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

竹房　あつ子[†,††]　松岡　聡[††,†††]

広域ネットワークに接続された計算資源を 1 つの計算基盤として提供する Grid システムが将来の高性能計算アプリケーションの計算基盤として注目されている．Grid システムとして，ネットワーク上で計算資源とともにサービスの提供を可能にする Network-enabled Server（NES）が複数提案されている．NES は一般的にクライアント・サーバ型アーキテクチャとなっており，分散した Grid 計算資源上にサーバを用意する．しかしながら，複数サーバ，複数のクライアントを想定した Grid のスケジューリングに関する議論が十分に行われていない．また，将来の NES システム運用での課金に伴い，Grid ユーザはジョブ実行時間を最短にすることから最小コストの資源群を利用して規定時間内に処理を終了させることを要求するようになる．本稿ではデッドラインスケジューリングに着目し，その性能特性を Grid のスケジューリング手法の評価システム Bricks を拡張して調査した．まず，複数サーバ，複数クライアントを想定した，デッドラインスケジューリングアルゴリズムを紹介するとともに，そのアルゴリズムの性能を高めるメカニズム，Load Correction と Fallback を提案する．次に，Bricks システムを用いたシミュレーションによる評価より，Grid 上での NES システムのデッドラインスケジューリングの有効性を示す．

Performance of a Deadline Scheduling Scheme on the Computational Grids

Atsuko Takefusa[†,††] and Satoshi Matsuoka[††,†††]

The *Computational Grid* is a promising platform for the deployment of high-performance computing applications. A number of projects have addressed the idea of software as a *service* on the network. These systems usually implement client-server architectures with many servers running on distributed Grid resources and have commonly been referred to as *Network-enabled servers* (NES). An important question is that of *scheduling* in this multi-client multi-server scenario. Note that in this context most requests are computationally intensive as they are generated by high-performance computing applications. The Bricks simulation framework has been developed and extensively used to evaluate scheduling strategies for NES systems. In this paper we discuss a *deadline scheduling* strategy that is appropriate for the multi-client multi-server case. We propose a simple deadline scheduling algorithm and then augment it with Load Correction and Fallback mechanisms which could improve the performance of the algorithm in our context. We then give simulation results and draw conclusion on the value and feasibility of deadline scheduling for NES systems on the Grid.

1. はじめに

広域ネットワークに接続された計算資源を 1 つの計算基盤としてユーザに提供する，Grid と呼ばれる高性能広域計算システムの研究が盛んに行われている[1),2)]．Grid コンピューティングソフトウェアに関する基礎技術は研究開発だけでなくその標準化[2)] および運用も開始されている[3)~7)]．一方，Grid 上で高性能計算アプリケーションを実行する際の重要な課題としてスケジューリングが挙げられる[8)]．従来のスケジューリングでは，並列アプリケーションの実行時間 (makespan) の短縮など，1 人のユーザの 1 アプリケーションの実行をいかに速く処理するかに重点がおかれていた[9)~13)]．

Grid システムには，ネットワーク上で計算資源とともにサービスの提供を可能にする Network-enabled Server（NES）[3),7),14),15)] がある．NES システムは一般的にクライアント・サーバ型アーキテクチャとなっており，RPC ベースのプログラミングモデルをユーザに提供する．科学技術計算には独立した多数のタスクからなる Parameter Sweep 型アプリケーションが多数あり[16),17)]，これらのアプリケーションは NES の提供するプログラミングモデルと適合しやすい．よって Global Grid Forum[2)] では NES の標準化が進められている．

NES の提供する計算資源群を同時に複数のユーザが利用する状況では，複数クライアントジョブのスケジューリングが重要な課題となる．広域ネットワーク上の特定

† 日本学術振興会 JSPS
†† 東京工業大学 Tokyo Institute of Technology
††† 科学技術振興事業団 JST

多数のユーザに対して提供する Grid 上の計算資源はその性能や質が多岐に渡るため，将来的な Grid システムの運用における課金方法もそれに応じてバラエティに富むようになる．よって，ユーザは従来の計算機利用のように単純な経験則に基づいて最短実行時間を目標としてジョブを投入するのではなく，規定時間（デッドライン）内に処理を終えるための最小コストの計算機群を要求するようになる．すでにこのような経済モデルの概念を利用したスケジューリング手法が複数提案されている[18)~20)]が，その有効性や利用可能性は明らかでない．

デッドラインスケジューリングは経済モデル下でのスケジューリング手法の 1 つである． NES システム Nimrod[21)] では，性能や計算資源の需要の高さから計算資源にプライオリティをつけ，はじめはプライオリティの低い資源にタスクを割り当てて，各タスクの実行時間と計算資源との関係から計算資源に対する評価を改めて，多数のタスクからなるアプリケーションの全体の実行時間がユーザの求めるデッドラインを超えないように次のタスクを割り当てていくデッドラインスケジューリングを実現している．

本稿では経済モデル下でのデッドラインスケジューリングに着目し，デッドラインスケジューリング手法の提案とその性能特性を調査する．評価では， Grid ユーザが発行するジョブにはあらかじめデッドラインが指定されているものとし，デッドラインスケジューリングの失敗（指定されたデッドラインまでに計算が終了しない）率を少なくすることを目的とする．評価には Grid コンピューティングのスケジューリング手法の評価システム Bricks[22),23)] を改良して用いる．

