

異種ミドルウェア間に跨るワークフロー ジョブの実行方式と実装

田中義一[†] 合田憲人[†]

研究室のようなグリッド環境に含まれない計算資源とグリッド環境の計算機資源との連携には、未だ煩雑な処理が必要となる。我々は、研究室の計算資源とグリッドミドルウェアがインストールされる情報基盤センタ等の計算資源を連携させるミドルウェアの検討を行っている。本報告では、研究室レベルの計算資源と、情報基盤センタレベルの計算資源に対して、シームレスなジョブ投入やモニタリングを可能とするワークフローツールの設計と実装について述べる。

Design and Implementation of Workflow Tool Utilizing Resources on the Multiple Grid Environments

Yoshikazu Tanaka[†] and Kento Aida[†]

Users need complicated operations to federate computing resources in a small laboratory, which is not operated with grid middleware, and computing resources in the grid. We are now planning functional enhancement of grid middleware for utilizing the laboratory level resources such as cluster machines in the laboratory. This report describes design and implementation of Workflow Tool in the laboratory, which provides them with seamless interface to manipulate workflow jobs in the laboratory and grid environment.

1. はじめに

グリッド技術の普及に伴い、世界各国でグリッド環境の整備が進められている。国内においても、NAREGI (National Research Grid Initiative) ミドルウェアを用いたグリッド環境の整備が、9 大学の情報基盤センタで進められている 1)2)。しかし、グリッド運用には高度な技術が必要とされるため、グリッドにより管理される資源は情報基盤センタなどで使われる高性能計算機に限られており、研究室に設置されるような計算機には及んでいない。そのため、ユーザが、研究室の計算機とグリッド上の計算機を連携させたい場合には、煩雑な処理が必要となる。

ジョブを連携実行させるシステムとして、Unicore, Condor, GridAnt/Karajan, Taverna など数々のワークフローツールが知られている 3)4)5)6)。例えば、Condor は、DAGMan スクリプトで書いたワークフロージョブを DAGMan ワークフローエンジンで実行を行うシステムである。GridAnt/Karajan は Java Commodity Grid(CoG)Kit に付属しているワークフローのスクリプト言語である。Taverna はデータベース検索などの Web サービスをワークフロー化するツールである。NAREGI ミドルウェアでも、ワークフローツールを提供している。しかし、これらのワークフローツールは、1 つのミドルウェアに対応するもので、複数の異なるミドルウェアに跨ってジョブを実行することができないという問題がある。

著者らは現在、研究室の計算資源(LLS: Laboratory Level System)と情報基盤センタ等でグリッド運用される計算資源 (NIS: National Infrastructure System) 間でシームレスなジョブの実行や複数のジョブの連携実行を実現するミドルウェア技術の検討を行っている。本稿では、利用者が計算機の運用形態を意識することなしに、LLS や NIS に対するシームレスなジョブ投入やモニタリングを可能とするワークフローツールの設計と実装について述べる。具体的には、LLS の計算機の運用のためのミドルウェアとして、負荷が軽く手軽にインストールできるミドルウェアとされる UK e-Science 開発の GridSAM サービスや、Globus Tool Kit のジョブマネージャである WS-GRAM サービスを用いる 3)10)。NIS の計算機の運用のためのミドルウェアとして NAREGI サービスを用い、両システムがユーザにとってシームレスに見えるワークフローツールを設計し、実装と評価を行う。アプリケーションレベルのワークフローの記述のために NAREGI ミドルウェアで導入されている NAREGI-WFML をベースに、ジョブサービスサブリットとして、各ミドルウェア対応のジョブの実行、モニタリングを行える方式を提案する。

[†] 国立情報学研究所
National Institute of Informatics

2. NAREGI-WFML

NAREGI ミドルウェアのワークフローツールでは、科学計算分野で多種多様な要求（大量なバルクジョブ、広域 mpi 実行など）に応えるため、ワークフローの各ジョブの実行要件の指定に標準化されつつある JSDL(Job Submission Description Language)を独自に拡張したものを適用している 7)8)。また、ワークフローの記述に関しては以下のような 2 段構造となっている。すなわち、NAREGI-WFML(NAREGI WorkFlow Markup Language)というアプリケーションレベルの記述と、ワークフロージョブを NAREGI 環境に投入する際に、NAREGI-SS(Super Scheduler)が提供するジョブ実行に関するグリッドサービスを使用するために、Web サービスの連携言語である BPEL による記述からなる 8)9)。

