社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－HPC－102（6）
2005／6／2

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

畑中　正行 †1　中野　恭成 †1　井口　裕次 †1
大野　利男 †2　佐賀　一繁 †3　秋岡　明香 †3 (†6)
中田　秀基 †4,†3　松岡　聡 †5,†3

本稿では、OGSA-EMS のサービス・アーキテクチャに沿った NAREGI スーパースケジューラの設計と実装について述べる。NAREGI スーパースケジューラの実装をとおして OGSA-EMS アーキテクチャの実現可能性を確認すると同時に、ヘテロかつ多数台の計算資源を要求する MPI 並列ジョブの自動資源割当における OGSA-EMS 仕様の問題点を明確化するとともに それを解決する OGSA-EMS 構成要素への拡張を提案する。

Design and Implementation of NAREGI Super-Scheduler based on OGSA Architecture

Masayuki Hatanaka †1, Yasumasa Nakano †1, Yuji Iguchi †1,
Toshio Ohno †2, Kazushige Saga †3, Sayaka Akioka †3 (†6),
Hidemoto Nakada †4 , †3 and Satoshi Matsuoka †5 , †3

In this paper, we describe design and implementation of NAREGI Super-Scheduler based on OGSA-EMS Architecture. Through our experience of its design and implementation, we made sure that OGSA-EMS architecture is feasible. Also, we clarify the issues for the specification on resource allocation of a MPI parallel job that requires heterogeneous and many computational resources, and propose a set of extensions to OGSA-EMS components to resolve the issues.

1. はじめに

本稿では、 GGF (Global Grid Forum) の OGSA (Open Grid Services Architecture) 仕様 [1] の中の実行管理サービス群アーキテクチャに沿った NAREGI スーパースケジューラについて述べる。

NAREGI スーパースケジューラは、グリッド環境において高度なグリッド・アプリケーションに対する実行を可能にするための、ヘテロかつ多数台の計算資源を要求する MPI 並列ジョブの自動資源割当及び実行を可能にするグリッド・サービス群である。

本稿では、連成ジョブのような高度なグリッド・アプリケーションの資源割当における OGSA アーキテクチャの問題点を明確化するとともに、NAREGI スーパースケジューラにおいてなされた、これら問題点への解決方法及び OGSA 関連仕様の拡張について説明する。それからその実装・評価をとおして、OGSA アーキテクチャの妥当性について述べる。

本稿の構成は次のとおりである。2節で OGSA 仕様の概要を述べる。3節で NAREGI スーパースケジューラの設計概要を説明する。4節でスーパースケジューラの実装を概説する。5節でまとめと今後の課題を述べる。

†1 富士通株式会社 Fujitsu Limited
†2 富士通プライムソフトテクノロジ Fujitsu Prime Software Technologies Limited
†3 国立情報学研究所 National Institute of Informatics
†4 産業技術総合研究所 National Institute of Advance Industrial Science and Technology
†5 東京工業大学 Tokyo Institute of Technology
†6（現所属）ペンシルベニア州立大学 Pennsylvania State University

2. OGSA の概要

OGSA (Open Grid Services Architecture) [1] 仕様は、グリッド・コンピューティング分野の標準化団体である GGF (Global Grid Forum)の中の OGSA-WG において策定された仕様である。この仕様は、グリッド・システムにおける中核のグリッド・サービス群のアーキテクチャを定義する。 OGSA 仕様では、この基

本的かつ共通のウェブ・サービス (OGSA サービス) の上に、各固有の問題領域のサービスが定義されることを想定している。OGSA サービスは、(1) 実行管理サービス群、(2) データ・サービス群、(3) 資源管理サービス群、(4) セキュリティ・サービス群、(5) 自己管理サービス群、(6) 情報サービス群の6つに大別されており、さらに、(1) の実行管理サービス群 (Execution Management Services; 以降 OGSA-EMS と略す) は次のように3つに分類される。

1. SC (Service Container) サービス
2. JM (Job Manager) サービス
3. 資源選定サービス (Resource Selection Services)

SC サービスは、ジョブやサービスの実行環境を提供する。 SC は、分散資源管理のインターフェイスを介して、サービス固有の属性として資源を他のサービスに開示する。JM サービスは、ジョブの起動から終了までのジョブ実行の全処理過程をカプセル化する上位のサービスである。JM は、ジョブ文書を管理し、WS-DM 仕様のインターフェイスを介して他のサービスにジョブ文書を開示する。ジョブ文書は、ジョブ投入記述言語 (JSDL; Job Submission Description Language) [3] の文書や、契約書 (Agreement document) や、ジョブ状態等のサブ文書から構成される。JM は、バッチ・キュー、ポータル、ワークフロー、ジョブアレイ等の上位サービスである。資源選定サービス (以降、OGSA-RSS と略す) は次の3つのサービスからなる。

