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

中田秀基[†1] 竹房あつ子[†2] 松岡聡[†3]
佐藤三久[†4] 関口智嗣[†1]

ネットワーク技術の発展にともない、グローバルコンピューティングシステムがいくつか提案されているが、その計算/通信リソースを十分活用するための手法はいまだ確立されていない。リソースの有効活用のために、広域分散計算に特有なリソースの変動と不安定さを考慮し、関連する複数のタスクに対して一括して適切にリソースを割り当てるスケジューリング手法が必要とされている。

本稿では、グローバルコンピューティング環境において個々のアプリケーションの性能とシステム全体のスループットを両立させるための、階層化されたスケジューリングフレームワークを提案する。さらに、このフレームワークに準拠したNinfシステムのメタサーバスケジューリングフレームワークの実装について述べ、試験的に実装したスケジューリング手法による評価を行う。この結果、本フレームワークのもつ可能性と、タスク間のデータ依存関係を考慮したスケジューリングアルゴリズムの重要性が明らかになった。

A Scheduling Framework for Global Computing

Hidemoto Nakada,[†1] Atsuko Takefusa,[†2] Satoshi Matsuoka,[†3]
Mitsuhisa Sato[†4] and Satoshi Sekiguchi [†1]

Rapid progress in networking technology is now making global computing systems feasible. Although there have been proposals of global computing systems, it is still a research issue as to how to achieve efficient usage of computing resources in global computing. In particular, we need to devise appropriate scheduling strategies/algorithms of computing resources over wide-area networks, which are often dynamic and unstable in nature. This paper presents our preliminary scheduling framework for unifying application and job scheduling in global computing. The proposed framework establishes a layer of scheduling and resource allocation subframeworks. We show our software framework Ninf metaserver which provides low-level scheduler and resource monitor. We also evaluate some scheduling strategies using the framework.

1. はじめに

ネットワーク技術の発展により、広域ネットワーク上に分散した計算リソースや情報リソースを積極的に活用して、大規模計算を実現するグローバルコンピューティングが可能となった。近年これを目的としたシステムがNinf[1)]をはじめ、複数提案されている[2)~5)]。しかしグローバルコンピューティングシステムの計算/通信リソースを十分活用するための手法はいまだ確立されていない。リソースを有効に活用するために、広域分散計算に特有のリソースの変動と不安定さを考慮し, 関連する複数のタスクに対して適切にリソースを一括して割り当てるスケジューリング手法が必要とされている。

従来のグローバルコンピューティングシステムでのスケジューリングの研究は、アプリケーションスケジューリングとジョブスケジューリングの2つに大別することができる。アプリケーションスケジューリングは、単一のプログラムの応答時間を短縮することを目的とし、アプリケーションプログラムの特性にしたがってプログラムを複数の並列タスクに分割してタスクをスケジュールする。これに対してジョブスケジューリングはグローバルコンピューティングシステム全体に投入した複数ジョブの総実行時間(スループット)を向上させることを目的として、各ジョブをスケジュールする。

アプリケーションスケジューリングの例としてはAp-

†1 電子技術総合研究所 Electrotechnical Laboratory
†2 お茶の水女子大学 Ochanomizu University
†3 東京工業大学 Tokyo Institute of Technology
†4 新情報処理開発機構 Real World Computing Partnership

pLeS[6]、Prophet[7], Globus[5] のDUROC/GRAM[8] などがある。これらのシステムはいずれも1つのクライアントプログラムがそのグローバルコンピューティングシステムを占有する状態を前提としており, 複数のクライアントのプログラムがグローバルコンピューティングシステムを共有し、お互いのタスクが干渉することは考慮されていない。Netsolve[2] のAgentはクライアントに適切なサーバを紹介する、ある種のアプリケーションスケジューリングシステムであるが、Netsolveの性質上単純なRPCの構造に対するスケジューリングのみを対象としており、複数の呼び出しを集合としてディスパッチするという観点には欠けている。

ジョブスケジューリングはMPPのように密に結合した環境では広く研究されてきた[9] が、計算/通信リソースが不安定に変動するグローバルコンピューティングシステムに関しては十分研究されていない。Condor[10] のMatchmakingメカニズムは計算/通信リソースとそれに対する要求を構造化された言語で記述し、リソースと要求の対応を取ることによってリソースの割り当てを行うものである。これは、複数のジョブをそれぞれ適切なプロセッシングユニットにディスパッチするという意味においては一種のジョブスケジューラではあるが、その主眼が特定の資質(アーキテクチャ/デバイスなど)をもつリソースを確保することにおかれており、複数のジョブ全体のスループット向上をめざすものではなく、通常の意味のジョブスケジューラではない。また、計算/通信リソースの負荷などの動的情報、複数のジョブ間の干渉、自分自身のサブタスク間の干渉が考慮されていない。