まず，柔軟かつ容易にスケーラブルな評価環境を設定可能にする Bricks のシミュレーションモデルについて説明する．次に，デッドラインスケジューリングの失敗率の軽減を目指す簡単なスケジューリングアルゴリズムを紹介するとともに，そのアルゴリズムの性能を高めるメカニズム， Load Correction と Fallback を提案する． Load Correction は Grid モニタリングシステムにより得られた資源情報をスケジューリング結果を用いて補正するものである． Fallback はサーバにジョブの入力データが到着した際にそのジョブがデッドラインを超えるかどうか予測し，超えると判断した場合は別のサーバにジョブを再投入させるメカニズムである．そして，特定多数のユーザが Grid の計算資源群に対してシングルタスクのジョブを複数投入するシンプルなモデルを想定し， Grid の負荷状況，計算資源コスト，性能予測の見積もり，および各スケジューリングメカニズムに対するデッドラインスケジューリングの有効性を調査する．本稿では計算資源の性能をその資源のコストとみなす簡単な経済モデルを適用する．評価では， PC クラスタ上で Bricks シミュレーションを数千回実行した．シミュレーションより，従来の実行時間を最短にすることを目的としたスケジューリング手法はデッドラインスケジューリングよりユーザのジョブが利用する計算資源のコストが高くなる，保守的にデッドラインを見積もるとスケジューリングの失敗率が下がる一方資源コストが高くなる， Load Correction メカニズムは有効でなく，

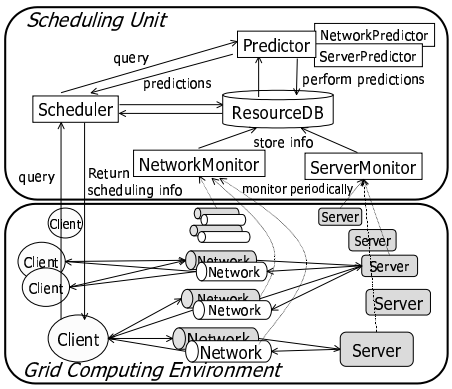

図 1 Bricks システムアーキテクチャ

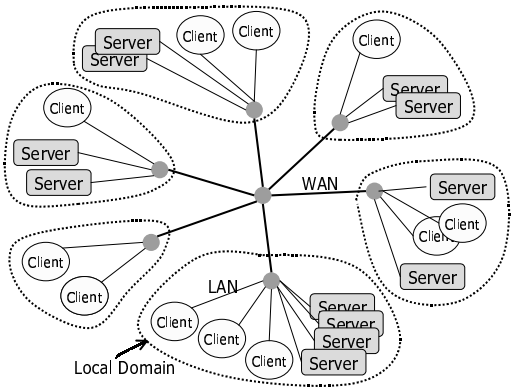

図 2 ネットワークトポロジ

Fallback は失敗率の軽減に非常に有効であることが明らかになった．

2. Bricks システムとその拡張

Bricks システムは， Grid システムのスケジューリング手法およびそのフレームワークの評価基盤を提供する Java で実装された離散シミュレータである．図 1に Bricks のシステムアーキテクチャを示す． Bricks は Grid をシミュレートする Grid Computing 環境と Scheduling Unit からなる． Grid Computing 環境では， Grid トポロジ，資源モデル（負荷トレース，待ち行列等），クライアントモデル（リクエスト到着時間等）のシミュレーションコンポーネントセットを提供する． Scheduling Unit では，スケジューラ，プレディクタ，資源モニタ，資源情報データベースなどの Grid でのスケジューリングに要する各モジュールを提供する[8)]． Bricks のユーザは Bricks が提供する宣言的なスクリプトにより，柔軟に環境設定を行い，多様なスケジューリングアルゴリズムの再現性のある性能評価実験を行うことができる．また， Scheduling Unit の各モジュールはユーザの実装したスケジューリングアルゴリズムや既存スケジューリングモジュールと容易に置換可能であり，その評価や機能試験を Bricks 上で行うことができる[22)]．

ただし，現状のBricksシステムではスケーラビリティが十分でない．よりスケーラブルで現実的なGrid環境での評価を可能にするため，Bricksシステムを拡張して図2のような木構造のネットワークモデルを取り入れた．クライアント・サーバ間の各ネットワークは中間ノードとLAN，WANの異なる通信バンド幅のネットワークにより構成される．この拡張により，ネットワークシミュレーションのための通信バンド幅およびトポロジを生成する既存システムGT-ITM[24]，tiers[24]，BRITE[25]等で生成されたネットワークトポロジが利用可能となる．

2.1 Gridシミュレーションモデル

Bricksでは，Grid上の計算資源（サーバ）と，クライアント・サーバ間のネットワークをそれぞれ待ち行列を用いて表現する．待ち行列で表現されるネットワーク，サーバモデルには以下の2つがある．

外乱データ／ジョブモデル　ネットワーク／サーバの混雑度をそのネットワーク／サーバに到着するGridシステムの利用者以外からの発行される外乱データ／ジョブの流量により表す．