1. EPS (Execution Planning Services)
2. CSG (Candidate Set Generator)
3. RS (Reservation Services)

OGSA-EMS アーキテクチャにおけるサービス間の典型的なやりとりを図1に示す。

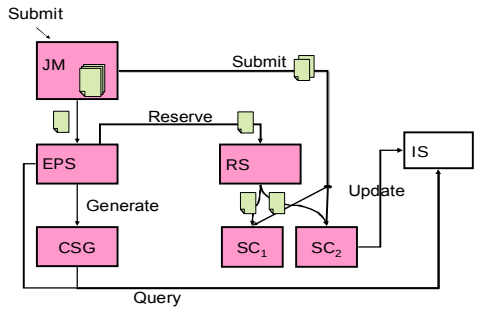

図1 OGSA-EMS の構成
サービス名の略記は、本文中に沿うものとする。ここで IS は、OGSA の情報サービス群を総称する Information Services の略である。

以下の各項では、OGSA-RSS の EPS, CSG 及び RS について概説する。

2.1. EPS (Execution Planning Services)

EPS は、ジョブと資源の間の時間制約付きの写像を構築するサービスである。この写像をスケジュールと呼ぶ。典型的には、EPS は、実行時間、費用、信頼性等の目的関数を最適化して、スケジュールを生成する。EPS は、スケジュールを生成するだけで、スケジュールの実行・管理は Job Manager によってなされる。EPS は通常、ワークフロー・エンジンのような上位サービスである Job Manager から呼び出され、スケジュールを生成するために CSG や情報サービスを呼び出す。

2.2. CSG (Candidate Set Generator)

CSG は、ジョブを実行可能なサービス・コンテナ (のロケーション等のメタデータ) のセットを生成する。CSG は通常、EPS から呼び出され、ジョブ要件を取り出すためにジョブ文書にアクセスし、資源を探索するために情報サービスに照会し、ジョブが実行可能かを確認するためにサービス・コンテナとやりとりする。

2.3. RS (Reservation Services)

RS は、グリッド上の予約可能な様々な資源に対し共通のインターフェイスを提供する。予約可能な資源には、計算資源だけでなく、ストレージ、ネットワーク帯域等がある。予約は複数の下位の予約の集積点である可能性がある。 RS は、ジョブの実行計画を保証するため EPS から呼び出される。

3. NAREGI スーパースケジューラの設計

NAREGI スーパースケジューラ (以降 NAREGI-SS と略す) は、OGSA 仕様のサービス・インフラストラクチャである WS-RF (Web Services Resource Framework) 仕様 [2] の上に構築される。また、NAREGI-SS は、OGSA の実行管理サービス群 (OGSA-EMS) のアーキテクチャに基づくサービスを実現する。

また、NAREGI-SS では、グリッド環境における高度なグリッド・アプリケーションに対する実行を可能にするため、ヘテロかつ多数台の計算資源を要求する MPI 並列ジョブの自動資源割当・実行を可能とすることを目標にした。こうしたアプリケーションの例として、溶液中の分子構造計算のための RISM-FMO 法の連成ミドルウエア Mediator [5] がある。NAREGI-SS では Mediator のような連成プログラムの資源割当・実行を一般的に解決することを目標にして、設計にあ

たり Mediator を事例として OGSA-EMS アーキテクチャ上での連成プログラムに対する自動資源割当について検討した。

3.1. JSDL へのヘテロ資源の記述拡張

OGSA-EMS アーキテクチャにおいて、ジョブの資源要件記述言語の候補仕様は、GGF の JSDL-WG で策定中の JSDL 仕様 [3] である。しかしながら、JSDL は単純ジョブを対象としており、Mediator のように1つの MPI ジョブの中で RISM 法が SMP 型の計算機を要求し、他方 FMO 法がクラスタ型計算機を要求する、MPMD (Multiple Program, Multiple Data) 型のジョブの場合、 JSDL はそのままではヘテロな資源要件を記述できない。このため、JSDL 仕様に対しヘテロ資源を記述するため拡張する必要がある。

