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

2007－ARC－172 (31)
2007－HPC－109 (31)
2007／3／2

F-Omega: サーバ稼動状況に適応するGridRPCアプリケーションの開発・実行フレームワーク

渡邊啓正† 平澤将一† 本多弘樹†

グリッドアプリケーションにはユーザの指示にしたがって計算ノードを動的に変更して処理を継続するフレキシビリティが必要である．本稿では，フレキシブルなグリッドアプリケーションを容易に開発・実行するためのフレームワーク **F-Omega** を提案する．F-Omega には以下の特徴がある．リモートライブラリの管理を隠蔽した平易なイベント駆動型プログラミング方式を採用している，計算資源の利用に関する制約を自動的に可視化する機能を備えた資源利用の動的制御インタフェースを持つ．適用実験の結果，リモートライブラリの管理が不要な見通しのよいグリッドアプリケーションを作成できること，計算資源の利用に関する制約を監視する手間が軽減され資源利用の制御を効率よく行えることを確認した．

F-Omega: A Framework for GridRPC Application with Adaptive Server Use

Hiromasa Watanabe,† Shoichi Hirasawa† and Hiroki Honda†

Grid applications need flexibility so that they can execute persistently even they have to change computation nodes according to user-specified schedule about resource usage. In this paper, we propose a framework **F-Omega** to easily develop and execute flexible grid applications. F-Omega has the following features; easy event-driven programming style for grid applications while concealing remote library management, automatic visualization of resource usage constraints and user interface for dynamic control of resource usage of grid applications. Experimental results for F-Omega applications show that F-Omega enables programmers to create grid applications easily while keeping simple and unmixed outlook without implementing remote library management. F-Omega also enables users to control resource usage efficiently by mitigating user's burden to monitor server constraints.

1. はじめに

近年，広域ネットワーク上の計算資源を用いた分散並列計算環境であるグリッドを実用化するための研究が盛んに行われている[1),2)]．グリッドの実用化に際しては，グリッド上で動作する分散並列アプリケーションプログラム (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あるいは数週間安定して動くことが要求される．一方，グリッド計算機の運用において生ずる，システムのメンテナンスや優先度の高いジョブの割り込みはグリッドアプリケーションの長時間実行を阻害する．そのため，長時間実行されるグリッドアプリケーションには，アプリケーションユーザの指示にしたがって利用する計算機を動的に変更し処理を継続する柔軟性が求められる．本稿ではこの柔軟性をフレキシビリティと呼び，その性質を持つグリッドアプリケーションを，フレキシブルなグリッドアプリケーションと呼ぶ．

本研究では，グリッドアプリケーションのプログラミングモデルとしてGridRPC[3)]を用いる．GridRPCではサーバ計算機への計算タスクの動的割り付けの実現が容易である．そのため，GridRPC はフレキシブルなグリッドアプリケーションの開発に向いている．しかしながら，GridRPC のエンドユーザ API では，サーバ計算機上のリモートライブラリを関数ハンドルで抽象化し RPC を実行する，プリミティブな機能のみが提供されている．そのため，利用するサーバが実行中に設定されるフレキシブルな GridRPC アプリケーション (クライアントプログラム) の開発には，サーバリスト・リモートライブラリリストを管理する機構が別途必要となる．従来，プログラマがアプリケーション毎にこの機構を実装する必要があり，この作業はプログラマにとって複雑で手間がかかる．また，従来のGridRPC プログラミング方式では，利用不可能・初期化済といった，異なった状態にあるリモートライブラ

† 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

リに対する処理をソースコード上で複合して記述する．この方式に則ったソースコードは複雑で入り組んでおり，見通しが悪い．一方，フレキシブルな GridRPC アプリケーションの開発後の実行において，ユーザはサーバ計算機の利用における制約にしたがってアプリケーションの資源利用を制御する．しかし，利用可能プロセッサ数といった，日時によって変化する制約を多数のサーバ計算機について監視するには手間がかかる．

