隔ごとに上位の GIIS に登録することも可能である。

Information Provider はそれぞれ OS 情報、CPU 情報、メモリ情報など情報ごとに複数存在する。OS の情報のようにいつまでも変わることのないような静的な情報に対してはキャッシュを更新する間隔は長く、CPU やメモリの情報のように刻一刻変更されるような動的な情報に対してはキャッシュを更新する間隔を短く設定されている。

MDS のもう一つのコンポーネントの GIIS は、下位の GRIS または GIIS から報告されたリソース情報を管理するコンポーネントになる。この働きにより、グリッド環境に参加しているマシンのリソース情報は、DNS のように、中央集権型ではなく分散化して管理することが可能になる。分散して管理することにより、耐障害性を高め、スケーラビリティを確保することが可能となる。

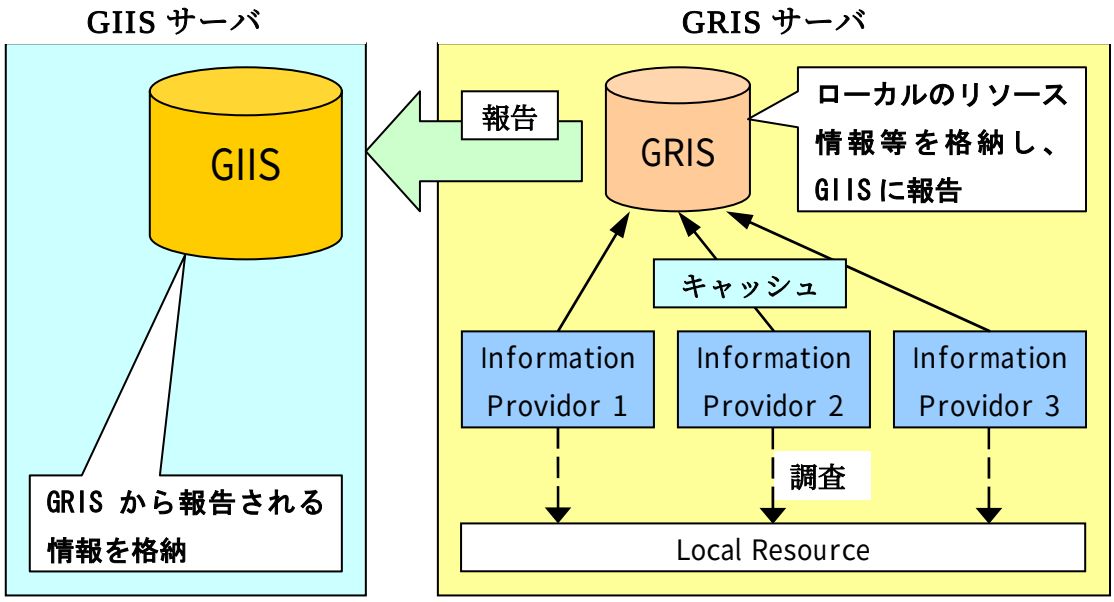

図 7：GIIS と GRIS

3.1.5　データの管理－GridFTP

「データ管理」を行っているのが「GridFTP」コンポーネントである。GirdFTP はグリッド環境にセキュアなデータ転送メカニズムを提供している。データ転送は GASS コンポーネントでも行うことができるが、GridFTP は GASS よりも高度なデータ転送を行うことができるので、通常 Globus Toolkit 2.4 におけるデータ転送は GridFTP を使用する。

Globus Toolkit 2.4 に含まれる GridFTP サーバは、オープンソースのソフトウェアである Wu-ftp サーバを拡張して作られたものである。Wu-ftp サーバの機能に加えて、表 3 で示している機能を拡張している。

GridFTP のファイル転送の流れを図 8 に示す。クライアントは GridFTP サーバ A が保持しているファイルを GridFTP サーバ B に転送している。このときには URL（「gridftp://＜ホスト名＞/＜ファイルのフルパス名＞」の書式）を指定して、転送する。

表 3：GridFTP の機能

機能	説明
GSI のサポート	GSI による認証や認可等のセキュリティ機能
サードパーティ転送	第三者間でのファイル転送を実現している。通常、自分、A、B というホストが存在し、A の持つファイルを B に転送する場合には、一度 A の持つファイルを自分に転送し、自分から B にファイル転送をするため、2 度のファイル転送が生じるが、サードパーティ転送では、A の持つファイルを A から直接 B に転送する。
パラレルデータ転送	1 つのデータ転送に対して、複数のコネクションを張ることによりパフォーマンスの向上を実現している。低帯域幅の場合、パケットロスを防ぎ、パフォーマンスが向上する場合がる。
ストライピング転送	ファイルをストライピングして、複数の独立したノードに分散保持する機能。
パーシャルファイル転送	ファイルの一部分だけを転送する機能。
リライアブルファイル転送	一時的なネットワーク障害やサーバの停止によって失敗したファイル転送を途中から再スタートさせる機能

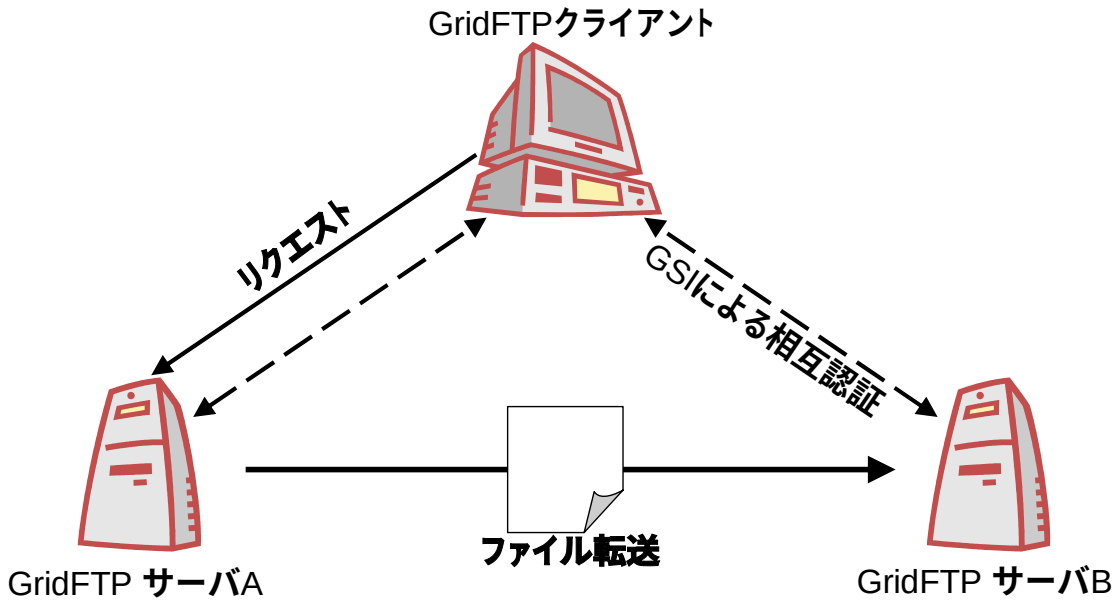

図 8：GridFTP のファイル転送

従来の FTP の機能では、図 8 のようなファイル転送を実行しようとした場合に、一度クライアントがサーバ A から転送したいファイルを get し、サーバ B に put とするという手順を踏む。GridFTP では、クライアントがサードパーティファイル転送を実行すると、クライアント－GridFTP サーバ A 間とクライアント－GridFTP サーバ B 間で GSI による相互認証が行われる。クライアントは認証にユーザ証明書から作成したプロキシ証明書を、GridFTP サーバは認証にホスト証明書を使用し、これに成功するとコネクションが確立し、GridFTP サーバ A、B 間で直接、ファイル転送が実行される。

3.2 Globus サーバの構築

図 2 の認証と情報取得を行う Globus におけるサーバの構築を行う。Globus Toolkit をインストールし、各種設定を行う。インストールする Globus Toolkit のパッケージは、GlobusToolkit を簡単にインストールするための Grid Packaging Tools と、Globus Toolkit 本体である。Globus Toolkit は、データマネージメント、インフォメーションサービス、リソースマネージメントから成り、それぞれに Client、Server、SDK の Bundle が用意されている。Globus サーバにはすべてのサービスの Client、Server、SDK をインストールする。

Globus Toolkit をインストールした後のデフォルトの認証局は、The Globus Alliance が用意している GlobusCA となっている。しかし、GlobusCA は 2004 年 1 月をもって閉鎖されたことと、キャンパスグリッドでは、大学内のプライベートネットワークを利用し構築するため、認証局を新たに構築しなければならない。よって、Globus Alliance が提供する簡易版の認証局作成パッケージ SimpleCA[2]を用いてキャンパスグリッドの認証局を構築する。

以上から、Globus サーバが提供するサービスは

- データマネージメントサービス
- インフォメーションサービス
- リソースマネージメントサービス
- SimpleCA による認証局

となる。

表 4 に Globus サーバとする計算機リソースを示す。

表 4：Tyranno リソース情報

ホスト名	Tyranno.hpc.cs.ritsumei.ac.jp
OS	Fedora Core 1 (2.4.22-1.2115.nptlsmp)
CPU	Intel(R) Xeon(TM) CPU 2.80GHz 2nd cache 512 KB
Memory	4.0GB
NIC	Intel(R) PRO/1000 Network

また、グリッドでの追加機能として MPICH-G2[7]をインストールする。これにより Globus Toolkit 上で MPI プログラムが動作する。

3.3 Globus クライアント群の構築

Globus クライアントはキャンパスグリッドにおける計算ホストも兼ねる。Globus クライアントは本研究室で構築した 2 組の PC クラスタを用いて構築した。両 PC クラスタのリソース情報を表 5 と表 6 に示す。

表 5：Goose クラスタのリソース情報

OS	Redhat Linux 7.3 (2.4.21-1SCORE)
CPU	Intel(R) Pentium III(TM) CPU 500MHz 2nd cache 512 KB
Memory	512MB
NIC	RealTek RTL8139 Fast Ethernet

