[卒業論文]

Grid として利用するコンピュータ環境の分類

（指導教員） 吉村賢治　教授

電子情報工学科　吉村研究室

学籍番号 TL021279

西村亮馬

[2005 年度]

福岡大学　工学部

卒業論文

論文題目

Grid として利用するコンピューター環境の分類

指導教員　　吉村賢治　教授

提出日　2006 年 1 月 20 日

提出者

学科	電子情報工学科
学籍番号	TL021279
氏名	西村亮馬

2005 年度　　卒業論文概要					
論文題目　　Grid として利用するコンピューター環境の分類					
電子情報工学科	TL021279	氏名	西村亮馬	指導教員	吉村教授

GridComputing を実際に導入した時に考えられる、レスポンスの低下について、回線速度と処理速度から、レスポンスの維持･向上を図る。

まずは実際に GridComputing を構築するため、Grid ミドルウェアである GlobusToolKit をインストールする。導入した Grid ミドルウェアを動かした時、GlobusToolKit にはデフォルトでスケジューラが搭載されていない。処理を依頼する側（人、またはコンピュータ。以下管理ホスト）は、処理をする側（以下実行ホスト）の存在を認識しているが、現在の状態（CPU 利用状況、回線利用中の有無）を知りえない。実行ホストに意図的に負荷を与え、表記された能力ではなく、潜在能力を把握することで、実行ホストの環境の分類を行い、管理ホストに指標を与える。指標を得た管理ホストは、その指標を元に環境毎へ処理の依頼の分類を行う事ができる。それぞれの得意な処理体系を持つ環境群へ、適切な処理を代入して行く事で、より円滑な処理を行えるようになるであろう。

本研究では、GridComputing による円滑な処理を大前提とした、管理ホストへの指標を与える処理、つまり実行ホストを２つの面から観測することで、実行ホストの分類を行う。

目次

図目次

表目次

第 1 章

序論

本章では、はじめに本研究における背景について述べる。次に本論文で扱う GlobusToolKit2.4 における従来の環境とその問題点について説明する。最後に本研究の目的を述べる（参考文献[1]）。

1.1　研究の背景

GridComputing とは、様々な場所にある複数のコンピュータを LAN やインターネットなどのネットワークで結び、CPU パワーやデータストレージなどのコンピュータ資源を共有して、あたかも 1 つの高性能コンピュータのように利用できるようにする技術のことである。電力網を意味する「Power Grid」を語源としており、電気を供給するように、必要な時に必要なだけコンピュータパワーを提供することを目指している。

GridComputing の大きなメリットは「コスト」と「スケーラビリティ」である。1 台 1 台のコンピュータが高性能なものでなくても、複数台で協調して処理を行うことにより、スーパーコンピュータに勝るとも劣らない性能を発揮する。従来は高価なスーパーコンピュータで行っていた大量の計算を、安価な IA サーバーなどで構成されるグリッドで処理することにより、高い演算性能を低コストで利用することが可能になる。また、より高い性能を必要とする場合にはコンピュータを追加するだけで良いため、非常にスケーラビリティの高いシステムを構築できる。

機種の異なる複数のコンピュータを接続して資源を共有するには、各コンピュータ間で統一されたグリッド基盤ミドルウエアが必要となるため、GGF（Global Grid Forum）という非営利団体によって標準化が進められている。なお、現在デファクトスタンダードと見なされている Grid ミドルウエアは、Globus Alliance が開発を進めている「GlobusToolkit」である。

現在、コンピュータの高性能化が進み、サーバ、デスクトップ PC などは、本来の性能をフルに使わなくても日常生活の処理を行うことができ、コンピュータのリソースが利用されていない時間も多い。つまり、全体としてみればコンピュータの処理能力は余剰状態にある。しかし一方で、遺伝子解析や金融シミュレーションなど、より高い処理能力を必要としている分野も存在している。グリッドを利用することで余剰のコンピュータ資源を、高い処理能力を必要とする分野に配分することができるようになる。

GridComputing の定義として、アルゴンヌ国立研究所兼シカゴ大学のイアン・フォスター氏は、「集中管理されていない分散した資源のコーディネート」、「オープンスタンダードなプロトコルやインターフェースの利用」、「単純には得られない質の高いサービスの提供」の３つを挙げている。

広義にとらえれば、「SETI@home」のような PC を中心とした大規模な分散処理も GridComputing に含まれるが、狭義では、高速な計算サービスを提供する“コンピューティンググリッド”、大規模データ処理のための“データグリッド”、Web を利用した信頼性の高いサービスをビジネス用途向けに提供する“ビジネスグリッド”に分類される。しかし、グリッドは発展途上の技術であり、これらの分類も研究の進歩に合わせ、日々変化していると言える。

この GridComputing を使う立場から問題点を挙げ、そのうちの一つ、実行する環境がどのような状態なのかというのを中心に、二つの側面から実験を行う。得られた結果をそれぞれ比較し、優劣・順位を付けることで、処理を依頼する立場に指標を与える。本研究を進めて行くことで、GridComputing のレスポンスを向上させ、遠くは GridComputing や情報工学の発展に貢献する。

1.2　従来の環境とその問題点

従来の処理させる側（人、もしくはコンピュータ）が、処理する側（コンピュータ）の情報をどの程度認識しているのか考えた時、その情報は必ずしも有用ではないことに気がつく。この処理する側の情報は、ホスト名、IP アドレス、メーカーが表記している CPU、メモリ、通信速度が考えられる。ホスト名や IP アドレスは GridComputing を使う上で欠かせない情報である。しかし、ホスト名や IP アドレスを変えたところで、GridComputing の処理能力には関係してこない。では、CPU やメモリ、通信速度はどうであろうか。

CPU パワーの低いマシンと、CPU パワーの高いマシンに同じ処理をさせた場合、CPU パワーの高いマシンの方が先に結果を出せるということは、当然の話である。しかし、CPU パワーの高いマシンが、長期の処理を任されていた場合、果たして同じ結果が出るであろうか。同様の話が通信速度にも言える。

GridComputing を利用してより速く結果を得たい、を前提として考えると、処理させる側（人、コンピュータ）が処理する側（コンピュータ）の実際の潜在能力を知る事で、より速く結果を得るという条件を満たすであろう。したがって、本研究では実際の潜在能力を知る、いわゆる処理させるコンピュータ環境の指標を得るという部分を行う。

1.3　本研究の目的

GridComputing を使いより速く結果を得るという前提を満たすには、様々な条件をクリアーしなければならない。大きく分ければ、処理する内容と処理する側の能力、という二つに分けることができる。処理する内容は、例えば処理するステップは多いのか、処理がループしていないか、処理しなければならないデータが大容量なのか、などが考えられるが、本研究では処理する内容については触れない事とする。つまり、本研究は、処理する側に関する指標を得る、という点を考える。

GridComputing を紐解いてみると、インターネットや LAN で構築されたサーバ群やデスクトップ PC 群である。上記の前提を満たすには、処理する側となるサーバやデスクトップ PC の処理能力が高く、また LAN やインターネットの通信速度が高い、という条件を満たせばよい。

ただし、これは処理する内容を考慮されてはいない。処理する内容を考慮に入れるならば、処理する側がどのような環境なのかを処理させる側へと与え、処理させる側で処理する内容と照らし合わせた上で、処理が得意な環境へ依頼するのが最も賢いといえるであろう。したがって、本研究ではその環境の状態を測定し分類することで、処理させる側が処理する内容と照らし合わせるのに使う指標を与える機能を実装する。

ここで、処理する側、つまり処理の内容を受け取り、実際に処理を行って結果を返すという一連の動作を任されるコンピュータの事を、実行ホストと呼ぶことにする。また、実行ホストに処理を依頼する立場にある、処理させる側のことを、管理ホストと呼ぶことにする。

第 2 章

GlobusToolKit2.4 の概要とその問題点

本章では、はじめに GridComputing のデファクトスタンダードな Grid ミドルウェアである GlobusToolKit について述べる。次に本研究のもとになる GlobusToolKit に実装されている機能、及び問題点について述べ、最後に本研究において実装する機能について述べる（参考文献[2]）。

2.1　GlobusToolKit2.4

グリッドシステムの構築を容易にするためのツールキットであり、米国アルゴンヌ国立研究所と南カリフォルニア大学の共同で開発された。デファクトスタンダードとして広く普及しており、世界中のほとんどすべてのグリッドプロジェクトで採用されている。バージョンには二つが並存している。今回はバージョン 2.4 を使用した。

GlobusToolKit は科学技術計算を指向しており、比較的クローズドな研究者集団が資源を共有するために開発された。ひとつのグリッドではリソースの数もユーザの数もそれほど多くないことを前提としている。ゆえに不特定多数の実行ホストは想定していない。

