大学院輪講配布資料　　平成 19 年 1 月 12 日

分散タスクスケジューラーの研究動向
A Survey of Distributed Task Scheduler

近山・田浦研究室　　電子情報学専攻　修士 1 年 56428 高橋 慧

Abstract

A grid environment offers vast computing resources, and it is used not only for numerical computations but also for data mining and natural language processing. For these data-intensive applications, distributed task scheduler offers useful environment. However, it is not easy to develop a smart scheduler which considers data locality and task dependencies.

This survey shows some exsiting scheduling algorithms and new researches which focus on dependency and data intensive conditions. A simple min-min heuristics performs well to schedule tasks without dependencies. To schedule data intensive tasks with dependencies, the makespan was halved by co-scheduling tasks and replicas with GA in a simulation.

1　はじめに

グリッド環境は、一般の環境では望めない莫大な計算資源を提供する。ユーザーは多数の CPU による計算力、広大なメモリ空間と、ディスク領域を用いることが出来るので、従来行うことが出来なかった大規模な処理を行うことが出来る。

一方で、並列アプリケーションは一般のプログラマにとって馴染みのあるものではなく、一般に記述は難しい。これに加えて、グリッド環境はノード性能・アーキテクチャーが不均質である上に、ネットワーク的にレイテンシが極端に大きいノード、バンド幅が小さいノードを含むことが珍しくない。このため各ノードの特性を考慮せずにプログラムを実行すると、性能が低下してしまう。

プログラムの並列化を考える際、一般に密な通信が必要なアプリケーションでは、専用の並列アルゴリズムを開発することが不可欠である。並列アルゴリズムでは、逐次プログラムと異なり、プログラムは複数のスレッドによって並列に実行される。また、必要になるデータが離れたプロセッサに存在する場合は、メッセージパッシングや共有メモリを用いて転送する必要がある。通信は通常のメモリアクセスに比べて遅いため、逐次プログラムとは異なったデータアクセスの工夫が必要になる。また、特にグリッド環境は各ノードが不均質なので、性能向上を図るには、環境特有のチューニングが必要になり、グリッドの性能を十分に発揮するアプリケーションを書くのは簡単ではない。

これに対し、近年重要度が増しているデータマイニングや統計的な自然言語処理は、処理の局所性が高いものが多い。このようなアプリケーションでは、あるデータに対して処理を行うプロセッサは、他のプロセッサと頻繁に通信をすることなく処理を行うことが出来る。このため、予めデータを適切に分割して、それぞれのノードでプログラムを走らせれば、複雑な並列プログラムを書くことなしに効率的にグリッド環境の性能を使うことが出来る。

グリッド環境において一連のタスクを実行するミドルウェアを、分散タスクスケジューラーと呼ぶ。[1] これは、たくさんの逐次タスクをノードに割り振り、かつ必要なファイルを適宜転送するものである。基本的にはタスクを遠隔のノードで実行し、必要に応じてファイルを転送できるシステムがあればよいが、一般に性能を向上させるためには、ランダムではない賢いスケジューリングを行う必要があることが知られている。まず基本として、性能が高いノードには多くのタスクが割り振り、低いノードには少ないタスクが割り振る。また、あるタスクの処理に必要なファイルがある場合は、なるべくそのファイルが存在するノードの近くに割り当てる。

こうした不均質な環境におけるスケジューリングは、既に多くの研究がある。しかし、近年新たにスケジュラーには、二つの要件が求められている。一つはワークフロー、もう一つはデータ・インテンシブなアプリケーションである。ワークフローは依存性を持ったタスクの集合で、アプリケーション記述の自由度が高い一方で、タスクの実行順序によって全体の実行時間が大きく変化する。また、あるタスクによって生成されたデータを他のタスクが利用するため、スケジューリングが複雑になる。データ・インテン

シブなアプリケーションでは、データの転送時間が全体の処理に対して占める時間が大きな割合を占めるので、効率よくデータ転送を行うのが不可欠となる。

本発表では、グリッド環境に対してタスクを分配するスケジューラについて、既存アルゴリズムを示した後、ワークフローやデータ・インテンシブな環境に対応した研究を紹介する。

