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

2005－HPC－102（8）
2005／6／2

GridRPCシステムNinf-Gにおける UNICOREおよびGT4によるジョブ起動

中田　秀基 †,†† 田中 良夫 †　関口　智嗣 †

GridRPC システム Ninf-G における UNICORE および Globus Toolkit 4 を用いたジョブ起動機構に関して報告する。Ninf-G は Globus Toolkit 2 および 3 を用いたジョブ起動機構を、クライアントライブラリに内蔵している。しかし UNICORE や Globus Toolkit 4 では C 言語でのクライアント API が提供されていないため、ジョブ起動機構をクライアントライブラリに内蔵することはできない。我々は、ジョブ起動機構を外部モジュールとしてクライアントライブラリから切り離すための汎用プロトコルを設計した。このプロトコルを使用することで、Ninf-G クライアントライブラリ本体に手を加えることなく、新たなジョブ起動機構を追加することができる。

Job Invocation via UNICORE and GT4 in the Ninf-G GridRPC system

Hidemoto Nakada ,†,†† Yoshio Tanaka †
and Satoshi Sekiguchi †

In this paper, we show an implementaion technique for remote job invocation in Ninf-G, for UNICORE and Globus Toolkit 4. Ninf-G have job invocation module embedded in the client library, for Globus Toolkit 2 and 3. Since there is no C-language client API for job submission in UNICORE and Globus Toolkit 4, it is impossible to implement the job invocation module embedded in the library. To cope with this, we designed the invocation module as a separate process and harnes with the client library via a generic protocol. By this way, we can add new job invocation module without changing the Ninf-G client library.

1. はじめに

Ninf-G[1),2)] は GridRPC[3)] と呼ばれる RPC(Remote Procedure Call) をグリッド上で実現するシステムである。現状の Ninf-G は、Globus Toolkit[4)] 2 および 3 の GRAM(Grid Resource Allocation and Management) に対応したリモートジョブ起動機構を、C 言語で記述されたクライアント側のライブラリに内蔵している。ジョブ起動機構をライブラリに内蔵する方法は性能的に有利であるが、拡張性に乏しく、さらに対象システムが C 言語の API をサポートしていない場合には、Ninf-G からそのシステムをジョブ起動機構としてサポートすることができない。

たとえば UNICORE[5),6)] は、ジョブ起動機構の根幹に Java 言語に依存した部分を持つため、原理的に C 言語からの API を持つことが難しい。また、Globus Toolkit 4[7)] では、C 言語で記述されたクライアントツールを提供しているにも関わらず、明示的に定義された API レイヤを持たない。このためこれらのシステムを従来どおりの内蔵する方式で Ninf-G に組み込むことは難しい。さらに、今後さまざまなグリッドミドルウェアが出現することが予想され、それぞれに対して個別にクライアントライブラリを変更して対処することは非効率である。

このような観点から、我々はリモートジョブ起動機構をクライアントライブラリに内蔵せず、独立した外部プロセスとする方式を設計、実装した。リモートジョブ起動機構とクライアントライブラリの通信プロトコルは、単純なテキストベースのプロトコルとなっており、外部ジョブ起動モジュールを実装する言語を制約しないよう配慮されている。

また、個々の外部ジョブ起動機構に依存した情報を Ninf-G クライアントライブラリが解釈する必要がないように、設定ファイルに記述した内容を直接外部ジョブ起動モジュールに渡すこととした。これにより、任意のジョブ起動機構を Ninf-G のクライアントライブラリを変更する必要なく、追加することができる。

本稿の構成は次のとおりである。2 節で Ninf-G について、3 節で, 本稿の対象システムである UNICORE と Globus Toolkit 4 について概要を述べる。4 節で汎

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

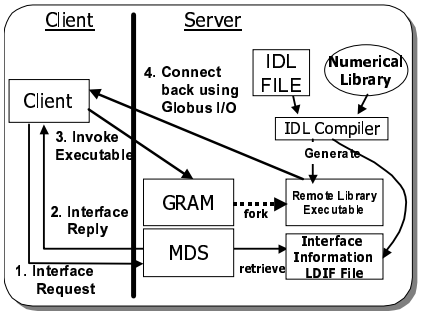

図 1 Ninf-G の概要

```
grpc_function_handle_t handle;
grpc_error_t res;
    ...
res = grpc_function_handle_init(&handle,
   "server.example.org", "test/func");
res = grpc_call(&handle, 100, A, B);
```