Globus がインストールされていても、それを使用できるのは以下の条件を満たしたユーザだけである。

・そのホスト上にアカウントを持っている。

・資源が認める認証局から発光された証明書を持っている。

・システム管理者が明示的に許可している。

また Globus はボランティアコンピューティングシステムではない。よって SETI@Home などの P2P とは根本的に発想が異なる。他にもプログラムから計算資源を守る機能がなく、プログラムの権限は通常のユーザと同じである。これに対し P2P 系では実行条件を規定するのが一般的である。また、グリッドの計算資源内に含まれている他のユーザから、プログラム（のデータ）を守る仕掛けも存在しない。通信は暗号化されているが、メモリ上のデータやファイルを暗号化はされていない。

そして、Globus はグリッド計算システムではない。Globus はあくまでもグリッド計算システムを構築するためのツールキットであり、実態は低レベルのライブラリとコマンドの集合体である。Globus をインストールしたからといって何かが解決するわけではない。

さらに、Globus はセキュリティに対して、完全ではないといえる。それはファイアウォールとの親和性が低く、双方向にコネクションを張るので NAT を介した運用は不

可能に近いといえる。また、多数のユーザが使用する場合の管理コストが非常に高く、各ホストに使用するユーザを登録しなければならない。

2.2　GlobusToolKit2.4 に実装されている機能

シングルサインオン認証の GSI、資源情報の取得をする MDS、リモートサーバ上でのプロセス実行をする GRAM、データ転送を行う GridFTP の４つで構成されている。

本研究で使用するのは、主に GRAM と GridFTP なので、以下この二つについて説明をしておく。

A)　リモートサーバ上でのプロセス実行をする GRAM

リモートサーバ上でプログラムを安全に実行する。シングルサインオンや UNIX アカウントと連動した認可を実装している。GASS と連動したプログラムの転送と入出力のリダイレクトを行う。実行ファイルや引数ファイルを自動的にステージングする。GRAM は以下の三つで構成される。

(1)Gatekeeper・・・計算機に常駐し、クライアントからのジョブ要求を待つ。
主な動作は、リクエストを受け付け、ユーザの権限で JobManager を起動する。

(2)JobManager・・・キューイングシステムにジョブを投入し、管理する。
主な動作は、プロセスの状態を監視、管理と、キューイングシステムを抽象化することである。

(3)キューイングシステム・・・プロセスの実際の起動をつかさどる。デフォルトでは単純な fork。

GRAM と SSH の違いは、GSI によるシングルサインオンを実現しているところにある。さらに別のサーバに接続することが可能であり、ホストとユーザ名の対応をユーザが管理する必要がない。他にもサーバ側でローカルなスケジューラの使用が可能であり、リモートジョブの詳細な管理を行うことができる。しかし、クライアントとの間にストリームはできない。

B)　ホスト間のデータ転送をする GridFTP

第三者転送をサポートしている。既存のサーバとの相互運用性を持ちつつ、様々な方向性に拡張している。

2.3　GlobusToolKit2.4 の問題点

上記 2.2 節で述べている機能を使っていく上で、問題となってくるのは、GridComputing を使えば、管理ホストはネットワークの先にある実行ホストの環境を考えなくて良い。（1.1 研究の意義より）しかし、実際にはどの環境で処理させるのかを指定することは、管理ホストに一任されている。例えば、仮に実行ホストになんらかの負荷がかかっていた場合、結果を受け取るのに莫大な時間を要する可能性が出てくる。他にも使いたい実行ホストへと繋がる回線を他の所用で使用中だった場合、転送速度は著しく低下するであろう。結果として、管理ホストが実行ホストの潜在能力を知りえない限り、性能の低い環境で処理させ続けたり混雑している回線を使用することで、結果としてレスポンスの低下を招くことになるであろう。

2.4　問題点に対する解決方針

この問題を回避するには、管理ホストは二つの側面を知る必要がある。一つは投入する処理がどういったものなのか、ということである。処理の種類によって、向き・不向きな環境を取捨選択することができるであろう。もう一つは、実行ホストとそれを取り巻く環境である。そしてこの実行ホストとそれを取り巻く環境で重要となってくるのは、処理演算能力と通信速度である。この二つを知ることで、特定の実行ホストへ、より速く結果を得ることができる処理を投入することができるであろう。

以下、実行ホストとそれを取り巻く環境を中心に、実行ホストの分類を実測値をもとに試みる。始めに環境の分類方法について述べ、実測値の取得方法について説明していく。

2.4.1　環境の分類

環境の分類方法については、2.4.2 節・2.4.3 節で述べる方法で得られた実測値（秒）を参考に、順位付けを行うことで分類とする。実行ホストの環境が複数になれば、平均値をボーダーラインとして大小の比較を行う。ジョブを投入するのが人の場合、これは目安になるであろう。ジョブスケジューラ等コンピュータプログラムが利用する場合、実測値のデータと平均値との差が重みとして機能する。

2.4.2　実処理時間の取得

実処理時間の取得は、実行ホスト上で実際になんらかのプログラムを動作させ、それの処理にかかる時間を計測することで得る。

具体的には第３章実処理時間の取得方法で述べる。

2.4.3　実転送速度の取得

実転送速度の取得は、管理ホストから実行ホストへなんらかのデータを転送させ、転送し終わるまでにかかる時間を計測することで得る。具体的には第 4 章実転送速度の取得方法で述べる。

2.5　まとめ

GridComputing を実現していく上で問題となってくるのは、GridComputing によって隠された実行ホスト群の状態にある。高負荷がかかっている状態の実行ホストで処理をさせようとしても、レスポンスが非常に悪い GridComputing となってしまう。そこで、実行ホストの環境を知る機能を実装し、管理ホストへ指標を与えることで、レスポンスの良い GridComputing を目指す。

第 3 章

実処理時間の取得方法

本章では始めに、実処理時間の取得方法に必要な事項を述べる。次にアルゴリズムを説明し、最後にまとめを述べる。

3.1　必要事項

実処理時間の取得において必要となってくるのは、実測値である。実際に対象コンピュータを使用した時に得られる具体的なデータが欲しいので、実際にプログラムを対象コンピュータで動作させる事とする。このプログラムの結果が出るまでの処理にかかる時間を計測し、実測値としてログに書き出す。単位は秒とする。ここで、1 度の試行の場合、極端に負荷がかかった状態である可能性がある。したがって、こちらから特定の条件を与え、その条件下での実測値を計測する。もし何らかの外部的要因があった場合、その実測値には変化が生じているであろう。しかし、数回の計測から予測可能なデータを得ることで、特別な事態を回避する。

今回は実際に動作させるプログラムとして、π の計算をするプログラムを作成した。この π のプログラムは、計算回数を与える事で計算し、より大きな数値を与える事で精度を増すように作成した。実測値の計測には、1000000 を与える事とする。

次に特定の条件を与えるプログラムとして、上記の π の計算をするプログラムを流用した。特定の条件（＝実測値の計測に関与しない、外部的要因となりえる演算）を満たすため、実測値計測より大きな値を与え、該当ホストが計算中の条件を作り出した。

3.2　実処理時間取得のアルゴリズム

実処理時間取得のアルゴリズムと、実際に両ホスト間でどのような動きをしているかを、以下図 3.2.1 と図 3.2.2 で記す。

実処理時間取得アルゴリズムは、図 3.2.1 の上部青い部分が管理ホスト側での処理の部分であり、下部の赤い部分が実行ホスト側での処理の部分である。管理ホスト部分で書出されるスクリプトは、実行ホストでログを取得するためのループを実現する。

