オーバーレイスケジューラ Jojo3 のグリッド RPC への適用

中 田 秀 基† 田 中 良 夫† 関 口 智 嗣†

複数サイトにまたがる大量の計算資源を用いた計算が，一般ユーザにとっても現実のものとなりつつある．我々は，このような環境で稼働するアプリケーションレベルスケジューラを容易に記述できる環境として，オーバーレイスケジューラ Jojo3 を提案している．Jojo3 はさまざまな環境上にオーバレイするレイヤとして機能し，プロセスの起動，およびプロセス間の通信機能を提供する．我々は，Jojo3 の有用性を検証するため，GridRPC 実装の一つである Ninf-G5 への適用を行った．Ninf-G5 はリモート実行モジュールの起動モジュールと，クライアントとリモート実行モジュール間の通信プロキシモジュールを独立したプロセスとしてコアモジュールの外部に持つ設計となっている．既存のモジュールをベースに Jojo3 を用いるように修正した結果，数十行から数百行程度の比較的軽微な修正によって，Jojo3 への適用が可能であることが確認できた．これによって，Jojo3 の提供する API の機能と記述力が十分であることが確認できた．本稿では，この過程で実装した，Python 言語による Jojo3 クライアントライブラリを合わせて紹介する．

A GridRPC implementation with Overlay Scheduler Jojo3

Hidemoto Nakada ,† Yoshio Tanaka † and Satoshi Sekiguchi †

We proposed Overlay scheduler Jojo3, which overlays various grid and cluster middle-wares and provide uniform, easy-to-use simple API to invoke jobs and to communicate with the jobs, for the application programmers. To prove effectiveness of Jojo3, we adapt Ninf-G5, which is one of the implementations of GridRPC, so that it uses Jojo3 for job invocation and communication. Ninf-G is designed to be modular: job invocation and process communication are taken care by dedicated, independent modules. We modified existing modules so that they use Jojo3, counted the modified lines, and confirmed that the modification was small and easy. Thus, we confirmed that Jojo3 was effective enough. We also show a jojo3 client library in Python, which was required to implement one of the modules above.

1. はじめに

グリッド技術とクラスタ技術の発達により，複数サイトにまたがる大量の計算資源を用いた計算が，一般ユーザにとっても現実のものとなりつつある． しかし，多種多様なグリッドミドルウェア，クラスタ上のキューイングシステムが普及している現状では，一般ユーザがこのような環境でアプリケーションを書くことは非常に難しい．Condor MW[1] や GridRPC[2] などのアプリケーションレベルのスケジューラを用いれば，ユーザによるアプリケーションの記述は容易となる [3] が，アプリケーションレベルスケジューラの作成は容易ではない．

我々は，このような環境で稼働するアプリケーションレベルスケジューラの記述を容易にする環境として，オーバーレイスケジューラ Jojo3 を提案している [4]．Jojo3 はさまざまな環境上にオーバレイするレイヤとして機能し，ノード状態のモニタリング，ノード上でのプロセスの起動，およびプロセス間の通信機能を提供する．

本稿では，オーバレイスケジューラの有効性を検証するため，GridRPC 実装の一つである Ninf-G5[5] への適用を行った．Ninf-G5 はリモート実行モジュールの起動モジュールと，クライアントとリモート実行モジュール間の通信プロキシモジュールを独立したプロセスとしてコアモジュールの外部に持つ設計となっている．これらのモジュールを Jojo3 対応にすることにより，比較的軽微な修正によって，Jojo3 への適用が可能であることが確認できた．

さらに，Ninf-G5 への適用の過程で，Python 言語による Jojo3 のクライアントライブラリを設計，実装した．このクライアントライブラリは Java 言語によるクライアントライブラリと同等の機能を持つが，Python 言語の特性に応じて若干の変更がなされている．