図 2 クライアントプログラムの例

用外部起動プロトコルの設計について述べ、5 節で、クライアントライブラリの変更点について述べる。6 節で関連研究について述べ、7 節でまとめと今後の課題を述べる。

2. Ninf-G の概要

Ninf-G[1),2)] は RPC(Remote Procedure Call) 機構をグリッド上で実現する GridRPC システムである。

図 1 に Ninf-G の動作概要を示す。Ninf-G は大きく分けてクライアントとリモート計算モジュールの二つのプログラムから構成される。クライアントは、サーバ上のリモート計算モジュールに計算を依頼し、結果を受け取る。ひとつのクライアントから、複数のリモート計算モジュールを同時に利用することで、並列実行を容易に実現することができる。

2.1 クライアントプログラム

Ninf-G のクライアントプログラムの例を図 2 に示す。ハンドルと呼ばれる構造を、サーバと実行プログラム ID を指定して作成し、それに対して`grpc_call`で引数を指定して計算を依頼する。

ユーザが引数のマーシャリングを明示的に行う必要がないのが、GridRPC の特徴である。

Ninf-G では Globus Toolkit[4)] 2 の、情報サービス (MDS)、ジョブ起動機構 (GRAM)、安全な通信機構 (Globus-IO) の 3 つを利用する。クライアントは、情報サービスを使用してリモートマシンの情報を取得し、ジョブ起動機構を使用してリモート計算モジュールをリモートマシン上で実行、安全な通信機構 Globus-IO 使用してリモート計算モジュールと通信する。

上記の 3 つの Globus モジュールのうち、ジョブ起動機構と通信機構は必須であるが、情報サービスは省略することができる。また、通信機構はリモート計算モジュールにビルド時にリンクしておくことが可能である。したがって、ジョブ起動機構を他の機構に置き換えることができれば、Globus Toolkit のサポートされていないサーバ環境でも、Ninf-G プログラムの実行が可能となる。

```
<SERVER>
hostname server.example.org
heartbeat 120
tcp_nodelay true
</SERVER>
```

図 3 設定ファイルの例

現状の Ninf-G は Globus Toolkit 2 および 3 の GRAM によるモートジョブ起動機構を、クライアント側のライブラリに内蔵する形でサポートしている。

2.2 クライアント設定ファイル

Ninf-G クライアントの動作を、クライアント側設定ファイルで制御することができる。図 3 に設定ファイルのサンプルを示す。このファイルでは、サーバ`server.example.org`への接続に対して、ハートビートの間隔を 120 秒に、TCP を NODELAY に設定している。

3. UNICORE と Globus Toolkit 4

3.1 UNICORE

UNICORE は、欧州富士通研究所が中心となって開発したグリッドミドルウェアである。UNICORE は、スーパーコンピューティングセンターのスーパーコンピュータを、遠隔地からシームレスに利用することを目的に設計されており、この使用法に対する要請から、ファイアウォールに対する配慮が十分になされている。

UNICORE の概要を図 4 に示す。UNICORE では、ファイアウォールに囲まれたサイトを Usite(UNICORE Site) と呼ぶ単位で抽象化している。各 Usite の中には、実際に計算を行う Vsite(Virtual Site) と呼ばれる単位が 1 つ以上存在する。Usite がスーパーコンピュータセンタに、Vsite がスーパーコンピュータや、計算機クラスタなどの計算システムに相当する。

UNICORE ではユーザのサブミットするジョブが、単一のタスクではなく、複数の依存関係を持つタスクからなるワークフローとなっている。このワークフローは Java のオブジェクトとして記述されており、このオブジェクトをシリアライズしたものをモジュール間でやり取りすることで、ジョブのサブミッションを行う。このワークフローを表現したオブジェクトを AJO(Abstract Job Object) と呼ぶ。

各 Usite には、Gateway と呼ばれるデーモンプロセスが 1 つ存在する。この Gateway が外部からの通信すべてを中継する。このため、Gateway だけを Fire-Wall 上に露出しておくだけで、Usite に含まれるすべ

