



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†

†電気通信大学 大学院情報システム学研究科

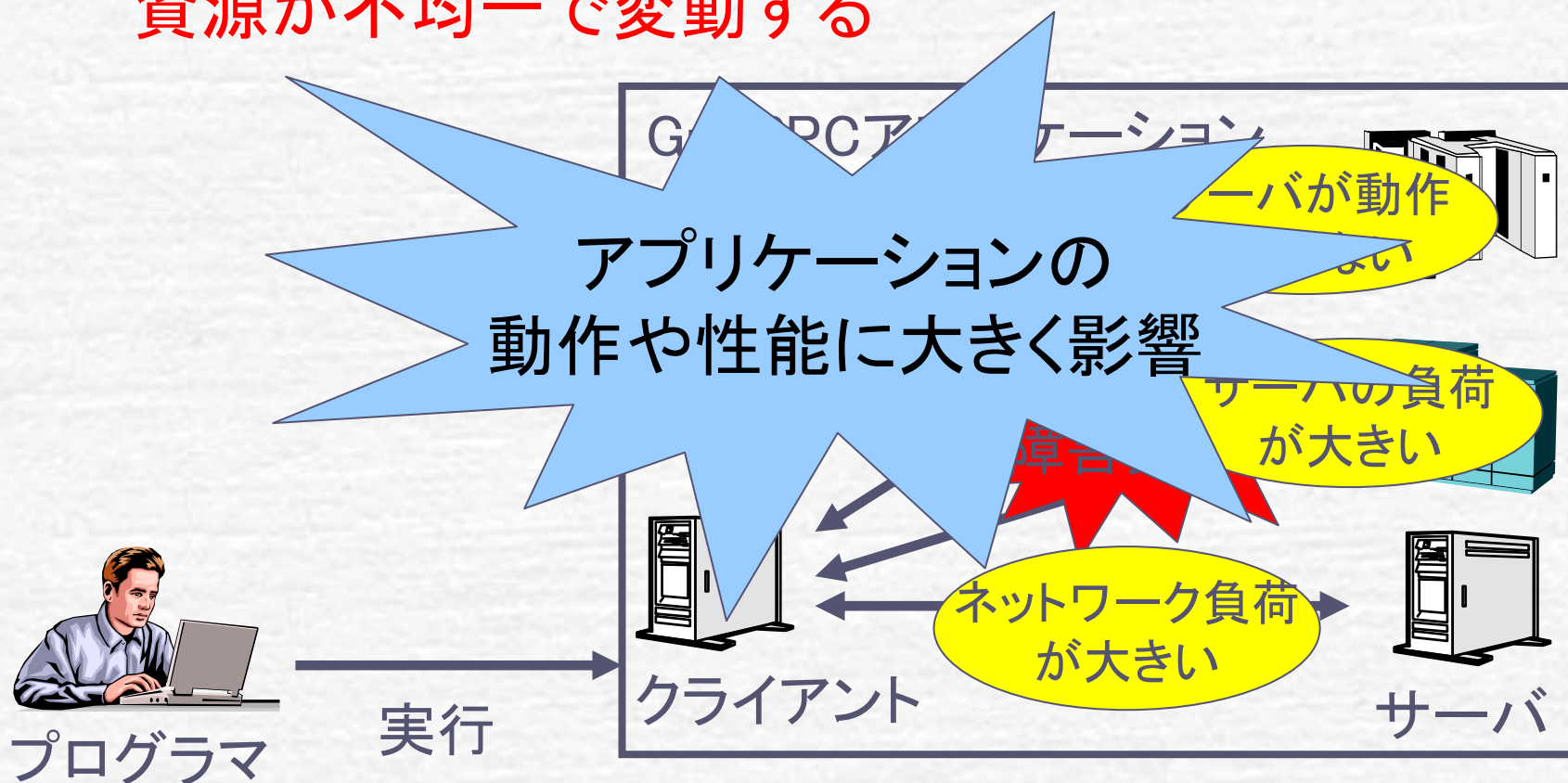


目次

1. 研究の背景
2. 研究の目的
3. ツールの設計および実装
4. ツールの有用性の検証
5. 結論
6. GridRPCシステムへの期待

GridRPCアプリケーション 開発における問題

資源が不均一で変動する



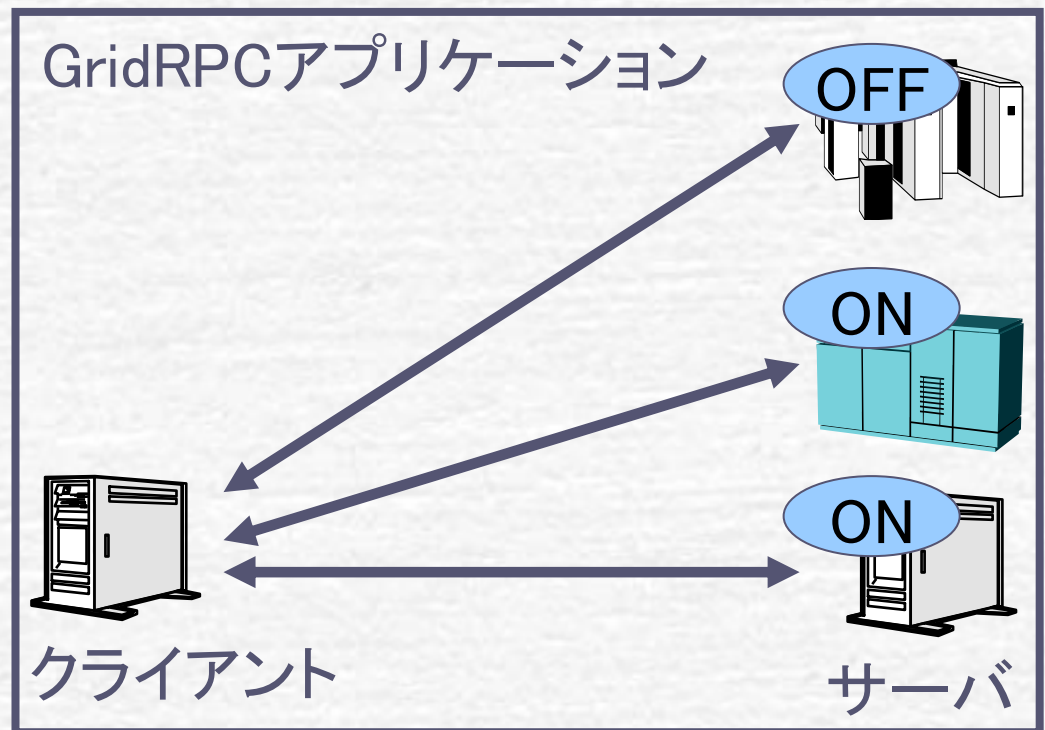