パラメータサーチに特化したグローバルコンピューティングシステムNimrod[11] は、スループット指向のジョブスケジューラを持つが、そのアルゴリズムは単純なセルフスケジューリングである。その後継システムに当たるNimrod/G[12] では、デットラインスケジューリングを行うとしているが、実装の詳細は明らかにされていない。

ここで重要なのは、従来のグローバルコンピューティングにおける研究ではアプリケーションスケジューリングとジョブスケジューリングが協調して論じられていないということである。これは、これまでのグローバルコンピューティングシステムの研究が事実上単一のジョブ/ アプリケーションをターゲットにした専用システムとして開発されてきたことに起因すると思われる。これまでは、グローバルコンピューティングシステムが未成熟だったこともあり、個々のシステム上では対象とするジョブ/アプリケーションを個別に動かすことしかできず、それに特化したスケジューリング機構で十分であった。フレームワークが成熟しつつある現在、アプリケーションスケジューリング的な観点とジョブスケジューリング的な観点をバランスよく組み合わせ、個々のアプリケーションの性能とシステムの総スループットを両立させるようなスケジューリングが求められている。これには、複数のジョブ間、サブタスク間の干渉を考慮にいれた上で、タスク全体を一括でスケジューリングするスケジューリングアルゴリズムとソフトウェアフレームワークが必要である。

本稿では、グローバルコンピューティングにおいてアプリケーションスケジューリングとジョブスケジューリングを統合するための基本的なスケジューリングフレームワークについて述べる。また、そのフレームワークの一部である低位レベルスケジューリングシステムの実例としてNinfシステムのメタサーバスケジューリングフレームワークの実装を示す。さらに、その簡単な評価を行う。

以後2では、グローバルコンピューティングでのスケジューリングフレームワークについて述べ、3でNinfメタサーバフレームワークについて説明する。4では、いくつかのスケジューリング手法について、メタサーバのプロトタイプを用いた実環境での評価について報告し、5でグローバルコンピューティングのスケジューリングに関する考察およびまとめを述べる。

2. グローバルコンピューティングスケジューリングフレームワーク

2.1 グローバルコンピューティングシステム

本稿で対象とするグローバルコンピューティングシステムは一般に以下の要素で構成される。

- クライアント: 計算要求を発行する主体
- 計算サーバ: 計算リソースを提供するサーバ
- ストレージサーバ: 計算の途中結果や共有されるべきデータを保持するサーバ
- スケジューラ: 計算要求に対して計算サーバやストレージサーバを割り当てる機構

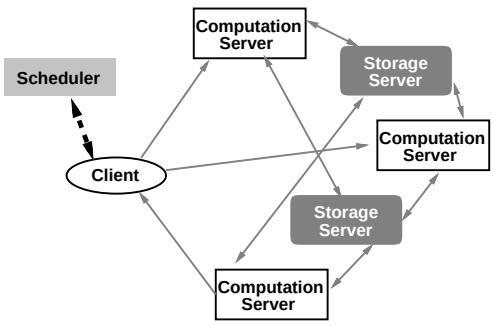

図1 グローバルコンピューティングシステムの概要

2.2 スケジューリングフレームワークの概要

単一ジョブの実行速度と複数ジョブの総合スループットを両立させるには、それぞれのジョブをより原始的なタスクに分割して、それを単一の枠組みでスケジュールすればよい。このための枠組みとして我々は図2に

示すような階層化したスケジューリングとリソース割り当てのサブフレームワークを提案する。

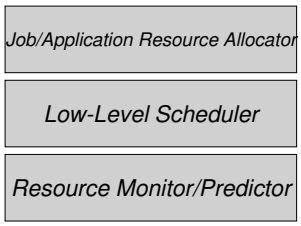

図2　グローバルコンピューティングスケジューリングフレームワーク

本フレームワークのポイントは、ジョブ/アプリケーションを、Network Task と呼ぶ、より原始的なタスクに分割してスケジュールすることにある。本システムでのアプリケーションの実行は、次の 2 段階に分けられる。