これらの問題を解決するべく，われわれは，フレキシブルなグリッドアプリケーションを容易に開発・実行可能とするフレームワーク **F-Omega** を提案する．F-Omega では，リモートライブラリの管理を隠蔽した平易なイベント駆動型プログラミング方式が利用可能である．また，サーバ計算機の利用に関する制約をユーザが容易に監視可能とするべく，制約情報ファイルを自動的にダウンロードするスクリプト (クローラと呼ぶ) を用いる．クローラは制約情報を収集し，収集された情報を Web アプリケーションがユーザに可視化する．

サンプル GridRPC アプリケーションに対する F-Omega の適用実験の結果，リモートライブラリの管理を必要としない見通しのよいグリッドアプリケーションを作成できることを確認した．また，計算資源の利用に関する制約情報を監視する手間が軽減されユーザが資源利用の制御を効率よく行えることを確認した．

2. フレキシブルな GridRPC アプリケーションの容易な開発・実行のための要件

本研究では，フレキシブルな GridRPC アプリケーションの容易な開発・実行のための要件として以下の二つを挙げる．

リモートライブラリの自動管理　初期化済といった，所定の状態にあるリモートライブラリに対する計算タスクの投入を容易に実装可能とするべく，サーバ・リモートライブラリの状態と，RPC の実行に必要な関数ハンドルをミドルウェアで自動的に管理する必要がある．

分離された処理記述　異なった状態に属するリモートライブラリ群に対する制御をソースコード上で見通しよく記述可能とするため，リモートライブラリの状態に応じて処理の記述を分離する必要がある．

これらの機能により，アプリケーションプログラマはリモートライブラリの状態に応じたアプリケーション依存の処理を個別に記述するだけで GridRPC アプリケーションを開発できる．そして，リモートライブラリの状態の管理に要する手間やソースコードの見通しの悪さが改善されると考えられる．

リモートライブラリの状態を自動的に管理するミドルウェアとして，タスクファーミング API[4] がある．タスクファーミング API はリモートライブラリを自動的に管理し，入力パラメータを変えながら多数の独立なタスクを実行する（ファーミングと呼ぶ）ための機能を提供する．実行されるタスクは初期化以外の依存関係が無く，逐次的に実行されるという制限がある．したがって，データの転送やタスクの割り当て先を明示的に指定するグリッドアプリケーションには適用が難しい．データの転送やタスクの割り当て先をアプリケーションから制御可能としながら，リモートライブラリの状態を自動的に管理する機能が必要である．

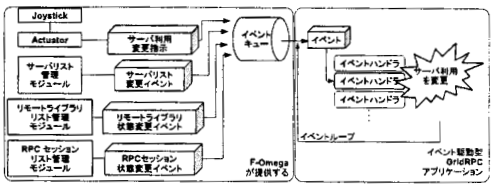

図 1　イベント駆動 GridRPC プログラミング
Fig. 1　Event-driven GridRPC programming

資源利用に関する情報の自動可視化　資源の利用に関する制約情報は利用するサーバのサイトの数に応じて増大し，情報の更新のタイミングはサイトによって異なる．したがってユーザが常に全ての制約情報を監視するのは煩雑であり，制約情報を自動的に収集してユーザに可視化する必要がある．また，実行中のグリッドアプリケーションの資源利用をユーザが容易に把握可能とするべく，グリッドアプリケーションで利用中のリモートライブラリの状態を自動的にユーザに可視化する必要がある．

グリッドアプリケーションの資源利用を制御する方法として，グリッドの資源利用を制御するスーパースケジューラが考慮されるべきである．しかし，スーパースケジューラシステムはまだ実用段階に至っておらず，ほとんどのグリッド利用環境においてスーパースケジューラが配備されていないと想定するべきであると考える．利用資源をユーザが直接指示することを想定した機能が必要である．