表 6：Raptor クラスタのリソース情報

OS	Redhat Linux 8.0 (2.4.21 SCORE version)
CPU	Intel(R) Pentium4(TM) CPU 3.20GHz 2nd cache 512 KB
Memory	2.0GB
NIC	Intel(R) PRO/1000 Network

Globus クライアントにインストールするパッケージは、Grid Packaging Tools と、Globus Toolkit 本体である。Globus Toolkit は、データマネージメント、インフォメーションサービス、リソースマネージメントの Client、Server の Bundle をインストールする。また、サーバ同様に、MPICH-G2 をインストールする。

4. 空き計算資源とネットワーク遅延に基づいたスケジューリング法

4.1 問題定義

グリッド研究で必要とされているのが、タスクの最適なホストの割り当てをする、スケジューリング法である。

Globus Toolkit は、グリッドインフラを構築するための道具であり、Globus Toolkit のサービスは計算機資源とユーザ、アプリケーション間をつなぐ部分を提供することであり、完全なグリッド環境を提供するわけではない。よって Globus Toolkit によって規定されるプロトコルを理解するアプリケーションの作成、及び協調動作の実現が必要である。

代表的な資源管理ソフトウェアとして、Condor-G[8]と呼ばれるスケジューラが挙げられ

る。Condor-G はウィスコンシン大学においてハイスループットコンピューティング分野で研究開発された Condor を Globus Toolkit のプロトコルに従ったインターフェースをもち、Globus と統合するための API（Application Programming Interface）として開発された。Condor-G の動作の様子を図 9 に示す。

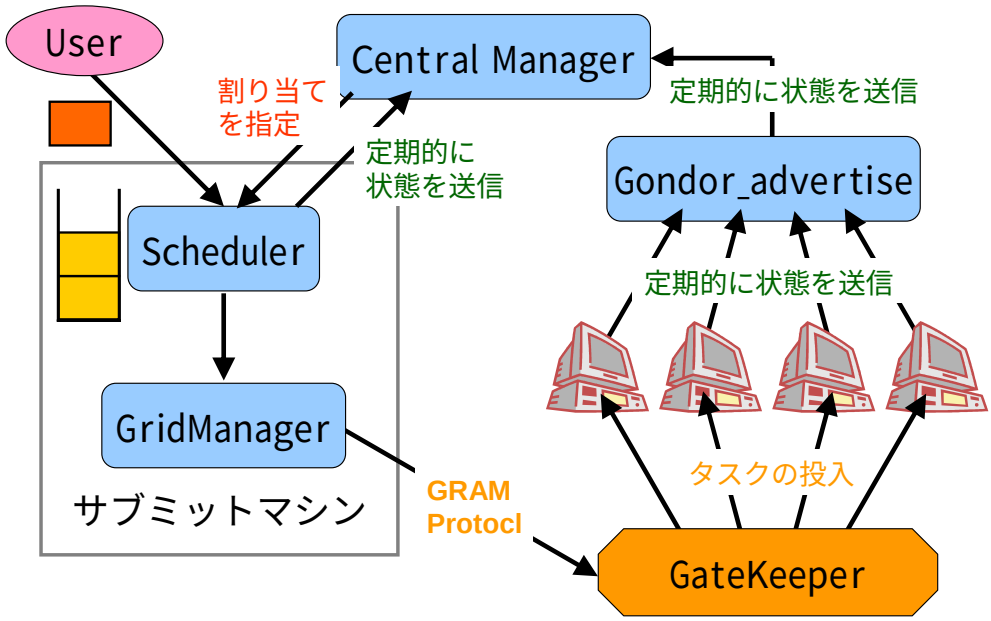

図 9：Condor-G

Condor-G の構成要素として、ユーザ、Condor-G のタスクスケジューラ、Central Manager からなる。Condor-G の定常状態として、ホストは Condor_advertise にジョブの実行状況や、資源空き状況などの情報を登録し、Condor_advertise は Central Manager に定期的に状態を送信する。また、サブミットマシンと呼ばれるクライアントマシンにおいて、タスクスケジューラは投入されるジョブを常に監視し、その状態を Central Manager に定期的に送信している。Condor-G を利用する場合、ユーザはサブミットマシンからジョブ（アプリケーション）を実行する。タスクスケジューラはジョブの投入を認め、Central Manager からホストの割り当ての指定を受信する。情報を元に GridManager が RSL を用いて、ジョブが要求する資源の情報が記述され、GRAM Protocol を介しグリッドホストの Gatekeeper にジョブが渡され実行する形となる[9]。

Condor-G は、ジョブの実行状態を定期的に記憶するチェックポイント機能や、チェックポイント機能を利用した別の計算資源上へのジョブ移行機能資源を持ち、情報を一元的に管理することにより、ジョブの要求に応えることができる。しかし、グリッドコンピューティングの観点から見ると、ネットワークを常に使用することにおける資源使用コストが高まること、スケジューラのタスク管理において、同時実行ができないという問題がある。

グリッド環境では主に次の特性がある。

・ 構成ホストの性能が多様

・ 構成ホスト間の通信コストが多様

また、キャンパスグリッドの特徴として

・ プライベートネットワーク上での高度なセキュリティ

・ 近距離ネットワーク環境

以上の特徴を考慮し、特にネットワークの情報、管理、そして、通信コストを最大限に考慮したタスクスケジューリング法の研究を行う。

4.2 タスクスケジューリング法

グリッド環境では、ヘテロジニアスな構成に成りうる。よって、ヘテロジニアスなホスト群の中から、最適なホストの選択が必要となる。そのため、計算機状態の監視による空き計算機資源の調査を行う次のタスクスケジューリング法を提案する。本スケジューリング法では、ネットワーキングを重要視し、空き資源探索を行う。

空き資源情報はあるサーバに集中管理するのではなく、各ホストが保有する分散管理モデルとする。この方法を取ることにより、サーバに常に情報を送るためのネットワークトラフィックを極力少なくすることができる。

計算機状態の対象となるデバイスは、CPU、メモリ、ネットワークインターフェース(NIC)であり、調査する値は、各スペック、各負荷を独自に実装するロードアベレージモニタで監視する。それぞれの値を用いて次の式(1)によって計算機資源の状態を値で表す。

$$P = \frac{Cc + Mm}{(n/N)} \qquad \cdots (1)$$

C : プロセッサ速度 (値:bogomips)　c : CPU の空き状態 (単位:%)

M : メモリのスペック (単位:MByte)　m : メモリの空き状態 (単位:%)

N : NIC のスペック (単位:Mbps)　n : NIC のトラフィック (単位:Byte)

bogomips とは、カーネルが起動時にコンピュータ上で一種のビジーループを実行し、プロセッサの速度を測った値である。また、ネットワークの状態の算出で乗算しているのは、ネットワークの負荷を強調するためである。

各負荷を 1 分間の間隔で取得し、1 分ごとに P 値を算出する。プロセッサ速度を 6370.09、メモリを 2048MByte、ネットワークインターフェースを 1000MByte として例を挙げる。

① ホストの定常状態、つまりホスト内ではサービスが立ち上がっているがユーザが利用してないアイドル状態を、CPU が 70%、メモリが 80%、ネットワークトラフィックが 500Byte とする。式(1)に当てはめると P は次のようになる。

$$P = \frac{6370.09 \times 70 + 2048 \times 80}{500 \div 1000} = 1219492.6$$

② ホストでネットワークを利用しないが負荷のかかるプログラムを実行しているとし、空き状況を CPU が 20%、メモリが 10%、ネットワークトラフィックが 500Byte とする。式(1)に当てはめると P は次のようになる。

$$P = \frac{6370.09 \times 20 + 2048 \times 10}{500 \div 1000} = 295763.6$$

③ ホストでネットワークを利用するが負荷の少ないプログラムを実行している、もしくはそのホストにファイルの転送しているなどネットワークだけ負荷がかかっているとし、空き状況を CPU が 60%、メモリが 70%、ネットワークトラフィックが 50MByte とする。式(1)に当てはめると P は次のようになる。

$$P = \frac{6370.09 \times 60 + 2048 \times 70}{50000 \div 1000} = 10511.308$$

④ ホストが過負荷状態であるとし、空き状況を CPU が 20%、メモリが 10%、ネットワークトラフィックが 50MByte とする。式(1)に当てはめると P は次のようになる。

$$P = \frac{6370.09 \times 20 + 2048 \times 10}{50000 \div 1000} = 2957.636$$

ネットワークの負荷を重点にロードアベレージの算出をしているため、③では CPU とメモリはある程度利用可能状態であるにもかかわらず、②のときの計算資源利用状態よりも低い数値となることがわかる。

P 値は過去 24 時間分記憶しておき、式(2)により 1 時間毎の平均から時間による重み付けを行い最終的なホストの空き状態の値として取得する。ただし、式(2)の計算はクライアントのリクエストがあるときにのみ行う。

$$P(t) = \frac{P_1}{t_1^{\,2}} + \frac{P_2}{t_2^{\,2}} + \cdots + \frac{P_i}{t_i^{\,2}} + \cdots \frac{P_{24}}{t_{24}^{\,2}} \qquad \cdots (2)$$

分散並列処理を行う場合、ホストでの処理が速くてもメッセージ送受信において反応が遅れればその分処理の遅れが顕著に現れる。メッセージ送受信によるネットワーク遅延を調べる。グリッド環境ではグリッドの特徴から、ある程度の遅延を認める。そのため、スケジューラでのネットワーク遅延の許容範囲を設ける。

以上のホストのロードアベレージとレスポンスタイムの値を用いて、ホストのランク付

けをする値を式(3)から算出する。

$$R = P(t) \times \frac{1}{respons\_time} \qquad \cdots (3)$$

4.3　タスクスケジューラの実装

実装するタスクスケジューラは Globus Toolkit とアプリケーションを接続するミドルウェアである。対象 OS は Linux、Unix 系とする。対象となるホストは、グリッドホスト群全体である。Globus Toolkit を利用し、また、Globus Toolkit と実装アプリケーション間の接続の役割を果たす機能を持つ。

