2005－HPC－104（1）
2005／10／7

高性能GridRPCアプリケーションの開発環境

小林孝嗣† 渡邊啓正† 本多弘樹†

GridRPC を用いることでプログラマはグリッドアプリケーションの作成を容易に行なうことができる．しかし，グリッドでは計算資源やネットワーク資源は不均一で変動するため，プログラマにとってアプリケーション実行中に起きた問題の原因究明や，資源を効果的に活用するアプリケーションの作成を行なうことは容易ではない．本研究では高性能 GridRPC アプリケーションの開発環境の一環として GridRPC アプリケーションの開発支援ツールの設計および実装を行なった．本ツールはアプリケーションのデバッグや性能改善の際のプログラマの手間を軽減すべく，計算資源の負荷情報や RPC 実行情報などを視覚的にプログラマに提示する．

Development Environment for High Performance GridRPC Application

Takatsugu Kobayashi,† Hiromasa Watanabe†
and Hiroki Honda†

GridRPC system allows programmers to develop a grid application easily. However, resources of a grid are heterogeneous and changeable. This makes it difficult to specify problems occurred in execution of a GridRPC application, or to develop a GridRPC application that uses resources effectively. We developed development assistant tool for GridRPC application, as a part of development environment for high performance GridRPC application. This tool visually shows programmers information about a load of resources and execution of GridRPC, in order to mitigate programmer's burden for debugging or performance improvement of GridRPC application.

1. はじめに

GridRPC[1] はグリッドにおける有力なプログラミングモデルの一つであり，GridRPC を用いることでグリッド上で動作する様々なアプリケーションを作成できることが報告されている[2)3)]．

グリッドでは計算資源やネットワーク資源は不均一で変動する．そのため高性能な GridRPC アプリケーションを作成する際に，正常な動作の妨げおよび性能低下に繋がる障害が起きる (表 1)．表中の 1〜4 の障害を特定するためには GridRPC の API によるエラーコードや RPC 実行の様子などの RPC 実行情報を調べなければならない．表中の 5〜7 の障害を特定するためには計算資源の性能情報および負荷情報といった計算資源情報を調べなければならない．しかし，これらの情報収集作業は以下の理由によりプログラマにとって非常に大きな手間となる．

- 資源の状態が実行のたびに変化する．
- 大規模な環境では情報量が増大する．
- 情報収集のためにアプリケーションのソースコードを変更しなければならない．

したがってプログラマが GridRPC アプリケーションの動作や性能に影響する障害の特定を行なうことは非常に手間がかかる．

本研究では高性能 GridRPC アプリケーションの開発環境の一環として GridRPC アプリケーションの開発支援ツールの設計および実装を行なった．本ツールは上記の問題を解決し，プログラマの手間を軽減するために以下の機能を提供する．

- RPC 実行情報の収集機能
- 計算資源情報の収集機能
- 収集した情報の可視化機能

なお本研究では GridRPC システムの一つである Ninf-G[4] を用いた GridRPC アプリケーションの開発を支援の対象としている．

本論文の構成は以下の通りである．第 2 章で提案するツールの設計について述べ，第 3 章でツールの実装について述べる．第 4 章で本ツールの利用手順と動作概要について述べ，第 5 章でツールの評価について述

† 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

表 1 アプリケーション実行中に起きうる障害の分類

1. 指定したホストが動作または存在していない.
2. サーバにリモートライブラリが存在しない.
3. ネットワーク性能が不十分である.
4. ネットワーク障害によりサーバとの通信が途絶える.
5. サーバの負荷が大きい.
6.RPC 実行中の計算資源に障害が起きる.
7. 計算資源の性能が不十分である.