3. F-Omega の設計

フレキシブルな GridRPC アプリケーションを容易に開発可能とするべく，F-Omega では次のモジュール群を提供する．

- サーバ・リモートライブラリ・RPC セッションのリストを自動管理するモジュール
- サーバリスト・リモートライブラリの状態・RPC セッションの状態の変化をイベントとしてアプリケーションに伝達するモジュール
- サーバリストやアプリケーションの資源利用を変更するモジュール (Actuator)

これらのモジュールは，アプリケーション実行中に発

生する状況変化を察知し，イベントとしてアプリケーションへ伝達する．ここで状況変化とは次のものである．

- Actuator によるサーバ利用の変更指示
- サーバリストの変化
- リモートライブラリの状態の変化
- RPC セッションの状態の変化

これらのモジュールを用いて，プログラマは，リモートライブラリの起動・終了，計算タスクの割り当て・キャンセルといった，サーバ利用を変更する処理をイベントハンドラとして実装する (図 1)．Actuator がサーバ利用を変更する指示をイベントとして出し，イベントハンドラがサーバ利用を変更することにより，適応的なサーバ利用が実現される．この実装方式をイベント駆動型 GridRPC プログラミング方式と呼ぶ．モジュール群は，サーバのホスト名・リモートライブラリの識別番号・RPC セッションの識別番号をメンバ変数として持つイベントを生成する．これにより，イベントのメンバ変数を用いることでサーバ利用を変更する処理を容易に記述できる．

本方式を用いた GridRPC クライアントプログラムのソースコードの一部を図 2 に示す．例では，イベントキューからイベントを取り出す関数が呼ばれ，Actuator からのリモートライブラリの追加指示イベントの場合に，イベントに含まれるサーバホスト名を用いてリモートライブラリ起動関数が呼ばれている．

以下，各モジュールの機能を説明する．

サーバリスト管理モジュール　利用可能なサーバのリストを保持する．Actuator によりサーバリストが変更された際，サーバリスト変化イベントをイベントキューに追加する．

リモートライブラリリスト管理モジュール　利用中のリモートライブラリのリストを保持する．リモートライブラリ起動関数の実行時に，リモートライブラリの制御に用いる関数ハンドルを保持するためのメモリ領域を自動的に確保して割り当てる．GridRPC API の実行によりリモートライブラリの状態 (利用可否・初期化済み等，拡張可能) が変化した場合に，リストを自動的に更新し，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追加する．

RPC セッションリスト管理モジュール　実行中の RPC セッションのリストを保持する．非同期 RPC の実行時に，RPC セッションの制御に用いるセッション ID を保持するためのメモリ領域を自動的に確保して割り当てる．また，RPC の終了を暗黙に待機するべく，非同期 RPC が実行される度に，RPC の終了を待機するスレッドを生成する．各スレッドは各 RPC セッションの状態を定期的に調査し，RPC が終了したと検知すると，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追加する．

```
main() {
  int unit_dispatch_count = 0;
  int unit_done_count = 0;
  grpc_initialize(); //GridRPC ミドルウェアの初期化
  fomega_initialize(); //F-Omega モジュール群の初期化
  while (can_exit == 0) { //イベントループ
    //イベントキューからイベントを取り出す (無ければブロッキング)
    event_t event = fomega_pop_event();
    switch (event.type) {
    case EVENT_LIBRARY_ADD: //リモートライブラリの追加指示
      //リモートライブラリを起動させる
      app_library_init(event.hostname, "solver/classa",
        event.library_num);
      break;
    case EVENT_LIBRARY_SUBSTRACT: //リモートライブラリの削除指示
      //リモートライブラリを終了させる
      app_library_destroy(event.hostname, "solver/classa",
        event.library_num);
      break;
    case EVENT_LIBRARY_READY: //リモートライブラリが利用可能になった
      if (unit_dispatch_count < NUMOFTASK) {
        //未割り当ての計算タスクが残っていたら RPC
        fomega_call_async(event.libraryid, "unit_task");
        unit_dispatch_count++;
      }
      break;
    case EVENT_TYPE_RPC_COMPLETED: //RPC が正常終了した
      unit_done_count++;
      if (unit_done_count == NUMOFTASK) {
        //終了要求イベントをイベントキューに追加する
        fomega_push_quit_event();
      } else if (unit_dispatch_count < NUMOFTASK) {
        //未割り当ての計算タスクが残っていたら RPC
        fomega_call_async(event.libraryid, "unit_task");
        unit_dispatch_count++;
      }
      break;
    case EVENT_TYPE_RPC_FAILED: //RPC が異常終了した
      unit_dispatch_count--;
      break;
    case EVENT_TYPE_QUIT: //終了要求
      can_exit = 1;
      break;
    }
  }
  fomega_finalize(); //F-Omega モジュール群の終了処理
  grpc_finalize(); //GridRPC ミドルウェアの終了処理
}
```