障害の特定

- 高性能なGridRPCアプリケーションの開発
 - 障害の発生が開発を困難にしている
 - 障害を考慮したプログラミング
 - アプリケーション実行中に起きた問題の原因究明
 - 障害の特定にはRPCの実行情報および計算資源の負荷情報を調べる必要がある
- ⇒様々な要因によりプログラマに手間がかかる

障害の特定

障害の特定を困難にしている要因の例

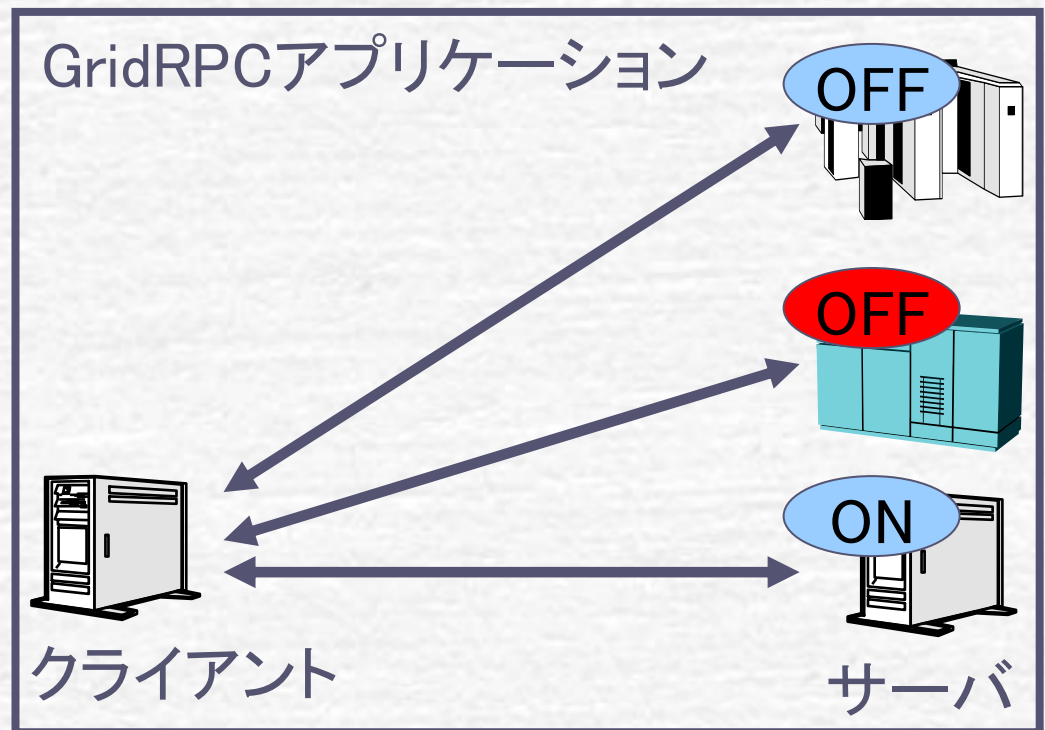
- 資源の状態が動的に変化
- 大規模環境では情報量が増大
- 情報収集のためのソースコード変更が必要