べる．第 6 章で関連研究について述べ，第 7 章で結論と今後の課題について述べる．

2. ツールの設計

本章ではツールが提供する機能 (図 1 網掛け部) の設計についてそれぞれ述べる．

2.1 RPC 実行情報収集機能

GridRPC に関する処理の開始時刻や所要時間などの情報を収集する．

これらの情報を収集する処理はプログラマがアプリケーションプログラムの内容を大きく変更することなく行なえるようにしなければならない．

2.2 計算資源情報収集機能

GridRPC アプリケーションが利用した計算資源の性能情報および負荷情報を収集する．

これらの情報も RPC 実行情報の収集と同様にプログラマに余計な手間をかけさせることなく自動で収集ができるようにしなければならない．

2.3 アプリケーション実行情報可視化機能

収集した RPC 実行情報および計算資源情報から GridRPC アプリケーションのデバッグや性能改善に有用な情報を可視化し，プログラマが効率良く把握可能とする．

以下では可視化ツールで提供する機能について述べる．

2.3.1 アプリケーション実行環境の情報提示機能

実行環境の情報を把握しやすい形でプログラマに提示する．

グリッドは不均一な計算資源から構成されていることが多いため，アプリケーションがどの資源を利用したのかという情報が GridRPC アプリケーションのデバッグおよび性能改善に必要である．

2.3.2 RPC 実行状況の可視化機能

RPC の実行や同期などの処理がどのようなに行なわれていたのか，どの程度の処理時間がかかっていたのかなどの情報を把握しやすい形でプログラマに提示する．

GridRPC アプリケーションにおいては RPC 実行

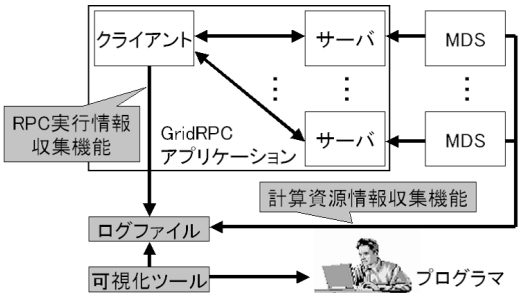

図 1 本ツールの動作概要

がアプリケーションの大部分を占めているため，RPC 実行の状況を容易に把握可能とする機能が必要である．

2.3.3 計算資源情報の提示機能

計算資源の負荷情報や性能情報といった計算資源情報を把握しやすい形でプログラマに提示する．

計算資源の性能や負荷状態が RPC に関連する処理の動作に大きく影響するため，計算資源の情報が GridRPC アプリケーションのデバッグおよび性能改善に必要である．

2.3.4 詳細情報の提示機能

RPC のハンドル初期化や同期処理などの処理それぞれに関する詳細な情報を表示する．

処理の詳細な情報が障害の特定および性能改善に必要となる．

2.3.5 フィルタ表示機能

全ての処理の中から特定の計算資源上での処理だけを表示するなど，指定した条件を満たす情報のみを表示する．

大規模な環境においては GridRPC アプリケーション実行に関する情報の量が増加するため，プログラマが必要とする情報のみに絞って表示する機能が必要である。

3. ツールの実装

3.1 RPC 実行情報収集機能

GridRPC の API の実行開始時刻および終了時刻を gettimeofday() 関数を用いて取得し，GridRPC の API からエラーコードやセッション情報を取得する．取得した情報はログファイルに出力する．そしてプリプロセッサを用いて GridRPC の API を前述の処理を付加した API に置換する．

プログラマはクライアントプログラムにヘッダファイルのインクルード文を追加するだけで RPC 実行情報の収集を行なうことができる．

また本ツールが提供する時刻計測用 API をクライアントプログラムの開始時および終了時に挿入するこ

とで，アプリケーション全体の実行時間を計測する．

3.2 計算資源情報収集機能

計算資源の性能情報および負荷情報を Globus Toolkit[5] の資源情報管理システムである MDS を利用して収集し，ログファイルに出力する API をプログラマに提供する．プログラマはこの API をクライアントプログラムに挿入することで任意の時点での計算資源情報を収集できる．