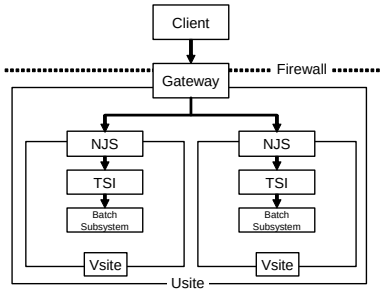

図 4 UNICORE の概要

ての Vsite の計算機資源を外部から使用可能になる。

Usite には 1 つ以上の Vsite があり、各 Vsite には NJS(Network Job Supervisor) と呼ばれるデーモンが存在する。NJS は実際にタスクを実行するモジュールで、ワークフローを解釈するワークフローエンジンであると同時に、後述の TSI と連動して、各タスクを処理するエンジンとしての役割も果たす。

TSI(Target System Interface) は、バッチサブシステムと NJS の間を取り持つモジュールである。

上述のように、UNICORE ではジョブの表現が Java オブジェクトそのものであり、通信は Java 固有のシリアライゼーションフォーマットで行われる。このプロトコルを C 言語でサポートすることは原理的には不可能ではないが、非常に困難である。

3.2 Globus Toolkit 4

Globus Toolkit 4(以下 GT4) は、グリッド上でのデファクトスタンダードとなっている低位ミドルウェアシリーズ、Globus Toolkit の 4 世代目である。Globus Toolkit は 2，3, 4 と各世代で実装テクノロジがまったく異なるため、お互いに互換性はない。GT2 は独自プロトコルを使用し、GT3 は OGSI(Open Grid Service Infrastructure) を使用していたのに対して、GT4 は WSRF[8](Web Services Resource Framework) と呼ばれる、一種の Web サービス技術をベースに実装されている。

GT4 のジョブ起動クライアントは C 言語と Java 言語で用意されている。Java 言語に関してはクライアント API が規定されているのに対して、C 言語では API が定義されていない。Web サービスとしてのインターフェイスを記述した WSDL 記述から、ツールによって自動生成されたインターフェイスは存在するが、抽象度が低く使用しにくい上、将来のバージョンで同じ インターフェイスが提供される保障がないため、これを使用することは好ましくない。

4. ジョブ起動モジュールインターフェイスの設計

4.1 外部ジョブ起動機構の概要

2 節で述べたとおり、Ninf-G のクライアントライブラリには Globus Toolkit 2 および 3 のジョブ起動機構が内蔵されている (図 5) 。

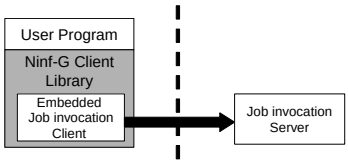

図 5 内蔵ジョブ起動機構

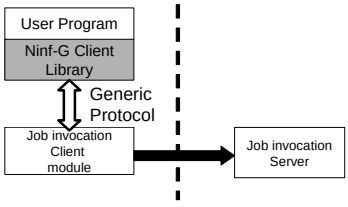

図 6 独立したジョブ起動モジュール

Globus Toolkit 2,3 以外のジョブ起動機構に対応したサイト上でリモート計算モジュールを実行したい場合は、Ninf-G クライアントライブラリの使用するジョブ起動機構を変更しなければならない。しかしジョブ起動機構をクライアントライブラリ内に内蔵するためには、C 言語で記述せざるを得ず、C 言語での API が提供されていない UNICORE や GT4 をサポートすることは難しい。

この問題に対処するために、我々は、ジョブ起動モジュールをクライアントライブラリから独立したプロセスとし、モジュールとライブラリをプロセス間通信で接続する手法を考案した (図 6) 。ジョブ起動モジュールを独立したプロセスとすることで、ジョブ起動部分を実装する言語に対する制約がなくなる。また、新しいジョブ起動機構に対応するために、複雑なクライアントライブラリの内部を変更する必要がなくなり、メンテナンスが容易になることが期待できる。

4.2 外部ジョブ起動プロトコルの要件

外部ジョブ起動モジュールには以下の機能が要請される。外部ジョブ起動モジュールとクライアントライブラリを結ぶ通信路のプロトコルは、これらの機能を実現するものでなければならない。