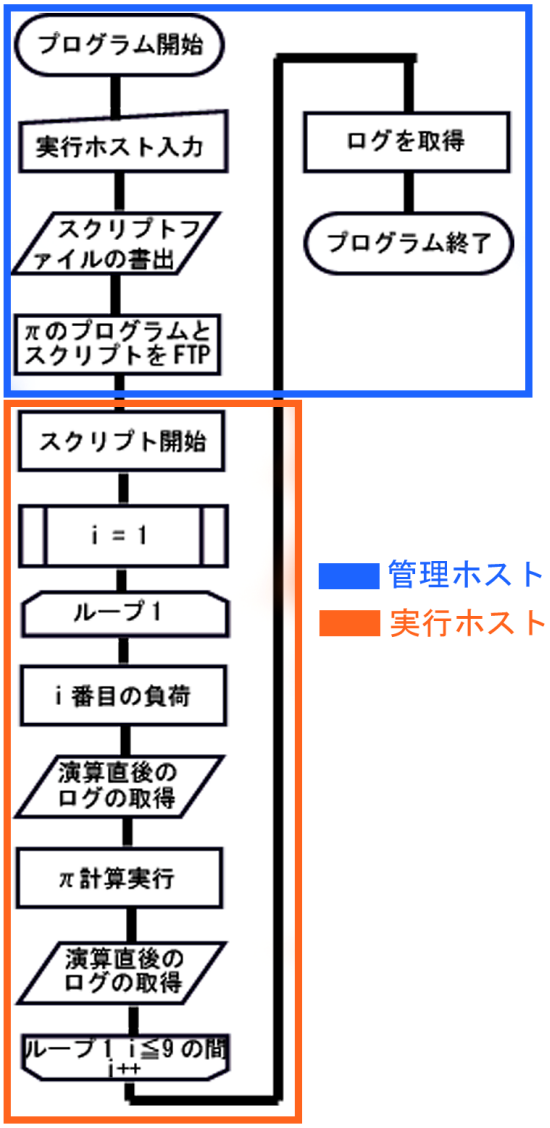

図 3.2.1　実処理時間取得アルゴリズム

次に、実処理時間計測での実行ホストと管理ホスト間のやりとりについて、時系列に沿ったやりとりを図 3.2.2 を示す。

前項の図 3.2.1 の内容を踏まえて、まずはプログラムを開始すると、管理ホストは生成されたスクリプトファイルとπのプログラムの両方を実行ホストへ転送する。そして実処理時間中に転送のラグを生じさせない為に、ログ取得とπの計算、負荷は一連の動作として、実行ホスト上で全て処理する。最後に実行ホストは残されたログファイルを管理ホストへと転送して、実処理計測を終了とする。

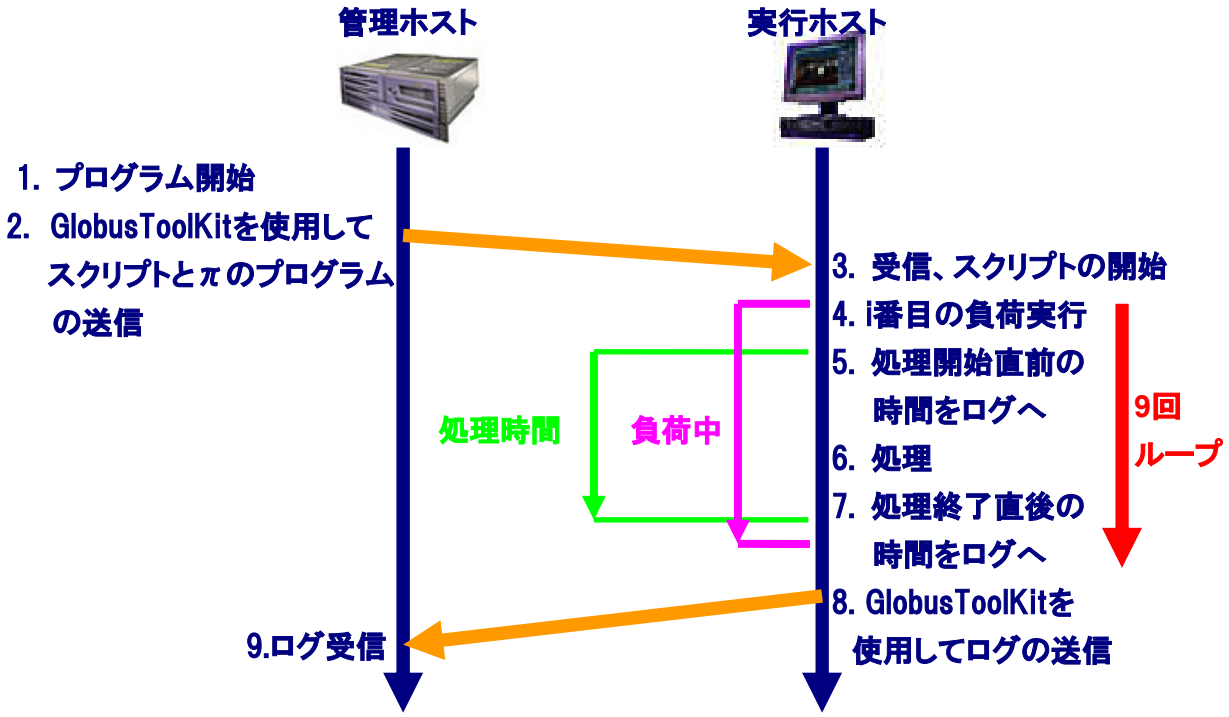

図 3.2.2　実処理時間取得の流れを示した時系列図

3.3　実処理時間計測についてのまとめ

管理ホスト上でプログラムを実行すると、引数に計測したい実行ホスト名を与える事で、実行ホスト上で実行されるスクリプトファイルが生成され、πの計算プログラムの本体と生成されたスクリプトファイルが実行ホストへと渡される。以下、3.2 節の図 3.2.1 アルゴリズム及び図 3.2.2 の時系列図を辿り、9 回の実測値を得る。

第 4 章

実転送速度の取得方法

本章では始めに、実転送速度の取得方法に必要な事項を述べる。次にアルゴリズムを説明し、最後にまとめを述べる。

4.1　必要事項

実転送速度の取得においても必要となってくるのは、やはり実測値である。実際に対象コンピュータ間での転送にかかる時間を得たいので、現物のデータを転送し、その転送にかかった時間を計測する。単位は秒とする。ここで、やはり 1 度の試行の場合、なんらかの外部的要因が存在すると考えられる。そのため、転送するデータの大きさを変え、その転送にかかった時間をそれぞれ取得して行く事で、理論的には法則性のある結果を得る。もし任意の実測値に異常があっても、周りの実測値より、特別な事態を回避する。

今回は実際に転送するデータとして、50MB、100MB、150MB・・・450MB を順番に送った。データ転送を始める直前にログ取得を開始し、GridFTP でデータを転送、転送終わった直後のログを取得して、一回の実測値とする。

4.2　実転送速度取得のアルゴリズム

実転送時間取得のアルゴリズムと、実際に両ホスト間でどのような動きをしているかを、以下図 4.2.1 と図 4.2.2 で記す。

実転送時間取得アルゴリズムは、図 4.2.1 の上部青い部分が管理ホスト側で、転送時間ログ取得までの処理の部分であり、中部の赤い部分が管理ホスト側から実行ホスト側への転送の部分である。実行ホストでは、転送時間の取得について特に関与はしない。

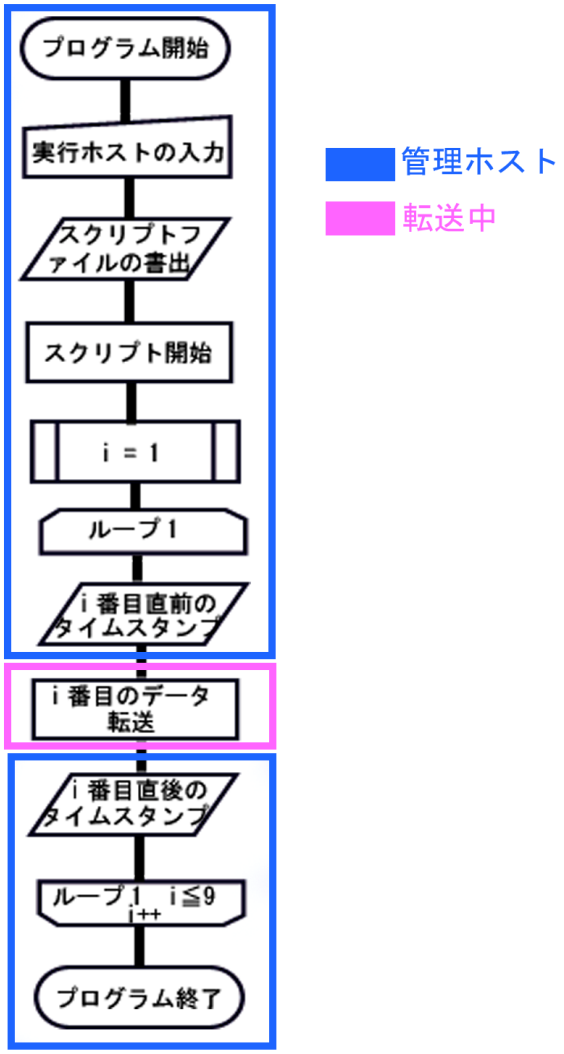

図 4.2.1　実転送速度取得アルゴリズム

次に、実転送時間計測での、実行ホストと管理ホスト間のやりとりについて、時系列に沿ったやりとりを図 4.2.2 を示す。

前項の図 4.2.1 の内容を踏まえて、まずはプログラム開始から、管理ホスト上で GridFTP のやりとりを書いたスクリプトファイルを実行する。そして実転送時間中に転送のラグを生じさせない為に、管理ホスト上でログ取得の処理をさせる。また、転送中に外的要因（意図しない回線への負荷）を避ける為、転送直後のタイムスタンプが終了してから、次の計測を開始する。