トレースモデル　実環境で得られた通信スループット／サーバ負荷の観測値のトレースにより，ネットワーク／サーバの混雑度を表す．

外乱データ／ジョブモデルは，パラメータの設定のみでシミュレーションが行えるものの，特にネットワークにおける通信スループット値の分散の調節が難い．また，シミュレーションの粒度を細かくして精度を上げるとシミュレーションコストが非常に高くなってしまう．一方，トレースモデルは実環境の変動をそのまま表現でき，シミュレーションコストが低いという利点があるが，事前にトレースデータを観測する必要があるため，シミュレーションで想定するネットワークトポロジが制限される．また，大規模評価環境ですべてのネットワーク／サーバに対してトレースモデルを用いると，大量のトレースデータを用意する必要がある．

本研究の評価実験では，サーバにはポアソン到着を用いた外部ジョブモデル，ネットワークにはトレースモデルを用いた．ネットワークトレースのトラフィックモデルには，Paxsonにより提案されたFractional Gaussian Noise（FGN）によるSelf-Similarモデル[26]を採用した．ネットワークのモデリングでは一般にポアソン到着を想定することが多いが，LANおよびWANでのネットワークトラフィックの性質はむしろSelf-Similarityがあるといわれており[27]，シミュレーション環境下でより現実に近い性質をもつネットワークトラフィックを表現するためにこのモデルを採用した．このモデルでは，まずSelf-Similarな性質をもつFGNのパワースペクトルを生成し，離散時間フーリエ変換で合成することによりサンプルパスを生成する．次に，このサンプルパスを指定した平均通信スループット，分散に従うように変形し，シミュレーションで用いる通信トラフィックトレースを生成する．よって，少数のパラメータの設定のみで様々なトレースを生成することができる．一方，高精度のサーバ負荷モデルがないため，サーバには外部ジョブモデルを用いてランダムに外部ジョブを生成させて負荷をシミュレートした．

3. デッドラインスケジューリング

既存NESシステム（Ninf[3]，NetSolve[7]）では，ユーザの計算要求に対してその計算時間が最短となるようにサーバを割り当てていく．これはユーザのデッドラインの要求を考慮しない積極的なアプローチ（以後Greedyとする）である．しかしながら，NESシステムがGrid上で広く利用されるようになると，デッドラインを考慮した計算資源選択が重要になる．デッドラインスケジューリングはユーザの要求するデッドラインまでにジョブの処理を終了させることを目的とした手法である．

3.1 スケジューリングアルゴリズム

本研究で用いたデッドラインスケジューリングのアルゴリズムを以下に示す．このアルゴリズムでは，複数ユーザの発行するジョブに対して適切なサーバを選択し，デッドラインを超える総ジョブ数を軽減させることを目的とする．

(1) ジョブのデッドラインまでの時間$T_{until\ deadline}$を算出する．

$$T_{until\ deadline} = T_{deadline} - Tnow \quad (1)$$

$T_{deadline}$はジョブに定められたデッドラインの時刻，T_{now}は現在の時刻を表す．

(2) 利用可能なサーバS_i（$0 < i \leq n$）にジョブを投入した場合のジョブ処理時間の予測値T_{S_i}を算出する．

$$T_{S_i} = W_{send}/P_{send} + W_{recv}/P_{recv} + W_s/P_{serv} \quad (2)$$

W_{send}，W_{recv}，W_sはそれぞれジョブの送信データ量，受信データ量，論理演算数を表し，P_{send}，P_{recv}，P_{serv}はクライアントからサーバ，サーバからクライアントへのネットワークバンド幅の予測値およびサーバでの処理速度の予測値を表す．Grid環境ではネットワークやサーバ計算機の負荷の変動が大きいため，一般にそれらの状況を定期的にモニタリングし，ジョブが割り当てられる際のネットワーク／サーバの負荷を予測して，その予測値をスケジューリングに利用する[28]．

(3) 処理時間の目標値T_{target}を算出する．

$$T_{target} = T_{until\ deadline} \times \mathsf{Opt} \quad (0 < \mathsf{Opt} \leq 1) \quad (3)$$

パラメータOptはGrid上での性能予測に対する精度の期待度を表す．Gridモニタリングシステムの提供する予測値には予測エラーが含まれているため，どの程度保守的に予測の失敗を見積もるかをOptにより調節する．Optを小さく設定することは予測失敗時のリスクを大きく見積もることを意味し，Optを1に設定することは予測の精度が非常に高いと判断したことを意味する．

(4) 利用可能なサーバの中から処理時間の目標値を超えず，目標値に最も近い処理時間となるサーバS_iを選出する．

$$Diff = T_{target} - T_{S_i} \geq 0$$
かつ，$Diff$が最小 (4)

ただし，$Diff \geq 0$ となる S_i が 1 つもない場合は，$Diff$ の絶対値が最小となるサーバを選出する．

3.2 Load Correction メカニズム

Grid 上のスケジューラは一般に NWS[28] 等から得られたリソースの負荷の測定値，予測値を利用する．前節で述べたアルゴリズムでもこれらの情報を用いている．しかしながら，実際にはサーバの利用状況に関する付加的な情報も保持している．すなわち，スケジューリングによりジョブが割り当てられたサーバの負荷は，そのジョブが投入されると高くなることがあらかじめ予測できる．一方，Grid のモニタリングシステムは定期的にモニタリングを行うため，即座にその負荷の変化を察知することができない．