- ジョブの起動
- ジョブのキャンセル
- ジョブの情報取得

さらに、プロトコルは以下の要件を満たす必要がある。

- 言語独立性
  外部ジョブ起動モジュールを実装する言語は規定しない。このためには、プロトコルが特定の言語、アーキテクチャに依存しないことが重要である。
- 簡潔性
  外部ジョブ起動モジュールの作成を容易にするた

めに、プロトコルは簡潔で明確に定義されている必要がある。

- マルチセッション・非同期
  クライアントプログラムが複数のリモートサーバを利用する場合、単一の外部ジョブ起動モジュールから複数のリモートサーバを利用できる必要がある。したがって、外部ジョブ起動モジュールは同時に複数のセッションを処理できるものでなければならない。
  また、クライアントプログラムはマルチスレッドで稼動する場合がある。複数のスレッドからのアクセスを効率的に処理できるプロトコルである必要がある。

4.3 外部ジョブ起動プロトコルの設計

前項で述べた要件に基づき、外部ジョブ起動プロトコルの設計を行った。

4.3.1 通信路とプロトコルの概要

外部ジョブ起動モジュールは、クライアントライブラリから fork、exec される。クライアントライブラリとモジュール間の通信路としては、標準入出力、および標準エラーを利用する。これは、モジュールを記述する際に通信路をモジュール側で確立する手間を省くためである。

通信には大別して、1) クライアントライブラリからのリクエスト、2) それに対するモジュールからのリプライ、3) モジュールからの非同期な通知、の 3 種類がある。本プロトコルではこれらを、外部ジョブ起動モジュールの標準入力、出力、エラーに割り当てる。

リクエストとリプライはアトミックとする。すなわち、クライアント側はリクエストを送信した後、対となるリプライを読み出すまで次のリクエストを送信しないことを、保障することとする。これによって、クライアントライブラリ内のジョブ起動モジュールと通信する部分の記述が容易になる。

クライアント内に複数のスレッドが存在する場合、リクエストとリプライをアトミックにするためには、クライアント側で通信路にロックをかけることになる。したがって、外部ジョブ起動モジュール側では、リクエストに対するリプライを速やかに送信する必要がある。

プロトコルのデータ形式としては ASCII のテキストを用いる。これは、言語、アーキテクチャ非依存にすることが容易だからである。また、モジュール開発時のテストも容易になることが期待できる。

4.3.2 プロトコルコマンドの概要

JOB_CREATE、JOB_STATUS、JOB_DESTROY、EXIT の 4 つのプロトコルコマンドを用意した。以下それぞれ詳述する。

4.3.2.1 JOB_CREATE

JOB_CREATE コマンドは、ジョブを作成、起動する際に用いるコマンドである。このコマンドだけは複数行にわたるコマンドとなっている。2 行目以降にジョブ作成に必要な情報をキーワードと値をスペースで区切って記述する。最後に JOB_CREATE_END で終端する。ジョブ作成に必要な情報は、ジョブ起動モジュールに依存するので、プロトコルレベルでは規定しない。

```
JOB_CREATE <Request ID> <改行>
<key> <value>  <改行>
<key> <value> <改行>
...
JOB_CREATE_END  <改行>
```

Request ID はクライアントが適当に定めるユニークな文字列で、後述する CREATE 通知との対応を付けるために用いられる。このコマンドに対するリプライの書式を下に示す。上段が成功時、下段が失敗時である。

```
S <改行>
F <Error String> <改行>
```

前述したように、リクエストとリプライはアトミックでなければならないため、起動モジュールはリクエストに対するリプライを即時に返却しなければならない。

JOB_CREATE コマンドの実際の実行には長時間がかかるため、コマンドのパーズに成功した時点でリプライを返答し、実際のジョブ起動の成功、失敗は後述する CREATE 通知によってクライアント側に返却する。

4.3.2.2 JOB_STATUS

JOB_STATUS コマンドは、ジョブのステイタスを取得する際に用いる。書式は下記の通りである。引数となっている Job ID は、CREATE 通知によって返却されたものである。

```
JOB_STATUS <Job ID> <改行>
```

このコマンドに対するリプライの書式を下に示す。成功時が上段、失敗時が下段である。ここで <Status> は、PENDING、ACTIVE、DONE、FAILED の 4 つのうちのひとつである。この状態集合は Globus Toolkit 2 のジョブの状態を踏襲したものであるが、十分に一般性を持つと考えられる。

```
S <Status> <改行>
F <Error String> <改行>
```