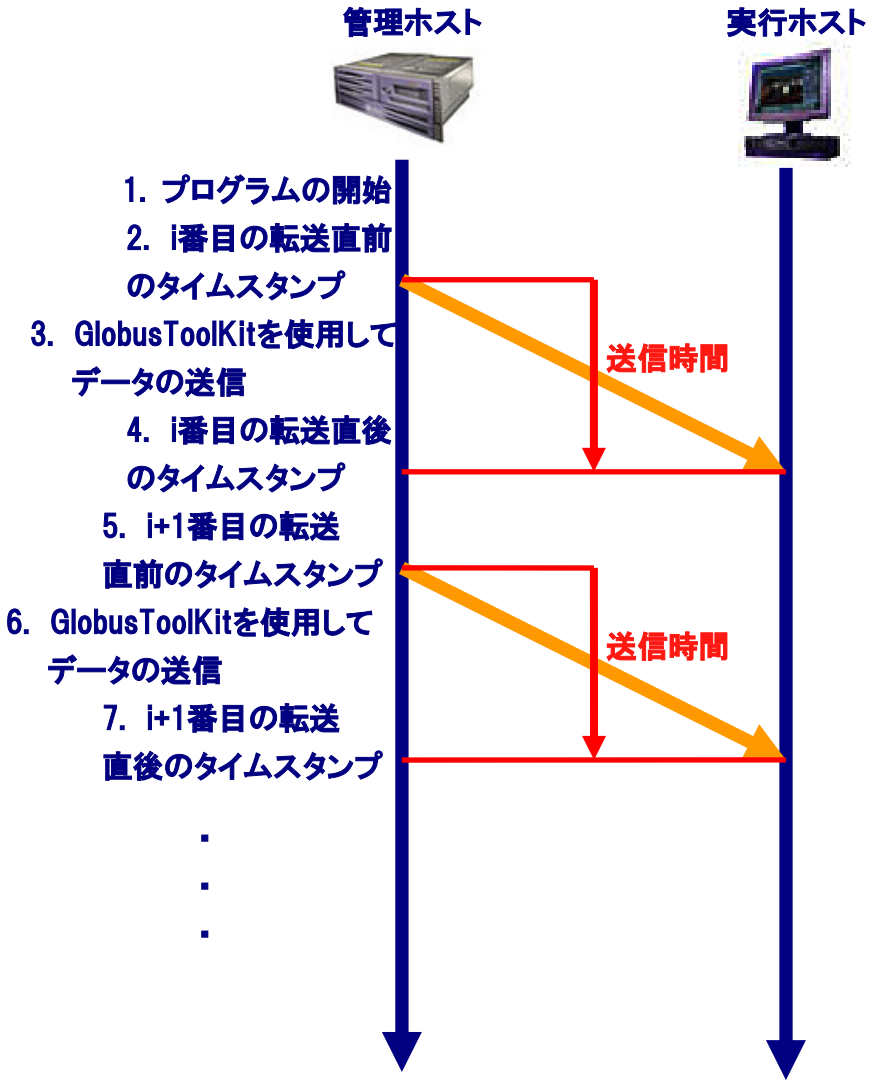

図 4.2.2　実転送速度取得アルゴリズム

4.3　実転送速度計測についてのまとめ

管理ホストに実測値を得たい実行ホスト名を与える事で、スクリプトを生成する。スクリプトには、GridFTP の命令が記述されており、順番に、タイムスタンプを取り、データを転送し、タイムスタンプを取る、という一連の動作を、転送するデータを変えながら繰り返す。

第 5 章

評価実験

本章では、２ステップで実測値を得る事とする。まず、管理ホスト１機に対し、実行ホスト１機を目的とする状態に関する評価実験を行う。次に管理ホスト１機に対し、複数の実行ホストがある場合を計測する。

5.1　一つの管理ホストと一つの実行ホストの評価実験

3 章の図 3.2.1 と 4 章の図 4.2.1 で述べた二つの計測アルゴリズムを用いて、一つの管理ホストと一つの実行ホストについて評価を行う。一連の流れとしてのアルゴリズムは、次項の図 5.1.1 評価実験の一連の流れに示した図 3.2.1 と図 4.2.1 のアルゴリズムを繋げた図である。ただし、図 3.2.1 と図 4.2.1 とで、実行ホスト名の入力の部分だけは役割が重なっているため、処理を省略している。また、図 3.2.1 と図 4.2.1 のアルゴリズムを連結する際に、実測値の明らかな異常があったため、300 秒のスリープを間に挿入している（理由は後述、5.1.4 節にて）。

今回の管理ホストと実行ホストの実験で使用した環境について表 5.1.1 に明示しておく。

表 5.1.1　スペック表 1

ホスト名＼項目	CPU（Hz）	通信速度（bps）
Grid00（管理ホスト）	Pentium3 1GHz	1000Mbps
Grid03（実行ホスト）	Pentium4 3GHz	1000Mbps

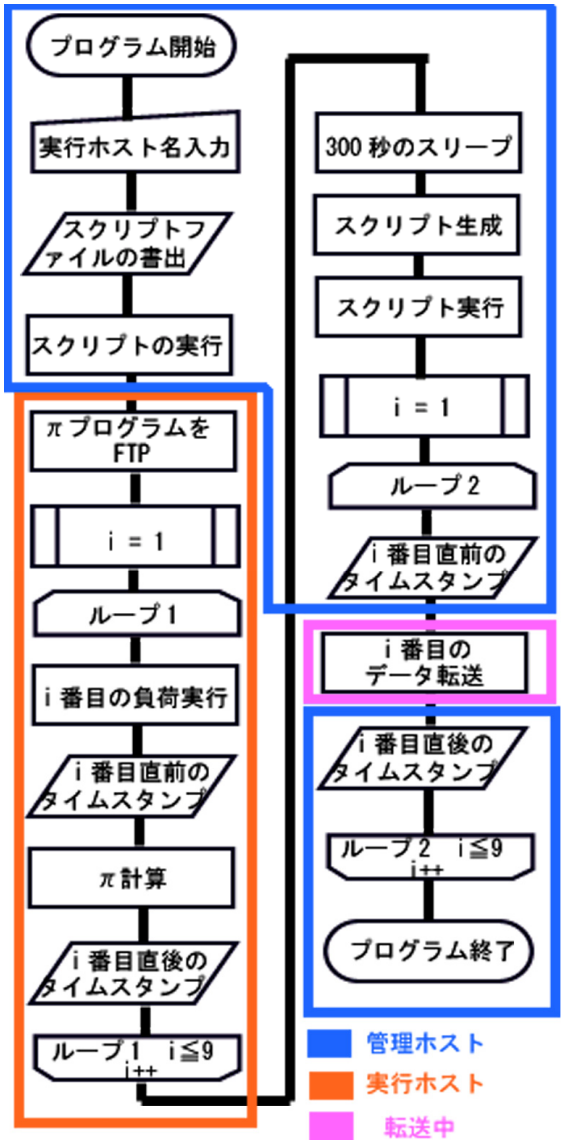

図 5.1.1　評価実験の一連の流れ

5.1.1　実処理速度の実測値

実際に実処理速度の実測値を計測することで得られたデータを表 5.1.2 と図 5.1.2 に示す。ここで、負荷数というのは、外的要因を指す。つまり、実測値を得るのとは別に、こちらが予め用意したダミーの負荷プログラムである。この負荷は計算回数を等差でリニアに増加させた負荷である。従って、負荷数が大きくなれば、負荷も大きくなることを表す。