障害の特定

障害の特定を困難にしている要因の例

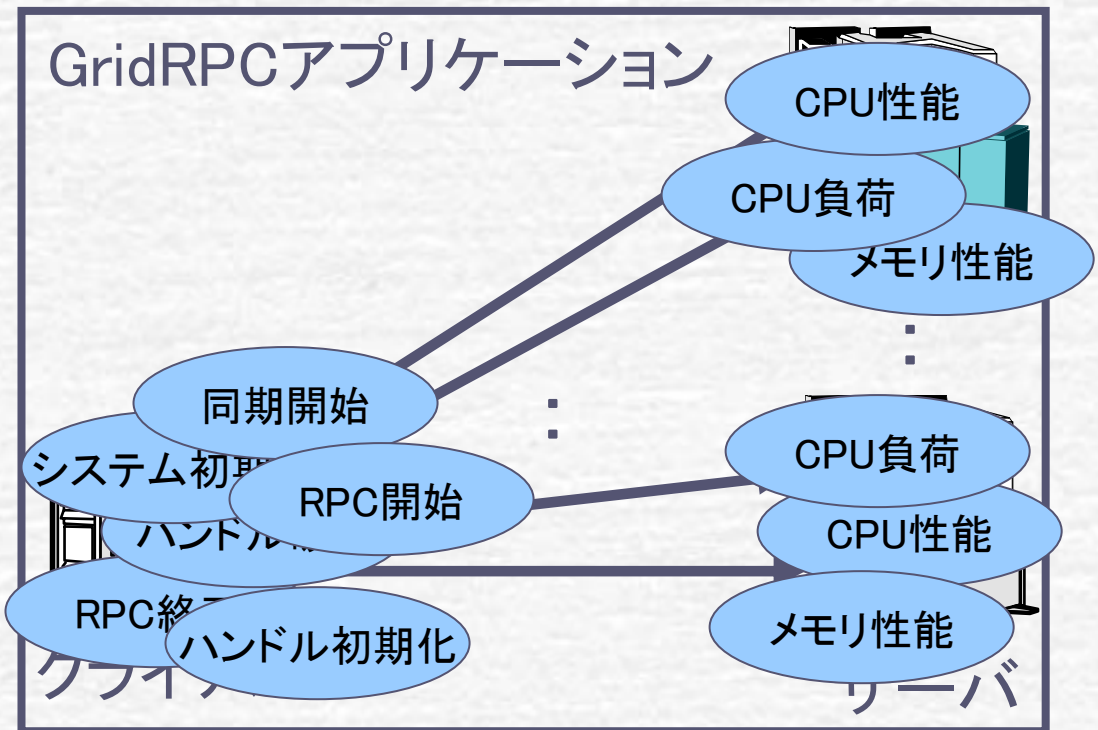
- 資源の状態が動的に変化
- 大規模環境では情報量が増大
- 情報収集のためのソースコード変更が必要



