情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 3　54–66 (Dec. 2008)

F-Omega：グリッドアプリケーションの自動サーバ切替えの枠組み

渡邊啓正[†1]　平澤将一[†2]　本多弘樹[†2]

サーバの性能が変動するグリッドを用いて長時間グリッドアプリケーションを実行させるには，サーバの適切な動的切替えが必要である．サーバの運用上の理由や障害などによって変動するサーバの稼働状況に対応したサーバ動的切替えを実現するには，従来，多数のサーバの運用計画を監視する手間，サーバ利用環境の管理のサーバ動的切替えへの対応，サーバ利用状況の監視の手間，代替サーバの選択とその指示の手間，サーバ可変グリッドアプリケーション開発の手間といった問題があった．本稿では，サーバの自動的な切替えのための枠組み **F-Omega** を提案する．F-Omega では，サーバ運用計画の自動監視機能，サーバ動的切替えに対応したサーバ利用環境の管理機能，サーバ利用状況の自動監視機能，代替サーバの自動選択と自動指示機能，サーバ可変グリッドアプリケーション開発の支援機能を新たに実装し，他の既存機能とともに統合した自動サーバ切替えシステムをユーザに提供する．適用実験において，2.8 日間にわたり 12 台のサーバ群の稼働状況が計 29 回変動したが，アプリケーションの利用するサーバが自動的に適切に切り替えられ計算が継続されることを確認した．

F-Omega: A Framework for Automatic Server Change in Grid Applications

Hiromasa Watanabe,[†1] Shoichi Hirasawa[†2] and Hiroki Honda[†2]

Appropriate dynamic server change is needed to make grid applications run for a long time using grid that servers' performances change by time. For changing servers according to scheduled server status change and server failure, conventional method has the following problems. There are laborious works for users, which are monitor of operation schedule of many servers, support to dynamic server change in implementation of management of server use environment, monitor status of server use, select server and send order about changing server to application and development of grid application that can change server. In this paper, we propose a framework **F-Omega** for dynamic server change. F-Omega implements an automatic monitoring function of server operation schedule, server use environment management function that supports dynamic server change, an automatic monitoring function of server use, an automatic server selecting function and an automatic order sending function about changing server and an assistance function to develop grid applications that change servers dynamically. F-Omega provides users with an automatic server changing system that unifies the above-mentioned functions and other existing functions. In application experiment, we made 29 changes of operational status on twelve servers over 2.8 days. And we confirmed that the application automatically changed its servers appropriately while continuing its computation.

1. はじめに

近年，WAN 上の計算資源を用いる分散並列計算環境「グリッド」の実用化に関する研究がさかんである[1),2)]．グリッドを実用的な計算環境とするには，グリッドで動作する分散並列アプリケーション（以下，グリッドアプリケーション）が計算をやり直すことなく数日から数週間計算し続ける必要がある．その一方で，サーバの稼働状況は運用上の理由や障害などにより変動する．このため，グリッドアプリケーションの長時間実行には，利用するサーバの適切な動的切替えが必要である．本稿では，この切替えにかかるユーザの手間を削減する機構について議論する．

サーバの動的切替えが必要となるサーバの稼働状況の変動には，運用計画に基づく計画的変動と，ハードウェア障害などによる突発的変動の 2 つが考えられる．

これらの変動に応じたサーバの動的切替えを実現するには，一般的に，次に列挙する機能が必要となる（2.1 節に詳述）．

(1)　サーバ運用計画の監視
(2)　サーバ障害の検知
(3)　サーバ利用環境の管理のサーバ動的切替えへの対応
(4)　サーバ利用状況の監視
(5)　代替サーバの選択とその指示
(6)　代替サーバへの動的切替え

†1 HPC システムズ株式会社
HPC Systems, Inc.
†2 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

(7)　代替サーバへのアプリケーションの配置
(8)　サーバ可変グリッドアプリケーション開発の支援

従来，上記の (1)，(3)，(4)，(5)，(8) の機能を実現しようとすると，以下の問題点があった．

- サーバ運用計画の監視については，Globus Toolkit [3]，Inca [4]，G-Monitor [5] などの資源監視ミドルウェアや，サーバ運用に関する電子メール・ウェブページを用いてユーザが手作業で監視作業を行わなければならない．
- サーバ利用環境の管理については，この管理処理をサーバの動的切替えにも対応できるようにユーザが実装しなければならない．
- サーバ利用状況の監視については，この監視処理をサーバの動的切替えやサーバの障害検知結果に対応できるようにユーザが実装し，ユーザが手作業で監視作業を行わなければならない．
- 代替サーバの選択とその指示については，ユーザがこれらを手作業で行わなければならない．
- サーバ可変グリッドアプリケーションの開発については，Ninf-G [6] といったミドルウェアで提供されている機能を用いても，サーバ切替え指示に応じてサーバを動的に切り替える処理をユーザが実装しなければならない．

そこで本稿では，上記の機能の実現に際してのユーザの煩雑さを軽減するべく，運用計画に基づく計画的変動とハードウェア障害などによる突発的変動に対応した自動サーバ切替え機構をユーザに提供する枠組み **F-Omega** を提案する（**図 1**）．F-Omega では，サーバ運用計画の自動監視機能，サーバ動的切替えに対応したサーバ利用環境の管理機能，サーバ利用状況の自動監視機能，代替サーバの自動選択と自動指示機能，サーバ可変グリッドアプリケーション開発の支援機能を新たに実装し，他の既存機能とともに統合した自動サーバ切替えシステムを提供する．これによりユーザはサーバ切替えにともなう煩雑な作業から開放され，より簡便にグリッドを利用できる．

F-Omega のミドルウェアによる実装例として，GridRPC [7] アプリケーションを対象として実装を行った．GridRPC はグリッドアプリケーション開発のための現実的なプログラミングモデルの 1 つである [1),8)]．また，MPI と組み合わせることで柔軟性・頑強性・高効率性を備えたグリッドアプリケーションを実現しやすいという特長がある [2)]．

F-Omega の適用実験では，2.8 日間にわたって 12 台のサーバ群の稼動状況を 29 回変動させた．実験中，自動サーバ切替えシステムによって，サンプルアプリケーションが利用するサーバが自動的に適切に切り替えられ，計算が継続されることを確認した．

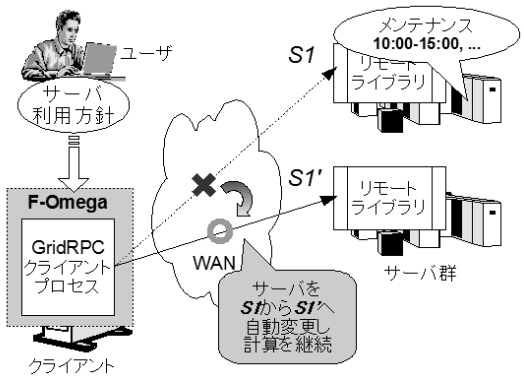


図 1　F-Omega の概観
Fig. 1　Overview of F-Omega.