以降の構成は以下の通りである．2 で，Jojo3 の概要を説明する．3 で，対象とする GridRPC 処理系である Ninf-G5 を概説する．4 で，Ninf-G5 に対する Jojo3

† 産業技術総合研究所 / National Institute of Advanced Industrial Science and Technology (AIST)

の適用について述べる．5 で，Python による Jojo3 クライアントライブラリについて述べる．6 で，まとめと今後の課題について述べる．

2. Jojo3 の概要

2.1 オーバレイスケジューラ

Jojo3 は，我々の提案するオーバレイスケジューラというコンセプトに基づく実行環境である．現在，グリッド環境，クラスタ環境ではさまざまなミドルウェアが利用されている．これらの間の相互運用性は貧弱であり，アプリケーションレベルスケジューラを記述する際には，個別にミドルウェアに対して対応する必要がある．

この問題を解決するために，我々はオーバレイスケジューラを提案している．オーバレイスケジューラは，グリッドミドルウェア，クラスタミドルウェアからなるベースレベルスケージュラ層と，アプリケーションレベルスケジューラ層の中間に存在する層であり，ベースレベルスケジューラの機能を抽象化し，基本的な機能を，上位レイヤに対して提供する．提供する主要な機能は，計算ノードのモニタリング，ジョブの起動およびモニタリング，ジョブ間の通信である．Jojo3 の主要なターゲットアプリケーションエリアは，マスターワーカ型の計算である．

グリッドミドルウェアのインターフェイスを抽象化する試みとしては，GAT[6] や Open Grid Forum の SAGA(Simple API for Grid Applications)[7] などがあるが，これらはクライアント側でしか機能せず，グリッド上で起動されるモジュールからアクセスできない点が，本質的に異なる．

2.2 構　造

Jojo3 はデーモンとクライアントライブラリから構成される．ユーザは Jojo3 のクライアントライブラリを利用して，アプリケーションレベルスケジューラを記述する．Jojo3 の概要を図 1 に示す．

デーモンモジュールは，既存のグリッドミドルウェア，クラスタミドルウェアを利用して，すべての計算ノード上に展開され，オーバレイスケジューラを構成する (図 1 (1), (2))．アプリケーションレベルスケジューラは，一つのジョブグループ (後述) に属するジョブ群から構成され，クライアントライブラリを通して，デーモンモジュールにアクセスし，オーバレイスケジューラの機能を利用する．

2.3 ジョブとジョブグループ

Jojo3 の想定するアプリケーションは，複数のノード上の複数のジョブから構成される．このジョブの集合をジョブグループと呼ぶ．あるジョブが起動したジョブは，自動的に親ジョブと同じジョブグループに属することとなる．したがって，ジョブグループは，ルートとなるジョブと，そのジョブから階層的に起動された一連のジョブの集合から構成される．ルートジョブはユーザが直接投入し (図 1(3))，その他のジョブは，ルートジョブから Jojo3 デーモンを経由して起動される (図 1(4))．

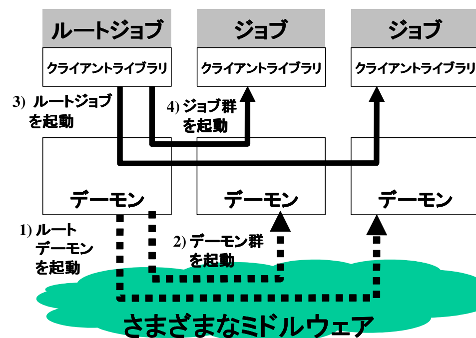

図 1　Jojo3 の概要

ルートとなるジョブは，Jojo3 デーモンに接続する際に，ジョブグループ ID を指定しない．Jojo3 デーモンは，新たなジョブグループを生成し，ジョブに返却する．ルートジョブから Jojo3 デーモン経由で起動されるジョブに対しては，起動時に環境変数として，ジョブグループ ID が渡される．