本スケジューラの対象 API は MPICH とする。コンパイラは Globus Toolkit に対応した MPICH-G2 を用いる。また MPICH-G2 はグリッド上で実行できるように RSL 記述による実行設定ファイルを自動生成するコマンドと、グリッド上で RSL ファイルを実行するコマンドが用意されている。しかし、ホストの選択はホスト名が書かれたファイルを読み込み、ファイルに書かれたホストすべてを利用して実行する。ホストの指定はユーザ任意であり、ホストのロードアベレージを考慮しない実行となる。

タスクスケジューラはサーバとクライアントの構成となる。サーバはホストのロードアベレージを調査し、クライアントからのロードアベレージの受信要求を受け取り返信するインターネットデーモンとする。クライアントはサーバにロードアベレージの値を受信し、それを元にホスト群から最適なホストを選択し、MPICH-G2 のコマンドを利用し Globus にジョブを与える。

4.3.1　タスクスケジューラの構成

(1)　スケジューラサーバ

- 機能

スケジューラサーバは毎分ホストの CPU の使用量、メモリの使用量、ネットワークトラフィック量を観測し、CPU 速度とそのアイドルレート、メモリ量とそのアイドルレート、ネットワーク速度とトラフィック量を元に 1 分間のロードアベレージを算出する。それと同時に毎分のロードアベレージを時間ごとに記憶する。

スケジューラサーバはスケジューラクライアントからロードアベレージの取得リクエストを待ち、受信するとホストの 24 時間のロードアベレージを算出し、スケジューラクライアントに返信を行う。

- モジュール構成

スケジューラサーバのモジュール構成は、ノード情報取得、毎分ロードアベレージ値の格納とインターネットサーバとなる。

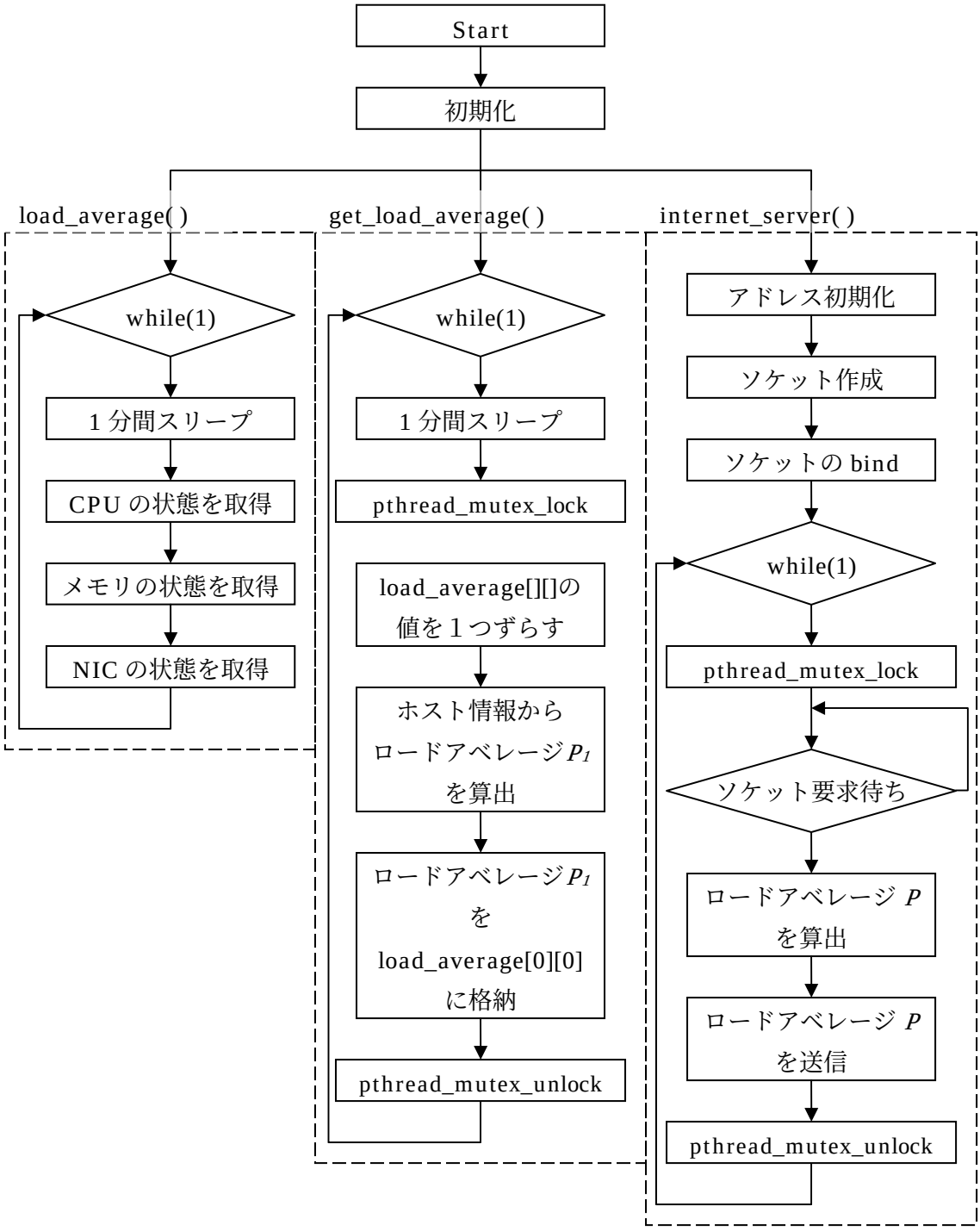

図 10：スケジューラサーバのフローチャート

■　ノード情報取得　load_average( )

ノードの負荷の状態を取得し、その値を元に CPU アイドルレート、メモリアイドルレート、ネットワークのトラフィック量を算出する。構成する関数を以下に示す。

・ stat_read( )　　：CPU Time の取得

・ meminfo_read( )　　：メモリ利用状態の取得
・ dev_read( )　　：ネットワークトラフィックの取得

■ 毎分ロードアベレージ値の格納　get_load_average( )

ノード情報取得と平行に実行し、ノード情報取得で得られた各値を元に、式（1）に当てはめ計算を行った後、配列に格納する。

■ インターネットサーバ　internet_server( )

アドレスを初期化し、クライアントはどのアドレスからもアクセスできるように設定。その後、クライアントからのリクエストを待つ。クライアントからのロードアベレージ取得リクエストを受信し、その瞬間のホストのロードアベレージを式（2）に当てはめ算出し、クライアントにロードアベレージ値を送信する。

以上の 3 つのモジュールはスレッドにより同時実行される。各モジュールの関係をフローチャートとして示す。

配列 load_average[][]は 1 次配列に時間、2 次配列に分が対応する。また load_average[][]は広域変数として宣言し、各スレッドが参照できるようにする。load_average[][]が更新している間に、ロードアベレージで計算を行わないように POSIX Thread の関数 pthread_mutex_lock と pthread_mutex_unlock で排他処理を行っている。

(2) スケジューラクライアント

● 機能

スケジューラクライアントは、グリッド環境に属するホストのホスト名が書かれた gridhosts ファイルを読み出し、グリッドホスト群に対しロードアベレージのリクエストを送信する。それと同時にグリッドホスト群に対するリクエストのレスポンスを観測する。各ホストのロードアベレージとリクエストのレスポンスから、ホストにランクを付け、降順に並べ替える。ユーザの要求ホスト数のみランクの上位から取り出し、マシンファイルを作成する。

● モジュール構成

■ ロードアベレージ取得

Globus Toolkit をインストールしているホストに対して、スケジューラサーバからホストのロードアベレージ情報をインターネット経由で取得する。また、クライアントのリクエストに対してサーバからのレスポンスを計測する機能も備える。

■　ホストソート

ホストのロードアベレージとレスポンスからホストのランク付けに用いる値を式(3)により算出し、値を元にホストをクイックソートにより降順に並べ替える。

4.3.2　モジュールの説明

(1)　line_option( )

スケジューラサーバ起動時に用いるオプションを読み取る。スケジューラサーバのオプションは、正常に値が格納されているかなどを表示するデバッグモードである。

(2)　stat_read( )

CPU の利用状況を読み込む関数である。プロセスの情報を含む擬似ファイル群 /proc にある stat ファイルを読み込み、CPU の利用状況を得る。取得する値は、ユーザーモード、低い優先度 (nice) でのユーザーモード、システムモード、タスク待ち (idle task)、でそれぞれシステムが消費した時間を jiffies (1/100 秒) を単位として計測した値である。

(3)　meminfo_read( )

メモリの利用状況を読み込む関数である。プロセスの情報を含む擬似ファイル群 /proc にある meminfo ファイルを読み込み、メモリの利用状況を得る。取得する値は、総メモリ量、メモリの空き容量である。

(4)　dev_read( )

ネットワークのトラフィック量を読み込む関数である。プロセスの情報を含む擬似ファイル群 /proc にある net/dev ファイルを読み込み、OS の起動からその時間までの総トラフィック量を得る。取得する値は、ネットワークインターフェースを通過した送受信のバイト量である。

(5)　load_average_check( )

以上の 3 つのモジュールを用いて、そのときのホスト情報を取得する。その値を式(1)に当てはめ、その瞬間のロードアベレージの算出を行う。

またオプションにデバッグモードを用いた場合、値の表示を行う。表示する値は、CPU のユーザーモード、低い優先度 (nice) でのユーザーモード、システムモード、タスク待ち (idle task)、メモリの総メモリ量、利用容量、空き容量、シェア容量、バッファ、キャッシュ、ネットワークの送受信バイト、パケット、ロードアベレージ P である。

4.4 Globus Toolkit との結合

実装したスケジューラを利用してグリッド上で並列プログラムを実行させる。その流れを以下に示す。

```
$ grid-proxy-init  ･･･プロキシ証明書を作成
Enter GRID pass phrase for this identity:
***** ユーザ証明書のパスフレーズを入力(表示されない)

$ client 4  ･･･クライアントを実行しマシンファイル(machines)を作成

$ mpirun -np 4 -dumprsl sample > run.rsl
                          ･･･sample を 4 台で実行するよう RSL ファイルを作成

$ mpirun -globus-rsl run.rsl  ･･･run.rsl をグリッド上で実行

$ grid-proxy-destroy  ･･･プロキシ証明書を削除
```