3.3 アプリケーション実行情報可視化機能

ログファイルに出力された RPC 実行情報および計算資源情報を Java を用いて実装した GUI ツールによって可視化してプログラマに提示する．以下では可視化ツールを構成する主要なコンポーネントについて述べる．

3.3.1 アプリケーション全体の情報表示部 (図 2-1)

アプリケーションの実行時間，RPC 実行回数，利用したサーバの数などのアプリケーション全体の情報を表示する．

3.3.2 実行環境の略図表示部 (図 2-2)

アプリケーションにおいて RPC の実行に利用された計算機の情報を視覚的に表示する．

本コンポーネントは性能低下やエラーの原因になっている計算資源の目処を付けやすくするために，負荷の高かった計算資源，エラーの起きた計算資源は異なった色で表示する．またこの部分において「平均 CPU 負荷が 80 %以上の計算資源」などの条件を指定することにより多数の計算資源を利用していた場合でもプログラマが必要な情報のみを表示させることができる．

3.3.3 RPC 実行状況のグラフ表示部 (図 2-3)

アプリケーションにおける RPC 実行の様子をグラフ形式で視覚的に表示する．

プログラマは本コンポーネントのグラフを見ることで，計算や通信に長時間を要した RPC などの特定を容易に行なうことができる．この部分においても実行環境の略図表示部と同様に RPC の計算時間などの条件を指定して一致するものだけを表示することができる．

3.3.4 計算資源の負荷状態のグラフ表示部 (図 2-3)

各計算資源ごとの CPU およびメモリの負荷の変動を折れ線グラフで表示する．

本コンポーネントは計算資源の負荷変動のグラフが RPC 実行状況のグラフと同じ部分に表示されるので，プログラマは容易に双方を比較することができる．

3.3.5 計算資源の詳細情報表示部 (図 2-4)

各計算資源ごとの性能情報などの詳細な情報を表示する．

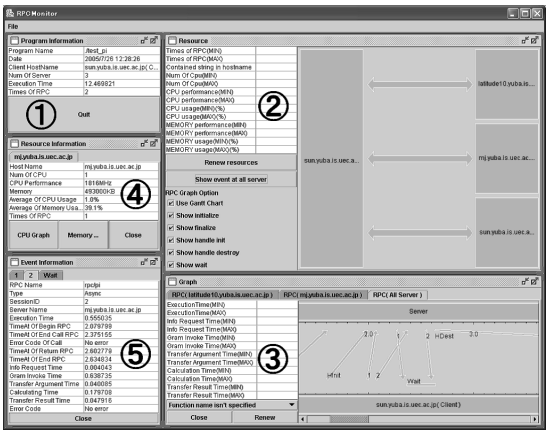

図 2 可視化ツールの動作画面

本コンポーネントは各計算資源ごとに性能情報，負荷情報，RPC 実行回数などの情報を一括してプログラマに提示する．

3.3.6 各処理の詳細情報表示部 (図 2-5)

GridRPC に関連する処理の詳細な情報を表示する．

本コンポーネントは各 GridRPC の API の処理開始時刻，所要時間，エラーコードなどを一括してプログラマに提示する．

4. ツールの利用手順と動作概要

本ツールおよび Ninf-G を用いてアプリケーション開発を行なう場合，プログラマは通常の Ninf-G アプリケーションの開発に加えて以下の作業が必要となる．

(1) クライアントプログラムに本ツールの提供するヘッダファイルのインクルード等のコードを数行追加する．
(2) 生成されたログファイルを GUI ツールで可視化する．

また可視化ツールの大まかな利用手順は以下の通りである．

(1) ログファイルを開くと GridRPC アプリケーションの実行環境の状態を表す図が表示される．
(2) 実行環境の状態を表す図から調べたい計算資源を選択するとその計算資源における負荷情報および RPC 実行状況のグラフが表示される．
(3) RPC 実行状況のグラフの各処理を表す図形をクリックするとその処理に関する詳細な情報が表示される．