2 分散タスクスケジュラー

2.1 一般的な要件

分散タスクスケジュラーは、与えられたタスク群をノードに割り当てる。このときに考慮する条件としては、以下の点が挙げられる。

- ノードが利用可能かどうか
- ノードの性能・混雑度
- タスクの実行コスト (時間)
- タスクに必要なデータ
- タスク実行に必要なデータの転送コスト (時間)

これらの条件を考慮しつつ、スケジューリングを行う。タスクの実行時間やデータの転送コストはノードによって異なり、しかも実行前に正確には予測出来ない場合も多い。このため実際のシステムでは、利用できるノードに対し、そのノードでタスクを実行可能かどうか、などの簡単なマッチングを行うだけでタスクを割り振っていくものもある。このようなシステムの例としては Condor [2] が挙げられる。しかし、タスクの実行時間やファイルの転送時間を全く考慮しない場合、性能が大きく低下する可能性がある。

計画的にタスクスケジューリングを行う場合、目的関数を選択する必要がある。これには転送量・混雑率の平均化など様々なものが考えられるが、ここでは makespan(一連のタスクの処理を開始してから全タスクが終了するまでの時間) を最小化するものについて考える。

2.2 ワークフロー

ワークフローとは、ある一連のタスクを、依存性に沿ってグラフで記述したものであり、あらゆる実世界の処理の抽象化として用いられる。条件分岐・繰り返しを含むもの、含まないものなど様々な記述の方法があるが、ここでは条件分岐は持たず、DAG (Directed acyclic graph) で表されるものを考える。

ワークフローは単純な並列タスク実行に比べ高い記述力を持つ。ここには必要不可欠な依存性のみが記述されており、具体的なノードの数や、ノードに対するタスクの割り振りが書かれているわけではない。このため、スケジュラーが実行時に利用できるノード数・ネットワークの状況などによって、動的に効率の良い実行順序を計画することができる。ユーザーは並列プログラミングに関する深い知識が無くても、簡単にグリッド環境を利用して、複雑な大規模な処理を行うことが出来る。ワークフローのスケジュラーとしては、Condor のサブシステムとして利用できる DAGman [3] が挙げられる。DAGMan はタスク群の DAG を読み込み、実行可能なタスクを Condor に投入する。

DAGman では行われていないが、ワークフローを分析することによって、並列に実行できるタスク・出来ないタスクが明らかになる。また、全体の処理を早く終わらせるため重要になるタスクが分かれば、それを早めに実行することで、全体の処理時間を短縮できる。

一方で、一般に依存性のあるタスクでは、前のタスクによって生成された中間データが必要になるが、このデータが生成される場所はスケジュールによって変化する。また、実行順序の制約があるため、実行不可能なスケジュールが生成され得る。このため、スケジューリングのアルゴリズムより複雑に必要になってしまう。

2.3 データ・インテンシブ環境

コンピュータの性能向上、特に記憶容量の飛躍的な増加によって、これまで望めなかった大規模な処理が可能になっている。例としては、Web 上のドキュメントに対する言語処理・遺伝子解析などが挙げられる。これらは処理自体は単純であっても、大規模なデータに対して処理を行うことで、これまで得ることが出来なかった成果が上がっている。データサイズはテラバイトからペタバイトに達し、並列処理が不可欠になっている。

グリッド環境でこのようなアプリケーションを扱う場合、多くの処理はタスク間の関連性が低いので、各ノードにデータがうまく配分されれば効率的に実行できる一方、全体の実行時間に占めるデータ転送時間が大きくなる。特に、グ

リッド環境ではレイテンシが大きく、バンド幅が小さいネットワークが混在するため、効率的なデータ転送計画を立てることが重要になる。

まず、データ転送を減らすようなスケジューリングが必要になる。タスク実行に必要なデータがあるノードにあるときは、可能ならそのノードで実行すると、データ転送が不要になる。また、あるタスクの出力するデータを利用するタスクが多数ある場合は、そのタスクはネットワーク的に近いノードが多いノードで実行することが望ましい。

