ユーザのステアリングを許す インタラクティブなジョブスケジューリングシステム

飯野　彰子 †　中田　秀基 ††,†
下平　英寿 †　松岡　聡 †,†††

複数のジョブから構成されるワークフローを、グリッド上でバッチ的に実行する研究は多くなされている。しかし、アプリケーションユーザの持つジョブの中には、基本的にはワークフロー的な構造を持つものの、各ジョブの実行においてユーザからの操作が不可欠なものも少なくない。このようなジョブは既存のワークフロー機構で実行することは難しい。

我々は、基本的にはワークフローに従った処理をしつつも、ユーザからの入力が不可欠な場面ではユーザに操作の要求を行う、ワークフロージョブスケジューリングシステムを開発した。ワークフローシステムのベースとしては、Condor の DAGMan を用いた。この DAGMan の機構を利用して、ユーザからのステアリングを受け付ける機構を実装した。また、本システムを系統樹推定問題に適用し、有効性を確認した。

An Interactive Job Scheduling System that Allows Job Steering by Users

Akiko Iino ,† Hidemoto Nakada ,††,† Hidetoshi Shimodaira †
and Satoshi Matsuoka †,†††

Since the grid environment is suitable for long-runnig workflow execution, the workflow engine for grid becomes one of the hot research areas. However, applications that require user steering during its workflow execution are not addressed with existing workflow engine research. Although there are lot of work on computational steering, the requirements and nature of the steering for the long-running workflow execution are totally different from conventional computational steering, and therefore, it should be addressed differently.

We designed and implemented a workflow scheduling framework that allows users to control the execution of their application. It is implemented using Condor DAGMan as a workflow engine and provides users steering capability via e-mail and web-enabled interface. We also evaluated the system with phylogenetic tree inference application.

1. はじめに

グリッドは長時間のジョブをバッチ的に実行する環境として適している。このため、複数のジョブから構成されるワークフローを、グリッド上でバッチ的に実行する研究が多くなされている。しかし、実際のアプリケーションには、基本的にはワークフロー的な構造を持つものの、各ジョブの実行においてユーザからの操作が不可欠なものも少なくない。このようなアプリケーションは、既存のワークフロー機構ではうまく実行することができない。

我々は、基本的にはワークフローに従った処理をしつつも、ユーザからの入力が不可欠な場面ではユーザに操作の要求を行う、ワークフロージョブスケジューリングシステムを開発した。ワークフローシステムのベースとしては、Condor[1] の DAGMan[2] を用いた。この DAGMan の機構を利用して、ユーザからのステアリングを受け付ける機構を実装した。また、本システムを系統樹推定問題に適用し、有効性を確認した。

本稿の構成は以下のとおりである。2 節で、本研究で使用したワークフローシステムである Condor DAGMan について述べる。3 節で設計について述べ、4 節で実装について述べる。5 節で系統樹推定問題への適用を通じた評価を行う。6 節で関連研究について述べ、7 節で今後の課題について述べる。

† 東京工業大学 Tokyo Institute of Technology
†† 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology (AIST)
††† 国立情報学研究所 National Institute of Information

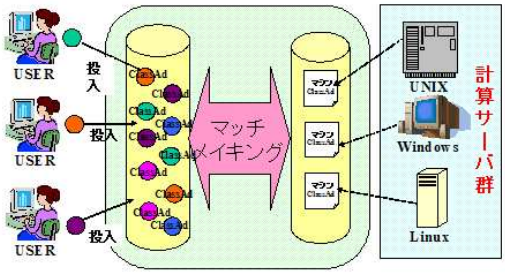

図 1　Condor の概要

```
Universe   = vanilla
Executable = test
output     = tmpOut
error      = tmpErr
log        = tmp.log
queue
```

図 2　サブミットファルの例

2. Condor と DAGMan

2.1 Condor の概要

Condorは、米国ウィスコンシン大学のグループが開発したハイスループットコンピューティングを指向したキューイングシステムである。当初はキャンパス内の遊休計算機を有効利用することを目的としていたが、現在では後述する Condor-G の機能により、Globus のメタスケジューラとしても広く使用されている。