ここで注意しなければならないのは、リクエストとリプライがアトミックであることである。ステイタスの取得に時間がかかる場合、リクエストを受けた時点でステイタスの取得を開始すると、結果として長時間通信路をロックしてしまうことになり、他のスレッドからのアクセスができなくなってしまう。このような場合には、定期的にステイタスを取得しておき、リクエストを受けた際にはその値を返却するようにする必要

がある。

4.3.2.3 JOB_DESTROY

JOB_DESTROY コマンドはジョブを破棄する際に用いる。ジョブ起動モジュールは、リクエストを受け取ると、まずリプライを返信する。その後、そのジョブが実行中の場合は、そのジョブを停止し、停止を確認した上で、ジョブ起動モジュール内の関連するデータ構造を破棄する。ジョブが停止したことは、次項でのべる STATUS_NOTIFY でクライアントに通知される。

このコマンドの書式を下記に示す。リプライの書式は JOB_CREATE のリプライと同じである。

```
JOB_DESTROY <Job ID> <改行>
```

4.3.2.4 EXIT

EXIT コマンドは、外部ジョブ起動モジュールに対して終了を指令するコマンドである。モジュールは、まずリプライを返し、起動中のジョブをすべてシャットダウンしてから、終了する。

EXIT の書式を下に示す。リプライの書式は JOB_CREATE の書式と同じである。

```
EXIT <改行>
```

4.3.3 通知の概要

通知は 2 種類ある。ひとつは、上記 JOB_CREATE コマンドでリクエストされたジョブの生成の結果を通知する CREATE_NOTIFY、もうひとつは、ジョブのステイタスが変化したことをクライアントライブラリに通知するための STATUS_NOTIFY である。

4.3.3.1 CREATE_NOTIFY

CREATE_NOTIFY の書式は下記の通りである。上が成功時、下が失敗時である。成功時には Request ID と JOB ID の対応を与え、失敗時には失敗理由をあらわす文字列を返す。

```
CREATE_NOTIFY <Request ID> S <Job ID> <改行>
CREATE_NOTIFY <Request ID> F <Error String> <改行>
```

4.3.3.2 STATUS_NOTIFY

STATUS_NOTIFY の書式は下記の通りである。ステイタスは JOB_STATUS の戻り値で使用されるものと同じである。

```
STATUS_NOTIFY <Job ID> <Status> <String><改行>
```

5. Ninf-G クライアント設定ファイルの拡張

ジョブ起動モジュールに関する設定項目を記述するために、クライアント設定ファイルを拡張した。

設定ファイルには、特定のジョブ起動モジュールを利用してジョブを起動する際に用いる情報を記述する必要がある。これらの情報は、各ジョブ起動モジュールに依存するため、設定ファイルのシンタックスが個々のジョブ起動モジュールに依存しないよう注意して設計する必要がある。

```
<SERVER>
hostname server.example.org
invoke_server UNICORE
invoke_server_option vsite VsiteName
invoke_server_option usite UsiteName
invoke_server_option keystore keyfile
invoke_server_option passfile passfile
</SERVER>
```

図 7 UNICORE 使用時の設定ファイル例

このために、下記の 2 項のクライアント設定ファイルの属性として追加した。

- invoke_server [システム名]
  ジョブ起動モジュールのシステム名を指定する。ジョブ起動モジュールのバイナリ名は、$(NG_DIR)/bin/ng_invoke_server.[システム名] となる。ここで$(NG_DIR) は Ninf-G のインストールされたディレクトリを指す。
- invoke_server_option [任意の文字列]
  ジョブ起動時に、JOB_CREATE コマンドでジョブ起動モジュールに引き渡される情報を指定する。この属性値にはスペースを含むことができ、そのまま JOB_CREATE コマンドに引き渡される。

図7 に UNICORE をジョブ起動機構として使用する際の設定ファイルの一部を示す。ジョブ起動モジュールのシステム名として UNICORE を指定し、UNICORE 固有の起動情報である Vsite 名、Usite 名、認証用鍵の入ったキーストアファイル名、そのパスフレーズを収めたファイル名を指定している。