NAREGI-WFML は、NAREGI ミドルウェアの基盤部分(資源管理、情報サービス、ブローリング等)のアーキテクチャの変化にあまり影響を受けずに、ユーザインタフェースであるワークフローツール(WFT)の開発ができるように、共通中間語的な位置づけで考案した言語である。ユーザが GUI で記述するワークフローを NAREGI-WFML の形に表現し、ジョブの投入時に、基盤モデルに依存した形に変換を行う。今までに開発したアーキテクチャ依存部は

- UNICORE4(AJO-AbstractJobObject-向け変換
- NAREGI-UNICORE(AJO JSDL 埋め込み)向け変換
- NAREG-SS(BPEL JSDL 埋め込み)向け変換

NAREGI-WFML のループ、ブランチ機能の処理を除くシンタックス構造を図 1 に示す。NAREGI-WFML は、ワークフローによるグリッドを構築するためのアプリケーション間でグリッドサービスの連携のために提案された GSFL (Grid Service Flow Language)の activity の単位をグリッドサービスレベルからタスクレベルに変えて設計開発した言語である 11)。NAREGI-WFML の構文の特徴として、ワークフローにおける最小の仕事の単位である activity を定義する部分と、activity 間の関連について規定する部分に分かれている。前者の定義を activityModel と呼び、後者の定義は compositionModel と呼ぶ。また、activity 全体で一つのワークフローを定義するだけでなく、小さな activity のグループ毎に部分的なワークフローを定義し、これを再び activity として再定義できる（この再定義された activity を importActivity と呼ぶ）。この特徴により、ワークフローを階層的に構築できることから、複雑なワークフローを生成することができる。さらに、適用するサービスを指定するため、ServiceProvider 要素がある。NAREGI では 1 つのサービスしかないので NAREGI-type のみの指定であった。この 3 つが NAREGI-WFML におけるトップレベルの構文である。activity には 5 種類ある。プログラムの実行を示す場合には、jsdl タグを用い JSDL によるプログラム要件を指定し、ファイルステージングのための transfer タグを用いソースとターゲット

の URI を指定する。図 2 に、NAREGI-WFML 構文と実際の GUI 上のアイコンの関係を示した。すべての制御フローの起点となるのが exportedActivity で、その中に、importActivity もしくは activity を含んでいる。また importActivity は、さらにその中に importActivity もしくは activity を含んでいる。アイコン図ではワークフローアイコン（アイコン部に図形が記述）が、importActivity に対応し、プログラムアイコンが jsdl-activity に対応する。また、アイコン間の四角い部分の間を結んだ線が transfer-activity に対応する。

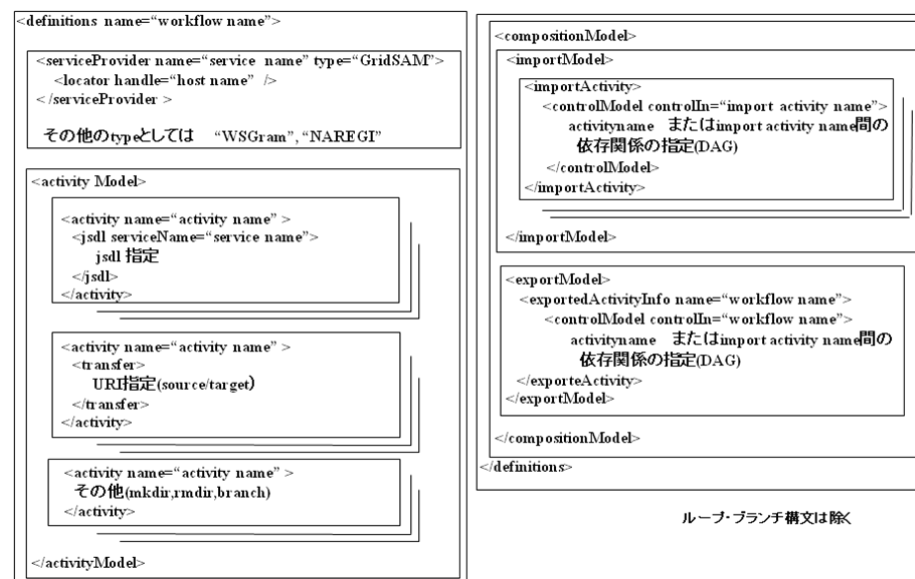


図 1 NAREGI-WFML の構造

3. 設計(実行方式)