Condorは、計算可能な計算機の集合を Condor プールと呼び、ユーザがサブミットしたジョブに対して、Condor プールに属する計算機に割り当て、実行する (図 1)。このジョブに対して計算機を割り当てる方法に、マッチメイキングと呼ばれる方法が使用される。さらに、計算が適宜チェックポイントされ、サーバフェイル時の再実行や優先順位の高いジョブによるプリエンプションがサポートされていることが、Condor の特徴である。

Condor でジョブをサブミットする際には、サブミットファイルと呼ばれるファイルを用意する。図 2 に、Condor のサブミットファイルの例を示す。

2.2 DAGMan の概要

DAGMan (Directed Acyclic Graph Manager) は、ジョブの依存関係を DAG(有向非循環グラフ) として記述しその順番どおりに実行していく、ある種のワークフローエンジンである。

ジョブ依存関係はサブミットファイルとは別に、DAG ファイルと呼ばれるファイルに記述する。DAG ファイルのサンプルを図 3 に示す。この DAG ファイルは図 4 に示す依存関係を表現している。このファイルで参照されている、`A.condor` などは、Condor のサブミットファイル名である。

```
JOB  A  A.condor
JOB  B  B.condor
JOB  C  C.condor
JOB  D  D.condor

RETRY A 3
Script POST  B B.post.sh

PARENT A CHILD B C
PARENT B C CHILD D
```

図 3　DAG ファルの例

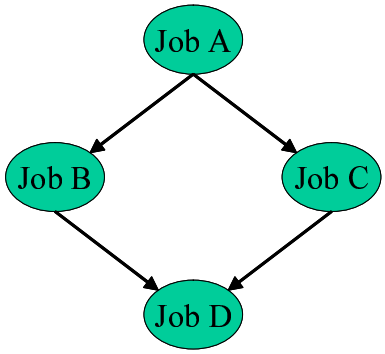

図 4　DAG の例

また、DAG はネストして使用することもできる。つまり DAG のひとつのノードを他の DAG にすることができる。これを行うには、内側に入る DAG ファイルを、`Condor_submit_dag` コマンドに`-nosubmit` オプションを付加して呼び出すことで、通常のサブミットファイルに変換する。通常のサブミットファイルとなった DAG は、他の DAG ファイルの中に自由に埋め込むことが可能となる。この機能を用いることで比較的複雑な構造を持つワークフローを表現することができる。

2.3 RETRY と POST スクリプト

DAGMan はワークフローエンジンとしては、単純で低機能ではあるが、いくつかの付加的なフローコントロール機能を提供している。

DAGMan はデフォルトでは、ワークフロー中のあるジョブが失敗した際 (ジョブの Exit コードが 0 でない場合)、そのジョブに依存するジョブの実行が停止し、最終的には DAG の実行全体が停止する。RETRY は各ジョブに対して指定できる属性で、ジョブが失敗した際にリトライする回数を指定する。指定したリトライ回数だけ再実行しても成功しない場合には、通常と同様に DAG の実行が停止する。

PRE、POST スクリプトは各ジョブに対してその前処理と後処理を行うスクリプトである。スクリプトはサブミットマシンで実行される。また、POST スクリプトはジョブの成功失敗に関わらず実行される。さ

らに POST スクリプトがある場合には、ジョブの成功判定は、ジョブ自身ではなく、その POST スクリプトの Exit コードで行われる。したがって、POST スクリプトを用いることで、ジョブが失敗した際にも、スクリプトでリカバーすることができれば、ワークフローの実行を続行することができる。

3. 設　　計

ワークフローをベースとしたシステムのジョブステアリングは、通常のジョブステアリングと大きく異なる。まず、バッチ的ジョブには長大な実行時間が想定されるうえ、キューイングシステムの状況によっていつ実際に実行されるかを事前に知ることは難しい。したがって、ジョブを実行する際に、ユーザが常に端末の前にいることを仮定することはできない。したがって、通常のジョブステアリングのように GUI と一体化したサブミット機構によって、ユーザにサブミットと操作を行わせるという手法は適用できない。

反面、ステアリングが必要となってから実際に入力を受けるまでの時間を短くすることはそれほど重要ではない。スケジューリングシステムには常に他のジョブが待ちキューに存在することが想定できるため、ユーザの入力を待っている間に、他のジョブを実行していれば、システム全体としてのスループットには影響がないからである。