表 5.1.2　実処理速度の実測値表 1

負荷数	1	2	3	4	5	6	7	8	9
Grid03(秒)	15	31	51	79	118	156	195	235	259

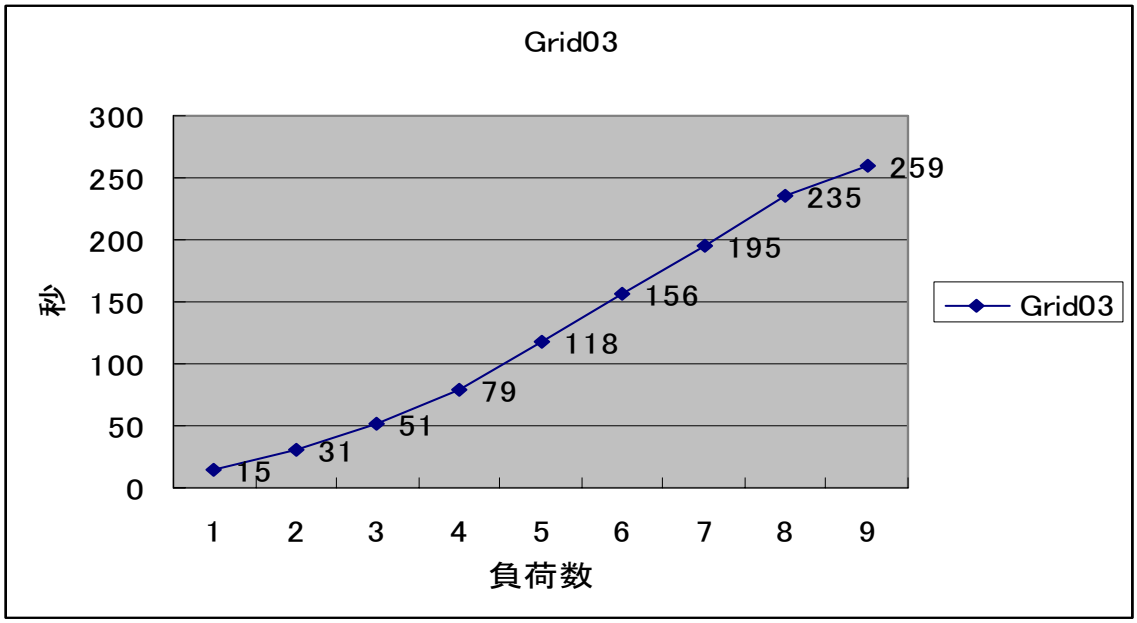

図 5.1.2　実処理速度の実測値グラフ 1

5.1.2　実処理速度の考察

X 軸である負荷数は、すべての試行において、計測プログラムのπ計算本体の回数が含まれている為、+1 されている。

予想では、負荷数は等差でリニアに増加しているため、得られる実測値もリニアに増加している結果を得られると予想される。

実際に得られた結果は、負荷をかければかけるほど予想通りなリニアに増加を示しており、理論的に正しい結果を得られていると言えそうである。

以上より、一つの管理ホストと一つの実行ホスト間の実処理速度の実測値は、結果が得られたと言える。

5.1.3　実転送速度の実測値

実際に実転送速度の実測値を計測することで得られたデータを表 5.1.3 と図 5.1.3 に示す。ここで、データは、実際に転送したデータの大きさ（単位は MB）を示す。

表 5.1.3　実転送速度の実測値表 1

データ(MB)	50	100	150	200	250	300	350	400	450
Grid03(秒)	126	130	63	43	106	101	132	111	168

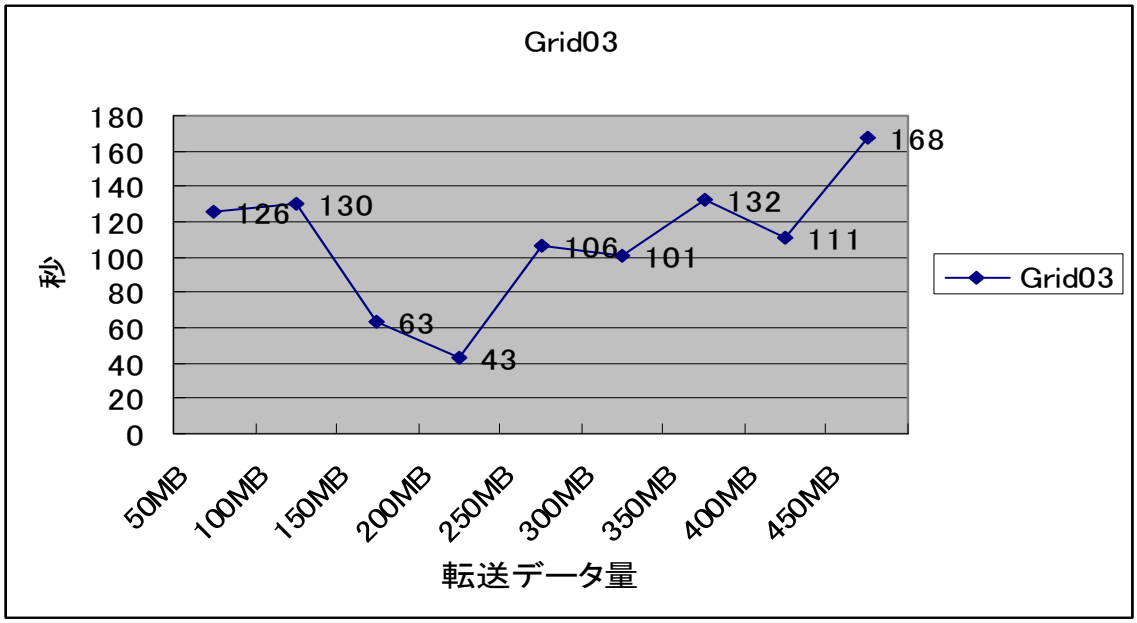

図 5.1.3　実転送速度の実測値グラフ 1

5.1.4　実転送速度の考察

X 軸である負荷数は、転送したデータの大きさを示す。

予想では、転送するデータは等差で大きくなっていく大きさのデータを転送するので、結果もリニアに増加する結果が得られると考えた。

しかし、転送したリニアに増加するデータの大きさから考えると、非常に予想からかけ離れた値が出たといえる。特に 50MB と 100MB の結果は、150MB から 350MB までの結果を超えており、予想とは大きくかけ離れた結果が得られたと言える。

そこで、50MB、100MB の両結果を足した値である 300 秒ほどずらして計測を行ってみた。計測結果は次項の表 5.1.4 実転送速度の実測値表（Sleep 後）と、そのグラフである図 5.1.4 実転送速度の実測値グラフ（Sleep 後）になる。

表 5.1.4　実転送速度の実測値表（Sleep 後）

データ	50	100	150	200	250	300	350	400	450
Grid03	0	0	38	57	82	90	108	122	146

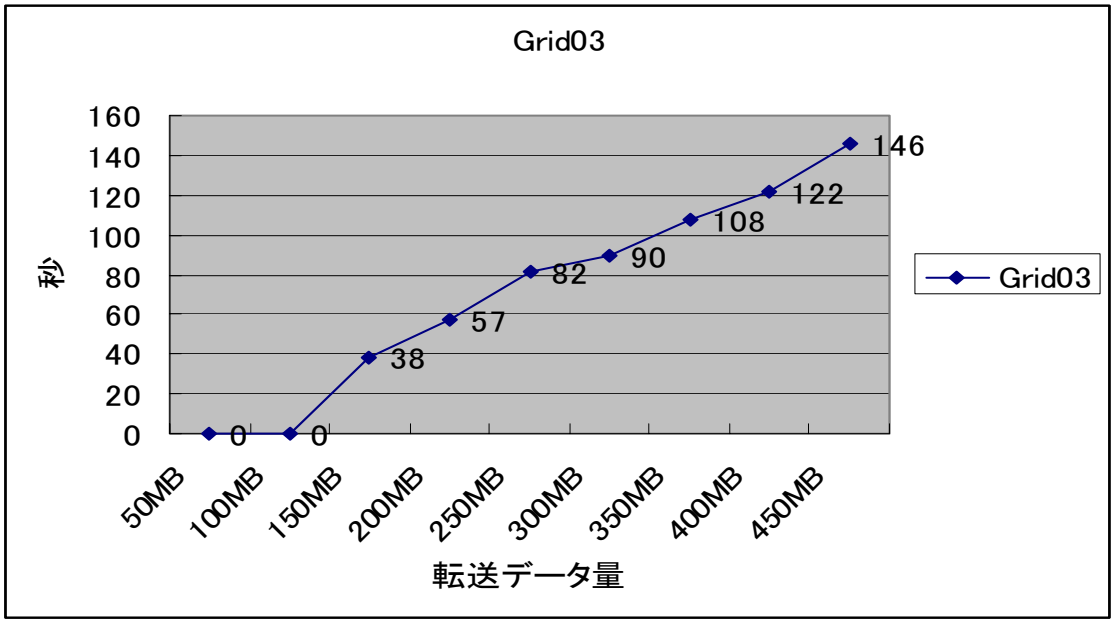

図 5.1.4　実転送速度の実測値グラフ（Sleep 後）

5.1.5　両実転送速度の計測の考察

5.1.4 節で得られた結果は、5.1.3 節で予想とはかけ離れた結果が得られた 50MB と 100MB の結果を考慮し、300 秒の Sleep を加えた後の結果である。5.1.3 節の結果に比べると予想に近いリニアに増加した結果になったといえる。50MB と 100MB については、0 秒というデータとしてふさわしいとは言えない数値が得られたが、これは 1000Mbps 回線を使用していることの誤差だと思われる。