スケジューリングアルゴリズムの性能を改善するため，スケジューリングの履歴情報を利用してモニタリングシステムから得られた負荷予測値を補正し，それをスケジューリングの際に用いる（以後 Load Correction と呼ぶ）．この手法は NES システムのオンラインスケジューラに適用できる．

Load Correction では，モニタリングシステムから得られたサーバ S_i の負荷予測値を $Load_{S_i}$ とすると，その補正値 $Load_{S_i corrected}$ は次のように求める．

$$Load_{S_i corrected} = Load_{S_i} + numjobs_{S_i} \times pload \quad (5)$$

$numjobs_{S_i}$ はその Grid システムのスケジューラにより S_i に割り当てられたジョブの総数，$pload$ は補正の大きさを調整する任意の値である．本稿の評価では，すべて $pload = 1$ としている．これは，評価で想定するアプリケーションは計算主体であり，一般に計算が主体となるタスクはその計算機の CPU 負荷を 1 つ高くするためである（例えば，計算主体の 3 つのタスクを実行している計算機の負荷はほぼ 3.0 となる）．

3.3 Fallback メカニズム

スケジューリングアルゴリズムでは，各ジョブに対して計算・通信時間の見積もり（予測値）を利用する．しかしながら，ジョブの実行に要する入力データの送信時間の見積もりを失敗すると，入力データがサーバに到着した後にそのジョブの実行がデッドラインまでに終了しないと予測できる場合もある．さらに，本稿の評価ではサーバは First Comes First Served（FCFS）でジョブを処理するものとしており，サーバにジョブが到着する順序が異なったために実行時間の見積もりを誤る場合もある．この問題を解決するため，サーバ自身にスケジューリングをサポートする機能をもたせる（以後 Fallback とする）．

Fallback では，ジョブの入力データがサーバに到着した際，サーバはそのジョブが要求されたデッドラインまでに終了するかどうか見積もる．サーバがそのジョブがデッドラインまでに終了しないと判断した場合，サーバはそのジョブをキャンセルしてジョブを発行したクライアントにそれを通知する．クライアントは再びそのジョブを処理するサーバをスケジューラに問い合わせ，改めて選択されたサーバにジョブを投入する．ただし，この Fallback のオーバーヘッドによりデッドライン内にジョブを終了できなくなる可能性があるため，本稿では再投入する回数を制限する．

表 1 シミュレーションで用いたパラメータ

Grid Computing 環境		
Local Domain 数	10	固定
Local Domain のノード数	5-10	一様分布
クライアントとサーバの比率	1:1	固定
平均 LAN バンド幅	50-100[Mbps]	一様分布
平均 WAN バンド幅	500-1000[Mbps]	一様分布
サーバ性能	100-500[Mops/s]	一様分布
サーバの平均負荷	0.1	固定
サーバの外乱ジョブ命令数	50[Mops]	固定
論理パケットサイズ	10[Mbits]	固定
クライアントジョブ特性		
送信 / 受信データ量	100-5000[Mbits]	一様分布
ジョブ論理演算数	1.5-1080[Gops]	一様分布
ジョブ発行間隔（Int）	60/90/120[min]	ポアソン
Deadline パラメータ（DF）	1.0-3.0	一様分布
Scheduling Unit		
モニタリング間隔	5[min]	固定
スケジューラ	Deadline /Greedy	
Opt	0.5/0.6/0.7/0.8/0.9	
Load Correction メカニズム	on / off	
Fallback メカニズム	$N_{max.\ fallbacks}$ = 0/1/2/3/4/5	

4. スケジューリング手法の評価

拡張した Bricks システムを用いてデッドラインスケジューリングアルゴリズムの性能を示す．評価では東京工業大学松岡研究室の Presto II クラスタ（Dual PentiumIII 800MHz，× 64 nodes）上で，スケジューリングアルゴリズム，クライアント，サーバ，ネットワークトポロジ，Opt，および Grid システムの負荷を変化させたシミュレーションを約 2500 回行った．Java 2 Runtime Environment (1.3.0) と Java HotSpot Client VM では，1 シミュレーション（75 ノード 24 時間）あたり 30-60[min] 要した．また，各シミュレーションの実行には APST ソフトウェア[10] を用いた．APST は独立した多数のタスクを利用可能なクラスタノードに自動的に割り当てる Grid ミドルウェアである．

4.1 評価環境

シミュレーションでは，まず従来のアプローチである Greedy アルゴリズムと第 3.1 節のデッドラインスケジューリングアルゴリズム（以下 Deadline ）を比較する．次に，パラメータ Opt，Load Correction および Fallback メカニズムのスケジューリングの失敗率と選択したサーバのコストに対する影響を比較する．第 1 節で述べたように，評価ではサーバ計算機の性能をそのサーバのコストとする経済モデルを適用する．

シミュレーションで用いるパラメータの詳細を表 1 に示す．表 1 の Grid Computing 環境におけるパラメータを用いて，5 つの異なる Grid シミュレーション環境を生成する．次節に示す評価実験結果は，この 5 つのシミュレーションの平均値である．シミュレーション実行時の LAN/WAN の通信バンド幅は，標準偏差