2.4 ジョブ間通信

ジョブは，同じジョブグループに属する他のジョブに対してネットワーク接続を行うことができる．このネットワーク接続は，デーモンを経由してルーティングされるため，必ずしも高速ではないが，NAT などの非対称なネットワーク環境下でも必ず接続が可能であることが保証される．より高速な通信が必要であれば，Jojo3 の提供する接続を用いて相互に情報を交換し，新たに直接通信を行うことを想定している．

3. Ninf-G5 の概要

本節では，Jojo3 の適用対象とした GridRPC システム Ninf-G5 について概説する．

3.1 GridRPC

GridRPC は，RPC(Remote Procedure Call) の API の一つで，Open Grid Forum で標準化された API 規格である．Ninf-G5 を含めていくつかの実装が存在する．

GridRPC 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．

3.2 Ninf-G5

Ninf-G5 は産総研を中心として開発が続けられてきた GridRPC システム Ninf シリーズの，最新の実装

```
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

である．さまざまな下位レイヤに柔軟に対応するべく，モジュラーデザインとなっている点に特徴がある．

Ninf-G5 によるアプリケーションは，クライアントプログラムとリモート実行モジュールで構成される．Ninf-G5 は下位のレイヤに対して，ジョブ起動サービスと通信サービスを期待する．これらのそれぞれに対して，ジョブ起動サーバ (Invoke Server)，および通信プロキシサーバ (Communicataion Proxy Server) を外部プロセスモジュールとして，コアライブラリから切り離すことによってコアライブラリのスリム化と，さまざまなグリッドミドルウェアに対する柔軟な対応を実現している．

これらのモジュールは，コアとなるライブラリと，簡単で実装言語を選ばないテキストプロトコルで通信するよう設計されている．実際，ジョブ起動サーバは，Java, Python, C など，さまざまな言語で実装されている．

3.3　ジョブ起動サーバ

ジョブ起動サーバ (InvokeServer)[8] は，リモート実行モジュールの起動に用いられる．

ジョブ起動サーバと，クライアントライブラリ間の通信路には，ジョブ起動サーバの標準入出力，エラーが用いられる．このうち標準入出力は同期通信に，標準エラーは非同期の通知に用いられている．

ジョブの起動，停止，ジョブ実行状態の監視などの機能を提供する．

3.4　通信プロキシ

通信プロキシは，クライアント側とリモート実行モジュール側の双方で対となって動作する．

クライアント通信プロキシ (以下 CCP) は，クライアントプログラムから，リモート通信プロキシは (以下 RCP)，リモート計算モジュールから起動される．クライアントと CCP，リモート計算モジュールと RCP はそれぞれ，制御用の通信路で接続される．制御プロトコルは，ジョブ起動サーバのそれに準じており，同様に標準入出力，エラーを用いて通信する構造となっている．

通信プロキシによる通信路確立の様子を，図 3 に示す．

(1)　クライアントが通信ポートをオープン．
(2)　クライアントが，CCP を起動．クライアントの通信ポートを通知．
(3)　CCP は，受信用ポートをオープンし，ポート番号をクライアントプログラムに通知．

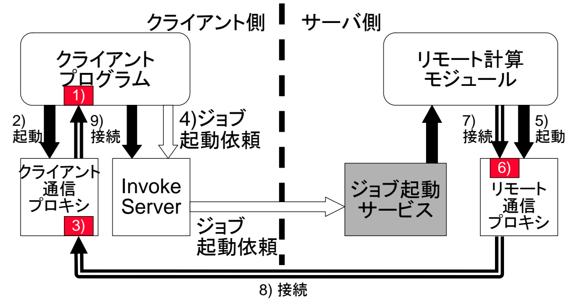

図 3　通信プロキシによる通信路の確立

(4)　クライアントは，ジョブ起動機構経由でリモート計算モジュールを起動．この際に，CCP に対して接続するための情報 (ポート番号)
(5)　リモート計算モジュールは，RCP を起動．この際に CCP のポート番号を通知．
(6)　RCP は受信ポートをオープンし，そのポート番号をリモート計算モジュールに通知．
(7)　リモート計算モジュールが RCP に接続．
(8)　RCP が CCP に接続．
(9)　CCP がクライアントに接続．

4. Ninf-G5 への Jojo3 の適用

4.1　設　　計

Jojo3 は Ninf-G5 が期待するジョブ起動とジョブ間通信の双方の機能を提供する．従って，ジョブ起動サーバと通信プロキシの双方を兼ねる Jojo3 のクライアントを記述すればよい．