2. サーバ動的切替えの要件と実現内容

本章では，サーバの動的切替えの一般的な要件を整理し，その中で本システムで実現した機能を明記する．さらに，本システムで実現していない機能についての対処と，その将来的な解決方針を述べる．

2.1　サーバ動的切替えの一般的な要件

サーバの稼働状況の計画的変動と突発的障害に対応してサーバを動的に切り替えるシステムには，一般的に以下の機能が必要である．

(1)　サーバ運用計画の監視
システムは次に関するサーバ運用計画を監視する．
- ハードウェア稼動状況（利用可能なプロセッサ数，メモリ容量など）
- ソフトウェア稼動状況（デーモンプロセスや計算ライブラリなど）

本機能は，利用可能なサーバをシステムが選択するために用いる．

(2)　サーバ障害の検知
システムがサーバ上の障害や異常を検知する．本機能は，システムが障害発生時にサーバの切替えを行うために用いる．

(3)　サーバ利用環境の管理のサーバ動的切替えへの対応

サーバの動的切替えに対応できるように，サーバの利用環境をシステムが動的に管理する．なお，GridRPC では，サーバの利用環境とはリモートライブラリの関数ハンドルや RPC セッションの ID を指す．

(4)　サーバ利用状況の監視
(6) のサーバ動的切替えと (2) の障害検知結果に応じて変動するサーバの利用状況をシステムが監視する．なお，GridRPC では，サーバの利用状況とはアプリケーションによってサーバ上で起動されているリモートライブラリの数を指す．本機能は，サーバの利用状況を考慮してサーバの切替えを行うために用いる．

(5)　代替サーバの選択とその指示
(4) のサーバの利用状況と (1) のサーバ運用計画情報に基づいてシステムが代替サーバを選択し，(6) の機能に対して代替サーバへの切替えを指示する．なお，システムが柔軟にサーバを選択するために，ユーザ定義のサーバ利用方針に基づいてサーバを選択する機能も必要である．

(6)　代替サーバへの動的切替え
代替サーバへの切替え指示に従って，アプリケーション実行中にシステムが切替え元サーバの利用を停止し，代替サーバの利用を開始する．

(7)　代替サーバへのアプリケーションの配置
利用するサーバに対して，必要に応じてシステムがアプリケーションを配置する．

(8)　サーバ可変グリッドアプリケーション開発の支援
サーバ切替え指示に従ってサーバを動的に切替え可能なグリッドアプリケーションを容易に開発可能とする API を，システムがアプリケーション開発者に提供する．

2.2　本システムで実現した機能と未実現の機能

2.1 節の要件のうち，本稿のシステムでは (1)，(3)，(4)，(5)，(8) を実現した（詳細は 4 章を参照）．(2) と (6) については，Ninf-G の機能を用いた．

(7) については本稿では実現しておらず，サーバすべてにアプリケーションをあらかじめ配置することで不要とした．将来的には Relis-G[9] などのアプリケーション自動配置システムを組み込んで実現するのが良いと考える．

3.　F-Omega におけるサーバの動的切替え

F-Omega の自動サーバ切替えシステムを用いたサーバの動的切替えの流れを示す（**図 2**，図中の番号は 3.1 節の列挙番号に対応）．

3.1　サーバ運用計画に基づく切替え

まず，次の手順で，サーバ切替え指示に応じてサーバを動的に切替え可能なグリッドアプリケーションを作成する．

(1)　プログラマが，F-Omega のイベント駆動型プログラミングモデル（4.2 節）に従って GridRPC クライアントプログラムを作成する．具体的には，サーバ切替え指示を解釈するモジュール（Actuator）と GridRPC 資源を自動管理するモジュールから作成されるイベントに対するイベントハンドラを記述する．

続いて，次の手順で，サーバ運用計画情報の監視，およびサーバ切替えの指示が行われる．

(2)　各サーバ管理者が，サーバ運用計画ファイルを作成し，各サイトのウェブサーバ上で公開する．

(3)　ユーザがサーバ利用方針ファイルを作成する．

(4)　ユーザが，(1) で作成された GridRPC クライアントプログラムを起動する．Actuator が自身と通信するための IP アドレスとポート番号を表示する．

(5)　表示された IP アドレスとポート番号，全サーバの運用計画ファイルの URL，サーバ利用方針ファイルのパスをコマンドライン引数として，ユーザがサーバ選択プログラムを起動する．

(6)　サーバ選択プログラムがサーバ運用計画・サーバ利用方針を読み込み，Actuator を介してサーバ利用状況を取得する．適切なサーバ切替え指示を作成し，TCP/IP 通信により指示を Actuator に送信する．

最後に，以下の手順で，サーバ切替え指示に応じてサーバの切替えが行われる．

(7)　受信した指示に応じて，Actuator がリモートライブラリの追加/削除指示イベントをイベントキューに挿入する．

(8)　対応するイベントハンドラが，指示に応じてリモートライブラリの起動/終了を行う F-Omega モジュール API を実行する．

3.2　障害発生時・初期実行時のサーバ動的割当て

グリッドアプリケーション実行中の，ハードウェア故障といった突発的なサーバ障害に対して，F-Omega は基本的な耐障害機能を提供する．これは具体的には次の流れで実現される．