が各平均バンド幅の 10% となる Self-Similar トレース（第 2.1節）により決定する．また，各サーバは FCFS でジョブを処理するものとし，その負荷は平均が 0.1 となるよう外乱ジョブの平均到着間隔を設定する（第 2.1節）．この際，外乱ジョブはポアソン到着に従ってサーバに投入される．

表 1のクライアントジョブ特性のパラメータは，各シミュレーション実行時に用いられる．様々な特性をもつジョブに対するスケジューリングの性能を調べるため，クライアントの発行するジョブの論理演算量は負荷を考慮しないサーバ全体の平均性能（300）に対して実行時間が 5-60[min] となるように 1.5-1080[GOps] に設定した．また，クライアントのジョブの平均発行間隔 Int は，Grid システム全体の負荷の度合いを調節するパラメータであり，Int が長くなるにつれ負荷が低くなる．評価では，負荷が低い場合（Int= 120），中程度（Int= 90 ），高い場合（Int= 60 ）を想定したスケジューリング性能を調査する．また，各ジョブのデッドライン $T_{deadline}$ は以下のようにして決定する．

$$T_{deadline} = T_{start} + ((W_{send} + W_{recv})/B_{net} + W_s/B_{serv}) \times \mathrm{DF} \quad (6)$$

ここで T_{start} はジョブが発行された時刻，W_{send}, W_{recv}, W_s はそれぞれジョブの送信量，受信量，論理演算数を表し，B_{net}, B_{serv} は各クライアント・サーバ間のネットワークおよびサーバの最大スループットの平均値とする．また，デッドラインパラメータ DF はデッドラインまでの長さを決定する．実際の Grid システムの運用では，デッドラインの長さは各ユーザの事情により決定されるため，本シミュレーションでは $1.0 \leq \mathrm{DF} \leq 3.0$ とした．

SchedulingUnit では，5[min] ごとにネットワーク／サーバの状況をモニタリングするネットワーク／サーバモニタを用意する．ネットワークのモニタリングは各クライアント・サーバ間のネットワークに対して定期的にプローブデータを流すことで測定されるが，すべてのネットワークでプローブを同時に開始するとプローブデータ同士の衝突により測定される通信スループットが著しく低下する恐れがある．よって，シミュレーションでは巡回してプローブするネットワークモニタを用いる．実環境下で Grid の負荷状況のモニタリング・予測するシステム NWS[28] でも，同様の方式でモニタリングを行っている．また，表 1の Scheduling Unit にはスケジューリングに関するパラメータを示す．$N_{max.\ fallbacks}$ は fallback 回数の最大値である．シミュレーションでは資源モニタの観測値をその資源の負荷予測値とした．これは，現状では Grid 環境でのネットワーク／計算機負荷の予測の精度が保証されていないため，まず基礎データとしてモニタリングのみから得られた情報をもとにスケジューリングした結果を調査するためである．

4.2 評価結果

図 3にスケジューリングアルゴリズム Greedy と Deadline のデッドラインスケジューリングの失敗率を示す．Deadline では異なる Opt を設定した結果を示している（図 3では，X 軸に D-Opt と表記する）．失敗率は総

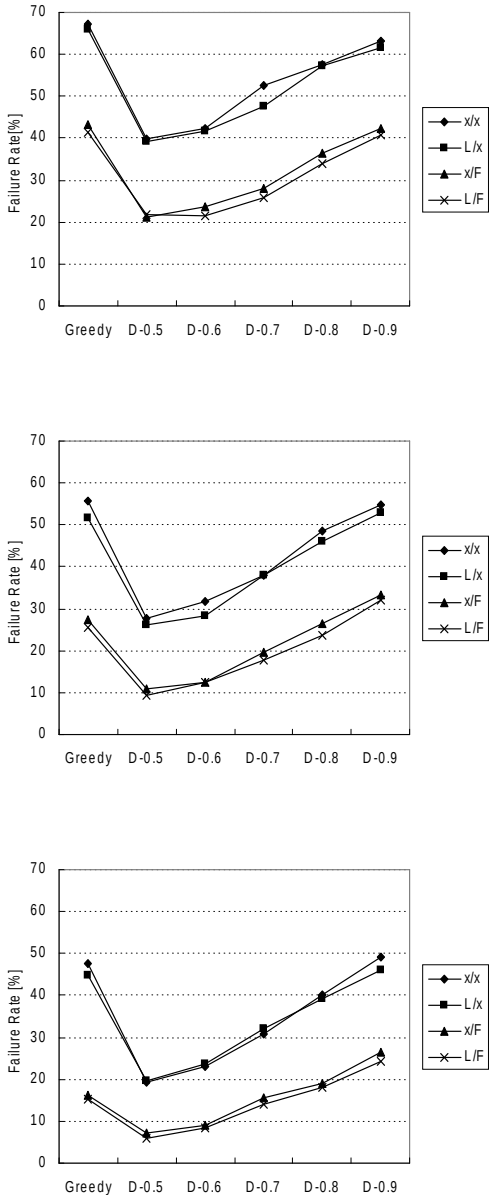

図 3 Greedy と Deadline のスケジューリング失敗率の比較（$N_{max.\ fallbacks}$ = 1，Int= 60（上），90（中），120（下））．