著者らは現在、ワークフローツールのプロトタイプ的设计と実装を進めている。本節では、本プロトタイプ的设计について述べる。

3.1 システム構成

図 3 は、本ワークフローツールを用いたシステム構成例を示している。研究室サイドにある資源は、LLS 向けポータルサーバ(LLS-Portal)と、研究室で使用可能な計算サーバである。研究室サーバ上には GridSAM または WS-GRAM サービスがインストールされている。この図で、NAREGI-VO1 とは、VO1 という仮想組織が使用を許可された NAREGI 資源である。LLS-Portal 中の WFT/WFT エンジンが、ユーザが記述したワークフロージョブを、activity ごとに指定された LLS 資源または NAREGI 資源にジョブを投入し、ジョブの実行順序の制御とモニタリングを行う。図では、pre/post process ジョブ および sim1 ジョブが研究室資源のサーバ側で実行され、sim2/sim3 ジョブが NAREGI 資源で実行される例を示している。なお、AHS は ApplicationHostingSystem で、プログラムのデプロイ等を行うために開発されるサービスである 12)。

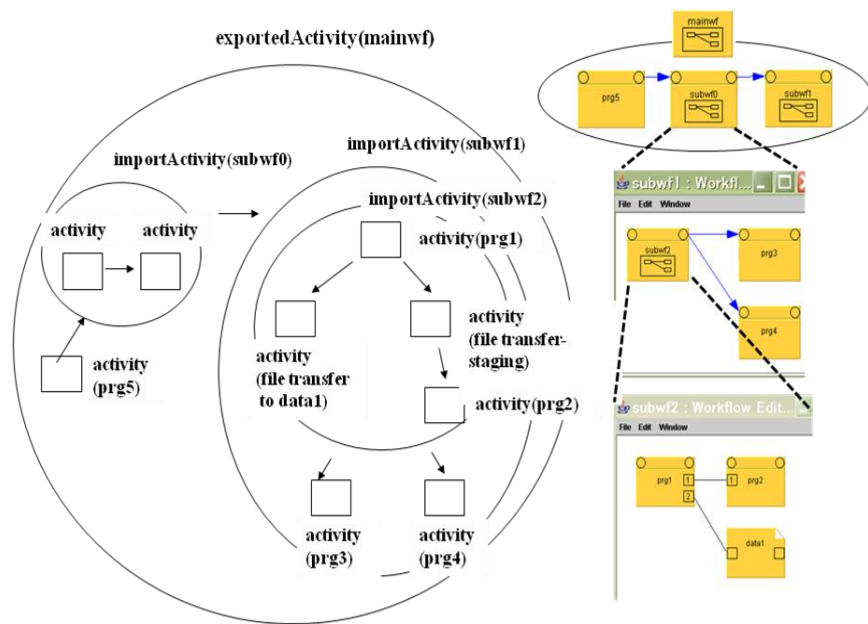


図 2 NAREGI-WFML と GUI

本プロトタイプでは、認証のための MyProxy/VOMS/UMS サーバを NAREGI 内のものを共有して使う仕組みとしている。このため、NAREGI の仮想組織のメンバとして登録していなければ、LLS-Portal を通して研究室側の資源を使うことができない問題はある。また、1つの NAREGI-VO に複数の研究室が LLS-Portal を建てて接続することが可能である。その際、NAREGI の VO 管理者としては、LLS-Portal の NAREGI 資源へのアクセスを許可する運用を行えばよい。