障害の特定

障害の特定を困難にしている要因の例

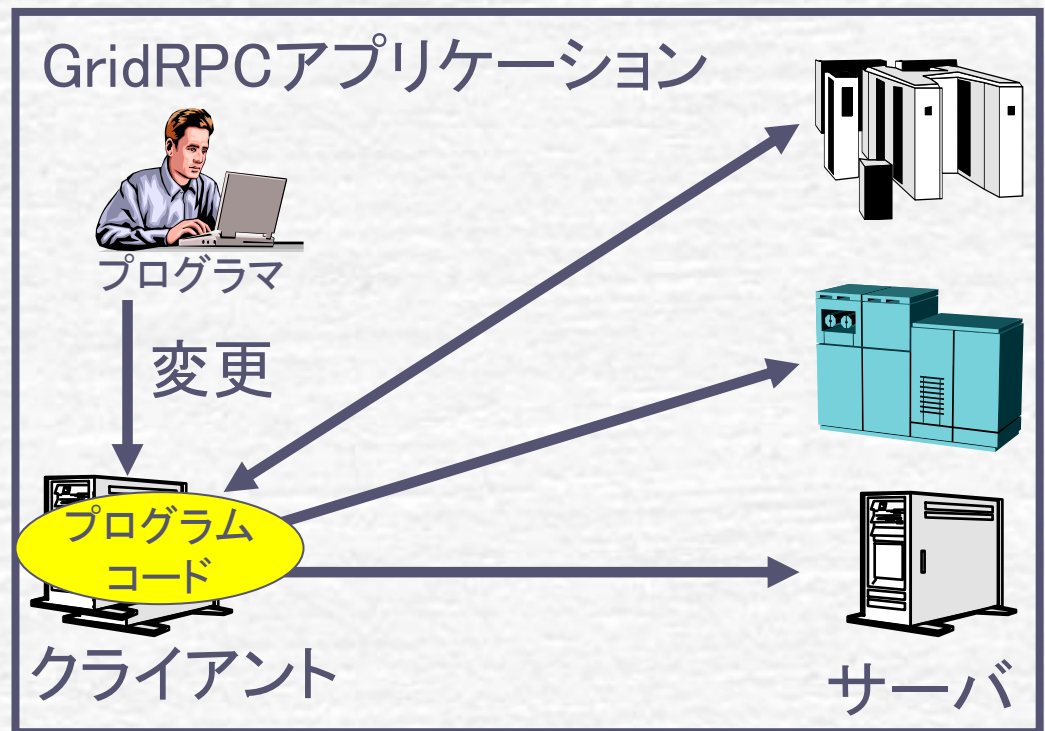
- 資源の状態が動的に変化
- 大規模環境では情報量が増大
- 情報収集のためのソースコード変更が必要