ここで問題になるのは，ジョブ起動サーバと CCP の双方が，Ninf-G5 クライアントプログラムから起動されてしまうことである．Jojo3 では，Jojo3 を経由せずに起動されたジョブは，同じジョブグループに属することができない．一方，ジョブ起動サーバが起動するリモート計算モジュールおよび RCP は，自動的にジョブ起動サーバと同じジョブグループに属することになる．従って，CCP と RCP は異なるジョブグループに属することになる．しかし，Jojo3 では同じジョブグループに属するジョブ間でしか通信を許さない．このため，CCP と RCP が直接通信できないことになってしまう．

この問題を解決するために，ジョブ起動サーバが CCP と RCP の間を中継する構造を取ることとした．図 4 に Jojo3 を用いた Ninf-G5 の実装を示す．RCP は CCP にでは無く，ジョブ起動サーバに対して接続し，ジョブ起動サーバが CCP に接続する構造となっている．

4.2　実　　装

4.2.1　ジョブ起動サーバ

ジョブ起動サーバは，Java 言語で実装した．この

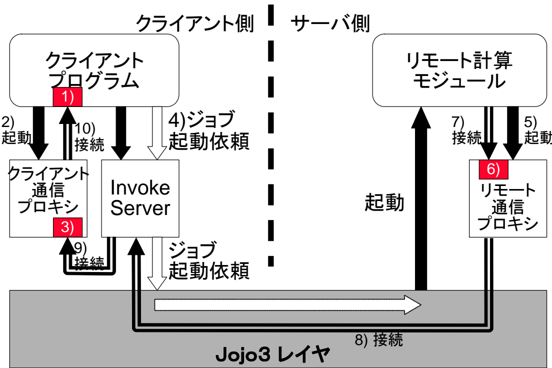

図 4 Jojo3 を用いた Ninf-G5 の実装

際, Ninf-G5 が従来から提供している, Condor[9] や NAREGI ミドルウェア [10] に対するジョブ起動サーバの実装に用いた汎用のライブラリを用いた．Ninf-G5 クライアントとの通信部分は, すべて汎用ライブラリが処理するため, 実装は比較的容易であった．

今回の設計では, ジョブ起動サーバが, 両通信プロキシ間のプロキシの役割を負うこととなる．具体的には, RCP からの接続を Jojo3 経由で accept し, CCP に対して connect して, フォワードしてやる必要がある．

この際, 問題になるのは, CCP が接続を待つポート情報の取得である (図 4 中の (3))．この情報は, 本来, RCP が知っていれば良い情報であるため, ジョブ起動サーバには明示的に知らされることはない．しかし, この情報は, リモート実行モジュールの引数を経由して RCP に引き渡されるため, リモート実行モジュールの起動リクエスト時に, ジョブ起動サーバから取得可能な状態で引き渡される．今回の実装では, リモート実行モジュールの起動リクエストからこの情報を抜き出すことで, ポート情報の取得を実現した．

4.2.2 通信プロキシ

通信プロキシは Python 言語を用いて実装した．この際, サンプルとして Ninf-G5 のパッケージに同梱されている, TCP/IP 通信のみを行うサンプル通信プロキシをベースとして利用した．

通信プロキシはクライアント側の CCP とリモートモジュール側の RCP の二つを実装しなければならない．しかし今回は, CCP の動作は, ベースとなるサンプル実装版と同じであるため, 特に手を加える必要は無かった．

一方, RCP に関しては, 一定の修正が必要であった．具体的には, ソケットで CCP に接続する代わりに, Jojo3 に対してルートジョブ (この場合はジョブ起動サーバ) への接続を依頼するよう変更した．

4.3 Jojo3API の記述力の評価

Jojo3API の機能と記述力を評価するために, ジョブ起動サーバ, 通信プロキシの記述行数を計測した．

表 1 ジョブ起動サーバの記述行数

	固有部	共有部	計
Jojo3	315		1491
Condor	340	1176	1416
NAREGI-MW	1535		2711

表 2 ジョブ起動サーバの記述行数

	固有部	共有部	計
Jojo3 リモート通信プロキシ	74		697
ソケットリモート通信プロキシ	51	623	674
ソケットクライアント通信プロキシ	52		675

4.3.1 ジョブ起動サーバ

ジョブ起動サーバの記述行数を表 1 に示す．汎用のジョブ起動サーバライブラリが 1176 行, Jojo3 固有部が 315 行であった．

比較のため, 同様の枠組みを用いて記述した Condor および NAREGI ミドルウェアに対するジョブ起動サーバの行数も示す． Condor 用ジョブ起動サーバは, Condor Java API[11] を用いて記述されている．Condor 用ジョブ起動サーバと Jojo3 用のジョブ起動サーバの行数は, ほぼ同程度である．これにより Jojo3 クライアント API の記述力は, Condor クライアント API と同程度であることが分かる．