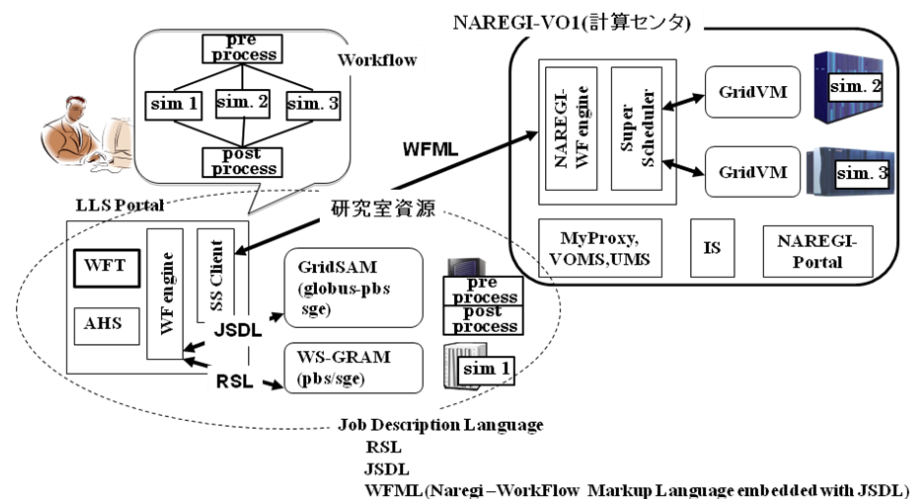


図 3 システム構成

3.2 LLS 向けジョブ実行ミドルウェア

LLS 向けミドルウェアとして、UKe-Science が開発した GridSAM および globus の WS-GRAM ミドルウェアをユーザインタフェースから選択できるようにした。それぞれ、ジョブの submit, cancel, query の java の API が提供されている。これらのインタフェースは標準化の動き (BES: Basic Execution Service) があるが、これらのミドルウェアはまだ対応していない^a 13)。

GridSAM は複数の PBS/SGE/globus 等のジョブマネージャを選択できるが、以下の観点から globus(gt2)を選択した 14)15)。研究室資源と NAREGI 資源でのデータの連携において広域共有ファイルシステムとして gfarm を用いることを考えているが、その

^a なお、GridSAM には試験的な実装が提供されている。

際の認証として gsi 認証がふさわしいとされているためである 16)。また、globus であれば、gridmapfile でマッピングされたローカルアカウントで実行されるが、他のジョブマネージャでは、すべてのジョブ実行が GridSAM サービスの実行アカウントで実行が行われる。このため、計算結果を各ユーザのアカウントのファイルに転送するための処理が必要となる問題がある。なお、globus のジョブマネージャは、設定により pbs/sge 等を選択ができるが、GridSAM からは globus のジョブマネージャを選択できる機能がないため、globus のデフォルト設定のジョブマネージャが使用される。なお、GridSAM は jobPersistence 等に優れているミドルウェアとされている。

WS-GRAM の場合、WS-GRAM の中から、globus のジョブマネージャを選択できる API があるので、WFT からジョブマネージャとして、sge/pbs(torque)が選択できるようにしている。本来は、ジョブマネージャの違いは意識させないほうが望ましいので不要の機能であるが、評価で述べるようにその差が完全には隠ぺいできなかったため、残してある。

3.3 ユーザインタフェース

ユーザが、ワークフローの各ノードのジョブやデータファイルの要件を記述するには、NAREGI と同様にジョブに関してはプログラムアイコンのプロパティ画面で記述

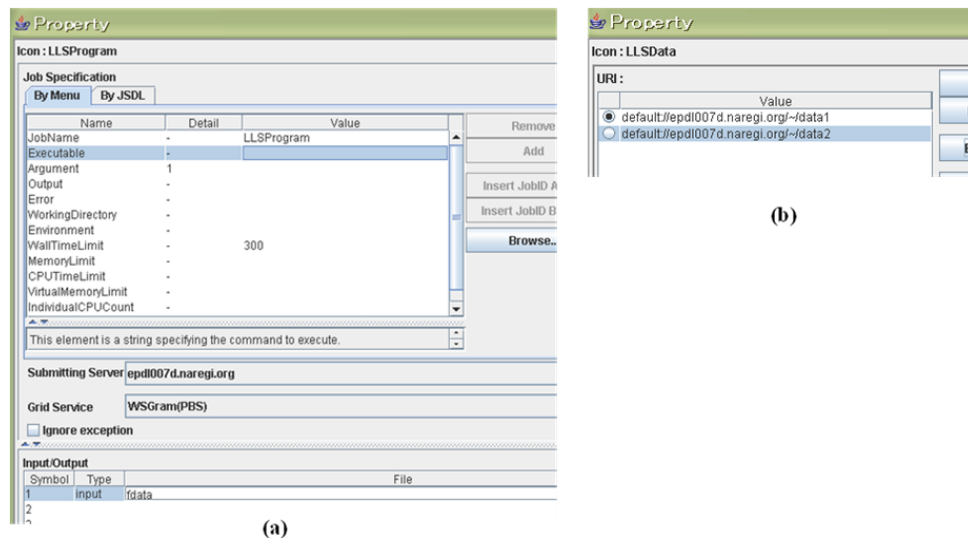


図 4 プログラムアイコン(a)とデータアイコン(b)のプロパティ

し、データファイルに関してはデータアイコンのプロパティ画面で記述する。記述は、LLS 部分に対しても NAREGI と同様に標準化されつつある JSDL の指定項目を指定する。図 4(a)は LLS プログラムアイコンのプロパティ画面であるが、NAREGI の画面との違いは SS(SuperScheduler)がプロカリングを行うために必要な要件のフィールドが存在しないだけである。データファイルを指定するデータアイコンに関しては、図 4(b)の如く、URI を指定する。複数の URI が指定可能で、その場合、ジョブ投入前に実際に必要なファイルを選択する必要がある。データアイコンと、プログラムアイコンを結ぶことで、プログラムアイコンの Input/Output 領域で指定したファイルとステージングが行われる。