障害の特定

障害の特定を困難にしている要因の例

- 資源の状態が動的に変化
- 大規模環境では情報量が増大
- 情報収集のためのソースコード変更が必要



障害の特定

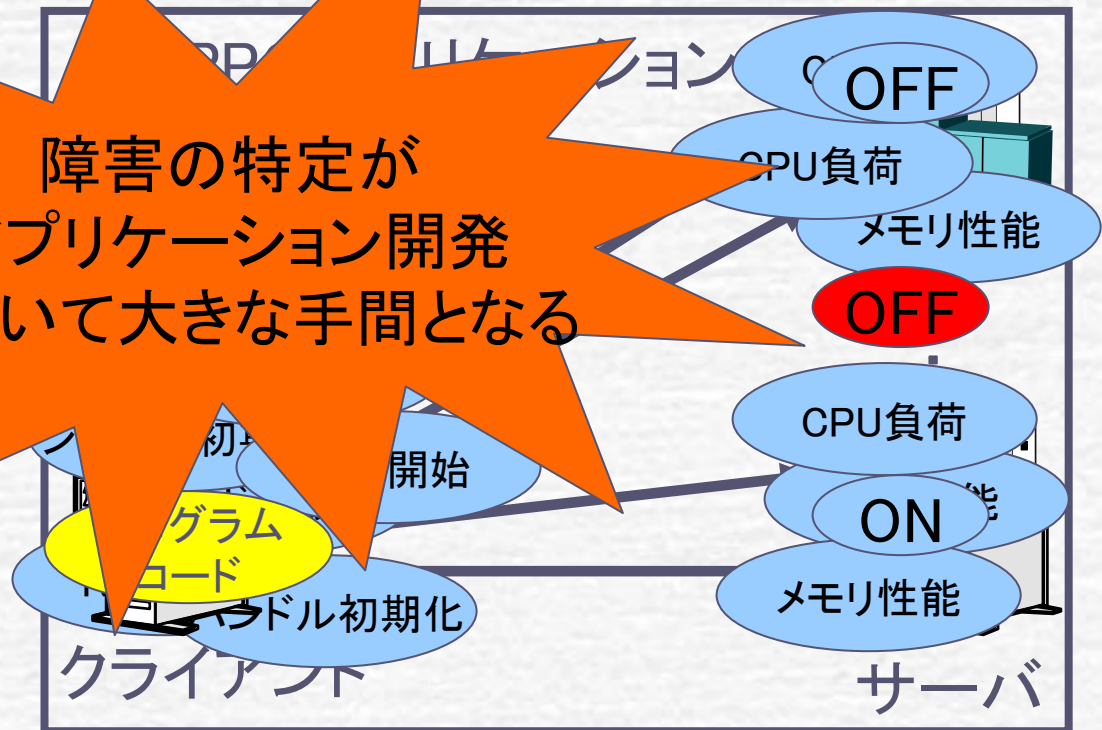
障害の特定を困難にしている要因の例

資源の状態が動的に変化する

大規模環境で情報量が増える

情報収集のためのソースコード変更が必要

障害の特定がアプリケーション開発において大きな手間となる



研究の目的

GridRPCアプリケーションのデバッグおよび性能改善を支援するツールを開発

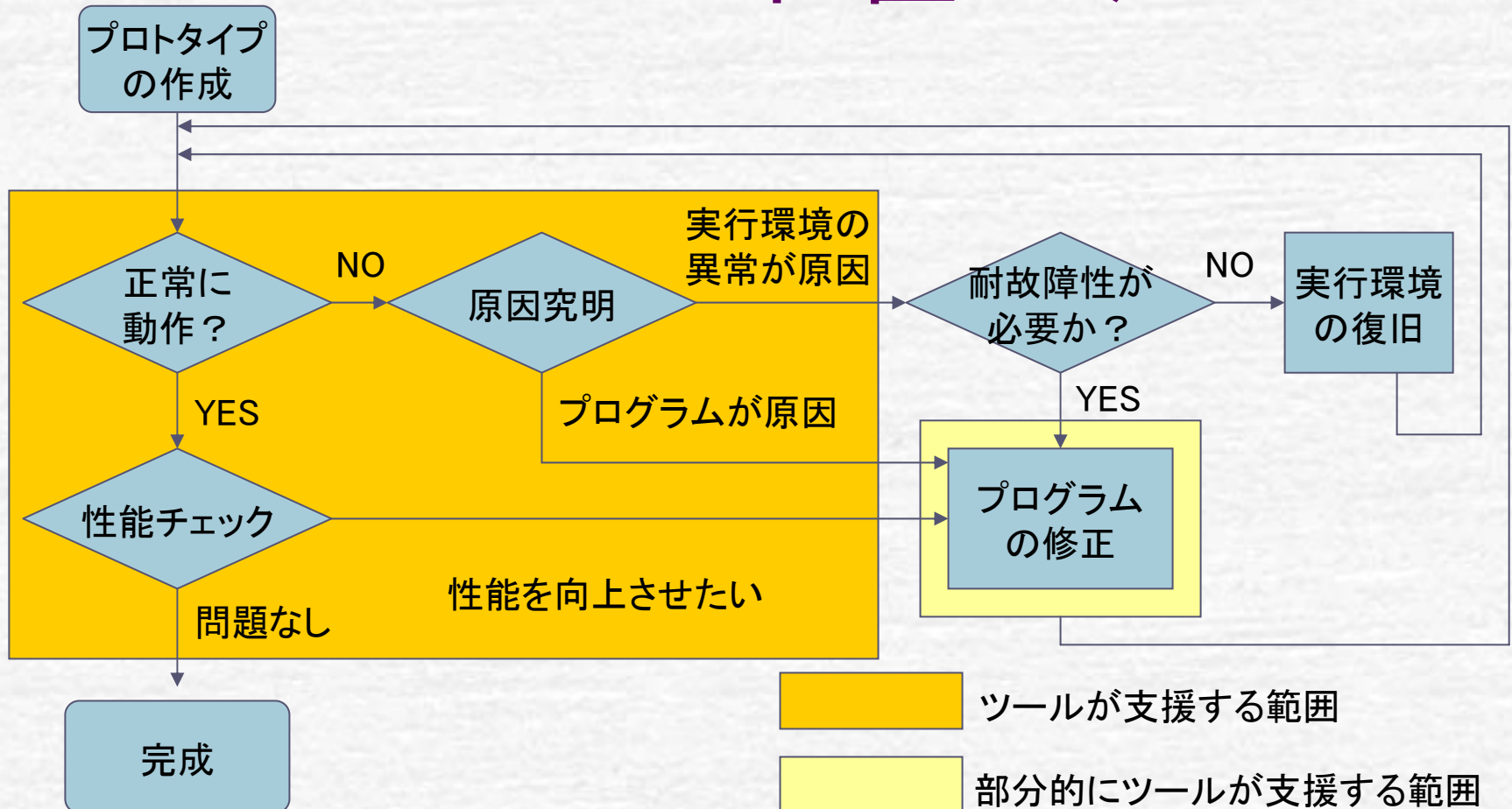
- RPC実行情報の収集
- 計算資源情報の収集
- 収集した情報の可視化

⇒プログラマの障害特定のための手間を軽減する

補足：本研究ではNinf-G[1]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

[1] 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.89-95(2003).