以上より、一つの管理ホストと一つの実行ホスト間の実転送速度の実測値は、結果が得られたと言える。

5.2　一つの管理ホストと複数の実行ホストの評価実験

本節では、前節 5.1 で使用したアルゴリズムを、対複数の実行ホストを考慮して、アルゴリズムに手を加えた。改良するに当たっての問題点は、複数の実行ホストに対して同じ条件下で測定をしているかと言うことである。理想では完全に同じ条件下という状態でないと、本研究の比較対象は難しい。同じ条件下、すなわち同程度の外的要因がかかっている状態で、同時に計測をすることが望ましい。実処理時間計測については、それぞれを同時に開始しても問題が無いため、この条件を満たし、かつ、5.1 節と変わりない。しかし実転送速度計測は、同時に計測開始した場合、管理ホストからデータを転送するネットワークインターフェイスの部分にトラフィックが集中すると考えられる。よって、トラフィックが集中することは、何らかの想定の範囲外となる外的要素が加わったと考える事ができるため、実際に欲しい実測値が得られなくなると考えられる。すなわち、同時に計測を開始することはできない。この問題を解決するために、以下の方法を取った。

まず、一度の計測は別のホストとの転送を行わない。つまり排他的な計測となる。

次に、出来る限り同時という条件を満たす為、複数の実行ホストに対し、一つの実行ホストとの計測が終わると、データの大きさを変えず、計測する実行ホストを変更する。計測したい実行ホスト全ての計測が終わった後、データの大きさを変える事で同時という条件をよい近い状態で満たす。ただし、計測したい実行ホストが大量に存在する場合、この計測方法では、始めのうちに計測した実行ホストと、最後の方に計測した実行ホストとの間にラグが生じる事が考えられる。この事については、6 章、6.3 節今後の課題で触れる事とする。

以上の事をアルゴリズムとして、次項の図 5.2.1 評価実験の一連の流れ改良版に示す。

また、今回の一つの管理ホストと複数の実行ホストとの実験で使用した環境について、表 5.2.1 スペック表 2 に示す。

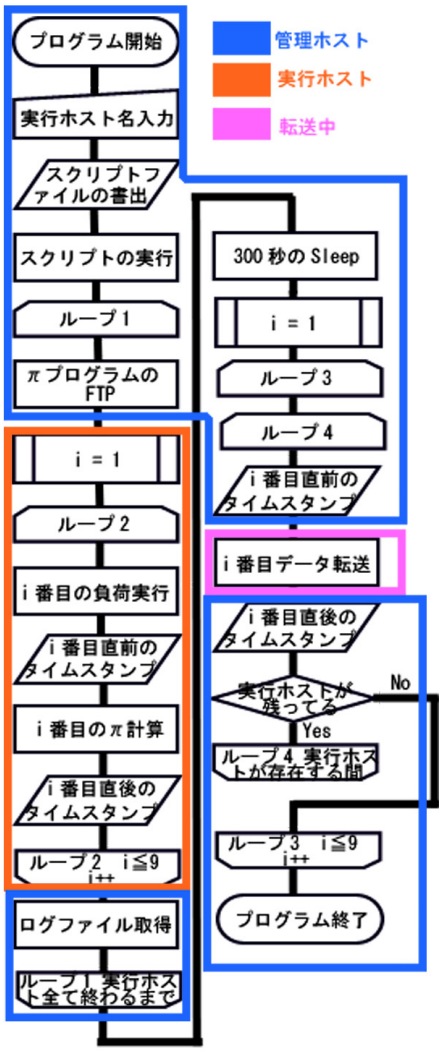

図 5.2.1　評価実験の一連の流れの改良版

表 5.2.1　スペック表 2

ホスト名	CPU（Hz）	回線 1（bps）	回線 2（bps）
Grid00	Pentium3 1GHz	1000Mbps	10Mbps
Grid01	Pentium4 3GHz	1000Mbps	10Mbps
Grdi02	Pentium4 3GHz	1000Mbps	10Mbps
Grid03	Pentium4 3GHz	1000Mbps	10Mbps
Grid04	Pentium4 3GHz	1000Mbps	10Mbps
Juptire	Cerelon 300MHz	1000Mbps	10Mbps

5.2.1　実処理速度の実測値

実際に実処理速度の実測値を計測することで得られたデータを表 5.2.2 と図 5.2.2 に示す。ここで、負荷数というのは、前節 5.1 と同じ、外的要因を指す。

表 5.2.2　実処理速度の実測値表 2

ホスト名＼負荷数	1	2	3	4	5	6	7	8	9
Grid01(秒)	16	30	55	80	111	152	199	232	255
Grid02(秒)	15	30	54	87	109	146	203	233	251
Grid03(秒)	15	31	51	79	118	156	195	235	259
Grid04(秒)	16	31	53	86	116	138	202	227	259
Jupiter(秒)	—	258	454	706	961	1279	1656	1935	2211

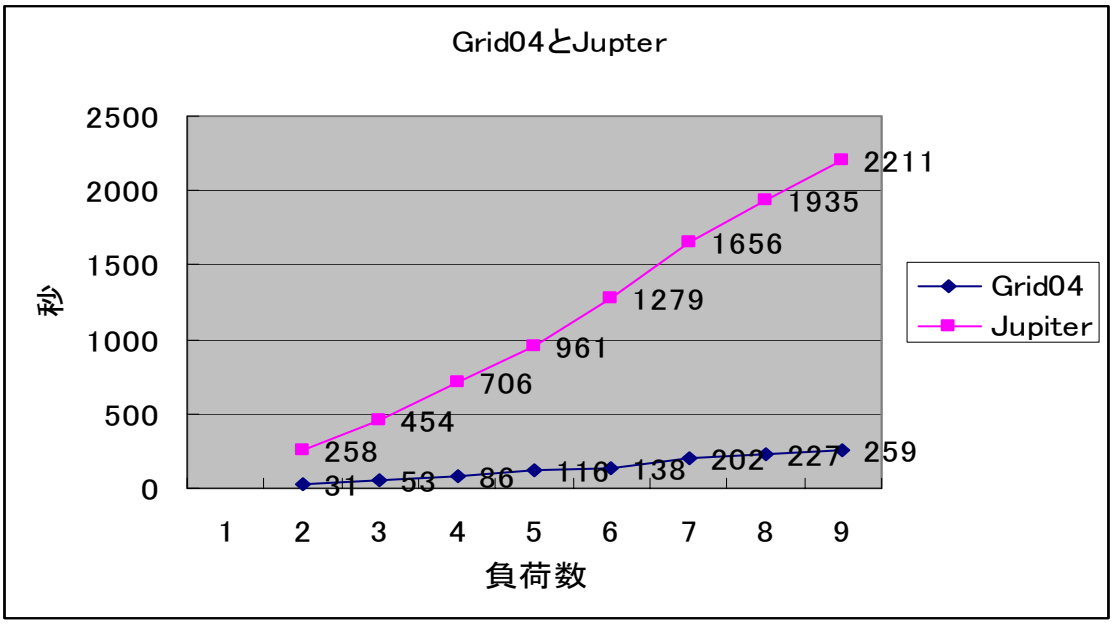

図 5.2.2　実処理速度の実測値グラフ 2

5.2.2　実処理速度の考察

結果から見ても分かる通り、Grid01 から Grid04 はスペックが全く同じである為、結果はスペック表からも誤差の範囲である。よって本項では Grid04（高スペック）と Jupiter（低スペック）について考察する。この結果をグラフにすると図 5.2.2 のグラフとなる。

前節 5.1.2 の考察と同様、X 軸である負荷数は、すべての試行において、計測プログラムのπ計算本体の回数が含まれている為、+1 されている。また、負荷をかければかけるほどリニアに増加しており、想定の範囲内にある理想的な結果を得られていると言える。

注目すべきは、高スペックである Grid04 と低スペックである Jupiter に明確な差がある事と、Jupiter の第 2 回目の結果と Grid04 の第 9 回目の結果が逆転していることである。性能に明確な差があることは、スペック表からでも明らかであるが、これが実測として証明されたと思われる。また逆転状況も生まれた事によって、Grid04 が高負荷状態にある場合には、Jupiter が処理能力で上回る事が実証されたと思われる。

5.2.3　実転送速度の実測値

実際に実転送速度の実測値を計測することで得られたデータを表 5.2.3 と図 5.2.3 に示す。ここで、データは前節 5.1 と同じ、実際に転送したデータの大きさ（単位は MB）を示す。

表 5.2.3　実転送速度の実測値表

回線＼データ量	50	100	150	200	250	300	350	400	450
10M(秒)	1	1	135	179	222	266	309	354	402
1000M(秒)	1	2	44	67	82	108	120	140	162