3.4 Workflow 制御

ワークフローに関しては、ユーザが GUI で記述したアイコンをもとに、NAREGI-WFML を生成してワークフローの制御を行う。NAREGI-WFML においては、適用するサービスを指定するため、ServiceProvider 要素で、NAREGI, WS-GRAM, GridSAM のサービスの定義を行い、各 activity が提供したいサービスを指定する。

全体のワークフローの記述から WFT 内のワークフローエンジンは、次のようにジョブを切り出して、各グリッドサービスにジョブを投入しモニタリングを行う。WFT は、LLS 計算機へのジョブの投入とモニタリング、LLS 計算機間のファイルステージングを直接実施し、NAREGI 資源内の計算機に対するワークフロージョブ投入とファイルステージングは NAREGI-SS サービスを経由して行う。なお、LLS と NAREGI 間でのデータ連携は gfarm を用いた広域共有ファイルシステムを用いる。ここで、図 5 のようなワークフロー例での Workflow 制御を示す。この図で、黄色のアイコンが LLS 内の計算資源に対するジョブ、黄土色のアイコンが NAREGI 計算資源でのジョブを意味する。即ち、ファイル転送(transfer1,2,3)が NAREGI 内、(transfer4,5)が LLS 内、job1 が NAREGI 内、job2,job3 が LLS に対するジョブ実行であり、ワークフローアイコンで示される WorkflowJob1 は、この中の子孫はすべて NAREGI での実行であると仮定する。全体のワークフローを表現した NAREGI-WFML を解析し、以下の処理で Workflow の制御を行う。

1. transfer1,transfer2,job1,transfer3 のファイルステージングと job1 実行をまとめて表現する WFML を作成して、NAREGI にワークフロージョブの投入とモニタリングを行う。
2. ファイル転送 transfer4, LLS ジョブ実行 job2, ファイル転送 transfer5 を順次 LLS 内で実行する。ファイル転送に関しては WFT から gridftp によるファイル転送を行い、LLS ジョブに関しては、指定の LLS ジョブサービスにジョブの投入とモニタリングを行う。ファイル転送の URI は WFML から

求め、job2 に対するジョブ要件の記述である JSDL を含む WFML を作成する。

3. 1.2 が終了後に、LLS ジョブである job3 を LLS のサービスにジョブ投入とモニタリングを行う。
4. 3.が終了後、WorkflowJob1 のサブワークフローを表現する WFML を作成して NAREGI-SS にワークフロージョブの投入とモニタリングを行う。なお、WorkflowJob1 の子孫全体が NAREGI に対するものでなければ、同様に、LLS 部分と NAREGI 部分を切り出す処理を行う。

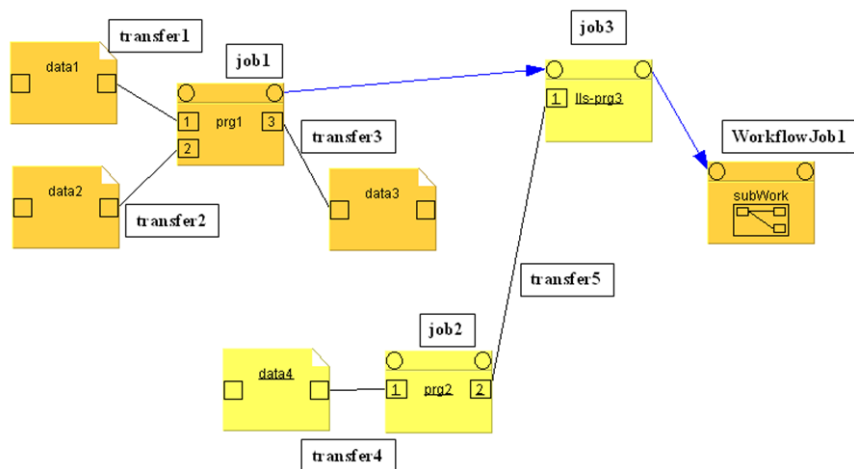


図 5 Workflow の例

4. 実装と評価

4.1 実装方式

NAREGI では、ワークフローエンジンの機能は NAREGI-SS に存在したので、NAREGI -WFT にはワークフローのエンジンの機能はない。LLS-WFT にはワークフローの制御のためにエンジンの実装が必要である。また、対応するミドルウェアは基