JSDL 仕様では XML InfoSet の記法で表現すると、以下のように資源要件を記述する(説明上、関係のない要素は省略されている)。

```
<JobDefinition>
 <JobDescription>
  <Application>
   <POSIXApplication … /> ?
  </Application> ?
  <Resource … /> *
 </JobDescription>
</JobDefinition>
```

ここで、**?** は出現回数 0..1、***** は 0..N、指定なしは 1..1 であることを示す。現状の仕様では POSIXApplication 要素は Application 要素の中に 0 もしくは 1 回しか現れてはならない。つまり、RISM,FMO 及び Mediator の三種類のプログラムを記述できないことがわかる。これに対し Resource 要素は JobDescription 要素の中に 0 回以上現れて構わない。しかし、POSIXApplication 要素と Resource 要素の間を関係づけることができないため、ヘテロな資源を要求できないことがわかる。

そこで、NAREGI-SS では、MPMD 型の MPI 並列ジョブを扱うため、POSIXApplication 要素の出現制約 maxOccurs を unbounded に変更し、プログラム記述の POSIXApplication 要素と計算資源記述の Resource 要素とを対応づけるため、Resource 要素に xsd:ID 型の属性 id と POSIXApplication 要素に属性 ref とを新たに追加した。この拡張は次のように表現される(下線部分)。

```
<JobDefinition>
 <JobDescription>
  <Application>
   <POSIXApplication ref="xsd:ID" … /> *
  </Application> ?
  <Resource id="xsd:ID" … /> *
 </JobDescription>
</JobDefinition>
```

これを具体例で示すと、次のようになる。

```
<JobDefinition>
 <JobDescription>
  …
  <Application>
   <ApplicationName>foo</ApplicationName>
   <Description>MPI</Description>
   <POSIXApplication ref="cluster">
    <Executable>a.out</Executable>
   </POSIXApplication>
   <POSIXApplication ref="smp">
    <Executable>b.cout</Executable>
  </POSIXApplication>
 </Application>
 <Resource id="cluster">
  <CPUArchitecture>x86</CPUArchitecture>
  <OperatingSsystem>
   <Type>LINUX</Type>
  </OperatingSystem>
 </Resource>
 <Resource id="smp">
  <CPUArchitecture>powerpc</CPUArchitecture>
  <OperatingSystem>
   <Type>AIX</Type>
  </OpratingSystem>
 </Reource>
 </JobDescription>
</JobDefinition>
```

この例では、a.out 及び b.out からなる MPMD 型の MPI プログラム foo を、それぞれ計算資源 x86/LINUX 及び powerpc/AIX のマシン上で実行することを要求する(新しく導入した識別子 smp 及び cluster で、アプリケーションと資源が結合される)。

3.2. CSG へのヘテロ資源の対応

CSG サービスは、節 2.2 に説明したように、JSDL からジョブの実行要件を取り出し、そのジョブを実行可能な SC の候補を返す。単一のジョブに対し単一の計算資源を探索し可能な SC の一覧を返す場合に比べ、単一のジョブに対し複数の多様な資源を返す場合は単純ではない。例えば、ヘテロな計算資源を割当てたり、複数台のクラスタにまたがってノードを割当てたりする場合である。こうした場合、CSG はより複雑な資源の組を返す仕組みが必要である。

節 3.1 では、ジョブ投入記述言語である JSDL を拡張し、1つのジョブ要求に対し複数資源の探索記述が可能なように拡張したことを述べた。NAREGI-SS の CSG では、この拡張された JSDL を入力として、資源の探索結果をこの JSDL の中に埋め込み返すという、

インターフェイス設計とした。典型的には CSG は JSDL の Resource 要素の中の HostName 要素に候補となる SC のメタ情報を追加する。

これに対応して呼び出し側の NAREGI-SS の EPS は、CSG よって返された資源候補つきの JSDL を評価し、資源を選定し、スケジュールを作成する。それから EPS は選定結果に基づいて、各実行環境の提供サービスである SC に対しそれぞれ JSDL を生成する。