次に、ファイルレプリカの利用が挙げられる。これは、多くのノードが必要とするファイルは効率的に転送するために必要になる手法で、データを予めいくつかの代表的なノードにコピーしておき、あるノードそのファイルを必要になった際は、近くのレプリカからコピーするというものである。輻輳を防ぐ点でも、同じデータの転送は何度も行わないことが望ましい。

最後に、中間ファイルの問題がある。グリッドの計算ノードが持つローカルディスクは、あまり大きくない場合も多い。このような場合、あるデータを用いるタスクが全て実行されるか、そのデータが他のノードにコピーされた場合、そのデータは必要に応じて削除できる。逆に自動的に削除されない場合、各地のノードでディスクが圧迫され、タスクの実行が不可能になってしまう。

3 スケジューリングアルゴリズム

一群のノードに一群のタスクを割り当てる問題を、スケジューリング問題と呼ぶ。ノードが不均質で、あるタスクを処理するのにかかる時間が大きく異なる場合、スケジュールによって全体での処理時間は大きく変化する。データインテンシブな問題では、あるタスクの実行時間は、必要なデータを転送する必要がない場合・近くに転送する場合・遠くに転送する場合でまたタスク間に依存性がある場合は、どのタスクを優先的に処理するかも大切な制約条件になる。

不均質な環境下でのスケジューリングは新しい問題ではなく、これまでに様々なアルゴリズムが提案されている。これまでは各ノードの計算能力が低かったので、動的に計画でき、計算コストが小さいスケジューリングアルゴリズムが提案されてきた。しかし、近年の性能向上により、GA のような大きな計算量を必要とするスケジューリングアルゴリズムも実際的になりつつある。

以降、単純化されたモデルにおいて、いくつかのスケジューリングモデルを説明する。ここで用いられる概念に、ETC 行列 (estimation time of completion) というものがある。これは、スケジュール可能な全てのタスクについて、各ノードで処理した場合、処理終了までにどれだけの時間がかかるかを示すものである。この時間は、以下の三つの要素を合計して求める。

- ノードが利用可能になるまでの時間
- ファイル転送時間
- タスク実行時間

ファイル転送時間は、データサイズとバンド幅から推定する。多くの場合、ネットワークの輻輳は考慮されていない。タスク実行時間も、実際のアプリケーションの挙動から推測されたり、またユーザーが明示的に指定したりする。

ETC 行列の属性として、consistency というものがある。あるタスク t_0 の処理時間をノード n_0 とノード n_1 で比較した場合、n_0 の方が処理時間が短かった場合、あらゆるタスクについて n_0 による処理時間が n_1 による処理時間より短くなるとき、ETC 行列は consistent であるという。一般に大きなデータを必要としない、計算がメインのアプリケーションでは、ETC は単純にプロセッサの速度に反比例するため、consistent になる。このような条件下では、良いスケジュールと悪いスケジュールの差が小さいので、ランダムなスケジューリングでも全体の処理時間があまり悪化しないことが知られている。これに対し、タスクによってノードごとの処理時間の長短が異なる場合を inconsistent と呼ぶ。一般にタスクに必要とされるデータが各地に分散している場合、タスク処理時間に含まれるデータ転送時間はノードによって大きく異なるので、ETC 行列は inconsistent になる。ランダムなスケジューリングで Inconsistent なタスク群をスケジューリングすると、処理時間が大きく延びてしまう。

3.1 Greedy なスケジューリング

スケジューリングにおいて、古典的に良い結果を示す手法として、欲張り法 (Greedy algorithm) を用いる手法が提案されている。これらは、タスクが全くスケジュールされていない状態から出発し、一ステップにつき一つのタスクを割り振って、最終的に完全なスケジュールを生成する。

まず、最初の状態における ETC 行列が生成される。この ETC 行列が計算されたら、次にどのタスクを優先的にスケジュールするかを決定する。優先されるタスクの選び方として、ここでは Max-min・Min-min・sufferage の三手法を説明する。

Max-min, Min-min においては、各タスクについて、最小の完了時間を選択する。これらを基準に、最も早く処理が完了するタスクを選択するのが Min-min という手法で、最も遅く処理が完了するタスクを選択するのが Max-min である。Min-min は軽いタスクが優先的にスケジューリングされ、Max-min では重いタスクが優先的にスケジュールされる。