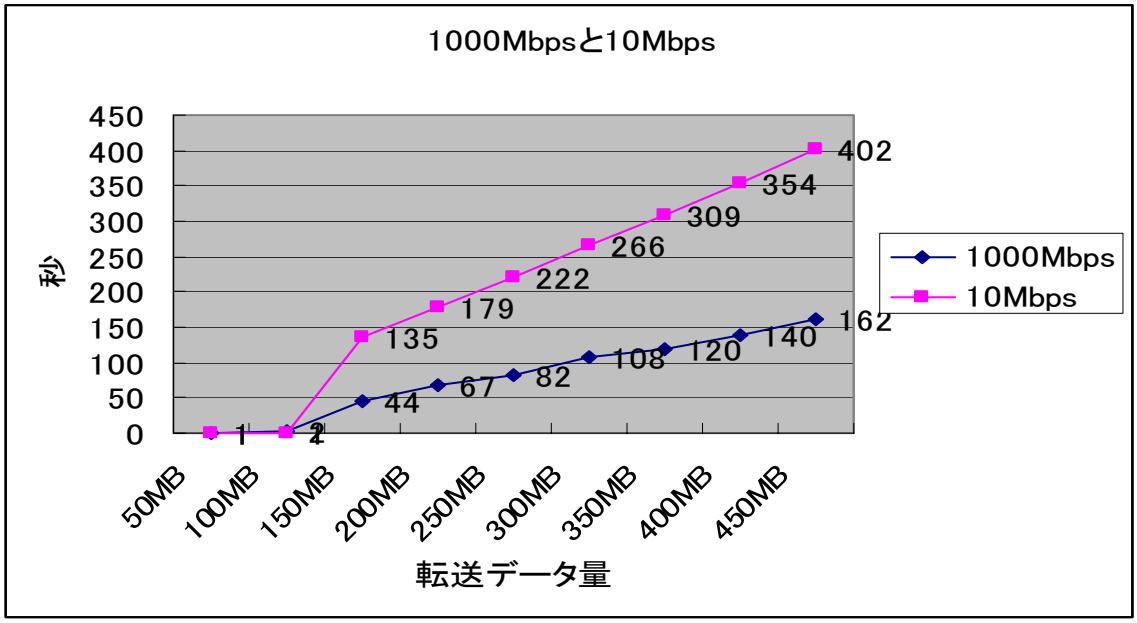

図 5.2.3　実転送速度の実測値グラフ 2

5.2.4　実転送速度の考察

図 5.2.3 は、前節 5.1.3 での 50MB と 100MB の予想とは大きくかけ離れた結果を得られるという問題を解決した、途中に 300 秒の Sleep を挿入する事を既に導入している。しかし、前節 5.1.3 では、1000Mbps の影響であると思われた 50MB と 100MB のデータ異常が、今回は 10Mbps にも現れている。これでは、1000Mbps によって得られる転送速度の誤差という 5.1.4 節の予想の説明がつかない。これは Globus の仕様なのかもしれない。しかし結論を得るまでにはデータ数が足りないので、憶測の域を出ない。150MB 以降の結果については、概ね理想的な数値が得られている為、分類に使うには十分な結果といえる。ちなみに、両データについては Grid04 と Jupiter 両方で計測し、その結果の平均を用いている。

第 6 章

考察

本章では、第 5 章の実験で得られた実測値から考察をする。まず、実測値を得た上で出てきた問題について考察する。次にその問題を考慮した上で、各実行ホストの環境を分類する。最後に今後の課題として、精度の向上と取得時間の短縮について触れる。

6.1　実転送速度取得について

前節 5.1.3、5.1.4、5.2.3、5.2.4 でも触れたが、実転送速度の実測値を見ると、異常な事に気がつく。50MB と 100MB の場合、値が極端に理論的な数値とずれる。この原因を回避するために 300 秒ほどの Sleep を加えると、今度は逆に転送時間がほぼ 0 という結果にたどり着く。この問題について何らかの仮説を得るまでには時間がなくて至らなかった。もし Globus の仕様だとすると、現状では GlobusToolKit の内部を改良しない限り手の施しようが無いのかもしれない。

6.2　分類

以上の結果を用いて、実行ホストの環境の分類をしていく。単純に数値を比べた場合と、特別に外部的要因を加えた場合とで見比べてみる。必要となったのは処理能力と転送能力の二つであり、低処理能力と高処理能力、低速回線と高速回線であるため、4 つの環境に分類することができる。

まずは通常の外部的要因が何もない状態。前節 5.2 と同様、Grid01 から Grid04 については同様のスペックであるため、Grid04 と Jupiter を比較すると次項の図 6.2.1 分類図 1 のような結果が得られる。

図 6.2.1 より、相対的な 4 つの状態に分類することはできた。

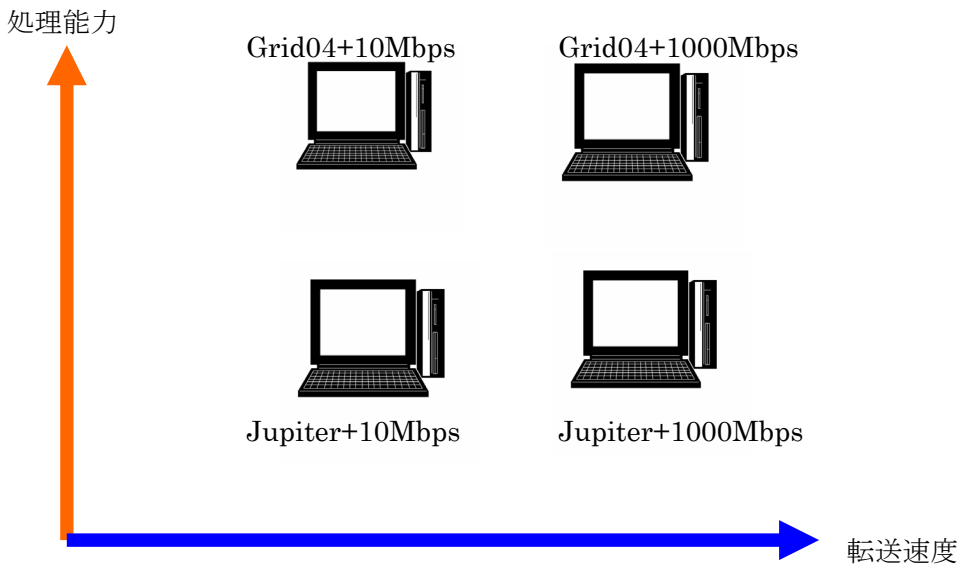
処理能力
Grid04+10Mbps
Grid04+1000Mbps
Jupiter+10Mbps
Jupiter+1000Mbps
転送速度

図 6.2.1　分類図 1

次に、Grid04 に特別に外部的要因を与えた分類を行ってみる。Grid04 で動画のエンコードを行った場合、処理能力は表 6.2.1 のように落ち込んだ。負荷数というのは、前節 5.1 と同じ、外的要因を指す。