図 2　イベント駆動型 GridRPC アプリケーションのソースコード例

Fig. 2　Example source code of event-driven GridRPC application

イベントキューモジュール　イベントキューを保持する．イベント取り出し関数はイベントキューが空のときブロックする．イベントからは，イベントの内容に付随して，関係するサーバのホスト名・リモートライブラリのハンドル・RPC セッション ID をメンバ変数として参照できる．これらの情報はサーバ・リモートライブラリ・RPC セッションの状態を参照するのに用いられる．

Actuator　Joystick(後述) との通信を行う TCP サーバスレッドを作成する．Joystick からの要求に従って，利用中のリモートライブラリの状態に関する情報を Joystick に返し，サーバリストを変更し，アプリケーションにサーバ利用の変更を指示する．Actuator はサーバ利用の変更要求リストを保持し，変更を実行すべき

日時に達したとき，サーバ利用の変更イベントを生成し，イベントキューに追加する．資源利用の変更指示は，特定の日時・サーバにおける利用プロセッサ数の変更値として与える．

F-Omega では，GridRPC アプリケーションの資源利用を制御するためのユーザインタフェースとして，ユーザからの指示を TCP/IP を用いたプロセス間通信で Actuator へ伝達する Web アプリケーション (Joystick と呼ぶ) を用いる．Actuator と Joystick の分離は Actuator と独立した制御インタフェースの機能拡張，および複数のグリッドアプリケーションの制御インタフェースの複合を意図して行った．Joystick は以下の機能を提供する．

Joystick　GridRPC アプリケーションで利用しているリモートライブラリの状態を可視化し，資源利用の変更のためのインタフェースを提供する．また，サーバ計算機の利用に関する制約情報をユーザが容易に監視可能とするべく，クローラを用いて制約情報を自動的に収集し，インタフェース上に可視化する．あらかじめ，各サーバ管理者はサーバの状態や利用上の制約をあらわすファイルを作成し，サイト内の Web サーバ上に配置してその URL を公開する．ユーザがその URL を Joystick のインタフェースに入力すると，Joystick はクローラを用いて制約情報ファイルを定期的にダウンロードし，解析結果をユーザに可視化する．

以上により構成される F-Omega は次の流れで機能する (図 3 を参照)．

(1) プログラマはリモートライブラリの管理を隠蔽したイベント駆動型プログラミング方式により GridRPC クライアントプログラムを記述する．GridRPC クライアントプログラムをコンパイルして，提案モジュール群とリンクさせる．

(2) GridRPC クライアントプログラムを起動する．このとき Actuator と通信するための IP アドレスとポート番号が標準出力に表示される．

(3) Joystick を Web アプリケーションサーバ上で動作させる．

(4) サーバ管理者がサーバ計算機の利用に関する制約をあらわすファイルを作成し，Web サーバ上で公開する．

(5) ユーザは Joystick を Web ブラウザで開き，Actuator の IP アドレスとポート番号を入力して，資源利用を制御可能な状態にする．

(6) ユーザが制約情報ファイルの URL を Joystick に入力すると，クローラが動作してサーバ計算機の利用に関する制約が Joystick 上に自動的に可視化される．