Globus Toolkit と MPICH-G2、実装したスケジューラを利用したプログラムの実行の一連の流れをシェルスクリプトで記述しグリッド実行コマンドとして用意する。次にシェルスクリプトの利用方法を示す。

./runMPI.sh［MPI 実行ファイル］［実行ホスト数］

このコマンドを実行することにより、Globus のシングルサインオンのためのパスフレーズを入力することで並列プログラムを実行することができる。

5. スケジューラの評価

実装したスケジューラを、連立 1 次方程式の解法である SOR 法と並列処理ベンチマークである NAS Parallel Benchmark を用いて評価する。テスト環境として、Tyranno と PC クラスタ Raptor 8 台を用い、Raptor05 と Raptor07 で常にプログラムを実行している環境とし、テストを行う。評価項目はホスト検索の評価、5 回実行の平均実行時間とする。1 台での通常実行結果とグリッド環境内から Tyranno、Raptor05、Raptor06、Raptor07 を選択し実行した結果を比較対象とする。

5.1 連立 1 次方程式の解法 SOR 法による評価

5.1.1 Red-Black SOR 法

対象とする連立方程式の定義をする。

図 11 のような正方領域で、領域内の変化を表す式(4)と定義する。

$$\nabla^2 u(x,y) = f(x,y) \qquad \cdots (4)$$

領域を辺ごとに N 等分し格子に区切り、格子のある頂点を以下のように表す。

$$u_{i,j} = u(x_i, y_j) \qquad \cdots (5)$$

$$h = 1/N \text{、} \quad x_i = ih \text{、} \quad y_j = jh \qquad \cdots (6)$$

$$(0 \le i \le N, 0 \le j \le N) \qquad \cdots (7)$$

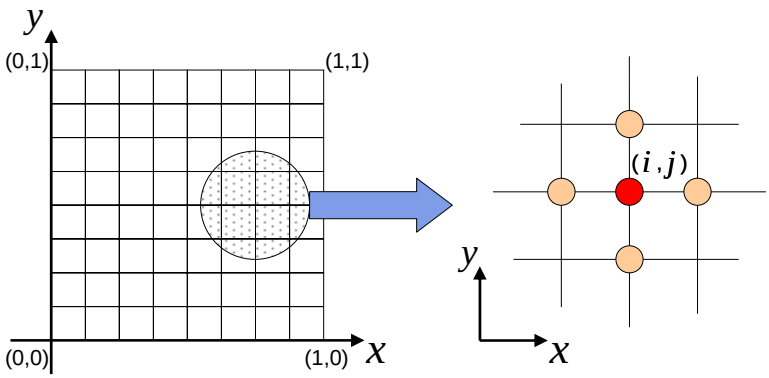

図 11：正方領域と SOR 法で用いる頂点

$u_{i,j}$ の近似値 $U_{i,j}$ を次の方程式の解として定義する。

$$U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j} = h^2 f_{i,j} \qquad \cdots (8)$$

上式を解くために SOR 法を用いる。SOR 法は連立 1 次方程式の解法のひとつで、ガウス・ガウデル法の発展形の解法である。この SOR 法を用いて、連立 1 次方程式の並列化を行った。SOR 法は図 11 のように、隣り合った頂点の値を用いるため、基本的に依存性が大きく、並列化は難しい。しかし、図 12 のように各頂点に色づけし、各色の頂点ごとに計算を行う Red-Black SOR 法というアルゴリズムを用いることで依存性を解消することができる。

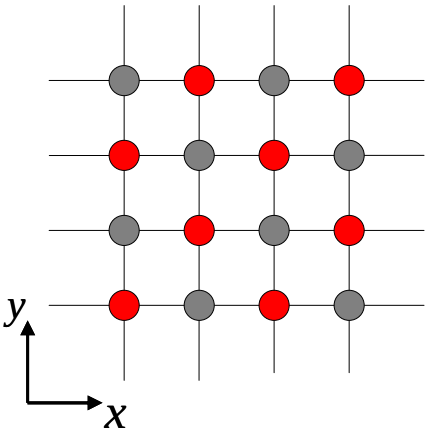

図 12：Red-Black SOR 法

Red-Black SOR 法により、赤色に定義した点と、黒色に定義した点は、同一色での処理では独立して処理できるためプロセッサファームアルゴリズムにより並列処理させる。

5.1.2 実行と評価

表 7 に Condor-G と本タスクスケジューラにおけるタスク投入までの時間を示す。

表 7：タスク投入までの時間

Condor-G	13.29
本スケジューラ	3.12

（単位：秒）

Condor-G では、Condor-G を利用したタスク投入のコマンドを入力した後、バックグランドで実行される。本タスクスケジューラでは、Condor-G でのタスク投入までの時間より短縮することができた。Condor-G では Condor-G 内でのスケジューラがホスト割り当ての処理を、Central Manager からの応答を待ち、その後行うため時間がかかっていると思われる。

次より、プログラムの実行評価を行う。実行条件として格子数を 800×800 で、ホスト数を 1 台、2 台、4 台で実行する。2 台の時 1 台、4 台の時 2 台ビジーホストを選択した場合と、本タスクスケジューラを使用する場合の実行結果を表 8 に示す。

ビジーホストを選択した条件ではホスト選択に、負荷のかかっている Raptor05 と Raptor07 を選択しているため、1 台と比べ 2 台では遅くなる結果となった。4 台では 1 台実行と相変わらない処理時間となっている。しかし、本タスクスケジューラを使用した場合、最適なホストを選択しているため 2 台で約 1.8 倍、4 台で約 2.6 倍の速度向上が得られた。SOR プログラムの特性から、ホスト数を増やすほどネットワークの使用率が増えるため 4 台での速度向上比約 2.6 倍というのは妥当な値だと考える。

表 8：Red-Black SOR プログラムの実行結果

	ビジーホストを選択	タスクスケジューラを使用
1 Host	**1050.1**	**1050.1**
2 Hosts	1308.9	**614.1**
4 Hosts	899.7	**403.5**

単位：秒

4 ホスト実行時にタスクスケジューラで得られたロードアベレージとレスポンスタイムを、ランク順に表示した結果を表 9 に示す。

表 9：ホスト 9 台のロードアベレージとレスポンスタイム

Ranking1:	Hostname:	raptor01	load averge:	2639984.781	delay time:	0.000961sec
Ranking2:	Hostname:	raptor00	load averge:	943826.039	delay time:	0.000973sec
Ranking3:	Hostname:	raptor06	load averge:	1176389.896	delay time:	0.000991sec
Ranking4:	Hostname:	raptor04	load averge:	1161437.018	delay time:	0.000989sec
Ranking5:	Hostname:	raptor02	load averge:	1140240.628	delay time:	0.000978sec
Ranking6:	Hostname:	raptor03	load averge:	1139886.945	delay time:	0.000989sec
Ranking7:	Hostname:	tyrano	load averge:	475890.688	delay time:	0.000594sec
Ranking8:	Hostname:	raptor07	load averge:	910806.217	delay time:	0.000984sec
Ranking9:	Hostname:	raptor05	load averge:	907133.254	delay time:	0.000993sec

以上の 9 ホストの中から上位 4 位までのホストを利用し実行したことになる。ランク 7 位の Tyrano ではロードアベレージが、ランク 8 位の Raptor07 のロードアベレージよりも低いにも関わらず、上位にランクされているのは、本タスクスケジューラがネットワークのトラフィック、遅延を考慮しているため、レスポンスタイムが早い方が上位になっていることがわかる。

5.2 NAS Parallel Benchmark による評価

5.2.1 NAS Parallel Benchmark の概要

NAS Parallel Benchmark は、米国の NASA Ames Research Center で作成されたベンチマークである。熱流体関連の数値計算プログラムであり分散メモリ型並列システムを対象とした権威あるベンチマークである。現在は version3.0 が最新となっており、MPI のほかに、OpenMP や HPF 用のソースコードも用意されている。このベンチマークは 8 つのプログラムで構成されており、EP、CG、MG、FT、IS の 5 つが kernel benchmark、残る LU、SP、BT が CFD(Computational Fluid Dynamics) application benchmark となっている。その中で、EP、CG、MG、SP、BT の 5 つを用いて評価を行う。この 5 つのプログラムの説明は以下のとおりである。

- Embarrassingly Parallel（EP）benchmark

  EP は、モンテカルロ法などで用いられる乱数発生プログラムである。基本的な計算過程は、乱数の種を求める関数と乱数列を求める関数を呼び出して、その結果得られた乱数列から乱数のペアを求め、それがどの領域に属するかを集計する。これらの処理を行うループが並列化の対象となっており、最後の集計に関する処理のところで、MPI_Reduce による通信が発生する。これ以外の部分は各ループの繰り返しごとに独立した処理を行っているので、並列化の効果が大きく現れるプログラムである。

- Conjugate Gradient（CG）benchmark

  CG は、正値対称な疎行列の最小固有値を共役勾配法によって近似的に解くプログラム

である。このベンチマークでは一定の通信が絶えず行われるため、ネットワーク性能がそのまま結果に反映されるという特徴を持つ。

- Multigrid（MG）benchmark

MG は、3 次元ポアソン方程式を、簡略化したマルチグリッド法で解くプログラムである。このベンチマークは、プロセス数が増加するにつれ、プロセス間の通信のメッセージサイズは小さくなる一方、通信回数が増える、という特性を持つ。

- Scalar Pent diagonal simulated CFD application （SP）benchmark

SP は、非優位対角なスカラ 5 重対角方程式を解くプログラムである。このベンチマークは、メッセージサイズの大きな通信が少ない回数だけ行われる。

- Block Tridiagonal simulated CFD application （BT）benchmark

BT は、非優位対角な 5×5 のブロックサイズを持つ 3 重対角方程式を解くプログラムである。基本部分は、3 次元 ADI（Alternative Direction Implicit）法が用いられている。演算量に対する通信の割合が異なる以外は SP と同じである。