これに対して NAREGI ミドルウェア用ジョブ起動サーバの記述量は非常に大きくなってしまっている．これは, NAREGI ミドルウェアの API が XML 文書ベースとなっているため, XML の生成, パーズをジョブ起動サーバで行わなければならなかったためである．

4.3.2 通信プロキシ

通信プロキシの記述行数を表 2 に示す．比較のため, ベースとなったソケット通信を用いる通信プロキシのサンプル実装の値も示す．これらの通信プロキシはクライアントとの通信のプロトコル処理や基本的な通信処理を行うライブラリ部を共有しているので, 表中では, 分けて記述してある．

ソケットを利用するリモート通信プロキシは, 固有部が 51 行であるのに対し, Jojo3 を用いるリモート通信プロキシは, 74 行となっている．また, Jojo3 版はソケット通信版をベースとしていることから, diff コマンドを用いて, 実際に記述が追加された行数を測定した．この結果 追加・変更された行数はわずか 33 行であることがわかった．これは十分小さい値であると言える．

この結果, Jojo3 の API の記述力が十分高く, 既存のソケットライブラリを用いたプログラムとの親和性も高いことが確認できた．

5. Python API

5.1 設　　計

Jojo3 の Java API はスレッドの使用を前提とした

設計とした．これは Java 言語においてはスレッドが言語レベルでサポートされており, 主要ライブラリの MTsafe 化が進んでいるため, 安心してスレッドを利用できるためである．また, select に相当する機能が限定的にしか利用できないため, スレッドを用いずに実装することは事実上困難である．

一方, Python では, スレッドはライブラリとしてサポートされているものの, 言語仕様レベルでのサポートは無く, ライブラリ群の MTsafe 化も進んでいない．このため, Jojo3 Python API ライブラリはまったくスレッドを使用しないこととした．

インターフェイスの設計としては, Java 版に準じる．ただし, Python では, Java と比較しコールバック関数の利用が容易であるため, リスナクラスではなくコールバック関数を用いるスタイルとしている．

5.2 API

図 5 に, Python による Jojo3 API の骨格を示す．Java API では, さまざまなコールバックメソッドを持つ Listener インターフェイスを, ユーザが実装する形を取っていたが, Python では個別のコールバック関数を必要に応じて提供する形になっている．

また, スレッドを用いていないため, クライアントライブラリに対して明示的に制御を渡す方法が必要となる．このために, `selectLoop()` メソッドが提供されている．

サンプルとして, Python クライアントライブラリを用いたマスタワーカプログラムのコア部を示す．マスタプログラムでは, クライアントの初期化時に, ワーカからの接続が来た際の動作として master クラスの addWorkerSocket を登録し, すべてのノードに対してワーカジョブを起動している．

ワーカプログラムは, クライアントを初期化し, ルートジョブに対して接続を行う．接続ができたらワーカオブジェクトを初期化し, 実行する．

5.3 JSON ライブラリ

Jojo3 ではクライアント, デーモン間のプロトコルとして, JSON を用いている．JSON 化のライブラリとして, JSON-py[12) を用いている．

ただし, JSON-py は比較的低レベルのライブラリであり, JSON 文字列を, Python のディクショナリ, およびリストに変換するだけで, Python オブジェクトへのマッピングは行わない．このため, JSON-py を補う形で, Python オブジェクトと JSON 文字列との間の直接変換を行うライブラリを開発し, これを用いた．

ここで問題になるのが, JSON 文字列から Python オブジェクトへの変換の際に必要となるオブジェクトの型情報である．Python では通常オブジェクトの各フィールドに対して静的に型を定義しないため, 型情報を取得することができない．そこで, フィールド (インスタンス変数) 名と同名のクラス変数を用いて, 型情報を定義する手法を用いた．この方法は django[13) や Google App Engine で, データベースモデルを定義するために用いられている手法である．図 7 に, ジョブ記述クラスを記述した例を示す．2 行目から 4 行目までの, コメントを付した行が型宣言のための行である．