本的に jobSubmit/jobCancel/jobQuery の機能があるが、BES のような標準インタフェースの実装はないため、最小限の工数で対応する必要がある。

図 6 に実装する LLS-WFT の構造を示す。ワークフロージョブに対して 1 つのワークフロージョブを表現するインスタンスをつくり、Java1.5 から導入された ExecutorService を用いたスレッドプールに投入して並列処理を実現する。また、各ワークフロージョブの中の、ワークフロー制御において、ジョブの状態を知る必要があるが、GridSAM は gt2 の WS-GRAM を使用しているため notification の機能がなく、NAREGI-SS はまだ notification の機能を公開していないため、polling によりジョブの状態を検出した。各スレッドの run() メソッドでは、ワークフローインスタンスの中で、実行可能な job に対して submit を行い、一定時間 sleep した後、モニタリングを行うというループとなっている。また、インタフェースの共通化を図るため、図のようにサービスのプラグイン化を図り、ミドルウェアによって異なる部分は各プラグインモジュールで対応するようにした。ジョブ submit の際に必要なジョブ情報としては、全体の WFML から投入単位（図 5 の例 taransfer1,2,3 job1 はまとめて NAREGI サービスへの 1 つの投入単位）の情報を表現する WFML を作成して、プラグインモジュールに送る。GridSAM プラグインでは、WFML から JSDL を抜き出して GridSAM サービスに投入し、WS-GRAM プラグインの場合は WFML から JSDL を抜き出し、RSL に変換して WS-GRAM サービスにジョブを投入する。NAREGI プラグインでは、入力

の WFML をそのまま、NAREGI-SS サービスに投入する。また、各種サービスのクライアントであるプラグイン部に関しては、クラスライブラリに競合があるので、別々の web アプリケーションとしてサーブレットに実装を行い、名前空間を分けることにより競合を回避した。

ジョブの状態(Pending,StageIn,Active..等)に関してミドルウェアによって異なるので、各モジュールは NAREGI-SS の状態の定義にマッピングして返却している。また、ユーザに提示する log 等の情報は、NAREGI の形式と同一にしている。そこでは、各 activity の情報を表示し、ミドルウェアの処理中に検出する例外は、cause 欄に表示している。GridSAM の場合は、各ステージ情報の保持する記述を取得し、WS-GRAM の場合は、GramJob の API から、フォールトタイプの記述である WS-BaseFaultsb から記述を取得して表示する。なお、ユーザプログラムのエラーは JSDL で指定した標準エラーファイルに書かれる。このファイルに関しては、WFT の log 画面に表示することにした。

b 異なるベンダなどにより開発された Web サービス間のインタオペラビリティを保つために、基本となる Fault メッセージの型と、その使用方法を定義

4.2 評価

4.2.1 簡単な性能評価

本研究の目的の1つは、LLSの資源とNAREGIに対するジョブ投入やモニタリングをシームレスなインタフェースで行えることである。ユーザに対して、LLSおよびNAREGIともJSDLでジョブ要件を指定するシステムを提供できたので達成できた。また、LLSとNAREGI間のジョブ連携に関しても、NAREGI-WFMLをベースにワークフローを制御する仕組みを実装したことで達成できた。と考える。

次に、LLSとして使用したGridSAMおよびWS-GRAMに関してNAREGIと比べた場合の負荷を評価する。以下に、/bin/hostnameを各種ミドルで投入した場合の、WFTからジョブを投入してから、結果が得られるまでの経過時間を示す。ジョブそのものの実行時間はほとんどないので、ミドルウェアの処理時間を意味する。