(1)　単一アプリケーションを複数の Network Task に分割する。
(2)　Network Task をサーバへスケジュールする。

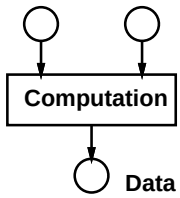

図3　Network Task

Network Task は計算とそれに対する入力データ、出力データで定義される (図 3)。アプリケーションは、この Network Task のグラフとして表現することができる。図 2 に示すフレームワークでは、ジョブ/アプリケーションリソースアロケータがアプリケーションの通信/計算特性を考慮しつつ Network Task への分割 (1) を行う。低レベルスケジューラではサーバやネットワークの現在の負荷や予測された負荷情報を考慮し、個々の Network Task を適切な計算ノードに割り当てる (2)。この際に、Network Task の構造に反映されているアプリケーションの構造を情報として用いることで、アプリケーションに特化したスケジューリングが可能になる。また、最下位のリソースモニタ/プレディクタは低レベルスケジューラにリソース情報を提供する。

このスケジューリングフレームワークでもっとも着目すべき点は、高レベルのアプリケーションスケジューリングとジョブスケジューリングの問題を低レベルの Network Task に統一するところにある (Network Task スケジューリング)。これにより、低レベルスケジューラがアプリケーションスケジューリングとジョブスケジューリングの要求をバランスよく採り入れたスケジューリングが可能になる。また、Network Task をベースとして考えることでグローバルコンピューティングの基本性能モデルが単純になり、計算サーバやネットワークの負荷情報だけを、スケジューリング決定時の本質的なパラメータとすることができる。これにより、グローバルコンピューティングプラットフォーム固有の不安定性下でのスケジューリングが容易になる。

2.3　Network Task スケジューリング

低レベルスケジューラは Network Task グラフを個々の Network Task の計算量、Network Task 間の通信量を考慮しつつ実際の計算/通信リソースにマップする役割を持つ。この際問題となるのは、過去にスケジュールして未だ実行中である Network Task 群や、同一グラフ内の他の Network Task との間に生じる干渉である。広域ネットワーク環境下では、計算/通信リソースのモニタリングは、そのコストや計算への影響を考慮すると、高い頻度で行うことはできない。さらに、モニタ情報の転送自身にも時間がかかるため、収集されたリソース状況は常に過去のものであることになり、これをスケジューリングの基礎情報としてそのまま用いることはできない。このため、現在の状況を予測する必要がある。さらに、同一グラフ内の Network Task の配置によるリソースの状況変化は、予測する以外に知る方法はない。この予測を可能にするために、広域に分散した計算・通信リソースの情報を収集し集中的に管理する DB マネージャとそれを参照するプレディクタを導入する。

低レベルスケジューリングのためのサブフレームワークは以下のモジュールから構成される。

- 計算リソースの負荷モニタ
- ネットワークリソースのモニタ
- リソース情報 DB マネージャ
- リソース状況プレディクタ
- リソーススケジューラ

負荷モニタは計算サーバの情報を収集する。ネットワークリソースモニタは、システムを構成するすべてのサーバ/クライアント間のスループットを測定する。これらのモニタは定期的に観測をおこない、その結果をリソース情報 DB マネージャに報告する。リソース状況プレディクタは DB に格納された情報に基づいて、そのリソースの現在と未来の状況を予測する。スケジューラは DB とプレディクタの情報を踏まえて、各 Network Task に対して適切な計算ノードを割り当てる。

3.　Ninf メタサーバシステムによる試験実装

Ninf システムは、RPC ベースのグローバルコンピューティングシステムである。Network Task スケジュー

リングを行うフレームワークとして、2.3に基づくNinfメタサーバスケジューリングフレームワークのプロトタイプを設計・実装した。

3.1 Ninf システム

Ninfシステムは基本的にリモートサーバ上の計算ルーチンをクライアントから実行するシステムであり、通常の使用法では、計算に用いるデータはすべてクライアントから供給し、結果もクライアントに書き戻す(図4:A)。これに加えて、Ninfには計算に用いるデータを任意のストレージサーバから取得し、計算結果を任意のストレージサーバに書き込む機構がある(図4:B)。ストレージサーバの機能は、現在Ninfサーバ自身に実装されており、すべてのNinfサーバがストレージサーバとしての役割を果たすことができる。この機能を用いるとNetwork TaskグラフをNinfサーバ群にマップして実行することができる。