5.2.2 実行と評価

ベンチマークの使用時には「クラス」を指定することによって、プログラム規模（配列のサイズや演算の反復回数）の変更が可能である。本評価には、A、B２つのクラスを用いる。実行台数は、EP、CG、MG については 1 台、2 台、4 台で実行し、SP、BT については 1 台と 4 台で実行し比較評価する。

表 10：EP ベンチマークの実行結果

	ビジーホストを選択		タスクスケジューラを使用	
Class	A	B	A	B
1 Host	**86.16**	**344.83**	**86.16**	**344.83**
2 Hosts	**83.32**	**333.6**	**43.72**	**178.86**
4 Hosts	**41.55**	**166.6**	**22.12**	**86.57**

単位：秒

表 10 に EP ベンチマークの実行結果を示す。EP ベンチマークは各プロセスが独立して乱数発生させることから、ホストの計算資源の利用状況により速度向上が変化する。ビジーホストを選択した場合では負荷ホスト選択による速度向上飽和が発生しているのに対し、

スケジューラは 4 台では 4 倍以上の速度向上を出している。4 倍以上の速度向上を得られたのは、ホスト選択にハード構成が Dual CPU タイプである Tyranno を選択したためである。

表 11：CG ベンチマークの実行結果

	ビジーホストを選択		タスクスケジューラを使用	
Class	A	B	A	B
1 Host	**6.76**	**594.58**	**6.76**	**594.58**
2 Hosts	**9.82**	**407.06**	**5.01**	**223.92**
4 Hosts	**8.34**	**243.19**	**1.86**	**73.23**

単位：秒

表 11 に CG ベンチマークの実行結果を示す。CG ベンチマークは、ネットワーク性能がそのまま結果に反映されるという特性を持つ。本実験では、ネットワーク性能は 1000M であり、ネットワークに負荷をかけていないためビジーホストを選択した場合のクラス A 以外ではホスト数に見合った速度向上が得られた。ビジーホストを選択した場合のクラス A では、負荷状態での計算処理であるのと、計算範囲が小さいく、ロードバランスが無いために、速度が得られなかったのだと考える。本タスクスケジューラを用いた場合における速度向上は 4 台実行時に 8 倍以上の結果となっている。本実験条件が PC クラスタとあるサーバという構成となっており、その中で各ホストのネットワーク距離が極めて近いホストを選択したためこのような速度向上が得られたと考えられる。

表 12：MG ベンチマークの実行結果

	ビジーホストを選択		タスクスケジューラを使用	
Class	A	B	A	B
1 Host	**6.91**	**63.35**	**6.91**	**63.35**
2 Hosts	**8.83**	**41.36**	**6.51**	**32.33**
4 Hosts	**7.57**	**34.23**	**5.34**	**20.84**

単位：秒

表 12 に MG ベンチマークの実行結果を示す。MG ベンチマークは、プロセス数と通信回数が比例するという特性を持ったベンチマークである。この特徴が各クラス A のときに現れた。クラス A のデータ範囲が小さく、台数を増やしてもデータ分割とネットワークの使用率が見合っていないため速度向上が得られなかったと思われる。ビジーホストを選択

した場合に対して、タスクスケジューラは 4 台実行では約 2.9 倍の速度向上が得られた。

表 13：SP ベンチマークの実行結果

	ビジーホストを選択		タスクスケジューラを使用	
Class	A	B	A	B
1 Host	**273.85**	**965.29**	**273.85**	**965.29**
4 Hosts	**208.72**	**641.40**	**138.36**	321.19

単位：秒

表 13 に SP ベンチマークの実行結果を示す。SP ベンチマークは、MG ベンチマークよりも多く通信を行う特性を持つ。ビジーホストを選択した場合では、ホスト内での負荷とベンチマークの特性であるネットワーク通信負荷により速度向上が得ることができなかった。しかし、タスクスケジューラを用いた場合、ネットワーク通信の負荷のみになるためクラス A ではホスト 4 台で約 2 倍、クラス B では約 3 倍の速度向上が得られた。

表 14：BT ベンチマークの実行結果

	ビジーホストを選択		タスクスケジューラを使用	
Class	A	B	A	B
1 Host	**316.83**	**658.22**	**316.83**	**658.22**
4 Hosts	**254.2**	**993.1**	**82.31**	**336.48**

単位：秒

表 14 に BT ベンチマークの実行結果を示す。BT ベンチマークは SP よりも通信回数が少ないという特性を持つ。タスクスケジューラのクラス A では約 3.8 倍の速度向上が得られたが、クラス B では約 1.9 倍の速度向上となった。通信回数が少ないが送受信するデータが大きくなるのではないかと考える。Globus＋MPICH-G2 とタスクスケジューラと比較すると、クラス B の 4 台実行では速度向上が改善されている。

5.2.3 評価

本実験では、2 台のホストに負荷をかけた状態で実験を行った。本タスクスケジューラで Condor-G のタスク投入までの時間より短縮することができた。実装したタスクスケジューラは、ロードアベレージとレスポンスタイムを元に 9 台のホストから 2 台、4 台の最適なホストの選択ができた。特に EP ベンチマークの実験では、9 台のホストの中で、Dual CPU ホストを選択することで、4 台のときに 4 倍以上の速度向上が得ることができた。また、グ

リッドへタスクを投入するときに用いていたコマンド 5 つを、シェルスクリプトを用いて 1 つにまとめ、シングルサインオンにおけるパスフレーズのみでよくなり、グリッドへのタスク投入がし易くなったと言える。しかし、レスポンスタイムはクライアントとサーバ間での時間でしかない。クライアントとサーバ間のレスポンスタイムを計測したのは、ネットワーク使用時における条件と、ホストの距離を見るためであった。実際の並列アプリケーションの実行には、グリッドホスト間の実行であるため、クライアントとサーバ間だけでなく、ホスト間のレスポンスタイムも重要であると考察できた。今回は PC クラスタ内のホストを用いたため、ネットワークに問題が出なかった。

今後の課題として、ネットワークトラフィックを考慮し、グリッドホスト間のレスポンスタイムをどのタイミングで計測するか、また、どのように計測するか検討する必要がある。

6. おわりに

本論文では、大学内で限られたネットワークでのグリッドコンピューティングとグリッドに関する研究を行うキャンパスグリッドプロジェクトについて検討しその内容を述べ、Globus Toolkit を用いてキャンパスグリッドの計画における初期段階の研究室内でのグリッド環境構築を行った。本研究室で構築した PC クラスタ 2 組を統合し、グリッド環境を構築することができた。

また、ヘテロジニアス環境でのタスクスケジューリング法を提案し、タスクスケジューラを実装した。タスクスケジューラは、ホスト内でロードアベレージを取得し、ネットワークトラフィックを抑えるため、情報を各ホストに持たせ情報の分散管理を行う。また、ロードアベレージとネットワーク遅延を元にホストにランク付けし、最適なホストの探索を行う。研究室内のグリッド上でタスクスケジューラの評価を、Red-Black SOR プログラム、NAS Parallel Benchmark を用いて行った。ホストの選択では、ロードアベレージとレスポンスタイムを基にランクを付けることができた。また、並列プログラムの実行ではあるホスト 2 台に負荷をかけ、全ホスト 9 台のうちから 4 台を選択し、タスクスケジューラを用いて 4 台を選択する場合と、過負荷状態にある 2 台を含めた 4 台で実行する場合と比較した。タスクスケジューラを用いた場合の速度向上の最大は Red-Black SOR プログラムでは約 2.6 倍、EP ベンチマークでは約 4.2 倍、CG ベンチマークでは約 8.1 倍、MG ベンチマークでは約 2.9 倍、SP ベンチマークでは約 2.1 倍、BT ベンチマークでは約 3.8 倍の結果を得ることができた。

今後の展開として、1 つ目に、現在までに構築した、研究室内でのグリッド環境ではキャンパスグリッドとして称することができない環境である。グリッドに関する研究を行う場合、ネットワークに関する検討を行う必要があるため、研究室内でのグリッド環境を他研究室の PC クラスタ、ホスト接続しネットワークに距離を持つキャンパスグリッドを構築することである。

2 つ目に、本論文で評価したタスクスケジューラは、Raptor クラスタを用いた評価であった。しかし、ネットワーク距離が近いためグリッドでの評価としては低い。よって本研究室の Goose クラスタを加えた評価と、ネットワークに距離を持たせたグリッド環境における評価を行う。また、ネットワークに負荷をかけた評価を行う必要がある。

3 つ目に、タスクスケジューラのレスポンスタイムは、クライアントとサーバ間の時間だけである。実際に並列プログラムを実行した場合には、選択したホスト間での通信を行うため、ホスト間のレスポンスタイムを計測することも重要だと考察できた。本研究ではネットワークのトラフィックと遅延を考慮しているため、どのようにホスト間のレスポンスタイムを計測するか検討する必要がある。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導をいただきました山崎勝弘教授、小柳滋教授に深く感謝いたします。また、キャンパスグリッドプロジェクトのメンバーである渡部氏、Jia 氏、本研究にあたり、励ましの言葉や様々な貴重なご意見を頂き、また質疑に答えていただくなどした本研究室の皆様に心より感謝いたします。

参考文献

[1] The Globus Alliance http://www.globus.org/

[2] Globus Simple CA Package
http://www.globus.org/security/simple-ca.html

[3] Ian Foster,Carl Kesselman,Jeffrey M. Nick and Steven Tuecke:The Physiology of the Grid –An open Grid Service Architecture for Distributed Systems Integration: International J. Supercomputer Applications, 15(3), 2001

[4] 関口智嗣,水田秀行 編グリッドコンピューティング:情報処理学会誌 Vol.44 No.6, pp.573-614, June 2003

[5] 日本アイ・ビー・エム システムズ･エンジニアリング株式会社 著:”グリッド･コンピューティングとは何か Globus Toolkitではじめるグリッドの基礎”,ソフトバンク パブリッシング,2004