WFT(WS-GRAM):	13 秒
WFT(GridSAM):	32 秒
WFT(NAREGI):	40 秒 ^c

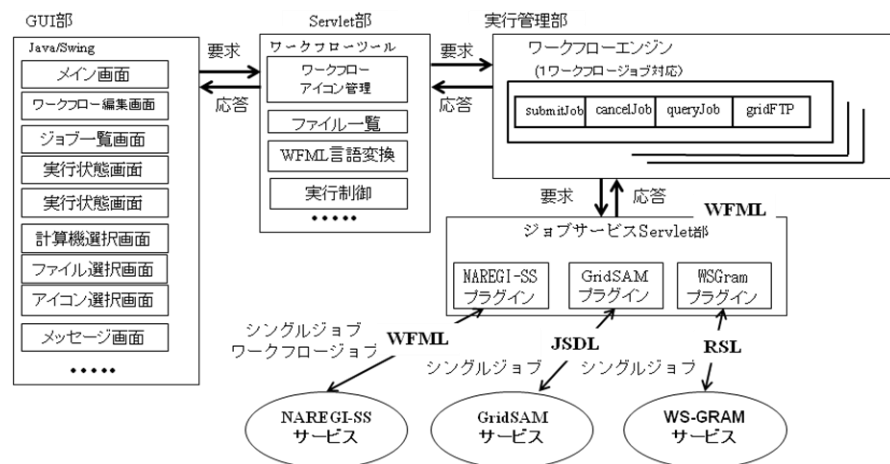


図6 LLS-WFTの構造

^c 標準のNAREGI-SSの設定では50秒以上の時間を要する。

なお、上記で使ったWFTが稼働しているサーバはXeon 2.8Ghz, 1Gbyteメモリの環境で、PBSがすぐジョブをスケジュールする設定で評価した。また、qsubコマンドによる経過時間は1秒程度、globusrun-wsコマンドを使用した場合は6秒程度であった。従って、LLS環境に、WS-GRAM等のミドルウェアを採用する効果は十分あると言える。

NAREGI-WFTと異なり、LLS-WFTでは、ワークフローエンジンの機能が新たに実装されるので、エンジンの負荷を評価する必要がある。WFTのスレッドプールの大きさを30と固定し、各ワークフロージョブのモニタリング間隔を10秒として、/bin/sleep 3000のジョブを100本投入した場合、topコマンドによるcpu利用率は、WS-GRAMとNAREGI-SSでは0%から最大25%、GridSAMの場合は、0%から最大90%程度まで周期的挙動を示していた。WS-GRAMとGridSAMでの差はおそらく、gt2とgt4の差であると思われる。また、この負荷を軽減するには、WS-GRAMの場合には、notificationの機能があるので、この機能を使いpoolingをやめることが必要である。

4.2.2 実装の際に判明した問題点

まだ、評価の途中であるため、以下に現在までに判明した問題点を書く。これらはBESなどのジョブ実行に関する標準化で解決されるだろう問題である。

1. GridSAMおよび、WS-GRAM(SGE)^dは、アプリケーションのexitCodeを知る機能がない。従って、アプリケーションが異常終了しても、ミドルウェアとしては正常に終了した場合には、ワークフローを止めるべきか判断できず続行してしまう問題がある。一方、NAREGI-SSでは、ミドルウェアとアプリケーションのexitCodeのANDをとった状態の返却を行っており、WS-GRAM(PBS)の場合は、アプリケーションのexitCodeが取得できるので、ワークフローの制御に利用が可能である。
2. ローカルスケジューラSGE/PBS(torque)の差が隠ぺいできないケースがある。同一名の標準出力エラーファイルがある場合、PBSの場合はオーバーライトされ、SGEの場合、アペンドされてしまう仕様である。
3. Globusのジョブマネージャの設計による差がユーザに見えてしまう事がある。Executableファイルが存在しない、または、存在しても実行属性がついていない場合、WS-GRAM(SGE)では、ミドルウェアの例外として示される。WS-GRAM(PBS)では、ミドルウェアは正常終了の扱いとなり、標準エラーファイルとして表示される。しかし、GridSAMの場合は、ミドルウェアは正常終了し、かつ、標準エラーファイルも出力されない問題が

^d SGEのglobus-jobmanerには、exitCodeを取得するAPIがあるが、バグがありすべてexitCode=0が返却される。現在のところバグ対策はされていないようである。