もう一つの sufferage という手法では、各タスクについて、(最小の完了時間) と、(二番目に小さい完了時間) の差が計算される。この値は、sufferage と呼ばれ、「そのタスクが最良のノードに割り当てられなかった時に、そのタスクの処理がどれだけ長くなってしまうか」を示している。例えば大きなデータを用いるタスクにおいて、データが存在するノードで処理する場合と、存在しないノードで処理する場合では、処理完了時刻に大きな差が出る。このようなタスクでは sufferage が大きくなる。次に、全てのタスクの中で sufferage が最も大きいノードが選択される。

いずれの手法においても、スケジュールするタスクを選択したら、そのタスクを最も処理が早く完了するノードに割り当てる。その後、ETC の行列は更新される。具体的には、そのタスクをスケジューリング可能なリストから削除し、割り当てたノードが利用可能になるまでの時間も更新する。更新が終わったら、新しいタスクの割り当てに移る。

これら Greedy な手法の特徴は、実装が簡単で処理が軽く、かつネットワークやノードの混雑度の変化に対して動的に対応できる点である。欠点としてては、スケジュールを作る過程において、一度スケジュールしたタスクの割り当ては二度と変更されないため、局所的最適解に陥りやすい点が挙げられる。また、ワークフローにおいては、たくさんの子タスクを持つノードが実行されないことにより、全体の makespan が延びてしまう可能性がある。しかし、依存性を持たないジョブの集合に対しては、GA などのに比べて遜色ないとの結果が報告されている。これについては後程触れる。

3.2 GA

静的なスケジューリング問題に対しては、GA や焼き鈍し法 (simulated annealing) も有力な手法として挙げられる。これらは、個体として一つの完全なスケジュールを用い、これをより優れたスケジュールに更新していく。一定時間実行後、ある程度良いスケジュールが得られたところで更新を終了し、そのスケジュールを採用する。ここでは GA によるスケジューリングアルゴリズムの実装について説明する。

これらのアルゴリズムを用いるのに必要になるのは、遺伝子の表現方法・評価関数・交雑方法である。遺伝子の表現方法としては、各タスクをどのマシンに割り当てたか、を書いた配列を用いるのが最も単純である。これは、依存性が無く、輻輳を考えないモデルの場合、あるノードに割り当てられたタスクの実行順序は、全体の makespan に影響しないためである。評価関数としては、今回の条件から makespan を用いる。交雑方法は、二つのスケジュールを途中で分割してつなぎ合わせたり、任意の要素を入れ替えたりといった手法が考えられる。

スケジューリング問題への GA の利用は広く行われていて、局所的最適解に陥りにくく、優れた探索性能を示している。一方で、処理自体が重く、探索空間の大きさに比例したループ回数が必要になるため、タスク数とノード数が大きくなった場合、計算コストが無視できなくなる可能性がある。これに対しては、タスク集合全体を予めいくつかの小さなタスク集合に分割し、それぞれについて別々にスケジューリングを行うことができる。また先に述べた遺伝子表現にはタスクの実行順序が含まれていないため、makespan を求めたりスケジュールとして実行することができない。また、表現方法によっては交雑の際に実行不可能なスケジュールが生成されていまう可能性もある。このため、遺伝子表現を工夫する必要がある。

3.3 各手法の比較

Tracy ら [6] は、11 種類のスケジューリング手法をシミュレーションを用いて比較した。タスクのデータは様々な条件を再現するようランダムに生成され、各手法を用いた makespan を比較した。

比較された手法はランダム・Max-min・Min-min・GA・焼き鈍し法・GSA・A*などで、8 ノード・512 タスクの問

題について、PentiumII 300MHz、メモリ 1GB の環境下で行われた。データは、まず各タスクについて、処理時間の基準となる値を一様乱数で生成し、これにさらに一様乱数を掛け合わせて生成されている。結果は、一つの条件に対し 100 個の異なった行列が生成され、各手法についての比較が行われた。結果の一例を図 1 に示す。