GridRPC アプリケーション実行における本ツールの動作概要は以下の通りである (図 1)．

- アプリケーション開始時，終了時に MDS に問い合わせて計算資源情報を取得してログファイルに

表 2　評価環境

	CPU(MHz)	ネットワーク
ホスト A	1794	100 BASE-T
ホスト B	1816	100 BASE-T
ホスト C	1816	100 BASE-T

出力する．

- RPC の実行，同期などの処理が起きるとその内容をログファイルに出力する．
- プログラマが本ツールの提供する計算資源情報収集の API をクライアントプログラムに挿入していた場合はその部分で計算資源情報を取得してログファイルに出力する．

5. ツールの評価

5.1 評 価 環 境

評価環境は表 2 の通りである．全てのホストで OS として Redhat Linux9 を，グリッドミドルウェアとして Globus Toolkit 2.4.3 および Ninf-G 2.3 を用いた．実行時オーバヘッドの評価ではホスト A をクライアント，ホスト B をサーバとして利用し，ツールの動作検証においてはホスト C をクライアント，ホスト A およびホスト B をサーバとして利用した．

5.2 実行時オーバヘッドの評価

本評価ではツールを利用して情報を収集することによりどの程度の実行時オーバヘッドが生じているのかを計測した．RPC 実行情報を収集する際のオーバヘッドは，RPC の実行回数を変えながら情報を収集した場合としなかった場合での実行時間を比較することによって求めた．計測にはモンテカルロ法により円周率を求めるプログラムを利用した．

計算資源情報を収集する際のオーバヘッドは，単純に指定した回数計算資源情報を収集するプログラムを用いて計測した．今回の評価では 1 回の収集につき計算資源一つの情報を取得している．

RPC 実行回数に対してのアプリケーション実行時間およびアプリケーション実行時間に対する RPC 実行情報収集のオーバヘッドの割合は表 3 の通りである．計算資源情報の収集回数に対するオーバヘッドの変化は表 4 の通りである．

表 3 から実行時オーバヘッドは最大でプログラム実行時間の 2.4 %程度でありアプリケーションの性能には殆ど影響しないことが分かった．表 4 から計算資源情報収集によるオーバヘッドは 1 回につき 4 ミリ秒程度で，通常のアプリケーションの実行時間から考えれば無視しても構わない程度であると分かった．

表 3　RPC 実行情報収集によるオーバヘッド

RPC 実行回数 (回)	情報収集なし (秒)	情報収集あり (秒)	オーバヘッドの割合 (%)
1	0.41	0.42	2.4
4	2.43	2.49	2.0
16	8.46	8.53	0.8
64	34.61	34.81	0.6

表 4　計算資源情報収集によるオーバヘッド

収集回数 (回)	オーバヘッド (ミリ秒)
1	4.0
4	10.6
16	60.8
64	245.8

5.3 障害の特定におけるツールの利用

本評価では実際にツールを利用することで問題の原因究明の手間を軽減できているかを検証した．評価に用いたアプリケーションはモンテカルロ法を用いて円周率を求めるものである．今回の例では利用するサーバに対してそれぞれ RPC を 1 回ずつ実行した．

本評価では上記のプログラムを一つの計算資源の負荷を意図的に高くした上で状態で実行する．すると通常なら 10 秒程度で終わるはずのプログラムが終了までに 60 秒近くかかる．これはアプリケーションの内容からして明らかに異常な速度であると言える．ここで一つの計算資源の負荷が高かったことが異常の原因であることを突き止められるかを検証する．