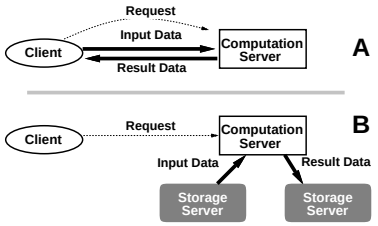

図4 Ninf サーバとストレージサーバ

3.2 メタサーバ

Ninf のメタサーバシステムは、2.3に基づいて設計されているが、効率的な実装を実現するために、モジュールは必ずしも1対1に対応していない。

メタサーバは以下のモジュールによって構成される。

- 計算リソースの負荷モニタ
- クライアントプロキシ
- リソース情報DBマネージャ
- リソース状況プレディクタ
- リソーススケジューラ

2.3 ではクライアントがスケジューラと直接交信、タスクの実行を行っているが、Ninfではクライアントプロキシがこの役割を担っている。これは、Ninfではさまざまな言語/システムにクライアントを実装することを目的のひとつとしており、クライアントプログラムに複雑な機能を実装することは避けたかったからである。

クライアントプロキシの役割で最も重要なことは、クライアントの代理としてリソーススケジューラと通信してスケジュールを実行することであるが、同時に各サーバとの間のネットワークのモニタリングも行う。サイト内すべてのクライアントからネットワークをモニタするかわりにクライアントプロキシが代表してモニタすることでモニタのコスト低減を図っている。さらに、クライアントプロキシからのネットワーク情報のリソース情報DBへの登録は、実際にスケジューリングを依頼する時点で行う。これにより、登録のコストが削減される。

サーバ間のネットワークの状況は各Ninfサーバ自身がモニタし、リソース情報DBマネージャに報告する。2.3におけるネットワークリソースモニタの機能は、クライアントプロキシと各Ninfサーバに分散して実装されていることになる。

リソース状況プレディクタには、それまでのスケジューリングの履歴が記録されており、この履歴とリソースDBに登録されている情報を用いて将来におけるリソース状況を予測する。たとえば、すでにタスクを配置したリソースは近い将来負荷が増大するだろう、といった予測が可能である。

具体的にメタサーバは次のように処理を行う(図5)。

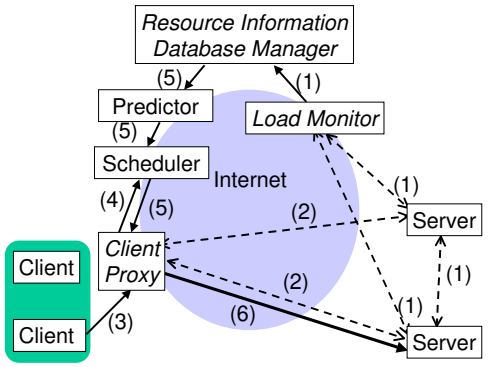

図5 Ninf メタサーバアーキテクチャ

(0) Ninfサーバは起動時に、自らの性能をベンチマークプログラムを用いて測定し、実装メモリ量/CPU数などの情報とともにリソース情報DBに登録する。

(1) ロードモニタは定期的に計算サーバの負荷情報をモニタし、それをリソース情報DBに格納する。同様にNinfサーバはNinfサーバ間のネットワークをモニタし、リソース情報DBに格納する。

(2) クライアントプロキシはクライアントプロキシと各サーバ間のネットワークの混雑度等の通信情報をモニタし、その結果をクライアントプロキシで管理する。
同様に、各サーバは互いの間のネットワーク情報をモニタし、定期的にリソース情報DBに報告する。

(3) 計算要求はNetwork Taskのグラフとして作成される。クライアントはクライアントプロキシにNetwork Taskのグラフを一括して送出する。

(4) クライアントプロキシは受け取ったNetwork Taskグラフに対するスケジュール要求をスケジューラに発行する。その際、クライアントプロキシ

とサーバ間の通信情報も通知する。

(5) スケジューラはリソース情報DBからリソース状況プレディクタを通して各計算サーバの負荷情報、サーバ間の通信情報を取得する。その情報に基づいて、Network Taskのグラフをサーバ群にマップする。
マップした結果はクライアントプロキシに送り返される。同時に、次の予測の際のヒントとしてプレディクタにも送られる。

(6) クライアントプロキシは、スケジューラから受け取ったスケジュール情報に基づいてそれぞれのNetwork Taskを個々のサーバに発行する。
この際、Network Taskの依存関係に基づいてNetwork Taskの発行順序をスケジューリングする。さらに、すべてのNetwork Taskの実行終了後、ストレージサーバに残った中間結果を削除する。