```
class client:
  def __init__(self,
               updateNodes=None,
               acceptHandler=None):
    """インスタンス生成メソッド．
    'updataeNodes' はいずれかのノードの状態が変化
    した際に呼び出されるコールバック関数．
    引数は，nodeInfo オブジェクトのリスト．
    'acceptHander' は，他のジョブからのコネクションが
    来た際に呼び出されるコールバック関数．
    引数は，ソケット．"""

  def getNodeInfo(self):
    """現在使用可能なノードの情報を取得する．
    返り値は nodeInfo オブジェクト．"""

  def getJobState(self):
    """ジョブグループ内のジョブの状態を取得．
    返り値は jobStatus オブジェクトのリスト．"""

  def connect(self, nodeId, jobId,
              connect_callback = None):
    """他のジョブに対して接続を依頼するメソッド．
    'connect_callback' は 接続した際に
    呼び出されるコールバック関数．これを指定しないと
    ブロック呼び出しになり，接続が成功（または失敗)
    するまで呼び出しがブロックする．"""

  def invoke(self, node, jobDesc,
             update_callback = None):
    """ジョブ起動メソッド．
    'update_callback' は，起動したジョブの
    状態が変化した際に呼び出されるコールバック関数．"""


  def selectLoop(self):
    """Jojo3 クライアントライブラリに制御を渡すための
    メソッド．"""
```

図 5 Python クライアントライブラリ API

JSON 文字列から Python オブジェクトへ変換する際には, read メソッドの第 2 引数として型を与える．例えば, ジョブ記述の配列へ変換する場合には次のように書く．

```
import json2
jdList = json2.read(jstring, [jobDescription])
```

6. おわりに

オーバレイスケジューラ Jojo3 の有効性を確認するために, GridRPC Ninf-G5 への適用を行った．Jojo3 を用いるための Ninf-G5 への改変は比較的軽微であった．これによって Jojo3 の有効性が確認できた．

また, Ninf-G5 への適用の過程で開発された, Python による Jojo3 クライアント API も紹介した．Python クライアントの基本的なインターフェイス構造は, Java 版のそれに準ずるが, 言語の相違に応じて若干の修正が行われている．

今後の課題としては以下が挙げられる．

- クライアントライブラリの拡充