(7) ユーザは制約情報を参考にしながら，Joystick に対して資源利用の変更指示を入力する．

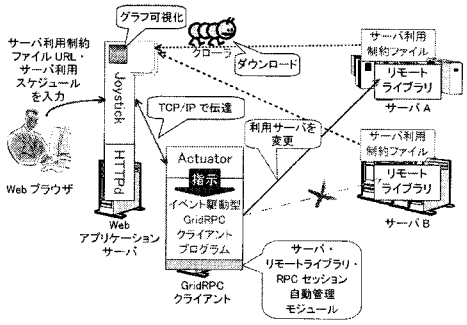

図 3　F-Omega のコンポーネント群の振舞い
Fig. 3　How components in F-Omega work

(8) 指示を受けた Actuator が GridRPC クライアントプログラムの資源利用を変更する．

4. F-Omega の実装

実装にあたり，GridRPC の参照実装である Ninf-G[5] を用いた．GridRPC クライアントプログラムにリンクさせるモジュール群は C++言語を用いて実装した．また，スレッドの作成やクリティカルセクションの制御のために POSIX スレッドライブラリを用いた．以下，各部分の実装について説明する．

サーバリスト管理モジュール　STL(Standard Template Library) の vector を用いてサーバ計算機のホスト名と状態を管理する．本モジュール内のサーバの状態情報を変更する関数で，サーバリスト変化イベントを生成し，イベントキューへ追加する．

リモートライブラリリスト管理モジュール　各リモートライブラリに一意のライブラリ ID(整数値) を割り当て，STL の map を用いてリモートライブラリの関数ハンドルと状態を管理する．リモートライブラリ状態変化イベントのメンバ変数としてライブラリ ID を設定可能とし，アプリケーションがイベントのライブラリ ID を用いてリモートライブラリの情報を取得可能とした．GridRPC API のラッパー API を作成することにより，リモートライブラリ起動関数が，関数ハンドル用メモリ領域の割り当て・状態リストの変更・状態変化イベントの生成とイベントキューへの追加を行う．

RPC セッションリスト管理モジュール　各 RPC セッションに一意の RPCID(整数値) を割り当て，STL の map を用いて RPC のセッション ID と状態を管理する．RPC セッション状態変化イベントのメンバ変数として RPCID を設定可能とし，アプリケーションがイベントの RPCID を用いて RPC セッションの情報を取得可能とした．GridRPC API のラッパー API を作成することにより，非同期 RPC 実行関数がセッション ID 用メモリ領域の割り当てを行い，RPC の終了を

図 4 リモートライブラリリストの可視化例
Fig. 4 Visualization example of a list of remote library

待機するスレッドを作成する．

イベントキューモジュール STL の deque を用いてイベントキューを管理する．イベントキューが空のときのイベント取り出し関数におけるブロッキングは POSIX スレッドの条件変数を用いて実装した．

Actuator TCP サーバの実装に PF_UNIX のソケットを用いた．Joystick からの要求の並列処理のため，要求毎に対応を行うスレッドを生成する．Joystick との通信に用いるメッセージプロトコルには軽量な JSON[6] によるテキスト表記を用いた．

Joystick Web アプリケーションの実装に Perl 言語による CGI を用いた．GridRPC クライアントプログラム内のリモートライブラリの状態はサーバ毎に正常・エラーの 2 つの状態で分類し表形式で可視化した (図 4 を参照)．サーバの状態や利用における制約情報を示すファイルは図 5 に示す XML 形式で記述した．XML は可読性と，任意の種類の制約項目を記述可能とする拡張性に優れている．本書式では，サーバの稼動状態・サーバの監視に使うツール・サーバの利用上の制約が記述される．また，記述内容が有効となる期間を startDate(有効となる時刻)，endDate(無効となる時刻) という属性で表す．クローラは Perl 言語で実装し，クローラの定期的実行には cron を用いた．スクリプトと Web アプリケーション間のデータ授受には，軽量な SQL データベースである SQLite[7] を用いた．Web アプリケーションでは，サーバ計算機の利用に関する制約情報を横軸を時刻，縦軸を項目の値とする積層型エリアグラフで可視化した (図 6 を参照)．グラフ作成には ChartDirector[8] を用いた．