メタサーバフレームワークのモジュール群はJavaで実装されている。スケジューラ/プレディクタモジュールのアルゴリズム部分は、独立したクラスとして記述されており容易に切り替えることが可能となっている。

4. スケジューリング手法の予備的評価

メタサーバスケジューリングフレームワークの有効性を確認するために、メタサーバ上に簡単なスケジューリング手法を実装し、サンプルプログラムを実行して実行時間等を測定した。

4.1 Network Taskグラフの記述

現在はNetwork Taskグラフを生成する上位スケジューラが未実装であるため、クライアントプログラムからの計算要求としてNetwork Taskのグラフを直接記述しなければならない。これには、Ninf の transaction 機能[13]を用いた。transaction は、複数のNinf呼び出し間のデータ依存関係を自動的に検出してNetwork Taskのグラフを作成する機能をもつ。下の例は行列 A, B, C, D を掛け合わせるコードである。

```
Ninf_transaction_begin();
Ninf_call("mmul", A, B, E);
Ninf_call("mmul", C, D, F);
Ninf_call("mmul", E, F, G);
Ninf_transaction_end();
```

この場合図6のようなNetwork Taskグラフが生成される。

4.2 スケジューリング手法

Network Taskのグラフをスケジュールするには、計算の配置だけでなく計算間でやり取りされる中間データの配置を行う必要がある。図6の例で言えば、3つの mmul の計算と データ E,F の配置を行うことになる。データ A,B,C,D,G は クライアントに存在しなければならないので、スケジューリングの対象外となる。ここでは以下の2通りの手法で、計算、中間データの配置を行った。

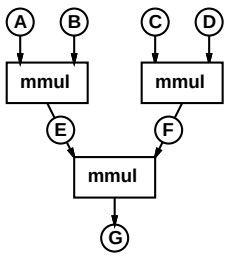

図6 トランザクションによって生成されるNetwork Taskグラフ

SimpleScheduling 個々の関数を個別にスケジューリングする。その関数を実装しているサーバのうち、もっとも実行時間が短くなると予測されるものを採用する。データはそのデータを出力する計算サーバに配置する。採用したサーバはプレディクタに通知されるため、特定のサーバに集中してしまうことはない。

FlowScheduling データフローをパスに切り分け、パス単位で配置を行う。基本的に一つのパス上に存在する計算/データをすべて同一のプロセッサに配置する。データはそのデータを出力する計算サーバに配置する。上記と同様に、採用したサーバをプレディクタに通知することで、負荷の集中を防いでいる。

FlowScheduling の具体的な手順を以下に示す。

(1) データフローグラフに含まれるすべてのパスをとりだす。

(2) 1のパスの中で最も計算コストが高いものを選び、パス中に存在する関数をすべて実装しているサーバのうち、もっとも「性能のよいサーバ」に配置する。

(3) 2で配置したパスをデータフローグラフから取り除く

(4) グラフがなくなるまで 1-3を繰り返す。

このアルゴリズムは、データ依存関係のある計算をなるべく同一の計算リソースに配置することで、計算サーバ間のデータ通信を削減することを主眼としている。図7にこの様子を示す。

図7:Aのグラフを対象とする。矩形が計算を示し、円形がデータを示している。このグラフには 1-4 の4本のパスが存在している(図7:B)。ここで パス2の計算量が最大であるとすると、パス2上の計算を一括して一つのCPUに割り振り、データフローグラフから取り除く(図7:C)。残ったグラフに同様の操作を行うと最終的に3つのグラフに分割される(図7:D)。

4.3 対象プログラム

対象プログラムとして、linpackのdgefaとdgeslを用いた。これらはそれぞれ、行列のLU分解とその結果を用いた求解を行うルーチンである。これらのルー

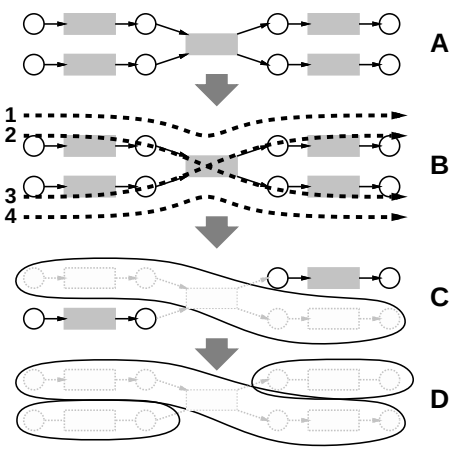

図7　FlowScheduling

チンを二つ連続して呼び出すことで、連立一次方程式の解を求める。両者の通信量/計算量を表1に示す。ここで n は行列のサイズである。

表1　dgefa とdgesl の特性

	通信量(入力)	通信量(出力)	計算量
dgefa	$8n^2$	$8n^2 + 4n$	$\frac{2}{3}n^3$
dgesl	$8n^2 + 12n$	$8n$	$2n^2$