(i)　3.1 節の (1)〜(5) によりアプリケーションが作成・実行されている．

(ii)　サーバ障害発生時，GridRPC ミドルウェアが障害を自動的に検知する．

(iii)　F-Omega の RPC セッション管理モジュールが RPC 失敗イベントを生成してアプ

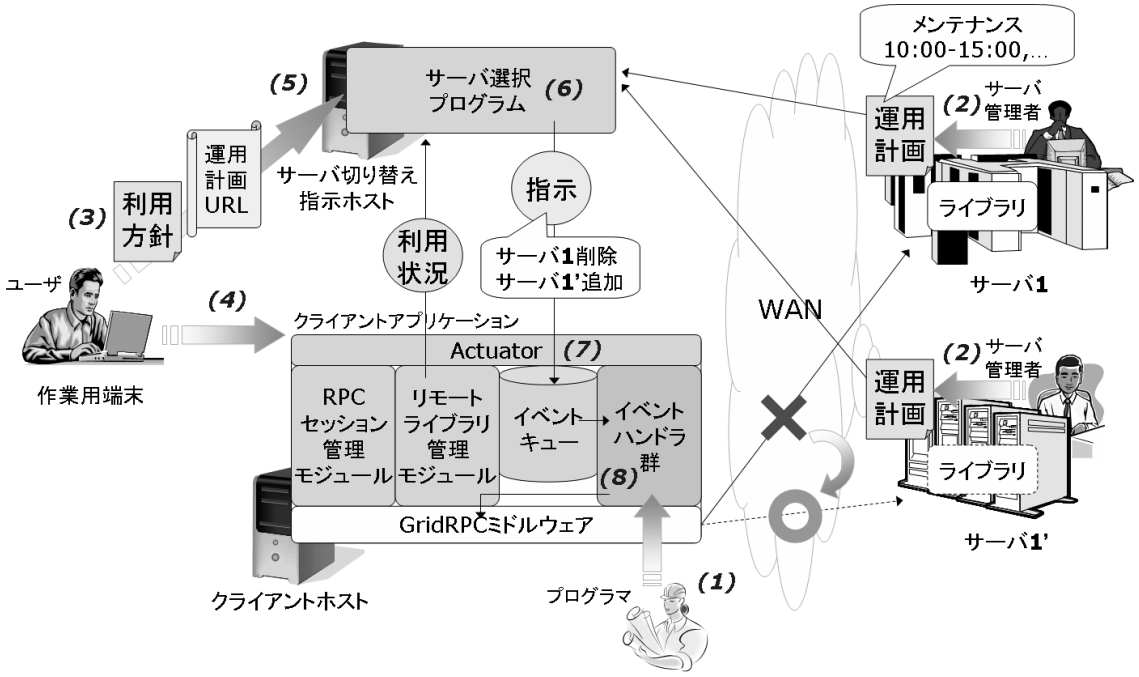


図 2　F-Omega のコンポーネント群の動作
Fig. 2　How components in F-Omega work.

リケーションへ通知する．さらに，リモートライブラリ管理モジュールがリモートライブラリのエラー状態を把握する．

(iv)　定期的に呼び出されるサーバ選択プログラムが，Actuator を介してリモートライブラリ管理モジュールからサーバ割当て不足を検知し，代替サーバを選択し，それを追加する指示を Actuator に与える．

(v)　Actuator がリモートライブラリ追加指示イベントを生成してアプリケーションに通知する．

(vi)　アプリケーションのイベントハンドラが代替サーバ上のリモートライブラリを起動する．

グリッドアプリケーションの初期実行時には (iii)〜(v) が実行され，サーバが自動的に追加される．

4. 設計と実装

F-Omega の自動サーバ切替えシステムの設計・実装について述べる．

本稿では，次の利点を考慮し，自動サーバ切替えシステムを Actuator（後述）とサーバ切替え指示システムに分離した．サーバ切替えシステムに新たな機能が必要になった場合，サーバ切替え指示システムは，F-Omega のイベント駆動型プログラミングモデルに従って書かれた GridRPC クライアントプログラムとは独立して拡張できる．たとえば，プロセッサやネットワークの性能を意識したサーバ選択アルゴリズムを新たに導入する際に有用である．また，複数のグリッドアプリケーションのサーバを切り替える場合，1 つのサーバ切替

え指示システムで十分である．

4.1　サーバ切替え指示システム

F-Omega のサーバ切替え指示システムは次の要素から構成される．

(a)　サーバ選択プログラム．(b)，(c)，(d) の情報を基に，利用するべきサーバを選択し，それに沿うようにサーバ切替え指示を作成する．そして指示を 4.2 節の Actuator に伝達する．本プログラムを定期的に実行することで継続的に自動的にサーバ切替えの指示が行われる．

(b)　サーバ運用計画ファイル．サーバのハードウェア・ソフトウェアの利用可能量を時系列に表す．

(c)　サーバ利用方針ファイル．ユーザが希望する，サーバ上のハードウェア・ソフトウェアの利用量を時系列に表す．

(d)　サーバ利用状況データ．各サーバでリモートライブラリがいくつ起動しているかを表す．4.2 節のリモートライブラリ管理モジュールから Actuator を介して取得する．

(a)〜(c) の詳細および実装について以下に述べる．

サーバ選択プログラム　サーバ運用計画とサーバ利用方針を遵守することを最優先し，それらで定まるサーバのハードウェア/ソフトウェアの利用可能量の上限値に沿うように，サーバ利用状況との差分をサーバ切替え指示として作成する．たとえば，ある時点でサーバ運用計画で提示された利用可能プロセッサ数が 4，サーバ利用方針で要求するプロセッサ数が 3，すでに 2 つのライブラリが起動されていた場合，(min(4,3)-2=1 より) 新たに 1 つライブラリを起動する指示を作成する．このとき，割当て可能なサーバが 2 台以上存在する場合には，1 台をランダムに選ぶ．また，サーバ利用方針ファイルで要求されたサービスがサーバ運用計画によりサーバ上で利用できない場合，利用可能プロセッサ数によらず，そのサーバを利用しないと決める．

サーバ運用計画ファイル　書式には，サーバの性能パラメータを簡潔に表せること，および，自動的に解析しやすいことが必要である．この理由で，書式に XML を用いる．ある期間でのサーバのハードウェア/ソフトウェア（サービスと呼ぶ）の利用可能量を，定量化したパラメータ値で表し，XML の属性代入文で記述する．ファイルの記述例を**図 3** に示す．この例では，利用可能なプロセッサ数を表す job_maxNoOfCPUs サービス，サーバ全体を表す server サービス（値が 1 であるとき利用可能を意味する），サーバ上のソフトウェアを表す globus_gram などの 5 種類のサービスの値が記述されている．また，利用可能量の計画的な変動を記述するべく，値が有効となる期間を startDate と endDate という属性で指定する．

```
<?xml version="1.0"?>
<gatekeeper hostname="grid00.yuba.is.uec.ac.jp">
  <constraints>
    <constraint job_maxNoOfCPUs="1"/>
    <constraint server="1"/>
    <constraint globus_gram="1"/>
    <constraint globus_mds="1"/>
    <constraint local_hdd="1"/>
    <constraint second_hdd="1"/>
    <constraint famous_compiler="1"/>
    <constraint server="0"
                startDate="2007/07/31 23:00:34"
                endDate="2007/08/02 09:05:16"/>
    <constraint job_maxNoOfCPUs="0"
                startDate="2007/08/04 05:17:50"
                endDate="2007/08/06 21:56:52"/>
    <constraint local_hdd="0"
                startDate="2007/08/04 20:06:47"
                endDate="2007/08/07 13:53:44"/>
    <constraint second_hdd="0"
                startDate="2007/08/07 23:15:50"
                endDate="2007/08/10 22:22:22"/>
    <constraint job_maxNoOfCPUs="0"
                startDate="2007/08/08 19:34:35"
                endDate="2007/08/09 00:43:14"/>
    <constraint famous_compiler="0"
                startDate="2007/08/10 10:40:37"
                endDate="2007/08/12 01:19:49"/>
  </constraints>
</gatekeeper>
```

図 3　サーバ運用計画ファイルの例
Fig. 3　An example of server operation schedule file.