5. 適用実験

実装した F-Omega をサンプル GridRPC アプリケーションに適用し，動作確認および 2 章に述べた要件に対する有効性の検証を行った．実験には LAN で接続されたクライアント PC1 台，サーバ PC4 台からなるグリッドを用いた．Joystick はクライアント PC 上で動作させた．グリッドミドルウェアには Ninf-G

```
<?xml version="1.0"?>
<gatekeeper hostname="asuka.yuba.is.uec.ac.jp">
  <states>
    <!-- サーバの稼動状態 -->
    <state availability="1.0" name="RUNNING"/>
  </states>
  <monitors>
    <!-- サーバ監視ツールへのリンク -->
    <monitor type="Ganglia"
     url="http://asuka.yuba.is.uec.ac.jp/~watanabe/"/>
  </monitors>
  <constraints>
    <!-- ジョブ投入時の制限 -->
    <constraint job_maxNoOfCPUs="2" endDate="1168355399"/>
    <constraint job_maxNoOfCPUs="1" startDate="1168355400"
     endDate="1168387799"/>
    <constraint job_maxNoOfCPUs="2" startDate="1168387800"/>
    <constraint job_maxMemory="256"/>
  </constraints>
</gatekeeper>
```

図 5 サーバ利用制約ファイルの例
Fig. 5 Example of server use constraint file

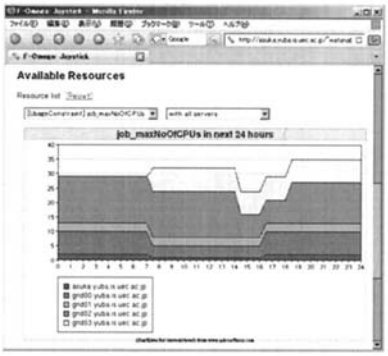

図 6 サーバ利用制約情報の可視化例
Fig. 6 Visualization example of a list of server use constraint

4.1.0 と Globus Toolkit 4.0.3[9] を用いた．

動作確認として，計算ノードの故障を人為的に再現するべく，実行中のリモートライブラリプロセスを kill によって強制終了させた．その際，リモートライブラリが利用不可能状態になったことを Joystick を介して 5 秒 (任意の監視間隔) 以内に確認できた．さらに代理サーバを利用するよう資源利用スケジュールを動的に再設定することで，故障時までの計算結果を失わずにアプリケーションの処理を継続できた．

5.1 資源管理の手間とソースコードの見通しの評価

図 2 は利用可能なサーバに対して非同期 RPC を実行する GridRPC アプリケーションを GridRPC イベント駆動型プログラミング方式を用いて記述した場合のソースコードである．

図 2 のソースコードから，サーバ名・関数ハンドル・リモートライブラリの状態といった情報がイベントのメンバ変数として供給されており，アプリケーション側でサーバリスト・リモートライブラリの状態リストを実装する必要がないことを確認できた．また，処理

対象とするリモートライブラリやRPCの状態がイベント内容から明らかで，イベント内容に応じて処理の記述が図中の枠のように分離されている．これにより，単純で込み入っていない，見通しのよい記述が行えていることを確認できた．

5.2 資源利用に関する情報の監視の手間の評価

F-Omegaを適用したサンプルGridRPCアプリケーションの実行において，ユーザがJoystickに対しサーバ利用制約ファイルのURLを一度入力するだけで，ユーザは常に最新の制約情報を視覚的に把握できた．一方従来では，ユーザが電子メールやサーバの運営情報サイトに定期的にアクセスするなどして，手作業で制約情報を収集・解析する必要があった．したがって，従来より少ない手間で資源利用に関する制約情報の監視を行えることを確認できた．