3.3. EPS への精緻化機能の拡張

RISM-FMO 法の連成ミドルウエア Mediator における FMO 法は、クラスタ計算機向きの計算である。こうした計算では、問題の規模や粒度に依存して MPI のプロセス並列度が増大する場合があり、資源の状況にしたがって地理的に分散した複数のクラスタ計算機にまたがって資源を割当てることが求められる。しかしながら、グリッド環境の中で刻々と変化する利用可能な資源に対して、並列ジョブの投入者が投入毎に何台のクラスタ計算機を使って実行すれば適切なのかを知ることは困難である。

よって、NAREGI-SS の EPS は、並列ジョブ投入者によって記述された抽象的かつ不完全な JSDL から、その中の要求プロセス並列数 (ProcessCount 要素) にしたがって、計算資源の選定処理で決定したクラスタ計算機 (SC) に対し具体的な JSDL を生成する。

3.4. JM への実行開始条件つきワークフロー拡張

一般的に、MPI でこうした複数のクラスタ間の MPI プロセスを連携するためには、異なる MPI 実装間や異なるシステム間の MPI プロセスの相互接続を可能にする Interoperable MPI (IMPI) 仕様 [4] のサーバ機能を実装したプログラム（例えば、インディアナ大で開発された impi-server）を起動する必要がある。MPI プログラム側も IMPI 仕様のクライアントを実装した MPI 言語ミドルウエア（例えば GridMPI）を使う必要がある。この場合 MPI プログラムの起動に先立って、IMPI サーバを起動する必要がある。

よって EPS は、資源の状況にしたがって、複数の SC に対する JSDL の生成だけでなく、こうした MPI ジョブに対し例えば図2のようなワークフローを生成する必要がある。このワークフローは、 Job Manager によって新しいワークフロー・インスタンスとして実行されるが、さらにこの実行にあたっては、ワークフロー・タスクの実行開始条件に次のような制約があることを、説明の簡単化のため 256 プロセスを要求する SPMD 型の MPI 並列ジョブを使って、具体的に見てゆく。

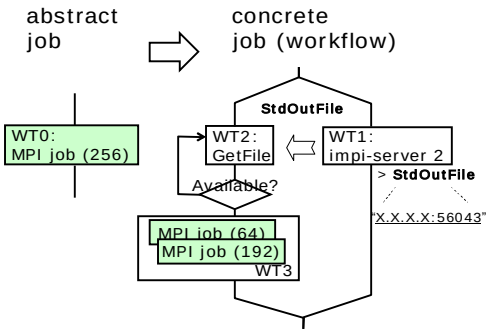

図2 MPIジョブの具体化

ワークフロー・タスク WT1 の impi-server は、IMPI 仕様のサーバ実装である。impi-server は引数として IMPI クライアントの数を指定することが必須である。このため、最初に WT3 タスクの MPI プログラム本体の資源を決定する必要がある。次に IMPI クライアントの数（この例では SC サービスの数で 2 である）を impi-server の JSDL に反映し資源を割当てスケジュールし実行する。次に、impi-server が起動時に動的にバインドした “<IP アドレス>:<ポート番号>” の形式の着呼用 TCP トランスポート・アドレス/ポート対が標準出力に出力される。WT2 タスクは、この出力をファイル転送または SC のジョブ資源アクセスメソッドの呼出しをとおして、データの中身を解析し IP アドレス/ポート対を取り出し、次のような JSDL の Environment 要素を生成する。

```
<Environment name=”IMPI_SERVER”>
  X.X.X.X</Environment>
<Environment name=”IMPI_PORT”>
  56043</Environment>
```

これらの情報は、WT3 の MPI プログラム起動時に GridMPI が使用する環境変数である。WT2 タスクの完了時点で、impi-server と MPI プログラム本体の起動順序の同期が図られる。しかし WT3 は、既に資源割当済のため、その JSDL の内容を変更する場合、再折衝が必要である。JM は、環境変数の追加のため、各 SC に対し契約書の再折衝を要求する。その後、事前予約した時刻に MPI プログラムが各 SC 上で起動され、与えられた環境変数 IMPI_SERVER 及び IMPI_PORT をもとに MPI 言語ミドルウエアから impi-server にアクセスし、全 MPI プロセスの同期後、MPI_init() から処理が始まる。

この考察のように、既存ツールを組合せて利用する場合、予め資源が決定している場合ことを前提とした部品が存在したり、既知のローカルなシステムでシェ

ルプログラムを使って簡単に実装したりしていたことが、自動の資源選定サービスと組合せる場合に、大きな影響がある。潜在的にこうした事例が数多く埋もれていることも想定した場合、グリッド環境におけるワークフロー・エンジンは資源割当及び実行の開始制御を汎用的に記述できることが望ましい。