記述者の負担を必要最低限に抑えるべく，複数の代入文において期間を重複することを許可する．これにより，記述者は重複期間以外の期間における値を記述しなおす必要がない．ファイル解析では，日時を最も狭く囲む期間に対し指定された値を使う．

サーバ利用方針ファイル　ある期間においてサーバで利用するハードウェア/ソフトウェアの希望利用量を XML で記述する．ファイルの記述例を**図 4** に示す．サーバ上で利用するサービス名が require 属性で記述され，サーバ上で実行するリモートライブラリの最大数が library タグで記述される．

記述量を抑えるべく，hostname 属性では，カンマ区切りで複数指定できるほか，

```
<?xml version="1.0"?>
<policies>
  <policy>
    <term startDate="2007/08/01 00:00:00"
          endDate="2007/08/31 23:59:59"/>
    <gatekeeper hostname="hpbla[1-8].yuba.is.uec.ac.jp"
                require="globus_gram, local_hdd"/>
    <library maxcount="12"/>
  </policy>
  <policy>
    <term startDate="2007/08/04 00:00:00"
          endDate="2007/08/05 23:59:59"/>
    <gatekeeper hostname="hpbla[1-8].yuba.is.uec.ac.jp"
                require="globus_gram, local_hdd"/>
    <library maxcount="20"/>
  </policy>
</policies>
```

図 4　サーバ利用方針ファイルの例
Fig. 4　An example of server use policy file.

“hpbla[1-8]” といった正規表現で複数記述をまとめることを許可する（解析時に展開する）．また，サーバ運用計画ファイルと同様に，利用方針を複数記述する際に期間を重複させることを許可する．

サーバ切替え指示システムのインストールおよび実行に必要なソフトウェアの構成を以下に述べる．必須のソフトウェアは次のとおりである．

- Perl
  サーバ選択プログラムの実装に用いた．また，Actuator との TCP/IP 通信に Perl 標準の IO::Socket モジュールを用いた．
- CPAN [10] の XML::DOM モジュール
  XML の解析に用いた．

以下のソフトウェアについては，同等の機能を持つ別のソフトウェア実装で代用できる．

- ファイル転送ソフトウェア
  サーバ切替え指示ホスト上で動作するサーバ選択プログラムが，サーバ上のサーバ運用計画ファイルを読み込むために必要である．本稿では GNU Wget [11] を用いた．
- コマンド定期実行ソフトウェア
  サーバ選択プログラムの定期的実行に必要である．本稿では cron を用いた．
- ウェブサーバソフトウェア
  サーバ上のサーバ運用計画ファイルの公開に必要である．本稿では Apache HTTPd [12] を用いた．

4.2　サーバ切替え指示に対応可能な GridRPC アプリケーションの開発方法

F-Omega では，サーバ切替え指示に応じてサーバを切り替える GridRPC クライアントプログラムをイベント駆動スタイルで記述する．そのために，F-Omega では次のアプリケーションモジュールをプログラマに提供する．

- Actuator．アプリケーションにサーバの切替えを指示する．
- GridRPC 資源管理モジュール群．リモートライブラリや RPC セッションを動的に管理し，それらの状態変化をアプリケーションに通知する．
- イベントキューモジュール．上記のモジュール群から作成された情報をイベントとして保持する．

プログラマは，これらによって生成されるイベントに対するアプリケーション固有の反応をイベントハンドラとして記述する．たとえば，イベントハンドラは，リモートライブラリの追加を指示するイベントを受け取った際にリモートライブラリを起動する．イベントハンドラはコンパイルされ，前述のモジュール群とリンクされて，完全な GridRPC クライアントプログラムとなる．このようにして，プログラムは Actuator の指示に応じてサーバを切り替える機能を持つ．

本方式のアプリケーションのソースコード例を**図 5** に示す．また，モジュールが提供する主な API 関数を**表 1** に示す（表中 `fomega_initialize` 関数の第 1 引数は Ninf-G のクライアントコンフィギュレーションファイル名を表す）．イベントループで `fomega_event_pop` 関数を用いてイベントキューから取り出されたイベントに応じ，イベントハンドラ（図中で点線の枠）が呼び出される．ここでは，Actuator（後述）がアプリケーションに対しライブラリを追加/削除するように指示したイベントと，RPC の正常終了/失敗を知らせるイベントに対して，イベントハンドラが定義されている．この例に限らず，通例，F-Omega のプログラミングモデルではこの 4 種類のイベントのみに対してイベントハンドラを定義する．各イベントハンドラの処理の詳細を次に記す．

- もしイベントがリモートライブラリの追加を指示していた場合，一番上のイベントハンドラが実行される．そこではイベント変数のメンバ変数（下線の箇所）として提供されるサーバのホスト名を用いて，F-Omega モジュール API のリモートライブラリ起動関数を呼ぶ．本関数は GridRPC ミドルウェア Ninf-G の `grpc_object_handle_init_np` 関数を用いてリモートライブラリを起動し，正常起動時に RPC 正常終了イベント（特殊な関数 CALL_INIT の正常終了を擬似的に表す）をイベントキューに挿入する．

```
int main(int argc, char *argv[]) {
  ...
  fomega_initialize(CONFIG_FILE, argc, argv);
  ...
  while (completed == 0) {
    fomega_event_pop(&event); //イベント取り出し
    switch (event.type) {
    case EVENT_LIBRARY_ADD:
      //Actuatorからライブラリ追加指示
      for (i = 0;i < event.operation.amount;i++)
        fomega_library_init(event.operation.hostname,
                            event.operation.attr);
      break;
    case EVENT_LIBRARY_SUBTRACT:
      //Actuatorからライブラリ削除指示
      for (i = 0;i < event.operation.amount;i++)
        fomega_library_delete_by_hostname(
                          event.operation.hostname);
      break;
    case EVENT_CALL_COMPLETED:
      //RPCが正常終了した場合
      fomega_call_get(&call, event.callid);
      printf("Task[%ld] completed¥n", call.taskid);
      ...
      //新たにRPCを発行
      fomega_invoke_async(event.libraryid,
        "get_number_of_solutions", taskid, n, depth,
        fixpos[taskid], &tmpcount[taskid]);
      ...
      break;
    case EVENT_CALL_FAILED:
      //RPCが失敗した場合
      fomega_call_get(&call, event.callid);
      printf("Task[%ld] failed¥n", call.taskid);
      ...
      break;
    }
    fomega_event_dispatch(event);
  }
  ...
  fomega_finalize();
}
```

図 5　F-Omega における GridRPC アプリケーションのソースコード例
Fig. 5　An example source code of a GridRPC application in F-Omega.

- もしイベントがリモートライブラリの削除を指示していた場合，上から 2 番目のイベントハンドラが実行される．そこでは，サーバホスト名を用いて F-Omega モジュール API のリモートライブラリ終了関数を呼ぶ．本関数は指定されたサーバに対して発行済みの RPC を GridRPC API の `grpc_cancel` 関数を用いて取り消し，RPC 失敗イベントをイベントキューに挿入する．関数ハンドルはイベントハンドラに続いて実行される `fomega_event_dispatch` 関数内部で Ninf-G の `grpc_object_handle_destruct_np` 関数を用いて破壊される．
- もしイベントがリモートライブラリの正常起動か RPC の正常終了を意味する場合，3 番目のイベントハンドラが実行され，新たに RPC を発行する．
- もしイベントが RPC の異常終了を意味する場合，4 番目のイベントハンドラが実行され，異常終了の旨を標準出力に表示する．