このような観点から、我々はメイルと Web ページをベースとした、ルースなジョブステアリング機構を設計した。ユーザのステアリングが必要になると、システムはユーザに対して Web ページへのリンクが書かれたメイルを送信する。このページにステアリングに必要な情報とステアリングのためのボタンが表示される。メイルと Web という遍在的なインターフェイスを使用することで、ユーザの環境に関する仮定を最小限にすることができる。たとえば、i-mode 携帯電話などからでもステアリングを行うことが可能となる。

このような機能を実現するためには、ワークフローエンジンからメイルの送付と、Web ページからの操作によるワークフロー実行の制御が必要となる。

4. 実　　装

4.1 システムの概要

われわれは、上述のワークフローエンジンとの連携を Condor DAGMan を利用して実現した。具体的には、Condor の DAGMan の POST 機構および RETRY 機構を利用することでジョブのステアリングを実現している。システムの概要を図 5 に示す。

今回の実装では、特定のジョブの出力結果が特定の条件を満たすかどうかをユーザに判断させ、条件を満たさなければ続行する機能を対象として実装した。

ステアリング機構は下記の 3 つの機構から構成される。

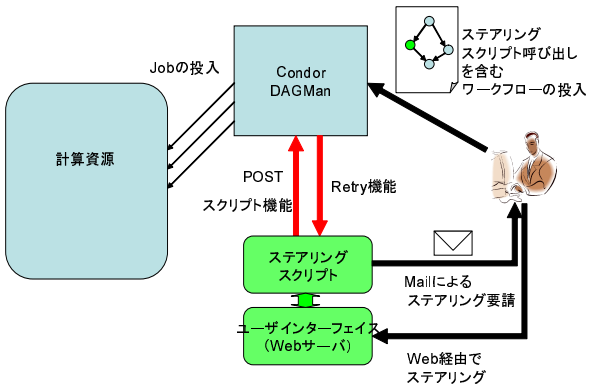

図 5　システム概要

- **Web ページの生成**
  ユーザに提示するステアリングのための情報として、ジョブの出力をグラフ化し、そのグラフを含む Web ページを生成する。続行、終了を選択するためのボタンも表示する。
- **Mail の送信**
  上記の Web ページへのリンクを含むメイルを作成、送信する。
- **Web ページからの制御**
  ユーザは Web ページ上のボタンおよびテキストフィールドへの入力を通じて、ジョブのステアリングを行う。ユーザの入力は CGI スクリプトで処理される。

これらの機能は DAGMan 内の POST スクリプトとして起動されるステアリングスクリプトから起動される。ステアリングスクリプトはまず Web ページを生成し、Mail を送信した後、ユーザからの入力をファイルを定期的にポーリングして待つ。CGI スクリプトは、ユーザからの入力データをファイルに書き出す。ステアリングスクリプトは、ポーリングでこの入力データを取得し、終了時の Exit コードによって DAGMan に対して、ジョブの実行、終了を指定する。

4.2 DAGMan によるコントロール

図 7 と図 8 に、本システムで使用する DAG ファイルの概要を示す。このサンプルは、2 つのジョブ A と B があり、A の終了をユーザが判定した後で、B の実行へ進む例である (図 6)。

図 7 に示したものが、直接の実行の対象となるトップレベル DAG ファイルであり、図 8 に示すものが、トップレベル DAG ファイルに埋め込まれる内部 DAG ファイルである。この例では、内部 DAG ファイルは `inner.dag.submit` というファイルに変換されて、トップレベル DAG ファイルから参照されている。

ジョブ A をジョブ W でラップしている。これは

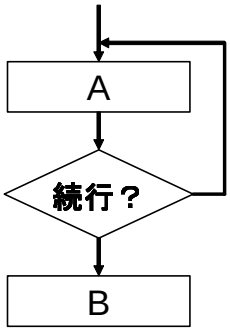

図 6　DAG サンプルの示すワークフロー

```
Job W inner.dag.submit
RETRY W 10
Job B B.submit

PARENT W B
```