ジョブ数に対するデッドラインを超えたジョブ数の割合である．図 3では，第 4.1節で述べたように 3 つの負荷が異なる（Int= 60，90，120）仮想 Grid システムの結果をそれぞれ示している．各グラフ中のスケジューリング手法”x/x”，”L/x”，”x/F”，”L/F”は，X 軸に表記した各スケジューリングアルゴリズムに対して何も用いないもの，Load Correction を用いたもの，Fallback を用いたもの，Load Correction および Fallback を用いたものの結果を示している（グラフ中の ”L”は Load Correc-

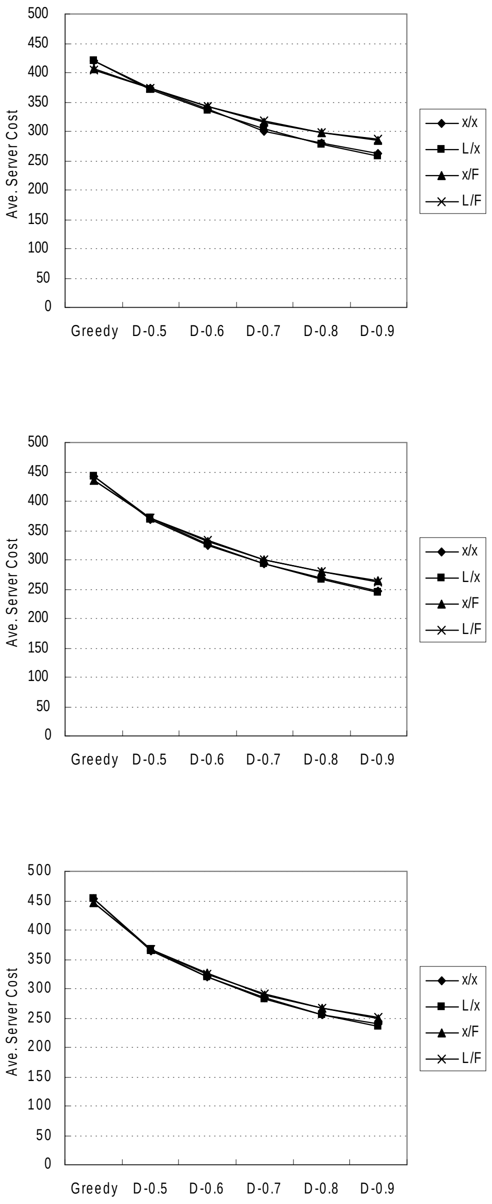

図 4 平均資源コストの比較（$N_{max.\ fallbacks} = 1$，Int= 60（上），90（中），120（下））．

tion，"F"は Fallback，"x"は各アルゴリズムを使用していない場合を表す）．図 3のシミュレーションでは，Fallback での最大 fallback 数（$N_{max.\ fallbacks}$）は 1 とする．Greedy かつ"x/x"の場合が従来のスケジューリングアルゴリズムの結果となる．

グラフより，負荷の異なるいずれの環境においても，Fallback が失敗率を低下させるために有効な手段であること，Load Correction の失敗率低下の効果は非常に小さく，多数のクライアントとサーバが NES システム上にある状況では有効ではないことが示された．また，

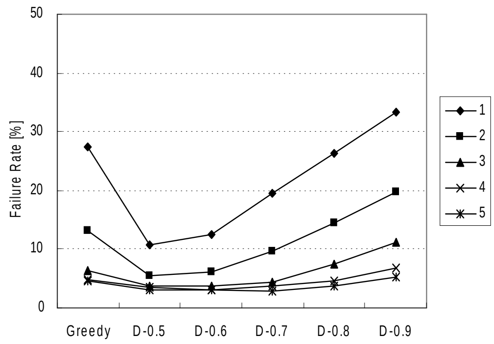

図 5 Greedy と Deadline のスケジューリング失敗率の比較（$N_{max.\ fallbacks} = 1/2/3/4/5$，Int= 90）．

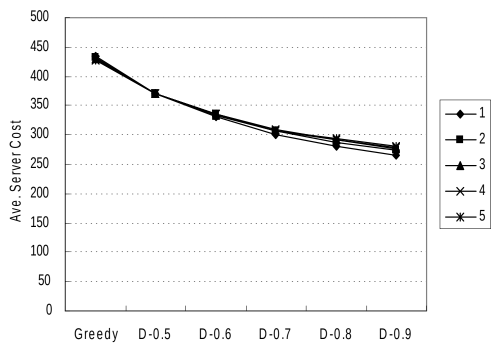

図 6 平均資源コストの比較（$N_{max.\ fallbacks} = 1/2/3/4/5$，Int= 90）．

Deadline アルゴリズムは Opt を小さく設定するとスケジューラは各ジョブに対してより厳しいデッドラインを想定するため，失敗率を低下させている．ただし，経済モデル下でのスケジューリングアルゴリズムの性能比較では，失敗率だけでなく各ジョブが割り当てられるサーバの計算資源コストを考慮する必要がある．

図 4に図 3と同じシミュレーションでの各アルゴリズムの平均資源コストを示す．グラフより，Opt を大きく設定すると平均資源コストが低下し，Greedy はいずれの Deadline アルゴリズムよりコストが高いことが分かる．すなわち，NES システムでは保守的なデッドラインスケジューリング（Opt が小さい場合）の方が，既存アルゴリズムより有効であることが示唆された．また，図 3，4より，Opt により Deadline アルゴリズムの保守性のレベルを変化させて失敗率とコストのトレードオフを調整可能にすることが示された．