実験では、これらの関数を一組として、複数組同時に発行する。下に疑似コードを示す。

```
Ninf_transaction_begin();
for (i = 0; i < TIMES; i++){
  Ninf_call("dgefa", A[i], n, &info);
  Ninf_call("dgesl", A[i], n, B[i]);
}
Ninf_transaction_end();
```

これによって生成されるNetwork Task グラフは図8のようになる。

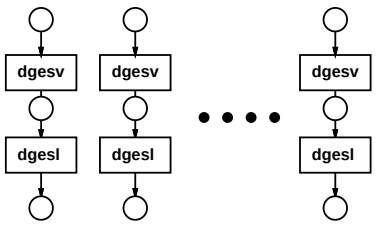

図8　対象プログラムのNetwork Task グラフ

表1からも分かるとおり、クライアントとdgefa、dgefa とdgesl の間に存在するデータ依存のデータサイズは オーダ n^2 とかなり大きい。クライアントとdgefa の間のデータ転送は不可避であるが、dgefa とdgesl の間のデータ転送は同じサーバに配置することで転送時間を削減することができる。従ってスケジューリングのポイントはペアとなっているdgefa とdgesl を同じサーバに配置することとなる。

4.4　実 験 環 境

実験環境として、電総研(ETL)、東工大(TIT)、お茶大(Ocha)の3つのサイトを用いた。電総研にクライアントおよびスケジューラ等のメタサーバのモジュールを設置し、東工大にはサーバA を、お茶大にはサーバB を設置した。それぞれのサーバの性能、及びサイト間の計測時のスループット、ping によるレイテンシを図9に示す。

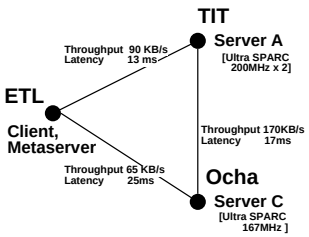

図9　実験環境

スループットの計測は100秒おき、ロードアベレージの測定は200秒おきに行っている。予測はロードアベレージに関してのみ行った。

4.5　実 験 結 果

測定は、行列サイズを 100, 300, 500, 700、並列して呼び出すdgefa とdgesl のペアの数を1,3,5 として行った。表2に測定結果を示す。上段がSimpleScheduling、下段がFlowScheduling の結果である。

実行時間はクライアントが観測した実行開始から終了まで時間である。全通信時間はすべての計算の入出力にかかった時間の総計である。同様に、全計算時間はすべての計算実行の総計である。

全般に、FlowScheduling のほうが実行時間が短縮されていることがわかる。

サーバA とサーバB が採用された回数と比率、及びdgefa とdgesl が同一のサーバに割り振られた回数と比率を表3に示す。

サーバ割り当ての比率は Simple も Flow も完全に同じになった。両者ともCPU 性能とロードにしたがってスケジュールしているため、同じ傾向になることは予期したとおりであるが、全く同一になったのは偶然である。サーバA に割り振られる量が サーバB の2倍以上なのは、図9に示す通りサーバA のCPU がサーバB のCPU よりも高速である上、サーバA が2CPU であるためである。

dgefa とdgesl が同一サーバに割り振られる確率はFlowScheduling が100%であるのに対し、Simple では 45.2%にとどまっている。

4.6　考　　察

図10に 5 並列実行時の FlowScheduling と Sim-

表2　スケジューリングの測定結果
(上段:SimpleScheduling, 下段:FlowScheduling)