図 7　トップレベル DAG ファイル例

```
Job A A.submit
Script POST A steering.sh
```

図 8　内部 DAG ファイル例

POST スクリプトと RETRY をひとつのジョブに対して指定すると、RETRY が先に評価されてしまうため、期待通りの動作をさせることができないためである。

内部 DAG ファイルで POST スクリプトとして指定されている steering.sh が前節で述べたステアリング用のスクリプトである。このスクリプト内で、Web ページの作成やメイルの送信などのステアリングに必要な操作がおこなわれる。

5. 評　　価

本システムの有効性を確認するため、生物の DNA 情報から進化の系統樹を推定する系統樹推定問題に適用した。このジョブのワークフローを図 9 に示す。

このジョブは、MrBayes、PAML、CONSEL という 3 種類のプログラムから構成される。このうち、MrBayes の終了判断はプログラムで行うことが非常に難しく、中間結果をグラフ化して、ユーザが判断しなければならない。

この問題に対して本システムを適用した。本システムの提示する Web ページを図 12 に示す。ユーザはこのページから 1 つ目のプログラムを終了するか、続行するかを選択することができる。

ジョブは Condor の DAG として投入する (図 10) 。システムはまず、Phase1 を実行する。Phase1 の実

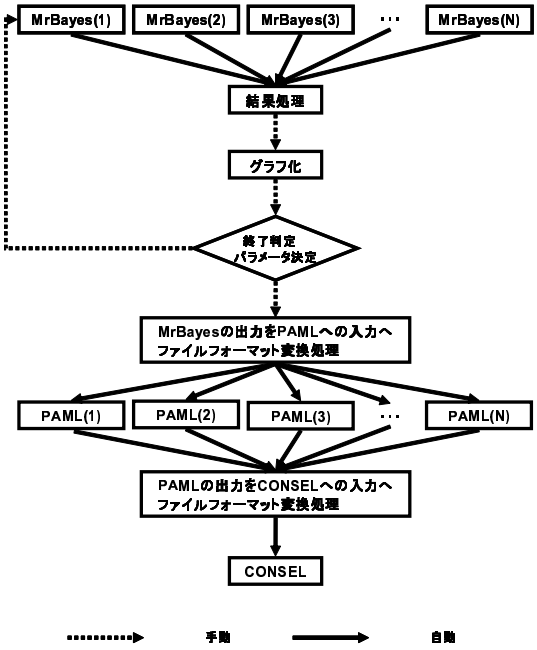

図 9　系統樹推定問題のワークフロー

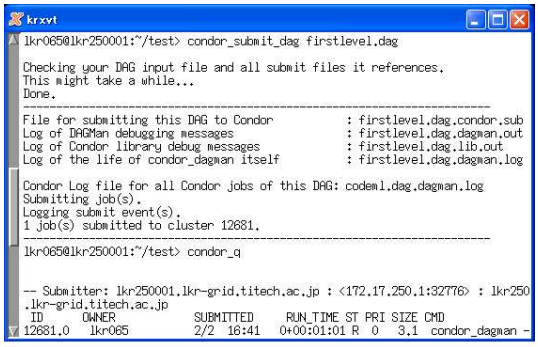

図 10　ジョブの起動

図 11　ユーザへの通知メイル

行がある程度まで進むと、Phase1 のジョブは一度終了し、POST スクリプトが起動され前節で述べたステアリング機構が起動する。

図 11 に、ユーザに送られてくるメイルを示す。このメイルには、ステアリングのための Web ページの URL が記述されている。

この URL を Web ブラウザでブラウズすると図 12

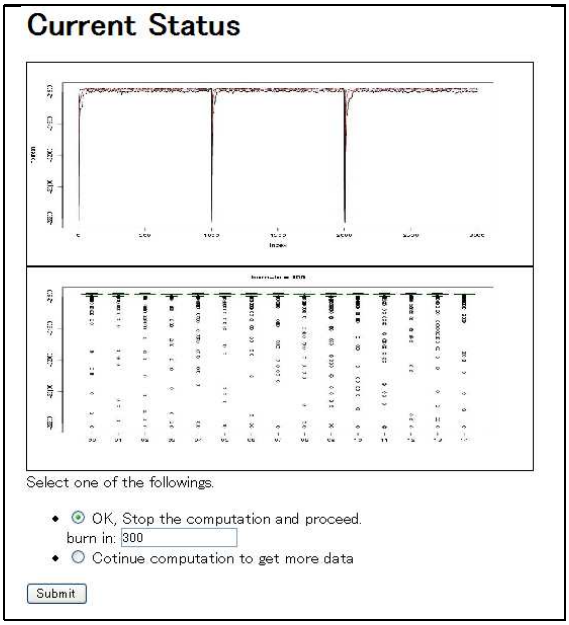

図 12 系統樹推定におけるジョブステアリング

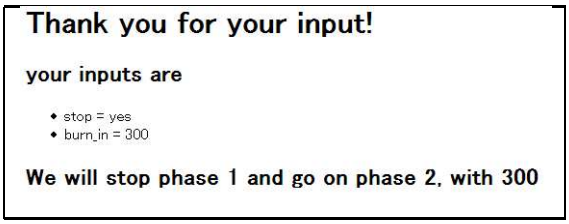

図 13 ステアリング終了画面

に示す、ステアリングのための Web ページが表示される。表示結果のグラフには、Phase 1 の実行結果が示されている。ユーザはこの結果をみて、Phase 1 を続行するか、Phase 2 へ進むかを決定し、該当するボタンを押す。図 13 に、ボタンを押して phase2 へ進む判断を行った後に表示される画面を示す。

6. 関 連 研 究

6.1 ワークフローエンジン

6.1.1 UNICORE

UNICOREO[3),4)] は、富士通ヨーロッパが中心となって開発した、強力なワークフローマネージメントを特徴とするグリッドミドルウェアである。