さらに表 6.2.1 の結果を Jupiter と比べた場合、図 6.2.2 のようなグラフとなる。

表 6.2.1 と図 6.2.2 より、次項の図 6.2.3 のように分類することができる。

図 6.2.2 と図 6.2.3 より、実行ホストの環境を分類するには、実処理速度と実転送速度を計測することで、十分分類することができると実証された。

表 6.2.1　Grid04 に特別な負荷をかけた場合の実処理時間実測値

負荷数	1	2	3	4	5	6	7	8	9
Grid04(s)	—	269	492	780	1063	1401	1848	2155	2460

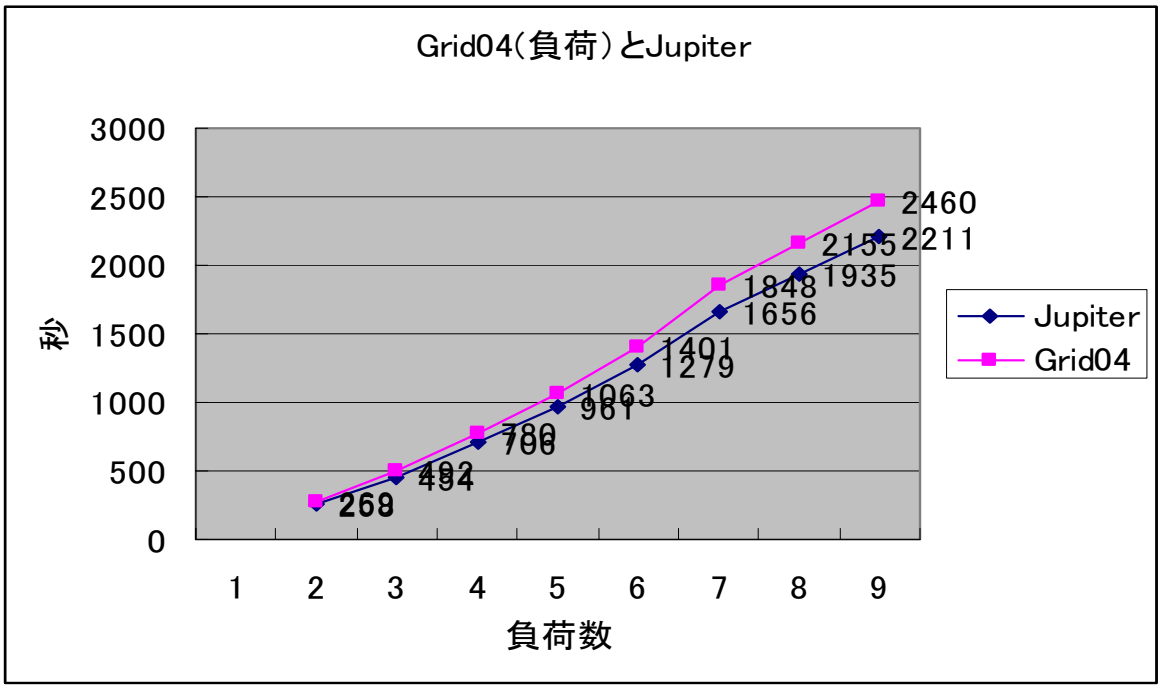

図 6.2.2　負荷をかけた Grid04 と Jupiter の比較グラフ

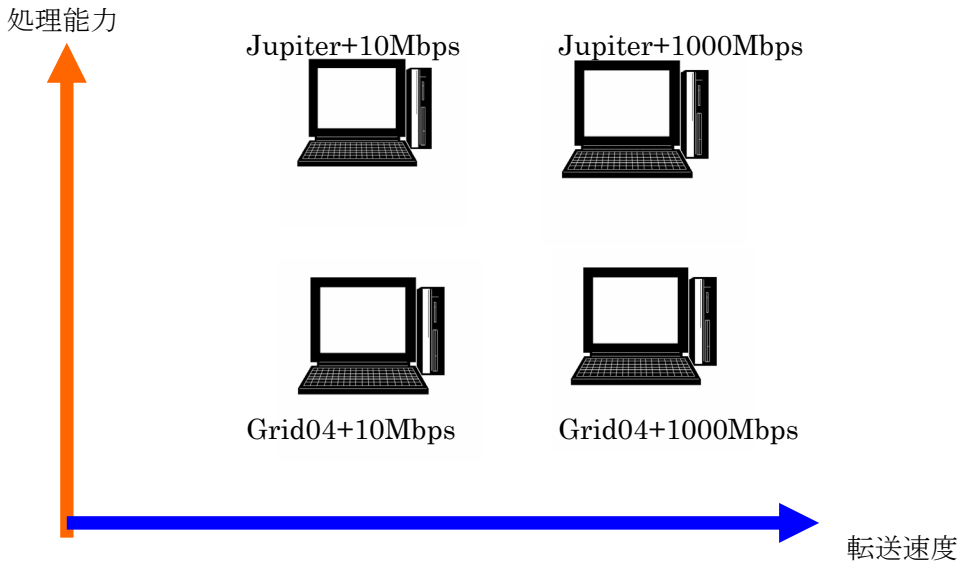

図 6.2.3　負荷をかけた Grid04 と Jupiter の分類

6.3　今後の課題

本研究で作成したスクリプトには、ある種致命的とも言える部分がある。この問題について現状での見解を述べる。

6.3.1　精度の向上

本研究では、比較する為のデータとして、秒単位での処理速度と転送速度の時間の計測を行っている。人から見たレベルでは十分な指標となりうるが、コンピュータから見たレベルでは少し物足りない。コンピュータから見た場合、１０００分の１秒レベルでの時間計測ができる事が望ましい。

また、5.2 節で問題となった、実転送速度を同条件下で計測するという問題も、任意の実行ホストについて同時に一度に計測することが、精度の向上に当たる。しかし現状の本研究では、同時に計測する効率の良い方法を考察するまでには至らなかった。

6.3.2　取得時間の短縮

本研究では、高スペックと低スペック、高速回線と低速回線の両極端な実験機を使用している。その関係上、明らかに差があるが、本研究の内容上、同じ条件を与えて計測している。高スペック・高回線で目に見えるデータを残した場合、低スペック・低回線では約 12 倍の計測時間がかかってしまった。仕方の無いことではあるが、もう少し改善の余地はあると思われる。処理にかける特定の条件とデータ転送の仕組みを根本的に見直す必要性がありそうだ。

6.3.3　分類の種類

本研究では、低処理能力･高処理能力という二つの項目と、低速回線･高速回線という二つの項目､合わせて4つの項目を知る事で､4つの指標を得る事を目標にしてきた。しかし、実行ホストの環境が増えれば 4 つの指標では物足りなくなるかもしれない。これは、投入する処理の内容と照らし合わせながら、分類数を増やしていくのが良いであろう。

二つの比較であれば、処理速度･転送速度の高低で振り分ける事ができるため、各実行ホストの結果を順番にならべて、ちょうど中間の値で二つに区切れば分類することができる。しかし分類の数を増やすならば、各実行ホスト毎になんらかの絶対的な値が必要になってくる。

本研究では、得られた結果が実測値であるため、測定結果が絶対的な値として使えるであろう。ただし、実験で得られた実測値が必ずしも理想的な値であるとは限らない。

そこで、負荷数を増やし、転送量を変えての実験が意味を持ってくるであろう。9 個の実測値が得られるが、この値の精度が問題になってくる。

例えば、得られた実処理速度の結果が図 5.2.2 のようなリニアに増加しているグラフになった場合、どの位置のデータで比べても、特に問題はなさそうである。しかし、図 5.1.3 のような波うったグラフの場合、絶対的な値としては使えない。

では、図 5.1.3 のような結果が得られた場合、どのように分類すればいいだろうか。

(A) 他の値はリニアに増加しているのに、ある一点だけ大幅にずれている場合
隣り合う数値を足して 2 で割る、いわゆる平均でおおよその値を出す事ができるであろう。

(B) 複数の値が波うってる場合
3 通りの考え方で、対処する。

(1) 処理の依頼が頻繁に起こっている･転送が頻繁に起こっている
長期的、1 日や 1 時間などの処理をさせる場合には、あまり気にならないが、短期的、1 時間や 10 分などで終わる処理には向いていないと考えられる。分類としては、単純に比較するのとは違い、もう一つ大きなカテゴリーとして分けるのが妥当であろう。

(2) 偶然、計測中に外部的負荷がかかった
時間をおいて、計測し直す事が望ましい。

(3) 近似する
もし、計測していく中で、他の実行ホストで得られた結果と照らし合わせると、いくつか重なるデータが存在している場合に限り、実行ホストの近似をすることもできるであろう。ただし、必ずしも正確な値であるとは言えない為、波うってる結果を得た実行ホストの優先度は低くなると考えられる。

いずれの場合にしても、結果が(B)の場合は、本研究の分類の指標としては、処理の投入時の優先度が低くなることであろう。

以上、(A)(B)の考えを導入し、実測値として安定していないデータは優先度を下げ、安定した値を得られた実行ホスト同士の、ある結果の数値に着目して分類することで、4 つの分類よりも、より多くの分類に耐えうる絶対的な値を得られるであろう。

第 7 章

結論

GridComputing を使っていく上で、環境を構築した際、先に本研究のスクリプトを実行し、どのくらいの潜在能力があるのかを予め管理ホストが認知しておくのが望ましい。本研究は、6.3 節で述べたように、ハイスペックな環境である Grid04 に比べて、ロースペックな Jupiter は Grid04 の約 12 倍のデータ取得時間がかかる。さらにリアルタイム計測をするという用途は考慮していない。実行ホストの本来の潜在能力を知り、管理ホストへ処理の振り分け用に指標を与えるスクリプトである為、ある程度の間隔毎、一日に一回や、週に一回、月に一回、または長期起動をしているなら、起動時に一回程度に実行できれば良いであろう。精度についてはまだまだ突き詰める余地はあるものの、実行ホストを分類するレベルでは十分及第点であると言える。

謝辞

本研究をおこなうにあたり日頃より暖かい御指導を賜わりました福岡大学工学部電気工学科　奥村勝　助教授に深く感謝いたします。

また、本論文を作成するにあたりましても、奥村勝先生には大変お世話になりました。あらためて深く感謝いたします。

参考文献

[1] [GridComputing] Grid Computing
http://premium.nikkeibp.co.jp/grid/
[2] [GlobusToolKit] Globus Pukiwiki
http://www.jpgrid.org/tech-info/pukiwiki/