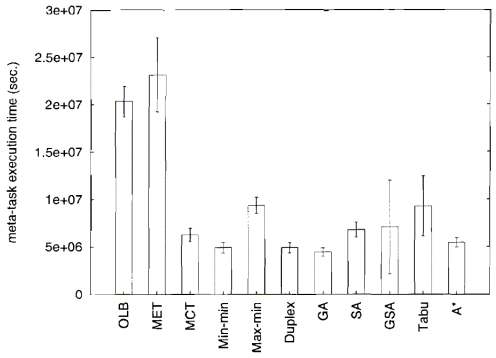

図 1. Comparing scheduling heuristics (Partially-consistent, high task, high machine heterogeneity data)

全体として、ほとんどの条件において GA によって得られたスケジュールが最短の makespan を与えたが、最適値を得るのに約一分を要した。一方で、Min-min 法によって得られたスケジュールは GA で得られたものに劣るものの、最も良いスケジュールから概ね 10%しか悪くなかった。このことから、タスク間に依存性が無く、輻輳を考慮しない状況においては、Min-min 法が実用的であるという結果が得られた。

4 Workflow のスケジューリング

GrADS プロジェクト [8] では、データ・インテンシブな環境に対応したスケジュラーの開発を行っている。

処理系として賢いスケジュラーを実装する上で必須になるのが、アプリケーションの実行時間やファイルサイズを予測することである。GrADS では、まず各ノードの性能を、整数演算性能・浮動小数点演算性能・CPU クロック・キャッシュサイズ・メモリクロック・メモリサイズなどを用いてパラメータ化し、また実際のプログラムを小さなデータで実行することでアクセスパターンを測定し、タスクの実行時間を推測する。またファイル転送の時間についても、各ノード間のバンド幅・レイテンシを測定することで予測する。こうして、未スケジュールなタスクに対する ETC 行列を計算することができる。

スケジューリングアルゴリズムとしては、Greedy な手法を用いている。まずワークフロー上で実行可能なタスクのうち一つに対しスケジューリングを行い、そのタスクを実行したことによって実行可能になったタスクを実行可能リストに加える。その後 ETC 表を更新し、順番に全てのタスクをスケジューリングするまで繰り返す。ただ完全に動的に行うのではなく、Max-min・Min-min・sufferage の三手法について全タスクを割り当て、完全なスケジュールを作成したのち、その中で最も Makespan が小さいものを選択する。

これらの工夫により、GrADS スケジュラーでは、既存のスケジュラーに比べて効率的なスケジューリングを実現した。これを検証するため、画像処理のタスクを用いて比較実験が行われた。実験には画像処理のデータが用いられ、データサイズは 2GB であった。これをデュアル Itanium・Opteron プロセッサなどが混在した 271 プロセッサの環境で実行した。なお GrADS はレプリカの配置は自動で行われないので、必要になるデータは予め各クラスタにコピーされ、データ転送はクラスタ間でのみ行われた。この一連のタスクについて、以下の四つのスケジューリング方法で実験が行われ、性能が比較された。

- RN : ランダムなスケジューリング
- RA : 完全な性能予測に対し、データ転送コストを考慮せず、性能に応じた数のタスクを割り付けたもの
- HC : CPU クロックのみによる簡単な性能予測ののち、Greedy なスケジュールを立てたもの
- HA : 完全な性能予測ののち、Greedy なスケジュールを立てたもの

この結果を図 2 に示す。

	HA	HC	RN	RA
makespans (min)	505	757	1121	762

図 2. Makespans with GrADS and random scheduler

このように、ランダムなスケジューリングを行った場合 (RN) に比べて約 2 倍の性能向上が見られた。スケジュールを分析したところ、性能予測によって、性能の高いノー

ドに多くのタスクを割り振られている様子が観察でき、各マシンのロードの平均化にも貢献した。

5 データ・インテンシブ環境に対応したスケジュラー

Huadong ら [10] は、データ・インテンシブな環境に対応したスケジューリングアルゴリズムとして、ファイルレプリカの配置をタスクスケジューリングと同時に扱う方法を提唱している。基本的なアイディアは、性能の高いノードを自動的に発見し、性能の高いノードで優先的に処理を行うことである。スケジューリングは 1 タスクずつ行われるが、一定個数のタスクをスケジュールすると、そのタスク群の全てのタスクが完了するまで新しいタスクはスケジューリングされない。