```
## マスタプログラム
class master:
  'master クラスの定義.'

  def addWorkerSocket(self, cid, s):
    'worker からの接続を追加'
    ...
ms = master()  # master オブジェクト作成

# クライアント初期化
cl = jojo3.client.client(
     acceptHandler=ms.addWorkerSocket)

# ノード情報取得
for node in cl.getNodeInfo():
  # ジョブ記述を作成
  jd = jobDescription(args = [...])
  # ノード上でジョブを起動
  cl.invoke(node, jd)

#制御をライブラリに渡す.
cl.selectLoop()

--------------------------------
## ワーカプログラム
class worker:
  'ワーカクラスの定義'
  def __init__(self, s):
    '初期化'

  def start(self):
    'worker 実行'

# client を初期化
cl = jojo3.client.client()
# ルートジョブの取得
rootJob = cl.getJobState()[0]
# ルートジョブに接続
(s, conId) = cl.connect(rootJob.nodeId,
                        rootJob.jobGId.jobId)

# ワーカスタート
worker(s).start()
```

図 6 Python によるマスタワーカの一部

```
class jobDescription(object):
    args = [str]           #文字列のリスト
    envs = {str:str}       #文字列のディクショナリ
    workingDirectory = str #文字列
    def __init__(self):
        self.args = []
        self.envs = {}
        self.workingDirectory = None
```

図 7 ジョブ記述クラス

現在 Jojo3 は Java と Python クライアントのみをサポートしている. 多くのアプリケーションが利用している, C 言語および C++言語に対するクライアントの作成を行う.

- アプリケーションへの適用
  コンセプトの有効性を立証するために, 他のアプリケーションへの適用を行う.
- 大規模環境での実証実験
  数百ノード規模の環境での実証実験を行いソフトウェアとしての頑健性, スケーラビリティに関する知見を得る.

参考文献

1) Goux, J.-P., Kulkarni, S., Linderoth, J. and Yorde, M.: An Enabling Framework for Master-Worker Applications on the Computational Grid, *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, Pittsuburgh, Pennsylvania, pp. 43 – 50 (2000).
2) GridRPC-WG: A GridRPC Model and API for End-User Applications. Open Grid Forum, GFD.052.
3) Aida, K., Natsume, W. and Futakata, Y.: Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)* (2003).
4) 中田秀基, 田中良夫, 関口智嗣: オーバレイスケジューラ Jojo3 の提案, 情報処理学会 HPC 研究会 2003-HPC-114, pp. 169–174 (2008).
5) 中田秀基, 田中良夫, 関口智嗣: グリッド RPC システム Ninf-G の可搬性および適応性の改善, 情報処理学会ハイパフォーマンスコンピューティングシステム研究会, Vol. 2007-HPC-112, pp. 37–42 (2007).
6) van Nieuwpoort, R. V., Kielmann, T. and Bal, H. E.: User-Friendly and Reliable Grid Computing Based on Imperfect Middleware, *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07)* (2007). Online at http://www.supercomp.org.
7) SAGA-WG: SAGA core WG home page. http://forge.ogf.org/sf/projects/saga-core-wg.
8) 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 GT4 によるジョブ起動, 情報処理学会ハイパフォーマンスコンピューティングシステム研究会, Vol. 2005-HPC-102, pp. 45–55 (2005).
9) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proc. of HPDC-7* (1998).
10) 中田秀基, 佐藤仁, 佐賀一繁, 畑中正行, 佐伯裕治, 松岡聡: NAREGI ミドルウェア β-gLite 間における相互ジョブ起動実験, 情報処理学会研究報告　2006-HPC-109 (2007).
11) Condor Java API, http://staff.aist.go.jp/hide-nakada/condor_java_api.
12) json-py: http://sourceforge.net/projects/json-py/.
13) django: http://www.djangoproject.com/.