[6] 産業技術総合研究所 グリッド研究センター 編:”グリッド ―情報社会の未来を紡ぐ―”,丸善,2004

[7] MPICH-G2 http://www3.niu.edu/mpi/

[8] Condor http://www.cs.wisc.edu/condor/

[9] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco,California,August 7-9,2001

[10] The NAS Parallel Benchmarks http://www.nas.nasa.gov/Software/NPB/

[11] 小阪隆浩:グローバルコンピューティング―Condor と Globus Toolkit の紹介―:PC クラスタ超入門 2000,2000

[12] 伊藤光裕,小出洋:グローバルコンピューティングに関する研究～Globus Toolkit の導入～:九州工業大学研究報告 No.75 pp.13-19,2003

[13] 石川裕,松田元彦,工藤知宏,手塚宏史,関口智嗣:GridMPI－通信遅延を考慮したMPI通信ライブラリの設計:情報処理学会ハイパフォーマンスコンピューティング研究会,Vol.95,No.17,pp.95-100,2003

[14] 松岡明香,松岡洋一:Grid におけるネットワーク負荷予測:情報処理学会ハイパフォーマンスコンピューティング研究会,Vol89,No.5,pp.25-30,2002

[15] 白勢健一郎,小川広高,中田良夫,松岡聡:グリッドコンピューティングにおけるモニタリングシステムの自律的構成:情報処理学会ハイパフォーマンスコンピューティング研究会,Vol.95,No.16,pp.89-94,2003

[16] 竹房 あつ子,松岡 聡:Grid 計算環境におけるデッドラインスケジューリング手法の性能:情報処理学会･電気通信処理学会 並列処理シンポジウム JSPP2001 論文集,pp.263-270, 2001

[17] 竹房 あつ子, 建部 修見, 松岡 聡, 森田 洋平:Grid Datafarm におけるスケジューリング・複製手法の性能評価:情報処理学会研究報告 SWoPP2002, pp.137-142,2002
[18] 竹房 あつ子, 建部 修見, 松岡 聡, 森田 洋平:Grid Datafarm におけるスケジューリング･複製手法の性能評価:情報処理学会･電気通信処理学会 SACSIS2003 シンポジウム 論文集, pp.121-128,2003
[19] 廣安知之:HPC 最新事情 第 11 回ビジネスグリッドを牽引する「Globus Toolkit」: COMPUTERWORLD,IDC Japan,11 号,2004
[20] 小橋博:第 2 特集 今日から始めるグリッドコンピューティング: UNIX USER,SOFTBANK Publishing,10 月号,2003
[21] 木ノ下稔:グリッドコンピューティングの調査と検討: 立命館大学理工学部情報学科卒業論文,2003

付録 A

グリッド環境の構築

以降、コマンド実行の説明には以下のように表記する。

user:griduser

```
$ /bin/date
2005 年　1 月　1 日 土曜日 00:00:00 JST
```

上記のように$の場合は一般ユーザ、”#”の場合は root ユーザを表す。

サーバへの Globus Toolkit のインストール [1] [6] [20]

(1)　Globus に必要なソフトウェア

Globus Toolkit をインストールためには、gcc、make、perl がインストールされている必要がある。

Globus Toolkit のインストール時、gcc を利用するが gcc のバージョンが 3.2 では正常にインストールされない。そのため gcc をアップグレードするか、もしくはダウングレードする。また Globus のインストールには Perl5.005 以上が必要となる。

(2)　Globus ユーザの作成

Globus では globus というユーザを作成し、インストールはすべてそのユーザで行うことを推奨している。もちろん root や個人ユーザでもインストールは可能だが、これはセキュリティの問題となるためセキュリティ機能が高い Globus では避ける。

(3)　環境変数

Globus のインストールには GLOBUS_LOCATION、GPT_LOCATION という環境変数が必要となる。GLOBUS_LOCATION は指定したフォルダに Globus を構築・インストールすることができる。GPT_LOCATION は Globus のインストールと Globus ツールとを分けてインストールするために設定する。我々は以下のようにそれぞれの環境変数を設定する。以下のような環境変数を、/etc/profile.d/フォルダ内に新たに作成する globus{.sh | .csh}にそれぞれに記入する。

/etc/profile.d/globus.sh

```
export GLOBUS_LOCATION=/opt/globus
export GPT_LOCATION=/opt/gpt
PATH=$PATH:$GLOBUS_LOCATION/bin:$GLOBUS_LOCATION/sbin:$GPT_LOCATION/sbin
export PATH
```

/etc/profile.d/globus.csh

```
setenv GLOBUS_LOCATION /opt/globus
setenv GPT_LOCATION /opt/gpt
set path=($path $GLOBUS_LOCATION/bin
                $GLOBUS_LOCATION/sbin $GPT_LOCATION/sbin)
```

以上の設定を有効にするには、一度ログインし直す、もしくは、source /etc/profile を実行する。

(4) インストールディレクトリ、Globus ツールディレクトリ

先ほど環境設定で指定したディレクトリを作成する。

user:root

```
mkdir -p /opt/{globus,gpt}
chown globus.globus /opt/{globus,gpt}
```

以上で Globus をインストールするための環境が整った。続いて GPT のインストールを行う。

The Globus Allience の以下のダウンロードページより必要なパッケージを入手する。

http://www.globus.org/gt2.4/download.html

(5) GPT のインストール

Globus Allience のダウンロードページから gpt-3.0.1-src.tar.gz をダウンロードする。次に以下のコマンドを実行する。

user:globus

```
$ gzip -dc gpt-3.0.1-src.tar.gz | tar xf -
...
$ cd gpt-3.0.1
$ ./build_gpt
```

GPT は Perl を内部で呼び出しているため PATH を設定してやる必要があるが通常 Linux では Perl は/usr/bin/perl にインストールされており、デフォルトで GPT のインストールが行えるが、それ以外の場所に Perl がインストールされている場合は

./build_gpt --with-perl=*Perl PATH*

でインストールができる

(6) bundle のインストール

bundle を Globus Allience からダウンロードする。Bundle は Date Management、

Information Services、Date Management の 3 つあり、それぞれには Client、Server、SDK と 3 つに分けて配布されている。これらはバイナリとソースの 2 種類あり、Fedora Core、Redhat Linux 9 の OS ではソースのインストールを推奨しており、それに従い、ソースインストールを行った。

Bundle のインストールコマンドは次のように行う。

user:globus

```
$ GPT_LOCATION/sbin/gpt-build [bundle-name] [flavors]
```

表：Bundle と Flavors

Bundle	Flavors
Data Management Client	gcc32dbg
Data Management Server	gcc32dbg
Data Management SDK	gcc32dbg
Information Services Client	gcc32dbgpthr
Information Services Server	gcc32dbgpthr
Information Services SDK	gcc32dbgpthr
Resource Management Client	gcc32dbg
Resource Management Server	gcc32dbg
Resource Management SDK	gcc32dbg

実際のコマンドは以下のようになる。

user:globus

```
$ gpt-build globus-data-management-client-2.4.3-src_bundle.tar.gz
gcc32dbg
$ gpt-build
globus-data-management-server-2.4.3-src_bundle.tar.gz
gcc32dbg
$ gpt-build
globus-data-management-sdk-2.4.3-src_bundle.tar.gz gcc32dbg
$ gpt-build
globus-information-services-client-2.4.3-src_bundle.tar.gz
gcc32dbgpthr
$ gpt-build
globus-information-services-server-2.4.3-src_bundle.tar.gz
gcc32dbgpthr
```

```
$ gpt-build
globus-information-services-sdk-2.4.3-src_bundle.tar.gz
gcc32dbgpthr
$ gpt-build globus-resource-management-client-2.4.3-src_bundle.tar.gz
gcc32dbg
$ gpt-build
globus-resource-management-server-2.4.3-src_bundle.tar.gz
gcc32dbg
$ gpt-build
globus-resource-management-sdk-2.4.3-src_bundle.tar.gz gcc32dbg
```

全 Bundle のインストール完了後、次のコマンドを実行し、Globus Toolkit をセットアップする。

user:globus

```
$ gpt-postinstall
```

(7) GSI のセットアップ

GSI の設定を行う。なお、デフォルトでは The Globus Allience の認証局を使用するように設定されている。

user:root

```
# $GLOBUS_LOCATION/setup/globus/setup-gsi
This script will overwrite the file --
     /etc/grid-security/certificates//grid-security.conf.42864e48
Do you wish to continue (y/n) [y] : y  ･･･y を選択

(1) Base DN for user certificates
         [ ou=hpc.cs.ritsumei.ac.jp, o=Globus, o=Grid ]
(2) Base DN for host certificates
         [ o=Globus, o=Grid ]

q  ･･･q を選択
```

この時点で GSI 関連のファイルが置かれる/etc/grid-security ディレクトリ以下が作成される。

(8) SimpleCA による認証局の構築

- SimpleCA のインストール

以下のコマンドと入力によって、SimpleCA のインストールを行う。

user:globus

```
$ gpt-build globus_simple_ca_bundle-latest.tar.gz gcc32dbg
$ gpt-postinstall

cn=Globus Simple CA, ou=simpleCA-tyrano.hpc.cs.ritsumei.ac.jp,
 ou=GlobusTest, o=Grid
Do you want to keep this as the CA subject (y/n) [y]:y  ･･･y を選択
Enter the email of the CA (this is the email where certificate
requests will be sent to be signed by the CA):  ･･･改行を入力
[default: 5 years(1825 days)]:  ･･･改行を入力
Enter PEM pass phrase:
****  ･･･認証局のパスフレーズを入力
```

ここで設定したパスフレーズは、認証書への署名するときに使用するので覚えておくこと。

- セキュリティの設定

SimpleCA の GSI 設定を追加する。ここで実行するコマンドは、(1)のインストール画面に表示されたものである。

user:root

```
# /opt/globus/setup/globus_simple_ca_78387d5e_setup/setup-gsi

This script will overwrite the file --
     /etc/grid-security/certificates//grid-security.conf.78387d5e
Do you wish to continue (y/n) [y] : y  ･･･y を選択

 (1) Base DN for user certificates
[ou=hpc.cs.ritsumei.ac.jp,ou=simpleCA-tyrano.hpc.cs.ritsumei.ac.jp,
ou=GlobusTest, o=Grid ]
(2) Base DN for host certificates
 [ ou=simpleCA-tyrano.hpc.cs.ritsumei.ac.jp, ou=GlobusTest, o=Grid ]

q  ･･･q を選択
```