この研究では、過去のタスクの実行時間からノードの性能を予想する。これは性能予測モデルに基づくものではなく、各タスクの重さがほぼ均一であるとの仮定に基づいて、一タスクの処理時間が短ければ性能が高いと考える。このようにして予測された各ノードの性能を元に、性能の高いと判断されたノードに優先的にタスクの割り当てを行う。また、一度割り当てたが結果が返ってこないタスクについては、より高速なノードが遊休状態になると、そのノードでも処理を開始する。このタスクに関しては、いずれかのノードが結果を返すと、二重に処理しているノードは処理を中止する。

データ･インテンシブな環境への対応としては、スケジュール可能なノードが複数ある場合、データ転送時間を考慮したスケジューリングを行う。この際のデータ転送時間は、実際にデータの一部を転送し、要した時間から推定する。こうすることで、静的なバンド幅だけでなく、実行時のネットワークの混雑度にも応じた予測が行える。

このシステムについて、性能を考慮しない方法・性能推定のみを行い、データ転送時間を考慮しない方法・本手法の三つの方法について、実際のアプリケーションで比較が行われた。実験は Planet Lab 上の 80 台のノードで行われ、全体のデータサイズは 6.7GB であった。なお、ノードは占有されておらず、常に他のユーザーのアプリケーションが動作していた。これらについて、一度にスケジュールするタスクの数を 200, 400, 600, 800 と変えて実験が行われた。この値が小さい場合には再割り当てが頻繁に起こり、タスクの処理時間は短くなる。結果を図 3 に示す。

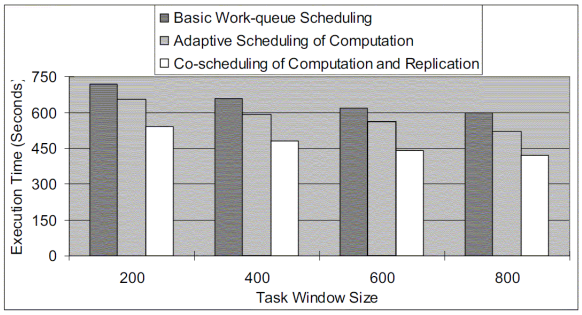

図 3.

このように、いずれの条件においても本手法が約 20%短い makespan を与えた。この研究では、スケジューリングは局所的に行われているため、大域的に良いスケジューリングが行われる保証は無い一方、実際に処理系として実装され、動的に各ノードの負荷やネットワークの混雑度もふまえたスケジューリングが行われる点で評価できると思われる。

6 データ・インテンシブ環境に対応した Workflow のスケジューリング

データ・インテンシブな環境下では、効率的なタスクスケジューリングとともに、効果的なファイルレプリカの配置が不可欠となる。Phan ら [11] は、データ・インテンシブな環境に対応して、タスクのスケジューリングとレプリカの配置を同時に計画するスケジューリングアルゴリズムを提案している。まずタスク集合としてワークフローが扱われており、

これは GA を用いており、遺伝子表現としてスケジュールとレプリカの配置の両方を用いることで、効率的なスケジューリングを可能にしている。

遺伝子は三つの配列によって構成されている。まず一つは、全タスクを一列化したものである。これは、ワークフローの実行順序制約を満たすようになっている。次に、各タスクに対して割り当てられるノードが書かれた配列がある。これは、先の依存関係を考えないスケジュラーにおいて用いられたのと同一である。最後に、各レプリカの配置を示す配列がある。この三つを指定することで、一つのスケジュー

ルを表すようになっている。評価関数としては makespan を用い、交雑はワークフローの制約を満たしつつタスク順序を入れ替えたり、レプリカの位置を交換することで行われる。

このスケジューリングアルゴリズムと、先の Min-min 法にランダムにレプリカを配置する手法が比較された。GA では、収束するまでは長く実行するほど良い解が得られる。しかし、スケジュラーとしての要件から、常に 100 世代のシミュレーションで打ち切る設定とされている。タスク数を 40, 80, 160 と変えて、実験が行われた。