本評価における原因究明の流れは以下のようになる．なおアプリケーションの動作検証に先立って本ツールが適用済みであると仮定する．

(1) 正常に動作しなかった際のログファイルを可視化ツールで開く．
(2) アプリケーションの実行環境の略図のウィンドウからある計算資源の負荷が高かったことが分かる (図 3)．
(3) 負荷の高かった計算資源上の RPC 実行状況を他の計算資源上のものと比較．
(4) 負荷の高かった計算資源上の関数ハンドル生成や RPC 実行などが他の計算資源上のものより大幅に時間がかかっていると分かる (図 4 下部)．
(5) 正常に動作した場合の RPC 実行状況と比較するとどの程度の遅延が起きていたのかが分かる (図 4)．
(6) 異常の原因は一つの計算資源の負荷が高く，そこで実行された処理が全体の足を引っ張ったためであると分かる．

以上により，偶然計算資源の負荷が高かったという

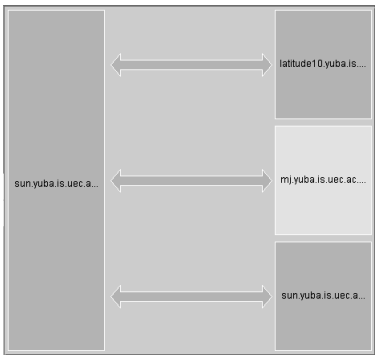

図 3 実行環境の略図

グリッド特有の再現性のない問題についても原因究明が行なえていることが確認できた．

今回の場合に従来の手法で原因を究明するには，プログラマが自分で MDS などのモニタリングシステムを検索するなどして各計算資源の負荷状況を調べることになる．しかし利用した計算資源が多い場合には，その分だけ計算資源情報の検索を行わなければならず手間が大きい．また今回のような問題は再現性がないために後から調べて原因を究明することが難しい．モニタリングシステムによっては一定時間前までの資源情報しか保持していない場合があり，その場合には GridRPC アプリケーションの実行直後にプログラマが自分で資源情報を別途保存しておかなければならない．これらのことからアプリケーションの動作検証を行なう際にあらかじめ本ツールを適用しておくことが有効であると分かる．

5.4 Grid アプリケーションの性能改善におけるツールの利用

前節で取り上げた問題点に対しては RPC にタイムアウト時間を設定するなどの対応が考えられる．タイムアウト時間は RPC が正常に動作した場合，計算資源の負荷が高くて処理時間が長かった場合を比較し，このタイムアウト時間を設定したらどうなるかなどを検討しながら設定しなければならない．そのような作業を行なう際には本ツールの RPC 実行状況のグラフ表示機能などが役に立つと考えられる．手作業で RPC の実行状況を調べようとした場合には GridRPC の API 呼び出しごとに printf() 文などをソースコードに追加しなければならないが，本ツールを利用すれば 3 行程度のコードを追加するだけで良い．

6. 関 連 研 究

MPI や OpenMP などの並列プログラムに対するデバッグ支援ツールや可視化ツールに関する研究は多数

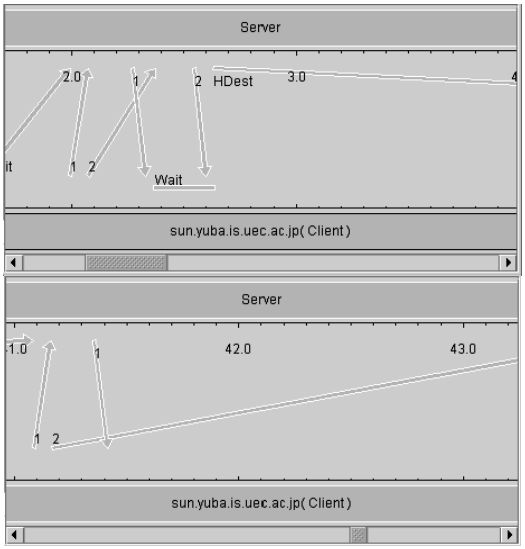

図 4 RPC 実行状況の比較

行なわれている[6)7)8)]．しかし，これら既存のツールでは以下の点を考慮していない．

- 計算資源およびネットワーク資源が不均一であり負荷状況が外乱により変動する．
- 利用する資源の数が増えて出力される情報の量が膨大になる．
- 実行環境を構成する計算資源の数が動的に変化する．