- デフォルト認証局の変更

デフォルトの状態では The Globus Allience の認証局を使用するので、SimpleCA の認証局を使用するように設定する。

user:root

```
# grid-default-ca
The available CA configurations installed on this host are:

1) 42864e48 -  /C=US/O=Globus/CN=Globus Certification Authority
2) 78387d5e -  /O=Grid/OU=GlobusTest/OU=simpleCA-XXXXX.hpc.cs.ritsumei.
ac.jp/CN=Globus Simple CA

The default CA is:  42864e48

Enter the index number of the CA to set as the default: 2
```

ここでは、インストールした SimpleCA の表示が 2 になっているので、2 を選択する。

以上で SimpleCA の構築が完了した。

(9) インストールチェック

インストールが正常にされているか、ベリファイチェックを行う。ここでエラーが表示されないことを確認する。

user:globus

```
$ gpt-verify
Verifying run-time dependencies...

Verifying setup dependencies...

Verifying setup packages...

The collection of packages in /opt/globus appear to be coherent.
```

証明書の作成と CA での署名

- ユーザ環境設定

Globus Toolkit のコマンドを実行する際には、事前に globus-user-env.sh シェルを実行

しておく必要がある。この時点で/etc/profile ファイルに以下の行を追加して、再度設定ファイルを読み込む。

user:root　　file:/etc/profile

```
...
source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

user:all

```
$ source /etc/profile
```

- 証明書の作成と CA での署名

証明書を作成する grid-cert-request コマンドを実行すると、３つのファイルが作成される。このうち証明書要求ファイル（*_request.pem）を認証局に渡して署名したものが正式な証明書となる。ユーザ認証書の場合は「usercert_request.pem」ファイルが要求ファイルになる。

秘密鍵ファイルについては他ユーザにコピーされたりしないように扱いに注意する。

■ ユーザ証明書の作成

ユーザ証明書の作成時には識別子（CN）を指定する必要がある。CN には実際の使用者やホスト上のユーザの名前等のユニークな名称を指定する。この値と GSI の設定時に表示されたユーザ証明書の BaseDN とを組み合わせたものがユーザ固有の DN となる。

user:griduser

```
$ grid-cert-request -cn “hayashi”
Enter PEM pass phrase：
*****　・・・ユーザ証明書のパスフレーズを入力
```

ユーザ証明書関連ファイル$HOME/.globus の下に作成される。

■ ホスト証明書の作成

user:root

```
# grid-cert-request -host tyrano.hpc.cs.ritsumei.ac.jp
```

ホスト証明書の関連ファイルは/etc/grid-security の下に作成される。

■ LDAP 証明書の作成

LDAP 証明書は MDS で使用されるものである。

user:root

```
# grid-cert-request -service ldap -host tyrano.hpc.cs.ritsumei.ac.jp
```

LDAP 証明書の関連ファイルは/etc/grid-security/ldap の下に作成される。

■ 証明書への署名

証明書は認証局によって署名してもらう必要がある。SimpleCA で認証局 tyranno に作成しているので、tyranno の各証明書は自分自身で行う。

ユーザ証明書への署名

user:griduser

```
$ cd /home/griduser/.globus
$ cp usercert_request.pem /tmp
```

user:globus

```
$ cd /tmp
$ grid-ca-sign -in usercert_request.pem -out usercert.pem
Enter password for the CA key:
*****   ・・・ユーザ証明書のパスフレーズを入力
```

user:griduser

```
$ cp /tmp/usercert.pem /home/griduser/.globus
(0byte のファイルに上書き)
$ chwon 644 / home/griduser/.globus/usercert.pem
```

ホスト証明書への署名

user:root

```
# cd /etc/grid-security
# cp hostcert_request.pem /tmp
```

user:globus

```
$ cd /tmp
$ grid-ca-sign -in hostcert_request.pem -out hostcert.pem
Enter password for the CA key:
*****   ・・・ユーザ証明書のパスフレーズを入力
```

user:root

```
# cp /tmp/hostcert.pem /etc/grid-security
(0byte のファイルに上書き)
# chwon 644 /etc/grid-security/hostcert.pem
```

LDAP 証明書への署名

user:root

```
# cd /etc/grid-security/ldap
# cp ldapcert_request.pem /tmp
```

user:globus

```
$ cd /tmp
$ grid-ca-sign -in ldapcert_request.pem -out ldapcert.pem
Enter password for the CA key:
*****    ･･･ユーザ証明書のパスフレーズを入力
```

user:root

```
# cp /tmp/ldapcert.pem /etc/grid-security/ldap
(0byte のファイルに上書き)
# chwon 644 /etc/grid-security/ldap/ldapcert.pem
```

一度署名したことのある証明書に対して署名を実行してエラーになる場合は、grid-ca-sign コマンドに-force オプションを付けて実行することで、エラーを回避することができる。

例：# grid-cert-request -host tyrano.hpc.cs.ritsumei.ac.jp -force

システムテストと設定 [1] [6]

- GRAM でのジョブ実行テスト

GRAM は通常、他ホストからの要求を受け付けるため xinetd で起動するサーバ（ゲートキーパー）として設定されるが、最初は一時的にパーソナルゲートキーパーと呼ばれるものを起動して、実行のテストを行う。

i. パーソナルゲートキーパーへのジョブ投入

user:griduser

```
$ grid-proxy-init -debug -verify  ･･･プロキシ証明書を作成
Enter GRID pass phrase for this identity:
***** ユーザ証明書のパスフレーズを入力(表示されない)

$ globus-personal-gatekeeper -start  ･･･パーソナルゲートキーパーを起動
GRAM contact:
  XXXXX.hpc.cs.ritsumei.ac.jp:33333:/O=Grid/OU=GlobusTest/OU=simpleCA-
XXXXX.hpc.cs.ritsumei.ac.jp/OU=hpc.cs.ritsumei.ac.jp/CN=Masaki Hayashi
                                ･･･接続用の文字列（=コンタクトストリング）

$ globus-job-run " XXXXX.hpc.cs.ritsumei.ac.jp:33333:/O=Grid/OU=Globus
Test/OU=simpleCA- XXXXX.hpc.cs.ritsumei.ac.jp/OU=hpc.cs.ritsumei.ac.jp/
CN=Masaki Hayashi" /bin/date
                ･･･コンタクトストリングを指定してジョブの実行の依頼を行う

Fri Dec 10 03:59:35 JST 2004

$ globus-personal-gatekeeper -killall  ･･･ゲートキーパーの終了
$ grid-proxy-destroy  ･･･プロキシ証明書を削除
```

ii.　サーバを起動する設定

OS の起動時に GRAM サーバ（ゲートキーパー）が自動起動する設定を行う。まず、/etc/service ファイルにポートの設定を追加する。

user:root

```
gsigatekeeper 2119/tcp
```

次に以下のファイルを/etc/xinetd ディレクトリに作成する。

file:/etc/xinetd.d/gsigatekeeper

```
service gsigatekeeper
{
        socket_type = stream
        protocol = tcp
        wait = no
        user = root
        env = LD_LIBRARY_PATH=/opt/globus/lib
        server = /opt/globus/globus-gatekeeper
        server_args = -conf /opt/globus/etc/globus-gatekeeper.conf
        disable = no
}
```

ゲートキーパーで接続を許可するユーザの DN を設定する。ユーザの DN は次のコマンドで確認できる。

user:griduser

```
$ grid-cert-info -subject
/O=Grid/OU=GlobusTest/OU=simpleCA-
XXXXX.hpc.cs.ritsumei.ac.jp/OU=hpc.cs.ritsumei.ac.jp/CN=Masaki Hayashi
```

次のコマンドでゲートキーパーに接続できるユーザをマップする。

user:root

```
# grid-mapfile-add-entry -dn "/O=Grid/OU=GlobusTest/OU=simpleCA- XXXXX.
hpc.cs.ritsumei.ac.jp/OU=hpc.cs.ritsumei.ac.jp/CN=Masaki Hayashi" -ln
hayashi
```

この設定は、/etc/grid-security/grid-mapfile ファイルに記述される。

以上の設定後、xinetd を再起動させる。

user:root

```
# /sbin/service xinetd restart
xinetd を停止中:                                           [  OK  ]
xinetd を起動中:                                           [  OK  ]
```

iii. ジョブの実行

次のコマンドを実行し、正しくジョブが実行されるか確認する。

user:griduser

```
$ grid-proxy-init  ･･･プロキシ証明書を作成
Enter GRID pass phrase for this identity:
***** ユーザ証明書のパスフレーズを入力(表示されない)

$ globus-job-run localhost /bin/date
Fri Dec 10 03:59:35 JST 2004

$ grid-proxy-destroy  ･･･プロキシ証明書を削除
```

- GridFTP でのジョブ実行テスト

OS 起動時に GridFTP サーバが自動起動する設定を行う。まず、/etc/service ファイルにポートを設定する。

user:root

```
gsiftp 2811/tcp
```

次に以下のファイルを/etc/xinetd.d ディレクトリに作成する。

file:/etc/xinetd.d/gsiftp

```
service gsiftp
{
        instances = 1000
        socket_type = stream
        wait = no
        user = root
        env = LD_LIBRARY_PATH=/opt/globus/lib
        server = /opt/globus/sbin/in.ftpd
        server_args = -l -a -G /opt/globus
        log_on_seccess += DURATION USERID
        log_on_failure += USERID
        nice = 10
        disable = no
}
```

以上の設定後、xinetd を再起動させる。

user:root

```
# /sbin/service xinetd restart
xinetd を停止中:                                           [  OK  ]
xinetd を起動中:                                           [  OK  ]
```

最後に GridFTP の実行テストを行う。テスト用のファイルを用意して、ローカルマイン内でのファイルコピーを行う。

user:griduser