タスク	条件	契機	操作	変数
WT1	1	allocate	import	NClusters
	2	allocated	export	Host
WT2	3	allocate	import	Host
	4	executed	export	AddrPort
WT3	5	allocated	export	NClusters
	6	reallocate	import	AddrPort

表1 割当／実行開始条件

「契機」の列で、allocate は資源割当の開始前に「操作」が完了済でなければ割当不可、allocated は資源割当の完了後に「操作」を実行、executed は、タスクの実行後に「操作」を実行、reallocate は、allocate 実行後、「操作」が完了済でなければ再割当不可であることをそれぞれ意味する。「操作」の列で、import は「変数」を読出しタスクに設定、export はタスクから「変数」に書込むことを意味する。

NAREGI-SS では、資源割当とタスク実行の処理開始条件の記述可能なワークフロー・エンジンを導入する。前提として、ワークフローの中の各タスクは、資源割当の完了の前に実行することはできない。表1は、図2で説明したワークフローの資源割当 / 実行の開始条件を記述している。このエンジンはこの開始条件に基づいて、ワークフローを次のように処理する。まずタスク WT1 の資源割当には、NClusters（IMPI クライアント数）が決定している必要がある(条件1)ので、(条件 5) より最初に WT3 の割当がなされる。WT1 の割当後 Host（WT1 の実行ホスト）が決定する(条件 2)ので、(条件 3)より WT2 の割当がなされる。WP1 と WT2 を実行した後、AddrPort（IMPI サーバのアドレス / ポート対）が決定する(条件 4)ので、(条件 6)より WT3 の再割当がなされ、WT3 の実行がなされ、ワークフローの実行が完了する。

4. NAREGI スーパースケジューラの実装と評価

4.1 NAREGI-SS の実装

NAREGI スーパースケジューラ (NAREGI-SS) では、Globus Toolkit 3.9.4 の提供する WS-RF 仕様 [2] の実装上に、OGSA-RSS の EPS, CSG, RS 及び SC の各サービスを実装した。また、一般的なグリッド・ミドルウエアで実現されている JM に相当するジョブ実行管理機能を重複して開発することは避け、既存の JM と、OGSA-RSS とが協調し合い、全体として OGSA-EMS を提供するような実装とし、NAREGI-JM では、グリッド・ミドルウエア UNICORE と開発した OGSA-RSS とを結合できるよう、UNICORE NJS (Network Job Supervisor) 側に OGSA-RSS 呼出し用インターフェイスである AJO Rewriter Interface を英国の UNICORE 開発チームと共同開発し、このインターフェイスを介した OGSA-RSS アクセス部を開発した。図3に、NAREGI-SS の全体構成を示す。

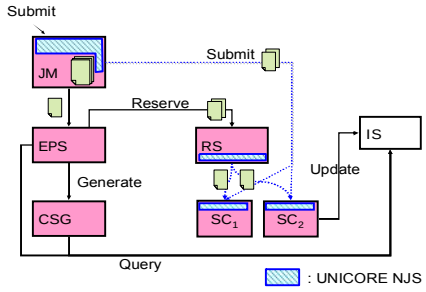

図3 NAREGI スーパースケジューラの構成

4.2 NAREGI-SS の評価

OGSA-EMS アーキテクチャの実現可能性と、高度なグリッド・アプリケーション向けの拡張の妥当性を確認するために実験を行った。

4.2.1 実験環境

実験環境としては、国立情報学研究所の「グリッド基盤ソフトウエア開発システム」を用いた。実験用の計算資源として、表2に示すとおり、4 台の x86 系の Linux クラスタ計算機を用意した。各計算機上では NAREGI-SC が動作し、NAREGI-SC は、利用可能な資源を IS に通知するよう構成されている。

システム名称	アーキテクチャ (OS)	計算ノード数
Cluster#1	x86 (RedHat Linux 8)	1
Cluster#2	x86 (RedHat Linux 8)	2
Cluster#3	x86 (RedHat Linux 8)	3
Cluster#4	x86 (RedHat Linux 8)	4