以下に，各モジュールの機能・設計を述べる．

Actuator　アプリケーションプロセス内に TCP サーバスレッドを作成し，サーバ選択プログラムと通信を行う．サーバ選択プログラムの要求に応じて，Actuator は実行中のリモートライブラリの情報を返したり，アプリケーションにサーバの切替えを指示したりする．Actuator はサーバ切替えの要求のリストを管理し，切替え日時に達した際，サーバ切替えイベントを作成しイベントキューへ挿入する．

リモートライブラリ管理モジュール　実行中のリモートライブラリ群を，ライブラリ ID（整数）を基に管理する．GridRPC API 関数によってリモートライブラリの状態（利用可能，エラーなど）の変化を検知すると，リモートライブラリ状態変化イベントをイベントキューに挿入する．

RPC セッション管理モジュール　実行中の RPC セッション群を，RPCID（整数）を基に管理する．非同期 RPC が実行される際，暗黙的に RPC の終了を待機するスレッドを作成する．スレッドは定期的に RPC セッションの状態を調べ，RPC の終了を検知すると RPC セッション待機関数を実行し，RPC セッション状態変化イベントをイベントキューに挿入する．

イベントキューモジュール　イベントキューを管理する．イベントキューにイベントが存在しない場合，イベント取り出し関数はブロックする．イベント構造体には，サーバのホスト名，リモートライブラリのハンドル，RPC セッションの ID をメンバ変数として含む．それらは，イベントに関連するリモートライブラリ・RPC セッションの情報を参照するのに用いられる．

モジュール群の実装には主に C++を用いた．Actuator でのネットワーク通信にはソケットを用い，リモートライブラリ・RPC セッション・イベントの管理に STL を用いた．

モジュール群のインストールおよび実行に必要なソフトウェアの構成を以下に述べる．以

表 1　F-Omega モジュール API
Table 1　F-Omega module API.

`fomega_initialize("config.conf", argc, argv)`	モジュール群の初期化
`fomega_library_init("hostname", attr_t)`	リモートライブラリの起動
`fomega_library_get(fomega_library_t *, ライブラリ ID)`	リモートライブラリ情報の取得
`fomega_library_delete_by_hostname("hostname")`	リモートライブラリの終了
`fomega_call_async(ライブラリ ID, "funcname", タスク ID, param1, param2, ...)`	非同期 RPC の発行
`fomega_call_get(fomega_call_t *, RPCID)`	RPC セッション情報の取得
`fomega_event_pop(fomega_event_t *)`	イベントの取り出し
`fomega_event_dispatch(fomega_event_t)`	イベント関連メモリ領域の開放
`fomega_finalize()`	モジュール群の終了

下のソフトウェアについては，同等の機能を持つ別のソフトウェア実装で代用できる．

- C++コンパイラ・リンカ
  モジュール群のコンパイル・リンクに必要である．本稿では GNU g++を用いた．
- POSIX スレッド規格に準拠したスレッドライブラリ
  モジュール内のスレッドの制御に必要である．本稿では GNU POSIX スレッドライブラリを用いた．
- GridRPC ミドルウェア
  本稿では Ninf-G ver. 4.2.1 を用いた．Ninf-G の下位のグリッドミドルウェアには Globus Toolkit 4.0.3 を用いた．

GridRPC ミドルウェアについては，Ninf-G でなくても，GridRPC のモデルに準拠したミドルウェアであれば代用できる．ただし，サーバの障害や異常を検知してそれをアプリケーション側に通知する機能が必要である．NetSolve/GridSolve [13]，DIET [14] などはこの要件を満たすため代用できるが，OmniRPC [15] は（執筆時のバージョン 2.0.1 では）この機能を持たないため代用できない．この対策として，F-Omega 自体が障害検知機能を実装する方法が考えられるが，F-Omega のシステムの実装が複雑化するという問題が生じる．障害検知処理（クライアント・サーバ間のネットワーク接続の監視など）を GridRPC ミドルウェアの実装に強く依存した方法でミドルウェアごとに本システムが実装する必要があるためである．このため，GridRPC ミドルウェアが障害検知機能を実装するべきと考える．

5. 評　　価

F-Omega のミドルウェア実装をサンプル GridRPC アプリケーションへ適用し，試験的グリッドにおいて，サーバの運用計画に基づく稼働状況の計画的変動に対するサーバの動的切替えの振舞いを観察した．また，サーバ運用計画ファイル・サーバ利用方針ファイルの記述コストを測定した．

表 2　実験環境
Table 2　Experimental environment.

(a) Pentium4 1.8 GHz × 1CPU，Fedora Core 4 × 4 台
(b) Xeon 2.8 GHz × 2CPUs，CentOS 4 × 8 台
(c) Xeon 2.8 GHz × 2CPUs，Fedora Core 3 × 1 台

実験環境には LAN で接続された 13 台の計算機を用いた（**表 2**）．各計算機に Globus Toolkit の GRAM（gatekeeper および jobmanager）を導入し，各々が独立したサイト（VO）の計算機であるかのように扱った．今日のグリッド実験[1] で用いられる計算機がせいぜい 10 サイトで構成されているため，これは現実的な構成といえる．GridRPC クライアント機として表中 (a) の計算機 1 台を用い，残り 12 台をサーバ機として用いた．サンプルアプリケーションには N-queens を用いた（N=20）．盤の K 列目（K=3）まであらかじめ配置した初期パターンを基に解を数え上げるルーチンを，非同期 GridRPC で並列に実行するものである．

F-Omega のイベント駆動型プログラミングモデルを用いて実際に並列分散アプリケーションを記述する方法を示すべく，本実験で用いた N-queens のソースコード（**図 6**）の概要を述べる．通例，アプリケーションは状態テーブル（本例では `taskstates` 配列変数）を用いて RPC タスクの実行状態を管理する．実行状態は基本的な，未割当て/計算中/計算完了の 3 種類とした（自由に拡張可能）．そして，各イベントハンドラは次の処理を実行する．

ライブラリの追加指示　F-Omega モジュール API のライブラリ起動関数を呼ぶ．
ライブラリの削除指示　F-Omega モジュール API のライブラリ終了関数を呼ぶ．

```
#include "fomega.h"
long *taskstates; //タスク実行状態の管理テーブル
#define STATE_UNALLOCATED (-1) //未割当て
#define STATE_RUNNING (-2)     //計算中
#define STATE_FINISHED (-3)    //計算完了
main(int argc, char *argv[]) {
  app_initialize(); //アプリケーション固有の初期化処理
  fomega_initialize("client.conf", argc, argv);
  while (!completed) {
    fomega_event_pop(&event);
    switch (event.type) {
    case EVENT_LIBRARY_ADD:
      //Actuator からのライブラリ追加指示
      for (i = 0;i < event.operation.amount;i++)
        fomega_library_init(event.operation.hostname,
          event.operation.attr);
      break;

    case EVENT_LIBRARY_SUBTRACT:
      //Actuator からのライブラリ削除指示
      for (i = 0;i < event.operation.amount;i++)
        fomega_library_delete_by_hostname(
          event.operation.hostname);
      break;

    case EVENT_CALL_COMPLETED: //RPC 正常終了
      fomega_call_get(&call, event.callid);
      if (strcmp("get_number_of_solutions",
        call.action) == 0) {
        result += tmpcount[call.taskid];
        taskstates[call.taskid] = STATE_FINISHED;
      }
      if (!app_find_another_job(&taskid)) {
        completed = 1;
        break;
      }
      //未割当てのタスクを割当て
      if (taskstates[taskid] == STATE_UNALLOCATED) {
        //Dispatch asynchronous RPC
        fomega_invoke_async(event.libraryid,
         "get_number_of_solutions", taskid, n, depth,
         fixpos[taskid], &tmpcount[taskid]);
        taskstates[taskid] = STATE_RUNNING;
      }
      break;

    case EVENT_CALL_FAILED: //RPC 失敗
      fomega_call_get(&call, event.callid);
      if (strcmp("get_number_of_solutions",
        call.action) == 0)
        taskstates[call.taskid] = STATE_UNALLOCATED;
      break;
    }
    fomega_event_dispatch(event);
  }
  printf("%d-Queens: result: %ld\n", n, result);
  fomega_finalize();
}
```