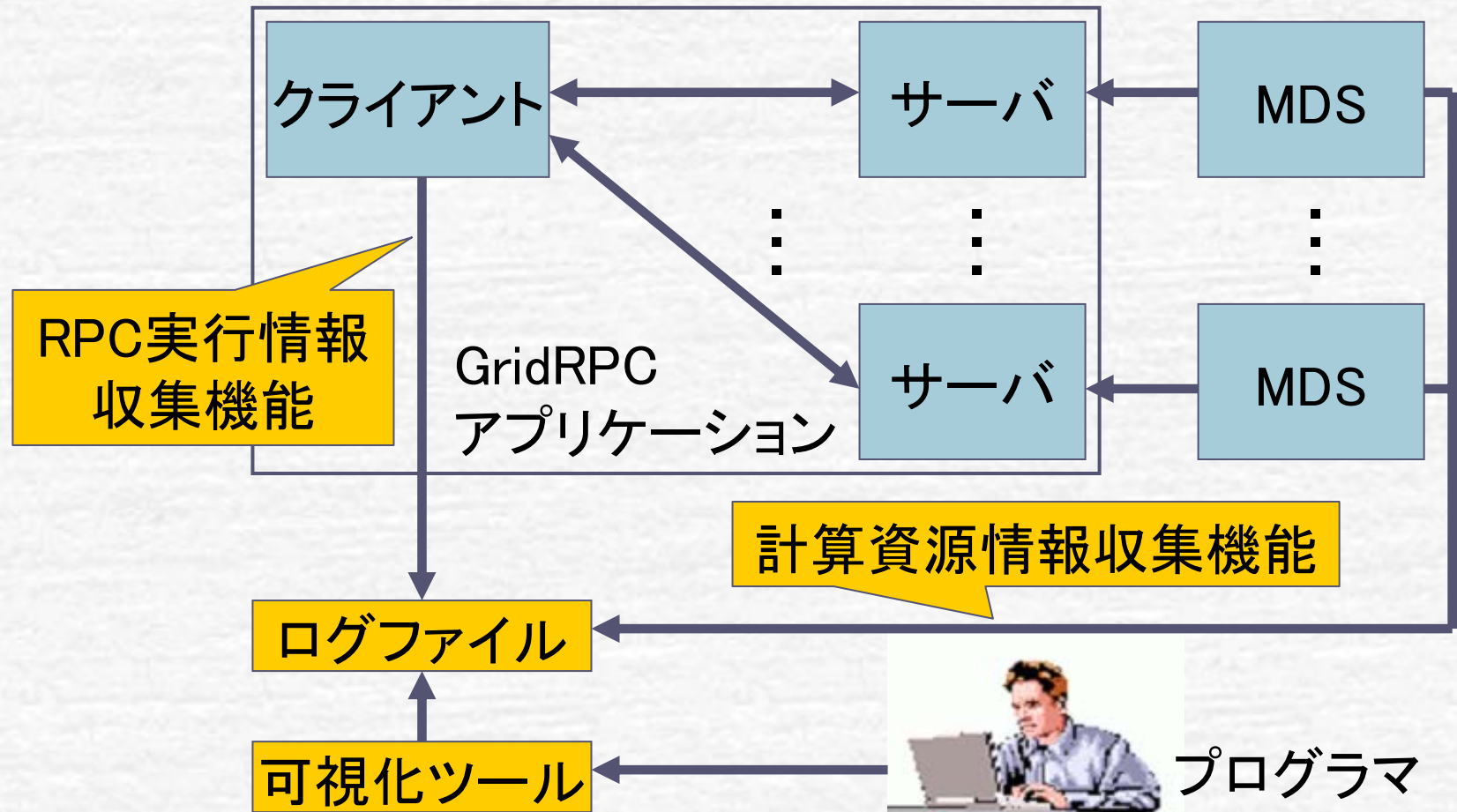
ツールの位置づけ



ツールの設計

- RPC実行情報および計算資源情報の収集機能
 - プログラマに手間をかけることなく自動的に行なう
- 収集した情報の可視化機能
 - 計算資源の状態やRPC実行状況を可視化する
 - プログラマが必要とする情報のみを提示する