size	回数	実行時間[s] 平均/標準偏差	全通信時間[s] 平均/標準偏差	全計算時間[s] 平均/標準偏差
100	1	7.26/5.43	3.06/2.71	0.02/0.00
	3	12.44/3.81	10.76/4.22	0.07/0.00
	5	14.84/4.83	16.80/5.43	0.12/0.00
300	1	14.19/8.91	10.53/6.91	0.64/0.04
	3	19.64/3.94	26.37/7.60	2.06/0.17
	5	25.75/3.86	52.96/8.99	4.62/0.52
500	1	24.21/16.04	17.43/16.24	3.50/0.22
	3	46.05/17.68	71.15/34.04	14.19/2.88
	5	64.19/10.51	137.98/22.22	31.36/5.89
700	1	47.29/28.21	32.89/23.99	13.02/4.48
	3	99.63/28.22	132.01/52.00	48.53/10.29
	5	131.05/14.13	309.65/43.42	90.19/12.50
100	1	2.24/1.88	0.51/0.41	0.02/0.00
	3	4.98/3.44	2.45/1.55	0.07/0.00
	5	8.74/6.03	5.54/3.72	0.13/0.01
300	1	7.16/6.24	4.36/5.26	0.64/0.04
	3	14.38/4.84	14.09/3.22	2.12/0.19
	5	16.71/3.79	25.02/6.08	4.58/0.60
500	1	18.95/12.80	12.63/11.07	3.54/0.23
	3	30.97/9.59	34.53/8.74	11.93/1.95
	5	35.77/6.89	70.73/13.54	29.47/2.29
700	1	45.76/8.76	31.10/6.33	11.99/3.77
	3	56.16/3.87	98.39/13.86	39.80/2.68
	5	67.98/11.35	119.64/23.70	102.90/8.14

表3　各サーバの呼び出された回数と比率

スケジューラ	サーバA	サーバB	同一サーバ
Simple	524(72.8%)	196(27.2%)	166(46.1%)
Flow	524(72.8%)	196(27.2%)	360(100%)

pleScheduling の所要時間の差を示す。

サイズが大きくなるほど、実行時間の差が大きくなっていくことがわかる。これは、FlowScheduling では確実に dgefa と dgesl が同じサーバに配置されるのに対して、SimpleScheduling では半分以上のケースで別のサーバに配置されるため、サーバ間でデータ転送が生じるが、このスケジューリング失敗に対するペナルティが転送される配列のサイズに従って拡大していくためである。全通信時間の差も同様にサイズに従って大きくなっていくことがわかる。

このことから、グローバルコンピューティング環境ではデータ転送の削減が重要であり、特に大規模なデータ転送を伴うプログラムの実行においては、データ転送量に着目したスケジューリングアルゴリズムが有効であることが分かる。

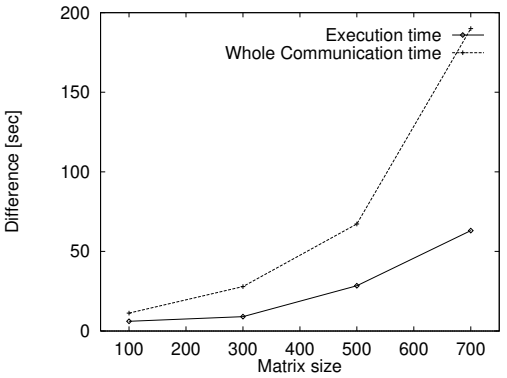

図10　5並列実行時の所要時間の差

5. まとめと今後の課題

本稿では、グローバルコンピューティングにおける基本的なスケジューリングフレームワークとして上位スケジューラと下位スケジューラに分離する枠組みを示すとともに、その下位フレームワークの一例として Ninf システムのメタサーバスケジューリングフレームワークの実装を示した。さらにメタサーバシステム上に簡単なスケジューリングアルゴリズムを実装し、本スケジューリングフレームワークの可能性を確認するとともに、スケジューリングアルゴリズムに関する評価を行った。評価の結果、データ依存関係を考慮しデータの移動量を削減するようスケジューリングを行う必要性が明らかになった。

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の研究はまだその端緒についたばかりであり、多くの課題が残されている。

高レベルスケジューラの開発　本稿で提唱したスケジューリングフレームワークを活用するには、アプリケーションを Network Task に分割する高レベルのスケジューラが必要になる。既存のアプリケーションスケジューラを参考に実装をすすめたい。また、Network Task グラフをユーザが直接記述することも考えられるが、その場合には AVS[14] に見られるような GUI によってサポートする必要があるだろう。

低レベルスケジューラの整備　本稿で示したスケジューラはサーバ間の通信リソース状態を十分考慮しているとはいえず不十分である。より高度なスケジューリングアルゴリズムを実装する必要がある。また、今回評価に用いた問題は人工的な問題であり、スケジューリング手法の評価に適切であるとはいいがたい。実用的問題での評価が望まれる。