図 6　F-Omega のイベント駆動型プログラミングスタイルで書かれた N-queens のソースコード
Fig. 6　Source code of N-queens in the F-Omega's event-driven programming style.

RPC 正常終了時　F-Omega モジュール API の RPC セッション情報取得関数を呼び，RPC 発行時にプログラマが指定したタスク ID を用いて RPC の実行結果を参照し，当該タスクの状態レコードを「計算完了」に更新する．続いて，未割当てのタスクが残っている場合に，非同期 RPC 発行関数を実行し（タスクをリモートライブラリに割り当て），そのタスクの状態レコードを「計算中」に更新する．

RPC 失敗時　失敗したタスクの状態レコードを「未割当て」に更新する．

イベントループ終了後，解の総数を表示して終了する．

5.1　サーバ切替え性能

提案する自動サーバ切替えシステムを用いて長時間の自動サーバ切替え動作の試験を行った．試験設定は次のとおりである．

- サーバ運用計画ファイルに記載されるサーバ稼働状況の計画的変動は，サーバのメンテナンスと外乱ジョブによるとした．ここでメンテナンスは，任意の期間，サーバ全体を表す server サービスか，あるいは他のサービス（4.1 節を参照）のいずれか 1 つを利用不可にする．また，外乱ジョブは，事前予約を用いるジョブを想定し，サーバ上で利用可能なプロセッサ数（job_maxNoOfCPUs サービスの値）を任意の期間減少させる．なお，本実験ではサーバのメンテナンスや外乱ジョブを実際には発生させず，それらに関するシミュレーションで求まる，サービスの稼働状況の計画的変動をサーバ運用計画ファイルに設定するのみとした．
- サーバ運用計画として，1 カ月間に各サーバのサービス群（7 種）のメンテナンスが計 10 回，各サーバ上の外乱ジョブが 10 回発生すると仮定した，シミュレーション結果を用いた．発生日時は一様乱数で与えた．また，各メンテナンスおよび外乱ジョブの長さは 1 時間から 3 日の間の一様乱数で与えた．頻度や長さは TeraGrid [16] のシステムイベントカレンダを参考にして設定した．具体的には，現実的設定値を以下のとおりと算出したが，執筆計画上，数日間でサーバ動的切替えの性能を検証する必要があったため，頻度のみ過剰に設定した．サーバメンテナンスの現実的設定値は，2007 年 1 月から 11 月までのクラスタメンテナンス情報から，頻度の期待値が 2 回/月，その長さは 1 時間から数日でまちまちである．外乱ジョブについては，同期間の Cross Site Run のための NCSA・SDSC・ANL といったサイトの予約情報から，頻度が約 5 回/月，その長さは 1 時間から数日までまちまちである．
- サーバ利用方針では，最大 12 台までプロセッサを用いるとした．また，リモートライブラリの起動に 2 種類のサービスが稼動していることを条件として設定した．

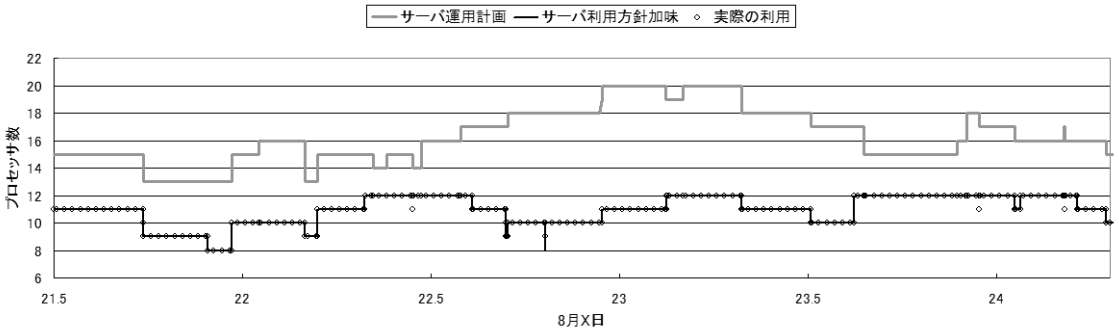


図 7　実験中の利用プロセッサ数の推移
Fig. 7　The number of used processors during operation test.

本実験中に実際に利用されたプロセッサ数の推移を**図 7** に示す．図中の「実際の利用」では，変化のない期間に限り，間引いてプロットしている．また，サーバ運用計画のみで決まる利用可能プロセッサ数と，サーバ利用方針を加味して決まる利用可能プロセッサ数も示す．図より，2.8 日間にわたって，実験中に利用されたプロセッサの数が，サーバ利用方針を加味した場合の利用可能プロセッサ数に正しく追随していることが分かる．なお本実験では，アプリケーションの実行を妨げる，サービスの稼働状況の変動が計 29 回生じたが，計算が正しく継続されていた．これにより，F-Omega によって，(提案するイベント駆動型で書かれた）GridRPC アプリケーションで利用するサーバが自動的に適切に切り替えられるといえる．

アプリケーションの実行性能，特に実行時間を考察する．実行時間は，(1) サーバ切替えに要する時間，(2) サーバ選択アルゴリズムの最適性に依存する．(1) について，従来の手動サーバ切替えよりも F-Omega の自動サーバ切替えの方が短い．F-Omega では本実験のようにサーバ稼動状況の変化に対してサーバ切替え指示システムが定期的（本実験では 1 分ごと）にサーバを切り替える．手動ではこのように短時間で変化を把握しサーバを切り替えることが大変難しい．(2) について，将来的に手動サーバ切替えと F-Omega で同等にできると考えている．F-Omega の現実装ではサーバ選択時にアプリケーションタスクの特性（計算量など）やサーバ性能（プロセッサ周波数など）が考慮されないが，これは F-Omega モジュール群とサーバ選択プログラムの実装の拡張により実現できると考えられる．総合すると，従来の手動サーバ切替えと比べて，F-Omega の自動サーバ切替えの方が，アプリケーション実行時間が短いと考えられる．

表 3　サーバ運用計画ファイルの行数の内訳
Table 3　Detail of the number of lines of server operation schedule file.