```
$ grid-proxy-init  ･･･プロキシ証明書を作成
$ echo "hello" > /tmp/test.txt  ･･･テスト用のファイルを作成
$ globus-url-copy gsiftp://XXXXX.hpc.cs.ritsumei.ac.jp/tmp/test.txt file:
///tmp/test2.txt
$ ls -l /tmp/test*.txt
$ grid-proxy-destroy  ･･･プロキシ証明書を削除
```

globus-url-copy テストにてエラーが起きた場合、ホスト名を”localhost”に変更して実行してみる。もしそれでもエラーが起きた場合は正常にインストールされていない。

異なるホスト間でのファイルコピーを行う場合は次のように指定する。

globus-url-copy gsiftp://＜hostname_1＞/＜source_filename＞ gsiftp://＜hostname_2＞/＜destination_filename＞

- MDS でのジョブ実行テスト

MDS のサーバを起動する。

user:root

```
# globus-mds start
Starting up Openldap 2.0 SLDAP server for the GRIS
```

自分自身の情報を表示する。

user:griduser

```
$ grid-info-search
SASL installing layers
dn: Mds-Host-hn=XXXXX.hpc.cs.ritsumei.ac.jp,Mds-Vo-name=local,o=grid
objectClass: MdsComputer
objectClass: MdsComputerTotal
objectClass: MdsCpu
...
```

検索条件を指定することができる。

例：CPU のクロック周波数を表示する

user:griduser

```
$ grid-info-search -LLL "(&(Mds-Device-Group-name=processors))" Mds-Cpu-
speedMHz

SASL/GSI-GSSAPI authentication started
SASL SSF: 56
SASL installing layers
dn: Mds-Device-Group-name=processors, Mds-Host-hn= XXXXX.hpc.cs.ritsumei
.ac.jp
 ,Mds-Vo-name=local,o=grid
Mds-Cpu-speedMHz: 2790
```

また、$GLOBUS_LOCATION/sbin/SXXgris という起動スクリプトが用意されており、Redhat Linux 系の場合はこれを/etc/rc.d/init.d にコピーしておき、システム起動時に自動で立ち上げる。

user:root

```
# cp $GLOBUS_LOCATION/sbin/SXXgris /etc/rc.d/init.d/gris
```

/etc/inittab でシステムの runlevel(「id:数字:initdefault:」の部分)をチェックした後、/etc/rc.d/rcX.d(X は runlevel)から S で始まるファイル名でシンボリックリンクを張る。例えば、テキストログインに設定している場合、runlevel は 3 であるので、次のようになる。

user:root

```
# ln -s /etc/rc.d/init.d/gris /etc/rc.d/rc3.d/S99gris
```

- 注意事項

Globus によるグリッド環境を再構築する際は、globus ユーザの.globus というフォルダと、/etc/grid-security フォルダをフォルダ名変更しバックアップすること。再構築する場合、両フォルダの設定を引継ぎセットアップされるため正常に動作しない。

クライアント群への Globus Toolkit のインストール

クライアントにインストールするパッケージはサーバ同様、gpt-3.0.1-src.tar.gz と全 Bundle そして、サーバで構築した SimpleCA パッケージである。各インストールはサーバ

へのインストールと同様に行うが、SimpleCA のインストールは次のように行う。

/opt/globus/setup/globus/setup-gsi を実行した後
user:globus

```
$ gpt-build .globus/simpleCA/globus_simple_ca_<Hash_Number>_setup-0.13.
tar.gz gcc32dbg

This script will overwrite the file --
     /etc/grid-security/certificates//grid-security.conf.78387d5e
Do you wish to continue (y/n) [y] : y  ･･･y を選択

(1) Base DN for user certificates
 [  ou=hpc.cs.ritsumei.ac.jp,  ou=simpleCA-tyrano.hpc.cs.ritsumei.ac.jp,
ou=GlobusTest, o=Grid ]
(2) Base DN for host certificates
 [ ou=simpleCA-tyrano.hpc.cs.ritsumei.ac.jp, ou=GlobusTest, o=Grid ]

q･･･q を選択
```

証明書の作成と CA での署名の作業は、証明書のリクエストファイルの作成、CA での署名作業はサーバでの作業と同様に行えばよいが、リクエストファイルや署名が行われた証明書ファイルの転送のやり取りはサーバでは同一マシン上で作業をしていたため、ファイルそのもののコピーで済んだが、クライアントの場合は、ネットワークを通じてファイルのやり取りをする必要がある。クライアントでの証明書の作成と CA での署名作業を Raptor00 を例に説明する。

■ ホスト証明書の作成

user:root

```
# grid-cert-request -host tyrano.hpc.cs.ritsumei.ac.jp
```

■ LDAP 証明書の作成

user:root

```
# grid-cert-request -service ldap -host tyrano.hpc.cs.ritsumei.ac.jp
```

LDAP 証明書の関連ファイルは/etc/grid-security/ldap の下に作成される。

■ 証明書への署名

ファイルのコピーを、ホスト間でセキュアなファイルの送信が行うことができる scp を用いて行う。

ホスト証明書への署名

user:root

```
# cd /etc/grid-security
# scp hostcert_request.pem tyrano:/tmp
```

user:globus

```
$ cd /tmp
$ grid-ca-sign -in hostcert_request.pem -out hostcert.pem
Enter password for the CA key:
*****   ・・・ユーザ証明書のパスフレーズを入力
```

user:root

```
# scp tyrano:/tmp/hostcert.pem /etc/grid-security
(0byte のファイルに上書き)
# chwon 644 /etc/grid-security/hostcert.pem
```

LDAP 証明書への署名

user:root

```
# cd /etc/grid-security/ldap
# scp ldapcert_request.pem tyrano:/tmp
```

user:globus

```
$ cd /tmp
$ grid-ca-sign -in ldapcert_request.pem -out ldapcert.pem
Enter password for the CA key:
*****   ・・・ユーザ証明書のパスフレーズを入力
```

user:root

```
# scp tyrano:/tmp/ldapcert.pem /etc/grid-security/ldap
(0byte のファイルに上書き)
# chwon 644 /etc/grid-security/ldap/ldapcert.pem
```

Globus サービスのシステムテストと設定は、サーバでのテストと同様に行う。

MPICH-G2 のインストール

- Globus テスト

MPICH-G2 をインストールする前に、Globus が正しくインストールされているか、Globus ベースの"Hello, World"プログラムを用いてテストを行う。

以下の手順に従ってテストを行う。

MPICH のホームページ上がっているテストプログラムをダウンロードする

・ hello.c

http://www.hpclab.niu.edu/mpi/DistributedFiles/hello.c

```
#include <globus_duroc_runtime.h>

int main(int argc, char **argv)
{

#if defined(GLOBUS_CALLBACK_GLOBAL_SPACE)
    globus_module_set_args(&argc, &argv);
#endif

    globus_module_activate(GLOBUS_DUROC_RUNTIME_MODULE);
    globus_duroc_runtime_barrier();
    globus_module_deactivate(GLOBUS_DUROC_RUNTIME_MODULE);

    printf("hello, world¥n");
}
```

・ Makefile

http://www.hpclab.niu.edu/mpi/DistributedFiles/Makefile.hello.v2.0

Makefile.hello.v2.0 を Makefile とリネームを行う。

```
RM = /bin/rm

include makefile_header

hello:
        $(GLOBUS_CC) $(GLOBUS_CFLAGS) $(GLOBUS_INCLUDES) -c
hello.c
        $(GLOBUS_LD) -o hello hello.o ¥
        $(GLOBUS_LDFLAGS) ¥
        $(GLOBUS_PKG_LIBS) ¥
        $(GLOBUS_LIBS)

clean:
        $(RM) -rf *.o hello
```

以上２つのファイルを用意したら、次のコマンドで makefile-header を作成する。

```
$ $GLOBUS_LOCATION/bin/globus-makefile-header -flavor=gcc32dbg ¥
globus_common globus_gram_client globus_io globus_data_conversion ¥
globus_duroc_runtime globus_duroc_bootstrap > makefile_header
```

次に hello.c をコンパイルを行う。

```
$ make hello
```

環境に合わせて RSL ファイルを作成する。

```
+ ( &(resourceManagerContact="XX.hpc.cs.ritsumei.ac.jp")
(count=2)
(label="subjob 0")
(environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
(LD_LIBRARY_PATH /opt/globus/lib/))
(directory=/home/hayashi)                    // プログラムのあるディレクトリ
(executable=/homes/hayashi/hello)            //プログラム名
)
```

Globus での動作を確認する。

```
$ globusrun -w -f hello.rsl
hello, world
hello, world
$
```

ここで何らかのエラーが出た場合は、RSL ファイルが間違っているか、もしくは Globus Toolkit が正しくインストールされていない。

このテストが正常に終了しない場合は、MPICH-G2 をインストールしない。

- MPICH-G2 のインストール

Globus Toolkit に MPICH-G2 をインストールする場合、MPICH version1.2.3 以降が必要である。MPICH のホームページから最新の MPICH 1.2.6(2005 年 2 月 15 日現在)をダウンロードする。インストール手順は以下のようになる。

(1) MPICH を解凍展開する。

$ gunzip -c mpich.tar.gz | tar xvf -

(2) MPICH を globus2 デバイス用に設定する。MPICH の configuration スクリプト多くのコマンドラインオプションが用意されている。globus2 デバイスの設定には -device= オプションと -arch= そして -prefix= オプションを用い、その他のオプションは使う必要はない。設定時には Globus flavor を指定する。

$ cd mpich-1.2.6

$ ./configure --with-device=globus2:-flavor=gcc32dbg -prefix=/opt/mpich-G2

アーキテクチャが 32bit または 64bit か識別可能な場合には明示的に -arch オプションを指定する。

(3) MPICH-G2 のビルド

$ make

(4) MPICH-G2 をインストールする。

$ su

make install

(5) 環境設定を行う。

bash,sh ユーザ

```
PATH=/opt/mpich-G2/:$PATH
export PATH
```

/etc/profile.d/globus.csh

```
set path=(/opt/mpich-G2/ $path)
```