プレディクタの整備　本稿で用いたプレディクタは割り当てられたサーバのロードを一時的に上昇させるだけのものである。グローバルコンピューティ

ングの性能評価モデルに基づくシミュレータ等[15]を用いたより予測を行うことが必要である。

Network Taskの拡張 本稿で提案したNetwork Taskでは比較的単純なデータフローグラフしか表現できず、任意の並列アプリケーションをNetwork Taskに分割することはできない。より一般的なタスク表現形をみいだし、それを扱えるようスケジューリングシステムを拡張する必要がある。

リソースの表現とリクエストの表現 現在の枠組みでは、リソースに関するpreferenceが設定できない。たとえば、課金するサーバとそうでないサーバがある場合のスケジューリングポリシなどをユーザが指定できるようにする必要がある。

ストレージサーバのセキュリティ ストレージサーバ上のデータは何らかの方法で保護する必要がある。セキュリティは計算サーバでも問題になるが、この場合のアタックがせいぜいサービス妨害攻撃程度である。これに対しストレージサーバ上のデータへの不正アクセスは直接的にユーザのプライバシをおかすこととなり、より深刻なものとなりうる。グローバルコンピューティングでは、あるストレージサーバに対して書き込んだ計算サーバとは別の計算サーバが読み出すことが一般的であるし、計算サーバ自体は複数のサーバに共有されるので、計算サーバに対してセキュリティポリシを設定しても無意味である。セキュリティポリシは、Network Taskグラフによって構成される計算のセッション全体に対して設定しなければならない。これを実現するためには、スケジューラがセッション全体に対して1つの鍵を生成し、これを計算実行時に計算/ストレージサーバに配るという手法が考えられる。

キューイングシステムとの協調 スーパコンピュータ等の高速計算機の多くは、何らかのバッチキューイングシステムを用いて運用されている。このため、投入したタスクがいつスケジュールされるかはシステムの状態に依存し、一般に不明である。グローバルコンピューティングにこれらの高速計算機を組み込むためにはそのスケジューリングシステムがバッチキューイングシステムとなんらかのかたちで協調して動作する必要がある。☆

参 考 文 献

1) Ninf: Network Infrastructure for Global Computing. http://ninf.etl.go.jp/.
2) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proceedings of Super Computing '96* (1996).
3) Grimshaw, A. S., Wulf, W. A., French, J. C., Weaver, A. C. and Jr., P. F. R.: Legion: The Next Logincal Step Toward a Natiowide Virtual Computer, CS94-21, University of Virginia (1994).
4) Arbenz, P., Gander, W. and Oettli, M.: The Remote Computation System, Technical Report 245, ETH (1996).
5) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications* (1997).
6) Berman, F., Wolski, R., Schopf, S. F. J. and Shao, G.: Application-Level Scheduling on Distributed Heterogeneous Networks, *Proceedings of Super Computing '96* (1996).
7) Weissman, J. and X.Zhao: Scheduling Parallel Applications in Distributed Networks, *Journal of Cluster Computing* (1997).
8) Czajkowski, K., Foster, I., Kesselman, C., Karonis, N., Martin, S., Smith, W. and Tuecke, S.: A Resource Management Architecture for Metacomputing Systems, *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing* (1998).
9) Feitelson, D. and Rudolph, L.: *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, Springer.
10) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proceedings of HPDC-7 (to apper)* (1998).
11) Abramson, D. and Giddy, J.: Scheduling Large Parametric Modeling Experiments on a Distributed Meta-computer, *Proceedings of PCW'97* (1997).
12) Abramson, D., Foster, I., Giddy, J., Lewis, A., Sosic, R., Sutherst, R. and White, N.: The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing, *Proceedings of ACSC97* (1997).
13) 中田秀基, 高木浩光, 松岡聡, 長嶋雲兵, 佐藤三久, 関口智嗣: Ninfによる広域分散並列計算, 並列処理シンポジウムJSPP'97論文集, pp. 281–288 (1997).
14) AVS: . http://www.avs.com/.
15) 竹房あつ子, 合田憲人, 小川宏高, 中田秀基, 松岡聡, 佐藤三久, 関口智嗣, 長嶋雲兵: 広域計算システムのシミュレーションによる評価 – Ninfシステムの広域分散環境でのジョブスケジューリング実現に向けて –, 並列処理シンポジウムJSPP'98論文集, pp. 127–134 (1998).

☆ SC'97においてGlobusシステムを用いたGUSTOのデモがおこなわれたが、Globus関連のジョブを最優先でスケジューリングするようにキューイングシステムを設定していた。この手法を常用するわけにはいかないだろう。