表2 実験環境の計算機資源

4.2.2 使用ジョブ

評価には、比較的単純な MPMD 型の 16 プロセスの MPI 並列ジョブを使用した。このジョブは、計算

ノード当り2種類の MPI プロセスが対になって動作することを要求するため、実行にあたって全体で 8 台の計算ノードが必要であり、表2より最低 3 台のクラスタ計算機を割当てる必要がある。また、異なるシステム間の MPI プロセスを相互接続するために、impi-server 用の計算機を同時に割当てる必要がある。

4.2.3 実験の概要

NAREGI-SS における多数台の資源探索・割当性能を測定する。JM 側で EPS への要求から応答までの時間を測定するとともに、サービス間の処理時間の内訳も測定する。

4.2.4 実験の結果

測定結果を表3に示す。この実験環境では、「CSG-IS」で4回の問合せが発生している。「RS-SC」でも MPI ジョブ用の 3 台のクラスタ計算機と impi-server 用の1台の計算機の合計4つの SC に対し予約要求が発生している。

JM-EPS	EPS 内訳		CSG/RS 内訳	
時間	処理名	時間	処理名	時間
46 秒	EPS 固有	11 秒	——	——
	EPS-CSG	21 秒	CSG 固有	9 秒
			CSG-IS	12 秒 (3 秒 ×4)
	EPS-RS	14 秒	RS 固有	2 秒
			RS-SC	12 秒 (3 秒 ×4)

表3 資源探索・割当の性能測定結果

ここで「EPS 固有」は、JM-EPS 間の通信オーバヘッド（通信遅延、HTTP/SOAP/XML メッセージ処理等）を含む。「CSG 固有」は、EPS-CSG の通信オーバヘッド、検索式生成、検索結果の処理時間を含む。「RS 固有」は、複数の資源間の予約時刻の同期処理の時間を含む。

サービス間のやりとりの回数は、合計 11 回(表3より 1+1+1+4+4 回）ある。使用した Globus における HTTP や SOAP や WS-RF のコストは1回の要求/応答当り約1秒かかることがわかっており、全体 46 秒のうち 11 秒 (23%) が WS-RF のオーバヘッドになっている。さらに複雑かつ多数台の資源要求では CSG から IS への資源の問合せ回数や RS から SC への予約要求回数が増大し、そうしたオーバヘッドが応答時間に積み上がる可能性が高い。このことから OGSA-EMS 実装では、WSRF のオーバヘッドや IS 及び SC 等の要求/応答のような基本性能の向上が重要であると言える。

しかしながら、こうした基本部品の性能向上は、大きな障害ではなく、今後の改善が期待できるため、OGSA-EMS アーキテクチャは、今後の高度アプリケーション向けの EMS の拡張に向けても、妥当な設計であると言える。

5. おわりに

本稿では、OGSA-EMS アーキテクチャに沿った NAREGI スーパースケジューラの設計及び実装をとおして、OGSA-EMS アーキテクチャの実現可能性を確認するとともに、OGSA-EMS のサービスの上での連成ジョブのような高度なグリッド・アプリケーションの実行のための拡張方式を提案した。

今後は、今回実装しなかったジョブ文書 (job document) のレジストリ・サービスの標準実装を進めるとともに、サービスの基本性能を向上させてゆく。

謝辞

本研究の一部は、文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施している「超高速コンピュータ網形成プロジェクト (NAREGI; National Research Grid Initiative) によるものである。

また、UNICORE AJO Rewriter Interface の設計・活用にあたっては、富士通欧州研究所の Sven van den Berghe 氏と David Snelling 氏には有益なご議論を頂いた。ここに記して謝意を表す。

参考文献

[1] I. Foster, H. Kishimoto, and et al., The Open Grid Services Architecture, Version 1.0, Jan. 2005. http://www.ggf.org/documents/GFD.30.pdf

[2] OASIS WSRF Technical Committee, Web Services Resource 1.2 (WS-Resource), Mar. 2005. http://www.oasis-open.org/committees/wsrf/

[3] A. Savva, and at el., Job Submission Description Language (JSDL) Specification Version 0.9.5-02, Apr. 2005. http://forge.gridforum.org/projects/jsdl-wg/

[4] IMPI Steering Committee, IMPI - Interoperable Message-Passing Interface, tech. rep., NIST, http://impi.nist.gov/, Jan. 2000.

[5] 真木淳, 何希倫, and 青柳睦, RISM-FMO 連成計算, NAREGI ナノサイエンス実証研究第2回公開シンポジウム, Feb. 2004.