内容	行数
ホスト名および外側タグ	5 行
サービス提供量の規定値	サービスの種類の数だけ増加
期間限定の値	メンテナンスごとに 1 行増加， 外乱ジョブごとに 1 行増加

サーバ上の突発的障害に対する F-Omega の耐障害機能は，実際の障害を用いた検証を行っていないものの，実用的と考える．理由は以下のとおりである．障害検出の基礎とした GridRPC ミドルウェア Ninf-G は実際に大規模長時間グリッドアプリケーション実行で使われ，種々の障害を検知できる[17]．また，前述の実験で，F-Omega の自動サーバ切替えシステムによりアプリケーションのサーバ利用不足時に自動的にサーバが追加されることを確認できた．さらに，従来プログラマにとって煩雑であった，リモートライブラリの管理のサーバ動的切替えへの対応・サーバ利用状況の監視処理の実装・代替サーバの選択とその指示の処理の実装が F-Omega では不要である．

5.2　サーバ運用計画ファイルの作成に要する手間

F-Omega で用いるサーバ運用計画ファイルの行数の内訳を**表 3** に示す．表より，メンテナンスや外乱ジョブごとに線形に増加することが分かる．また，期間を重複させた記述

表 4　サーバ利用方針ファイルの行数の内訳
Table 4　Detail of the number of lines of server use policy file.

内容	行数
最外側タグ	3 行
利用方針	利用方針ごとに 5 行
（以下内訳）	
期間の記述	1 行
ホスト名・要求サービス	1 行
リモートライブラリ総数	1 行
方針定義タグ（開き/閉じ）	2 行

（4.1 節）により，その増加分が 1 行ずつに抑えられている．

サーバ管理者が本ファイルでサーバ運用計画を記述する際の手間は，一般的に，従来の自然言語の電子メールやウェブページを用いた場合の手間よりも少ないか同程度である．なぜなら，本ファイルではサービスの稼動状況が端的な代入文で記述されているからである．

5.3　サーバ利用方針ファイルの作成に要する手間

サーバ利用方針ファイルの行数の内訳を**表 4** に示す．ユーザが指定するサーバ利用方針の数に応じて線形に増加することを確認した．

6. 関 連 研 究

グリッドアプリケーションにおけるサーバの動的切替えにはいくつかのアプローチがある．本稿で提案するアプローチは次を特徴とする．F-Omega では，サーバ運用計画情報の自動監視機能，サーバ動的切替えに対応したリモートライブラリの管理機能，サーバ利用状況の自動監視機能，代替サーバの選択と自動指示機能，サーバ可変グリッドアプリケーション開発の支援機能を新たに実装し，他の既存機能と統合した自動サーバ切替えシステムをユーザに提供する．ここでは，既存のアプローチと本稿で提案するアプローチを比較する．

Condor[18] は Matchmaking とジョブ管理の機能を提供しており F-Omega と似ているが，複雑なサーバ利用契約に応じたサーバの自動切替えが大変困難である．なぜなら，Condor ではアプリケーションプロセスのエラーによってサーバの切替えが始まるが，そのエラーを契約に応じて自動的に発生させるアクセス制御機構（プロセッサの利用を制限する機構やディスククォータなど）が現状のグリッドのほとんどの計算機に配備されているとはいえないためである．この機構の代わりに，Condor に対し手作業でサーバ動的切替えを指示する場合，数時間ごとにたびたび指示をしなければならず作業コストが大きいため，サーバ動的切替えが大変困難である．F-Omega では，前述のアクセス制御機構がサーバ上になくても，サーバ切替え指示システムがサーバ運用計画ファイルを用いてハードウェア・ソフトウェアの利用可能量を監視し，自動的にサーバを切り替えることができる．

Takemiya らの大規模長時間実行グリッドアプリケーション[1] では，サーバ切替え機構の再利用が困難である．グリッドアプリケーションのための共通モジュールからなる，本稿で提案するアプローチと異なり，この機構はアプリケーション自身に静的にアプリケーション独自の方法で実装されている（サーバ利用方針を記述したファイルを読み込み，サーバ利用をそれに適応させるアプリケーション処理工程からなる）．したがって，任意のアプリケーションにこのアプローチを適用することが難しい．

Globus Toolkit の DUROC（Dynamically -Updated Request Online Coallocator）[19] はサーバの性能パラメータを考慮してジョブに対して初期サーバ群を自動的に割り当てるが，F-Omega で提供するような，アプリケーション実行中のサーバ動的切替えの機能を持たない．

タスクファーミング API[20] では，GridRPC 計算タスクのためのサーバをサーバリストから自動的に割り当てる機能を提供する．しかし，F-Omega のような，サーバのハードウェア/ソフトウェア稼動状況に応じて自動的にサーバを切り替える機能が提供されていない．

アプリケーションの利用サーバを切替え可能とするべく，Moab Grid Suite[21]，Condor，ProActive[22] といったグリッドミドルウェアでは，プロセスマイグレーション機能が利用できる．F-Omega はサーバ切替え指示の枠組みとして位置づけられ，プロセスマイグレーション機構と相補的である．

7. まとめと今後の課題

本稿では，グリッドアプリケーションにおけるサーバの動的切替えのための枠組み F-Omega を提案した．F-Omega は，サーバ運用計画情報の自動監視機能，サーバ動的切替えに対応したサーバ利用環境の管理機能，サーバ利用状況の自動監視機能，代替サーバの自動選択と自動指示機能，サーバ可変グリッドアプリケーション開発の支援機能を新たに実装し，他の既存機能を統合した自動サーバ切替えシステムをユーザに提供することを特徴とする．

適用実験では，サンプルアプリケーションに F-Omega のミドルウェア実装を適用し，試験的グリッドにおいてサーバ利用の振舞いを観察した．その結果，2.8 日間にわたって，サーバ運用計画とサーバ利用方針に応じてアプリケーションが利用するサーバが自動的に適切に切り替えられることを確認した．また，サーバ上の突発的障害に対して有用な耐障害機

能を備えていることを考察した．すなわち，F-Omega により，サーバの動的切替えにおける煩雑さ（1 章参照）を軽減させることができた．以上より，サーバの動的切替えにおいて F-Omega の有用性は高いと考える．F-Omega はグリッドアプリケーションの長時間実行において高い利便性を提供し，グリッドの実用化を支援する．