ツールの動作概要



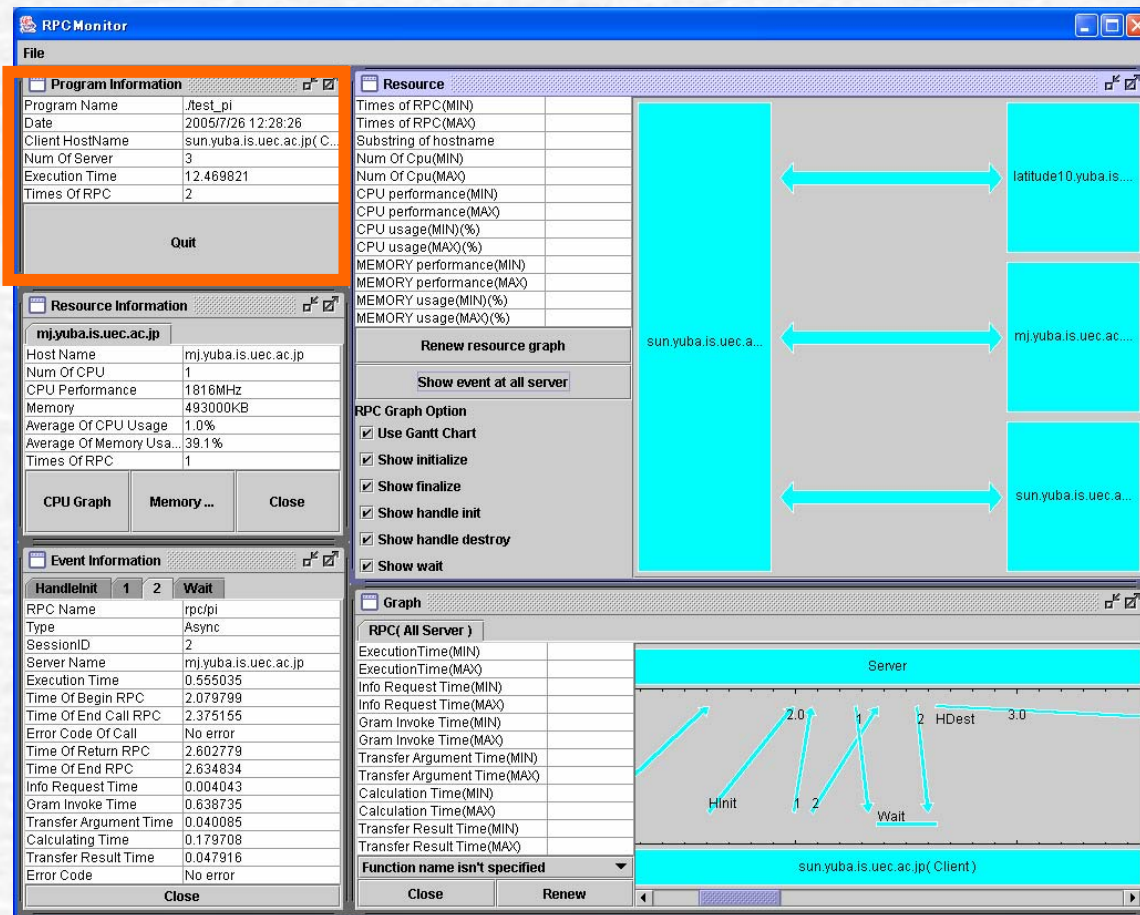
MDS: Globus Toolkit[2]における資源情報管理システム

[2] Globus Toolkit: <http://www.globus.org/>

ツールの実装

- 情報収集のためのヘッダファイルを提供
⇒インクルードすることで情報を自動収集
 - RPC実行情報
 - 計算資源情報
- 収集した情報はログファイルへ出力
- ログファイルをJavaで実装したGUIツールで可視化
- プログラマは数行のコードを追加するだけでこれらの機能を利用可能