図 5，6に，Load Correction を用いずに最大 fallback 数 $N_{max.\ fallbacks}$ を 1 から 5 に変化させた場合の失敗率と平均資源コストの比較結果を示す．ただし，同様の傾向を示していたため，Int= 90 の結果のみを示す．図 5より，$N_{max.\ fallbacks}$ を大きくするとすべてのスケジューリングアルゴリズムにおいて失敗率が低下

し，スケジューリング性能を向上させることが分かる．また，図6より，$N_{max.\ fallbacks}$を大きく設定した場合でも，資源コストが著しく増大しないことが分かる．よって，NESシステムのスケジューラでは，従来のスケジューリングメカニズムに加えて複数回のfallbackを行う機構を組み込むと非常に有効であることが示された．

5. 関連研究

第1節で述べたように，すでに経済モデルの概念を利用したスケジューリング手法が複数提案されている．Nimrod[21]では，Parameter Sweep型のジョブに対してself-schedulingによるデッドラインスケジューリング手法を提案・実装している．実環境での実験[21]では，デッドラインが近づくにつれ，スケジューラがコストの高い計算機を選択するようになる結果を示している．ただし，Nimrodでは1人のユーザから発行されるジョブのスケジューリングを行っている．

複数のタスクからなるParameter Sweep型アプリケーションのデッドラインスケジューリングでは，1つのタスクの処理が予測に反して処理時間が長くなった場合でも最終的なデッドラインを超えなければ問題にならないため，スケジューリングの失敗率はより低くなる．ただし，特定多数のユーザが同時にGridを利用することを想定すると，単にデッドラインに注目するだけでなく，Grid環境の負荷に応じて資源割り当ての積極さを変化させることでより低いコストの計算資源でジョブを実行できる可能性が高い☆．

より洗練された経済モデルとして，Zhao，Karamcheti[18]は，様々な管理ドメイン間のリソース共有のためのチケットと通貨を用いた共有合意メカニズムを提案している．また，G-Commerce[19]では，計算資源の提供者と消費者の間をBankが仲介して資源（コモディティ）の価格を決定し，最終的にオークションの概念を用いて各コモディティを消費者に割り当てていくスケジューリング手法を提案している．

また，Gridの評価環境ではMicroGrid[29]，Simgrid[30]が挙げられる．MicroGridはクラスタ計算機上にGlobus[4]ベースの仮想Gridシステムを構築するGridエミュレータである．実アプリケーションをそのままMicroGrid上で実行することが出来るため，アプリケーションのモデリングが不要でGrid上での試験的なアプリケーションジョブの実行に適している．ただし，ジョブの実行には実時間以上を要し，システムの制御に対する負担も大きい．

Simgridはトレースベースの離散シミュレータであり，Gridシミュレーションの基本的なコンポーネントセットを提供する．Simgridでは単一の並列アプリケーションの実行時間を最短にするタスクスケジューリングアルゴリズムの評価（処理時間の予測）を目的としている．スケジューリング手法の評価を行うには，Simgridの上の階層に別のシミュレータを実装する必要がある．

☆ 経済モデルを応用し，計算機の利用率（需要）が高い場合はその計算資源の価格を上げ，低い場合には下げる方針が複数提案されている[19),21)]．

6. まとめと今後の課題

Gridの計算基盤技術に関する研究は習熟してきたものの，そのスケジューリング手法に関する定量的な調査／解析はまだ不十分である．また，従来のGridシステムのスケジューリングではいかに速くジョブを処理するかに重点がおかれていたが，広域ネットワーク上の特定多数のユーザに対して計算資源を提供する場合，計算資源に対するアクセス権限や資源のプライオリティ等を考慮し，ユーザの求める計算性能を提供することも計算資源割り当て方針の重要な要素となる．

本稿ではデッドラインスケジューリングに着目し，簡単なスケジューリングアルゴリズムおよびスケジューリングアルゴリズムをサポートする2つのメカニズム，Load CorrectionとFallbackを提案した．Load CorrectionはGridモニタリングシステムにより得られた資源情報をスケジューリング結果を利用して補正するものである．Fallbackはサーバにジョブの入力データが到着した際にそのジョブがデッドラインを超えるかどうか予測し，超えると判断した場合は別のサーバにジョブを再投入させるメカニズムである．また，複数ユーザが複数サーバに対してジョブを投入する状況を想定し，拡張したGridコンピューティングのための評価システムBricks上でシミュレーションによるスケジューリングアルゴリズムの評価実験を行った．評価では，以下の知見を得た．

- Greedyアルゴリズムは Deadline より選択されるサーバのコストが高い．
- 保守的にデッドラインを見積もる（Optが小さい場合）と，資源コストは高くなるがスケジューリングの失敗率が低下する．
- Load Correctionメカニズムの性能向上効果は少なく，有益ではない．
- Fallbackメカニズムは失敗率を大きく低下させ，複数fallbackを行った場合でも平均資源コストが大きくならない．