6. 関連研究

武宮らによる大規模長時間実行Gridアプリケーション[1]では，利用するサーバのリストを更新するために，サーバの利用制約情報ファイルを再読込みする処理がアプリケーション内に静的に実装されている．F-Omegaでは，利用するサーバのリストの更新処理をアプリケーション内で静的に実装する必要がなく，Joystickからの更新要求を受けてActuatorが自動的に実行する．これにより，アプリケーションが単純となる．

GridRPCアプリケーションの長期安定実行を目指したNinf-C[10]では，チェックポインティング可能なスケジューリングシステムCondor[11]を基にして，GridRPCアプリケーションのマスタ・ワーカ双方における耐故障性を実現している．F-Omegaでは，GridRPCシステムが提供する耐故障機能を用いる(Ninf-Gでは故障検知機能を提供している)．Ninf-Cではジョブスケジューリングシステムが計算タスクの割り当てをアプリケーションと独立して自動的に行うため，アプリケーション単位で資源利用を変更することが難しい．

7. まとめと今後の課題

フレキシブルなグリッドアプリケーションを容易に開発・実行可能とするフレームワークF-Omegaの設計・実装を行った．サンプルGridRPCアプリケーションを用いた適用実験により，リモートライブラリの管理の手間や見通しの悪さが改善されて，プログラマがフレキシブルなGridRPCアプリケーションを容易に開発できること，計算資源の利用に関する制約を監視する手間が軽減されて，ユーザが資源利用の制御を効率よく行えることを確認した．F-Omegaはグリッドアプリケーションの長時間実行の実現において高い利便性を提供し，グリッドの実用的な利用を支援する．

本研究の今後の課題には以下が挙げられる．

- グリッド上のスーパースケジューラからグリッドアプリケーションの資源利用を制御可能とするべく，資源利用の制御インタフェースを，XML等を基にした相互運用性の高い規格に準拠したものとする．
- 継続的な制御作業に要するユーザの手間を軽減するべく，サーバの利用上の制約が変更された場合に，資源利用スケジュールの変更の必要性をF-Omegaが自動的に判断し，必要なときのみユーザに資源利用スケジュールの入力を促せるようにする．
- 計算タスクの再実行や冗長実行を容易にプログラミング可能とするべく，RPCで実行した計算タスクにアプリケーション依存のIDを割り付けるなどして，実行した計算タスクの参照を容易にする．

参考文献

1) 武宮博, 田中良夫, 中田秀基, 関口智嗣: 動的に計算量が変化する大規模長時間実行Gridアプリケーションの実現, 情報処理学会論文誌コンピューティングシステム, Vol. 47, No. SIG 18, pp. 31–43 (2006).
2) 谷村勇輔, 池上努, 中田秀基, 田中良夫, 関口智嗣: 耐障害性を考慮したNinf-Gアプリケーションの実装と評価, 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. SIG 7, pp. 18–27 (2005).
3) K.Seimour, H.Nakada, S.Matsuoka, Dongarra, J., C.Lee and H.Casanova: Overview of GridRPC: A Remote Procedure Call API for Grid Computing, *Proceedings of Grid Computing - Grid 2002*, pp. 274–278 (2002).
4) 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).
5) 武宮博, 田中良夫, 中田秀基, 関口智嗣: Ninf-G2: 大規模Grid環境での利用に即した高機能，高性能GridRPCシステムの実装と評価, 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. 11, pp. 144–159 (2004). http://ninf.apgrid.org/.
6) JavaScript Object Notation: http://www.json.org/.
7) SQLite: http://www.sqlite.org/.
8) ChartDirector: http://www.advsofteng.com/.
9) Globus Toolkit: http://www.globus.org/.
10) 中田秀基, 田中良夫, 松岡聡, 関口智嗣: 耐故障性を重視したRPCシステムNinf - Cの設計と実装, 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. SIG 11, pp. 160–170 (2004).
11) Condor: http://www.cs.wisc.edu/condor/.