あった。そのため、GridSAM の場合に限り、WFT がジョブ投入の前にファイルのチェックを `gridftp` を用いてチェックする実装を行なった。

4. 今回の発表では、LLS に関してはシングルプロセスジョブを想定しており、`mpi` ジョブは対象としていない。しかし、WS-GRAM(PBS)であれば、JSDL 中の `TotalResourceCount/IndividualCPUCount` に必要なノード数とノードあたりの CPU 数を指定すれば、Gobus4.0.5 の拡張 Job Description Support の `node selection parameter` を WFT が設定するので、以下のようなシェルスクリプトを JSDL 中の Executable として指定すれば、PBS のリソースアロケーションと連携が行われ `mpi` 実行が可能と考えられる 17)。

```
#!/bin/sh
NPROCS=`wc -l < $PBS_NODEFILE`
mpirunのパス -np $NPROCS -machinefile $PBS_NODEFILE
バイナリファイルのパス
```

5. まとめと今後の課題

本研究では、研究室 (LLS) および情報基盤センター (NAREGI) の計算機間でジョブを連携して実行するためのワークフローに関する検討を行い、プロトタイプシステムの設計と実装に着手した。具体的には、(1)研究室のユーザがワークフローを編集し、かつワークフロー中のジョブの実行制御を行うための LLS-WFT の設計と実装、(2)LLS-WFT から NAREGI-SS に対してジョブの投入およびモニタリングを行うためのインタフェースの設計および実装、(3)LLS-WFT から GridSAM または WS-GRAM サービスに対してジョブの投入およびモニタリングを行うためのインタフェースの設計および実装を行った。

以上の結果、図 3 に示す環境構築を行い、LLS-WFT から NAREGI-SS および LLS 環境として、GridSAM または WS-GRAM サービスにジョブを投入する機能、および投入後のジョブをモニタリングする機能の試作に成功した。

今後の課題としては以下の点があげられる。図 3 の構成では、LLS-WFT が実装される LLS-Portal ノードは、NAREGI 資源のあらゆるノードと通信が必要であり、運用管理上大きな問題となる。以下の方法により解決する予定である。LLS-Portal(WFT)と NAREGI 資源との通信相手は NAREGI-Portal ノードに限り、かつ `https` プロトコルに使用を制限する。他の NAREGI ノードに処理依頼をするために、LLS-WFT から NAREGI-Portal(WFT)ノードに連携の機能を追加する。

謝辞 本研究は、文部科学省の次世代 IT 基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」(研究コミュニティ形成のための資源連携技術に関する研究)の成果である。また、本研究を遂行するにあたり貴重なご助言、ご討論を頂いた(株)日立東日本ソリューションズの庄子智誉氏に感謝いたします。

参考文献

- 1) NAREGI, <http://www.naregi.org/>
- 2) Matsuoka, S. et al. : Japanese computational grid research project: NAREGI, Proceedings of the IEEE, Vol.93, No.3, pp.522-533(2005).
- 3) Globus Toolkit, <http://www.globus.org/toolkit>
- 4) UNICORE(Uniform Interface to Computing Resources): <http://www.unicore.eu/>.
- 5) Taverna: workbench: <http://taverna.sourceforge.net/>.
- 6) Yu, J. and Buyya, R. : A Taxonomy of Scientific Workflow Systems for Grid computing, <http://www.gridbus.org/papers/workflow-sigmod05.pdf>.
- 7) Job Submission Description Language(JSDL) Specification, Version 1.0: <http://www.gridforum.org/documents/GFD.56.pdf>.
- 8) 畑中正行 他: OGSA アーキテクチャに基づく NAREGI スーパースケジューラの設計と実装, 情報処理学会研究報告, Vol.2005, No.57, pp.33-38(2005)
- 9) Business Process Execution Language for Web Services version 1.1: <http://www.ibm.com/developerworks/library/specification/ws-bpel/>.
- 10) GridSAM- Grid Job Submission and Monitoring Web Service, <http://gridsam.sourceforge.net/>.
- 11) Krishnan, S. Wagstrom, P. and Laszewski, G.: GSFL: A Workflow Framework for Grid Services, <http://www-unix.globus.org/cog/papers/gsfll-papers.pdf>.
- 12) 宇佐見仁英 他: 異種グリッドミドルウェアに跨るアプリケーションホスティングサービス (AHS) の設計と実装 SWoPP 仙台 2009,
- 13) OGSA Basic Execution Service: <http://www.ogf.org/documents/GFD.108.pdf>.
- 14) TORQUE RESOURCE MANAGER: <http://www.clusterresources.com/products/torque-resource-manager.php>.
- 15) Sun Grid Engine 6.1: <http://jp.sun.com/products/software/gridware/>
- 16) Gfarm File System <http://sourceforge.net/projects/gfarm>
- 17) Job Description Extensions Support: http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Job_Desc_Extensions.html.