よって，将来のGrid NESシステムで複数ユーザにより計算資源の競合が起こる場合には，複数回のfallbackをサポートするデッドラインスケジューリングが非常に有効であることが明らかになった．

今後の課題として，より洗練された経済モデルでのスケジューリングアルゴリズムの評価をBricksシステムを改良して行い，シミュレーションでその有効性を調査する．また，シミュレーションから得られた知見をもとに，実際のGridシステムNinf[3]におけるデッドラインスケジューラを実装し，実用に向けて実環境での挙動を調査する．

謝辞 本研究を進めるにあたりご助言いただいたUCSDのHenri Casanova氏，Francine Berman教授，日頃ご討論いただくNinfプロジェクトの皆様，ならびにPCクラスタ利用に際して多大なご助力を頂いた松岡研究室のみなさんに深く感謝致します．本研究は文部科学省科学研究費補助金および科学技術振興財団（さきがけ２１）により支援されている．

参 考 文 献

1) Foster, I. and Kesselman, C.(eds.): *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (1999).
2) Global Grid Forum: `http://www.gridforum.org/`.
3) Ninf: `http://ninf.etl.go.jp/`.
4) Globus: `http://www.globus.org/`.
5) Legion: `http://www.cs.virginia.edu/~legion/`.
6) Condor: `http://www.cs.wisc.edu/condor/`.
7) NetSolve: `http://www.cs.utk.edu/netsolve/`.
8) Berman, F.: *The Grid, Blueprint for a New computing Infrastructure*, Morgan Kaufmann Publishers, Inc., chapter 12 (1998). Edited by Ian Foster and Carl Kesselman.
9) Berman, F., Wolski, R., Figueira, S., Schopf, J. and Shao, G.: Application-Level Scheduling on Distributed Heterogeneous Networks, *Proceedings of SC96* (1996).
10) Casanova, H., Obertelli, G., Berman, F. and Wolski, R.: The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid, *Proceedings of SC2000* (2000).
11) Goux, J., Kulkarni, S., Linderoth, J. and Yoder, M.: An Enabling Framework for Master-Worker Applications on the Computational Grid, *Proceedings of HPDC-9*, pp. 43–50 (1999).
12) Turgeon, A., Snell, Q. and Clement, M.: Application Placement Using Performance Surfaces, *Proceedings of HPDC-9* (1999).
13) Schopf, J. and Berman, F.: Stochastic Scheduling, *Proceedings of SC99* (1999).
14) Kapadia, N., Forter, J. and Brodley, C.: Predictive Application-Performance Modeling in a Computational Grid Environment, *Processings of the HPDC-8* (1999).
15) Czyzyk, J., Mesnier, M. and Moré, J.: NEOS: The Network-Enabled Optimization System, Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory (1996).
16) Stiles, J., Bartol, T., Salpeter, E., and Salpeter, M.: Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes, *Computational Neuroscience*, pp. 279–284 (1998).
17) Rogers, S.: A Comparison of Implicit Schemes for the Incompressible Navier-Stokes Equations with Artificial Compressibility, *AIAA Journal*, Vol. 33, No. 10 (1995).
18) Zhao, T. and Karamcheti, V.: Expressing and Enforcing Distributed Resource Sharing Agreements, *Proceedings of SC2000* (2000).
19) Plank, J., Wolski, R., Brevik, J. and Bryan, T.: G-Commerce: The Study and Building of Computational Economies for the Computational Grid (2000). Workshop on Clusters and Computational Grids for Scientific Computing `http://www.cs.utk.edu/dongarra/lyon2000/lyon-2000.html`.
20) Buyya, R., Abramson, D. and Giddy, J.: An Economy Driven Resource Management Architecture for Global Computational Power Grids, *Proceedings of PDPTA2000* (2000).
21) Abramson, D., Giddy, J. and Kotler, L.: High Performance Parametric Modeling with Nimrod/G: Killer Application for the Glocal Grid?, *Proceedings of IPDPS2000* (2000).
22) 竹房, 合田, 松岡, 中田, 長嶋: グローバルコンピューティングのスケジューリングのため の性能評価システム, 情報処理学会論文誌, Vol. 41, No. 5, pp. 1628–1638 (2000).
23) Bricks: `http://ninf.is.titech.ac.jp/bricks/`.
24) Calvert, K. L., Doar, M. B. and Zegura, E. W.: Modeling Internet Topology, *IEEE Communications* (1997).
25) Medina, A., Matta, I. and Byers, J.: On the Origin of Power Laws in Internet Topologies, *Computer Communication Review*, Vol. 30, No. 2, pp. 18–28 (2000).
26) Paxson, V.: Fast, Approximate Synthesis of Fractional Gaussian Noise for Generating Self-Similar Network Traffic, *Computer Communication Review*, Vol. 27, No. 5, pp. 5–18 (1997).
27) Paxson, V. and Floyd, S.: Wide-Area Traffic: The Failure of Poisson Modeling, *SIGCOMM '94*, pp. 257–268 (1994).
28) NWS: `http://nws.npaci.edu/NWS/`.
29) Song, H. J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K. and Chien, A.: The MicroGrid: a Scientific Tool for Modeling Computational Grids, *Proceedings of SC2000* (2000).
30) Casanova, H.: Simgrid: A Toolkit for the Simulation of Grid Application Scheduling, *Submitted to CCGRID2001*.