UNICORE では、クライアントがサーバに依頼する単位が、個々のタスク (コマンドの実行や、ファイルの転送など) ではなく、複数のタスクをまとめて依存関係で接続したもの (ワークフロー) である。サーバはワークフローを受け取り、ワークフロー中のタスクを個別に、ワークフローの指定する順序にしたがって実行する。このワークフローを AJO(Abstract Job Object) と呼ぶ。

AJO は Java のオブジェクトとして表現され、Java のシリアライゼーション機能を用いてエンコード、転送される。AJO の表現力は非常に高く、複雑なワークフローを表現することができる。

UNICORE にはいくつかのクライアントモジュールがある。そのうちのひとつの、Pallas[5)] 社が提供する GUI ベースのクライアントである Unicore Pro クライアントには、プラグインという概念がある。これは、アプリケーション固有のユーザインターフェイスをアプリケーションユーザが自由に GUI に追加できる機構である。この機構はコンピュテーション・ステアリングに利用することができるが、本稿で対象とする長時間にわたるバッチジョブのステアリングには適さない。

6.1.2 GridAnt

GridAnt[6)] は Globus Toolkit[7)] に対する多言語インターフェイスパッケージである CoG Kit の一部として開発された、ワークフローエンジンである。

GridAnt は、Java で記述されたプログラムのビルドを助けるシステムである Apache Ant[8)] に拡張モジュールを追加することで構成されている。Apache Ant にはもともと依存関係に沿ってビルドを進めるエンジンが組み込まれているが、このエンジンをそのまま利用してグリッドアプリケーションを依存関係にそって実行する。

GridAnt は通常のワークフローアプリケーションの実行には十分強力だが、ワークフローの管理がクライアント側で行われるため、実行中にクライアントを計算環境から切り離すことができないという問題がある。また、ジョブのステアリングを行うこともできない。

6.2 コンピュテーション・ステアリング

6.2.1 RealityGrid

RealityGrid[9),10)] は、英国の e-scicence プロジェクトの一部をなすプロジェクトで、グリッド上でのビジュアライゼーションと、アプリケーションステアリングを行う。特に、オンラインでリアルタイムのビジュアライゼーションに取り組む。RealityGrid は、実行途中の変数の値の外部への表示と外部からの変更を支援するライブラリを提供する。このライブラリを用いると、外部から変数を変更することでアプリケーションの動的な制御が可能になる。

また、RealityGrid によって提供される計算資源やサービスは、グリッドミドルウェア Globus Toolkit(GT2) に基づいて提供される (図 14)。