6. 関連研究

ジョブキューイングシステム Condor[9] において、他の外部起動モジュールに対する汎用インターフェイスを設計する試みがある[10]。Condor では、GAHP と呼ばれるテキストベースのプロトコルを用い、ClassAd と呼ばれる非常に汎用性の高いデータ形式をやり取りすることによって外部起動モジュールと通信する。我々のプロトコルは、このプロトコルを参考にして設計されている。Condor で用いられている GAHP プロトコルと我々のプロトコルの主な相違は以下の通りである。

- プロトコルの同期性
  GAHP プロトコルは、完全に非同期なプロトコルであり、リクエストとリプライにタグ付けすることによって対応をとっている。これに対して、我々のプロトコルはリクエストとリプライが同期的に行われる。この相違は呼び出し側プログラム

の構造に由来する。
Condor で外部起動モジュールと通信する Schedd は、完全なイベント駆動型で記述されており、リクエストを発行するコードと、リプライを解釈するコードは分離して書かざるを得ない。したがってプロトコルも非同期となるのが自然である。
これに対して、Ninf-G のクライアントライブラリは Globus の提供するスレッドモデルで記述されている。この場合はリクエストの発行とリプライの解釈を連続して行うことのできる同期的なプロトコルを用いることで、コードの保守性を向上させることが期待できる。

- ClassAd の使用
Condor の外部起動モジュールでは ClassAd を通信の要素として利用することで高い汎用性を得ている。
これに対して我々のプロトコルでは、単純なキーと値のペアを用いている。この理由は、以下の 2 つである。1)ClassAd の汎用性は過剰で、この目的にはキーと値のペアで十分である。2)Condor チームから提供されている ClassAd ライブラリは、Ninf-G の実装に用いられている C 言語がサポートされていないため、実装が困難である。

7. おわりに

本稿では、GridRPC システム Ninf-G に容易に外部ジョブ起動機構を追加するための、クライアントライブラリの設計および インターフェイスプロトコルについて述べた。
今後の課題としては以下が挙げられる。

- ジョブ起動速度の評価
ジョブ起動機構を外部に持つ構造は、潜在的には性能面で不利である。実アプリケーションを用いてジョブ起動にかかる時間を測定し、性能低下が実用上問題ない範囲におさまっていることを確認する必要がある。
- 複数の言語でのジョブ起動モジュールの実装
外部ジョブ起動機構プロトコルは、特定の言語に依存することなく、外部ジョブ起動機構を容易に記述できるよう考慮されている。この設計の有効性を確認するために、外部ジョブ起動機構を、スクリプト言語を含むいくつかの言語で実装する予定である。
- 情報サービスの外部モジュール化
今回は情報サービスに関しては使用しないことを前提に、ジョブ起動モジュールのみを外部プログラム化した。
情報サービス機構も外部プログラム化して拡張性を確保することは今後の課題のひとつである。

謝　辞

プロトコルの策定に当たってご意見いただいた、産総研 Ninf 開発チームの皆様に感謝します。
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。

参考文献

1) Nakada, H., Tanaka, Y., Matsuoka, S. and Sekiguchi, S.: *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, chapter Ninf-G: a GridRPC system on the Globus toolkit, pp. 625–638 (2003).
2) Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T. and Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *Journal of Grid Computing*, Vol. 1, No. 1, pp. 41–51 (2003).
3) Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C. and Casanova, H.: GridRPC: A Remote Procedure Call API for Grid Computing, submitted to Grid2002.
4) : Globus. http://www.globus.org.
5) : UNICORE. http://www.unicore.org/.
6) Romberg, M.: The UNICORE Architecture - Seamless Access to Distributed Resources, *Proc. of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pp. 287–293 (1999).
7) 中田秀基: グリッドコンピューティングの現在 - Globus Toolkit 4 & WSRF 詳報, *JavaPRESS*, Vol. 41 (2005).
8) : WSRF. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
9) Thain, D., Tannenbaum, T. and Livny, M.: Condor and the Grid, *Grid Computing: Making the Global Infrastructure a Reality* (Berman, F., Fox, G. and Hey, T.(eds.)), John Wiley & Sons Inc. (2002).
10) 中田秀基, Frey, J., 山田基弘, 伊藤泰善, 中野恭成, 松岡聡: Condor の汎用グリッドインターフェイスの設計と UNICORE への適用, 情報処理学会 HPC 研究会 2003-HPC-97, pp. 37–42 (2004).