本研究の今後の課題は次のとおりである．

- GridRPC 以外のグリッドミドルウェアへの対応
  GridRPC ではサーバごとにプログラムの起動/終了を制御できるため，サーバの切替えを実装しやすい．同じ理由で，ProActive などの分散オブジェクトモデルのグリッドミドルウェアに対しても F-Omega を適用しやすいと考えられる．一方，Grid-enabled MPI[23] などではサーバごとにプログラム起動/終了を制御し難い．F-Omega の適用には，プロセス間接続の復元なども考慮したサーバ切替え機構が必要である．
- 運用計画/利用方針ファイル間のマッチング項目の書式統一
  各サーバ管理者が作成するサーバ運用計画ファイル内のサービス項目と，ユーザが作成するサーバ利用方針ファイル内の要求サービス項目は，相互運用可能な書式で統一される必要がある．たとえば，`globus_gram="ON"` と `GT-GRAM="1"` が実際では同一のサービス項目を意味する場合でも，現時点では異なる項目と判断される．
- タスク特性やサーバ性能を考慮したサーバ選択
  アプリケーションの実行性能をさらに高めるべく，タスクの特性やサーバ性能を考慮したサーバ選択が行えるよう，F-Omega モジュール群とサーバ選択プログラムの実装を拡張する必要がある．
- サーバ利用方針ファイルと Ninf-G のクライアントコンフィギュレーションファイルの自動同期
  サーバ利用方針ファイルでは，Ninf-G のクライアントコンフィギュレーションファイルと同じ属性（ライブラリの動作のためのプロセッサ数など）も指定可能とするべきである．その際，これらのファイルの整合性を保つための作業コストが大きいため，自動的に同期させる機構が必要である．

謝辞　本研究を進めるにあたり，数多くのご助言をいただきました，産業技術総合研究所の中田秀基氏に深く感謝いたします．

参 考 文 献

1) Takemiya, H., Tanaka, Y., Sekiguchi, S., Ogata, S., Kalia, R.K., Nakano, A. and Vashishta, P.: Sustainable Adaptive Grid Supercomputing: Multiscale Simulation of Semiconductor Processing across the Pacific, *Proc. ACM/IEEE SC 2006 Conference*, p.23 (2006).
2) Tanimura, Y., Ikegami, T., Nakada, H., Tanaka, Y. and Sekiguchi, S.: Implementation of Fault-Tolerant GridRPC Applications, *Journal of Grid Computing*, Vol.4, No.2, pp.145–157 (2006).
3) Globus Toolkit. http://www.globus.org/
4) Smallen, S., Ericson, K., Hayes, J. and Olschanowsky, C.: User-level grid monitoring with Inca 2, *High Performance Distributed Computing, 2007 workshop on Grid monitoring*, pp.29–38 (2007).
5) Placek, M. and Buyya, R.: G-Monitor: Gridbus web portal for monitoring and steering application execution on global grids, *Proc. International Workshop on Challenges of Large Applications in Distributed Environments* (*CLADE 2003*) (2003).
6) 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). http://ninf.apgrid.org/
7) Seimour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C. and Casanova, H.: Overview of GridRPC: A Remote Procedure Call API for Grid Computing, *Proc. Grid Computing–Grid 2002*, pp.274–278 (2002).
8) Tanaka, Y., Takemiya, H., Nakada, H. and Sekiguchi, S.: Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid, *Proc. 5th IEEE/ACM International Workshop on Grid Computing* (2004).
9) 渡邊啓正，本多弘樹，弓場敏嗣，田中良夫，佐藤三久：Relis-G：計算グリッドのための遠隔ライブラリインストール機構，情報処理学会論文誌：コンピューティングシステム，Vol.45, No.SIG11, pp.196–206 (2004).
10) Comprehensive Perl Archive Network. http://cpan.org/
11) GNU Wget. http://www.gnu.org/software/wget/
12) Apache HTTP Server Project. http://httpd.apache.org/
13) YarKhan, A., Dongarra, J. and Seymour, K.: GridSolve: The Evolution of Network Enabled Solver, *Grid-Based Problem Solving Environments*, Gaffney, P.W. and Pool, J.C.T. (Eds.), Vol.239, pp.215–224, Springer, Boston (2007).
14) Caron, E. and Desprez, F.: DIET: A Scalable Toolbox to Build Network Enabled

Servers on the Grid, *International Journal of High Performance Computing Applications*, Vol.20, No.3, pp.335–352 (2006).
15) 佐藤三久，朴　泰祐，高橋大介：OmniRPC：グリッド環境での並列プログラミングのための Grid RPC システム，情報処理学会論文誌：コンピューティングシステム，Vol.44, No.SIG11, pp.34–45 (2003).
16) TeraGrid. http://www.teragrid.org/
17) 武宮　博，田中良夫，中田秀基，関口智嗣：MPI と GridRPC を利用した大規模 Grid アプリケーションの開発と実行：Hybrid QM/MD シミュレーション，情報処理学会論文誌：コンピューティングシステム，Vol.46, No.SIG12, pp.153–160 (2005).
18) Condor. http://www.cs.wisc.edu/condor/
19) Czajkowski, K., Foster, I. and Kesselman, C.: Resource Co-Allocation in Computational Grids, *Proc. 8th IEEE International Symposium on High Performance Distributed Computing* (*HPDC-8*), pp.219–228 (1999).
20) 谷村勇輔，中田秀基，田中良夫，関口智嗣：GridRPC を用いたタスクファーミング API の設計と実装，情報処理学会研究報告 2005-HPC-103，pp.67–72 (2005).
21) Cluster Resources, Moab Grid Suite. http://www.clusterresources.com/
22) Baduel, L., Baude, F., Caromel, D., Contes, A., Huet, F., Morel, M. and Quilici, R.: *Grid Computing: Software Environments and Tools, Chapter: Programming, Deploying, Composing, for the Grid*, Springer-Verlag (2006).
23) Karonis, N.T., Toonen, B. and Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing* (*JPDC*), Vol.63, No.5, pp.551–563 (2003).

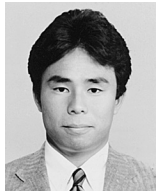
(平成 20 年 5 月 1 日受付)
(平成 20 年 9 月 24 日採録)

渡邊　啓正（正会員）

2003 年電気通信大学情報工学科卒業．2008 年同大学大学院情報システム学研究科博士後期課程修了．同年より HPC システムズ株式会社に勤務．計算グリッドのアプリケーション実行環境およびプログラム開発支援環境に関する研究に従事．情報処理学会第 65 回全国大会にて学生奨励賞受賞．

平澤　将一（正会員）

2000 年東京大学理学部情報科学科卒業．2002 年同大学大学院理学系研究科情報科学専攻修了．2005 年同大学院情報理工学系研究科コンピュータ科学専攻博士課程単位取得退学．2006 年電気通信大学大学院情報システム学研究科助手．2007 年同助教．高性能計算システム，プログラミング言語処理に関する研究に従事．ACM，IEEE CS 各会員．



本多　弘樹（正会員）

1984 年早稲田大学理工学部電気工学科卒業．1991 年同大学大学院理工学研究科博士課程修了．1987 年より同大学情報科学研究教育センター助手．1991 年より山梨大学工学部電子情報工学科専任講師．1992 年より同助教授．1997 年より電気通信大学大学院情報システム学研究科助教授．2007 年より同教授．並列処理方式，並列化コンパイラ，並列計算機アーキテクチャ，グリッド等の研究に従事．工学博士．電子情報通信学会，IEEE-CS，ACM 各会員．平成 15 年度山下記念研究賞受賞．