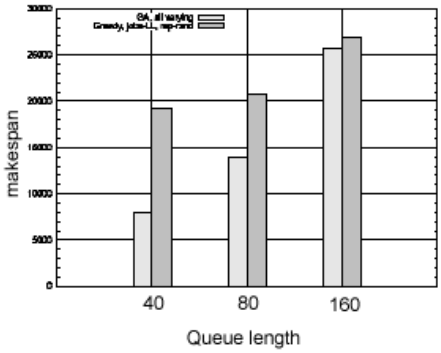

図 4. Makespans of GA (with automatically placed replicas) and Min-min (with randomly located replicas)

結果は、40 タスクの場合は、本手法が二倍ほど良い結果となったが、160 タスクの場合はその差はほぼなくなっている。これは、探索空間に比べて GA の世代数が少ないため、と説明されている。

7 おわりに

本発表では、グリッド環境にタスクを分配する分散タスクスケジューラについての調査を報告した。まず、一般的なタスクスケジュラーの考慮すべき条件について説明した後、ワークフローとデータ・インテンシブという、近年注目されている二つの環境を提起した。次に、既存のタスク分配アルゴリズムとして、Greedy な Max-min・Min-min・Sufferage の三つを説明し、これらの手法のシミュレーションによる性能比較を示した。この Greedy なアルゴリズムを用いて、ワークフロースケジュラーを実装した例として、GrADS プロジェクトの概要と成果を説明し、最後にデータインテンシブな環境に適合した、ワークフロースケジュラーの研究として、GA を用いた Phan らの研究を紹介した。

今後は、既存研究を参考にしつつ、実際のネットワークトポロジを考慮した、効率的なワークフロースケジュラーの実装を進めていきたい。また、データインテンシブなアプリケーションにおいては、中間ファイルの管理が重要な問題になると思われる。ローカルディスクが小さいノードにおいては、ノードに保持できるデータ量に制約がある。このため、既に必要とされなくなったファイルを適宜削除したり、必要に応じて再生成するなどの工夫も盛り込みたい。

参考文献

[1] Jia Yu and Rajkumar Buyya “A Taxonomy of Scientific Workflow Systems for Grid computing” Special Issue on Scientific Workflows, SIGMOD Record, ACM Press, Volume 34, Number 3, Sept. 2005

[2] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, “Condor-G: A Computation Management Agent for Multi-Institutional Grids.,” Cluster Computing, vol. 5, pp. 237-246, 2002.

[3] http://www.cs.wisc.edu/condor/dagman/

[4] Ewa Deelman1 et al. “Pegasus: Mapping Scientific Workflows onto the Grid” Lecture notes in computer science (Lect. notes comput. sci.) Grid computing (Nicosia, 28-30 January 2004, revised papers)

[5] Luiz Meyer “Planning spatial workflows to optimize grid performance” Proceedings of the 2006 ACM symposium on Applied computing Dijon, France (DSGC)

[6] Tracy et al. “A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems ” (TR-ECE 00-04)

[7] 町田悠哉 滝澤真一朗 中田秀基 松岡聡 “レプリカ管理システムを利用したデータインテンシブアプリケーション向けスケジューリングシステム” ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS2006), 2006, 9-16. 4,

[8] Mandal, A. et al. “Scheduling Strategies for Mapping Application Workflows onto the Grid. ” IEEE Symposium HPDC 14, 125.134. (2005)

[9] K. Cooper et al. “New Grid Scheduling and Rescheduling Methods in the GrADS Project” Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International (2004)

[10] Huadong et al. “Dynamic Co-Scheduling of Distributed Computation and Replication” Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID’06) - Volume 00 CCGRID ’06

[11] Phan et al. “Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm.” In Proceedings of the11th Workshop on Job Scheduling Strategies for Parallel Processing. Cambridge, MA. Springer-Verlag, Berlin, Germany 2005

[12] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, C. Pautasso, T. Heinis, R. Gronmo, Hjordis Hoff, Arne-Jorgen Berre, M. Glittum, S. Topouzidou “Developing scientific workflows from heterogeneous services ” June 2006 ACM SIGMOD Record, Volume 35 Issue 2

[13] Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan “Interpreting the Data: Parallel Analysis with Sawzall” http://abs.google.com/papers/sawzall-sciprog.pdf