したがって既存のツールの方式をグリッドに適用した場合，実行環境の状態によるアプリケーション性能の低下や大規模環境の利用によるアプリケーション実行情報量の肥大化などのグリッド特有の問題点を解決することが出来ない．

一方，グリッドにおいてアプリケーションの開発や動作検証に利用可能なツールの研究も行なわれている．GXP[9)] では分散環境における各計算資源上でのコマンド実行を一括して行なうことができる．例えば GXP を用いて ps コマンドを実行すれば各計算資源の CPU 負荷を調べることができる．しかし，利用する計算資源の数が多い場合にはテキストでたくさんの情報が出力されるため，必要な情報をその中から選び出すのは大変である．プログラマが必要な情報のみを提示する機能や情報の量が多くても可視化するなどして分かりやすく提示する機能が必要である．

7. お わ り に

本研究では GridRPC アプリケーションの開発におけるプログラマのデバッグおよび性能改善の際の手間を軽減すべく，GridRPC アプリケーションの開発支

援ツールの設計および実装を行なった．評価実験によりツールによる実行時オーバヘッドは無視しても差し支えない程度であると分かった．また有用性の検証によってグリッド特有の再現性のない問題に対する原因究明が可能であり，同時にその結果をアプリケーションの性能改善にも利用できることを示した．

今後の課題は以下の通りである．

(1) 大規模な環境における本ツールの有効性の検証
大規模な環境において GridRPC アプリケーションを動作させたときでも，本ツールを用いることでプログラマが GridRPC アプリケーションのデバッグおよび性能改善に有用な情報を容易に得ることができるかの検証を行なう．

(2) 可視化結果を元にしたアプリケーションの修正を支援する機能の実装
アプリケーションの修正を支援する機能としては可視化ツールから得られた情報を元に，利用する計算資源，RPC のタイムアウト時間などを記述した Ninf-G 用環境設定ファイルを自動で生成する機能の実装を行なう．

(3) 現在本ツールでは対応していない Ninf-G の API への対応
現在では grpc_call_arg_stack() など，いくつかの Ninf-G の API に対して RPC 実行情報収集機能の実装を行なっていないので，それらの API に対しても実装を行なう．

参 考 文 献

1) K.Seimour, H.Nakada, S.Matsuoka, Jack Dongarra, 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).
2) 武宮博, 首藤一幸, 田中良夫, 関口智嗣: Grid 環境上における気象予報シミュレーションシステムの構築, 情報処理学会論文誌コンピューティングシステム Vol.44 SIG 11-004, pp. 155–160 (2003).
3) 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).
4) 田中良夫, 中田秀基, 朝生正人, 関口智嗣: Ninf-G2: 大規模環境での利用に即した高機能，高性能 GridRPC システム, 情報処理学会研究報告 2003-HPC-95, pp. 89–94 (2003).
5) Globus Toolkit: http://www.globus.org/.
6) 丸山真佐夫, 津邑公暁, 中島浩: データ再演法による並列プログラムデバッキング, 先進的計算基盤システムシンポジウム SACSIS2005, pp. 61–70 (2005).
7) 上島明, 小畑正貴, 金田悠紀夫: Omni OpenMP コンパイラ用並列プログラム可視化ツール, 先進的計算基盤システムシンポジウム SACSIS2005, pp. 53–60 (2005).
8) Trace Analyzer: http://www.intel.com/cd/software/products/asmo-na/eng/cluster/tanalyzer/index.htm.
9) Kenjiro Taura: Grid Explorer : A Tool for Discovering, Selecting, and Using Distributed Resources Efficiently, *Summer United Workshops on Parallel, Distributed and Cooperative Processing 2004(SWoPP2004)*, pp.235–240 (2004).