RealityGrid は、本質的にビジュアライゼーションプログラムの外部からのコントロールを目的としているため、本研究の対象となるバッチ的なワークフローのステアリングには適さない。

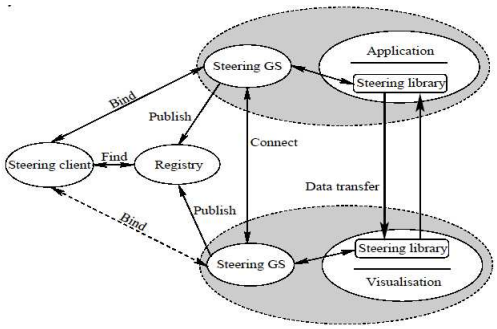

図 14 グリッド上のステアリング図

6.2.2 SCIRun

ユタ大学で開発された SCIRun[11] とは、ステアリング可能なアプリケーション開発、実行を目的としてシステムである。SCIRun では、アプリケーションの実行途中結果のモニタリング、ステアリングを C++ のクラスとして抽象化する。具体的には、ユーザは単一のジョブを一つの C++のクラスとして記述する。同クラスは、ジョブへの入力をフィールドとして持ち、ユーザはどのフィールドがステアリング可能か指示する。また、同クラスには、ジョブを実行する関数を定義する。SCIRun では複数のジョブの実行、ステアリングをそれぞれのクラスのインスタンスの関数を順次実行することで行う。また、Tcl/Tk の可視化環境を備えており、ユーザが特別なプログラムを記述することなく GUI の実行環境が利用可能である。しかし、アプリケーションを SCIRun のフレームワークにそって記述する必要があり、汎用性は低く、我々の目的には適さない。

7. おわりに

本稿では、ユーザからのステアリングが可能なワークフロージョブスケジューリングシステムを、Condor の DAGMan を利用して実装した。また、このシステムを系統樹推定問題に適用し、その有効性を確認した。

今後の課題としては以下が挙げられる。

- **汎用化**
  本稿で示した実装は、アプリケーション依存の部分とステアリングの部分が十分に分離されておらず、プログラマはステアリングを意識してアプリケーションを記述しなければならない。ステアリング機構をアプリケーションから分離し、任意のアプリケーションに対して容易に適用可能とする必要がある。
- **他の大規模アプリケーションへの適用**
  本稿では系統樹推定問題に適用して有効性を確認したが、他の大規模なアプリケーションに適用し、本システムが提供する機能が十分汎用で協力であることを確認する必要がある。
- **他のワークフローエンジンでの実装**
  今回の実装では Condor DAGMan を使用したが、Condor DAGMan はワークフローエンジンとして比較的低機能であるため、本システムの実装が困難であった。
  今後はたとえば UNICORE などの他のワークフローエンジンに適用し、本手法の有効性を確認するとともに、実際のユーザに提供していきたい。

参 考 文 献

1) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proc. of HPDC-7* (1998).
2) : DAGMan. http://www.cs.wisc.edu/condor/dagman/.
3) Erwin, D. W. and Snelling, D. F.: UNICORE: A Grid Computing Environment, *Lecture Notes in Computer Science*, Vol. 2150, p. 825 (2001).
4) Romberg, M.: The UNICORE Architecture Seamless Access to Distributed Ressources, *Proceedings of the Eight IEEE International Symposium on High PerformanceComputing*, pp. 287–293 (1999).
5) : Pallas. http://www.pallas.com/.
6) : GridAnt. http://www-unix.globus.org/cog/projects/gridant/.
7) : Globus Toolkit. http://www.globus.org/.
8) : Apache Ant. http://ant.apache.org.
9) : RealityGrid Project, http://www.realitygrid.org/.
10) Brooke, J. M., Coveney, P. V., Harting, J., Jha, S., Pickles, S. M., Pinning, R. L. and Porter, A. R.: Computational Steering in RealityGrid, *Proceedings of the UK e-Science All Hands Meeting* (2003).
11) : . SCIRun: A Scientific Computing Problem Solving Environment. Scientific Computing and Imaging Institute (SCI), http://software.sci.utah.edu/scirun.html, 2002.