アプリケーション全体の情報

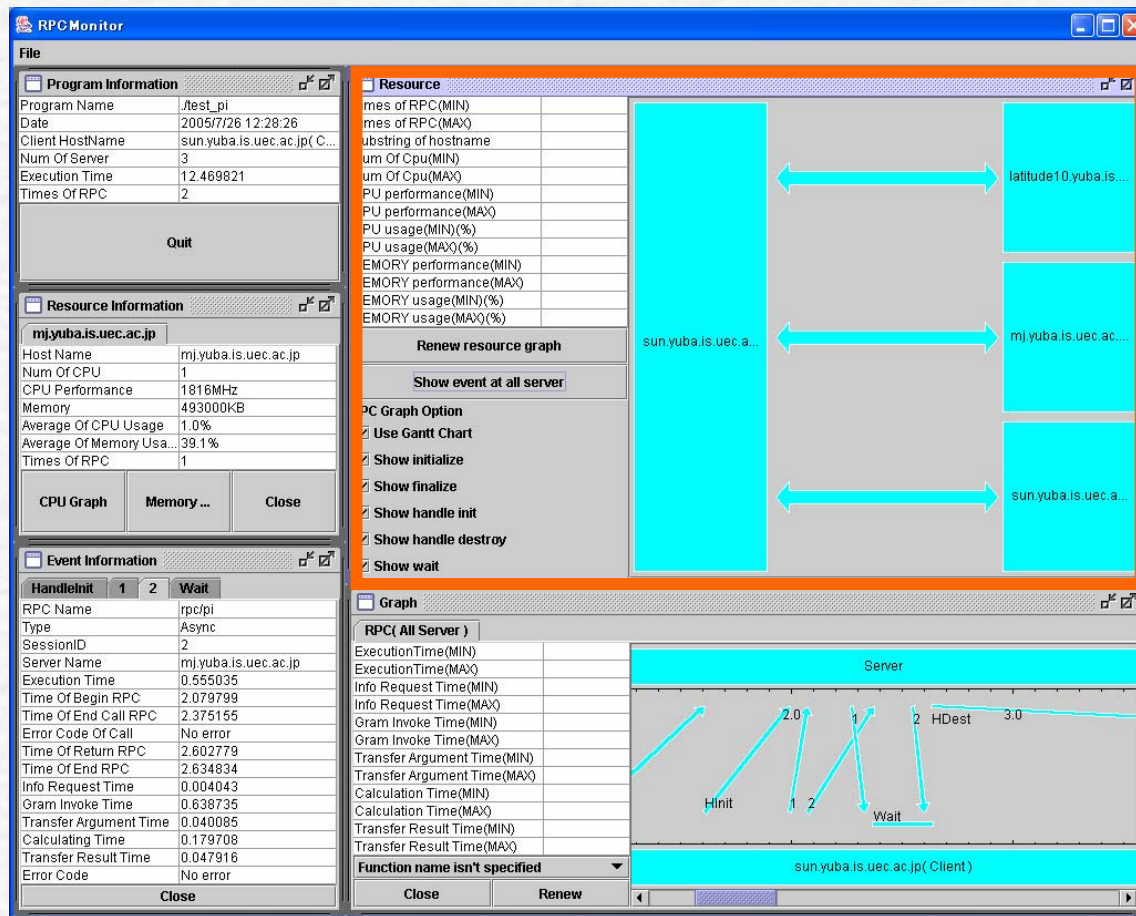


アプリケーション全体の情報

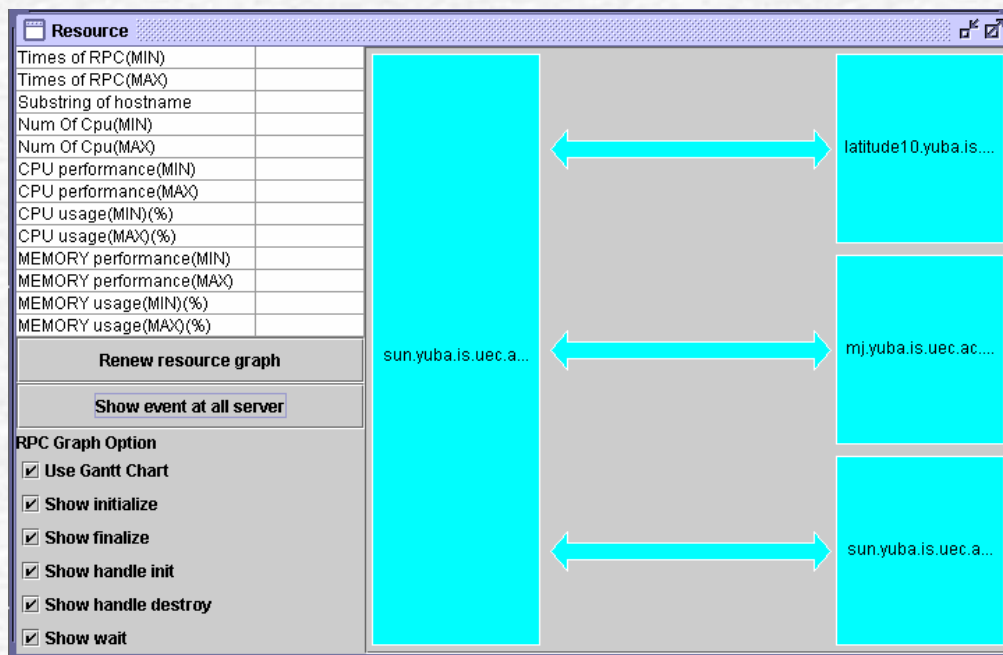
Program Information	
Program Name	./test_pi
Date	2005/7/26 12:28:26
Client HostName	sun.yuba.is.uec.ac.jp(C...
Num Of Server	3
Execution Time	12.469821
Times Of RPC	2
Quit	

- 開始時刻
- 実行時間
- RPC実行回数
- etc...

実行環境の略図

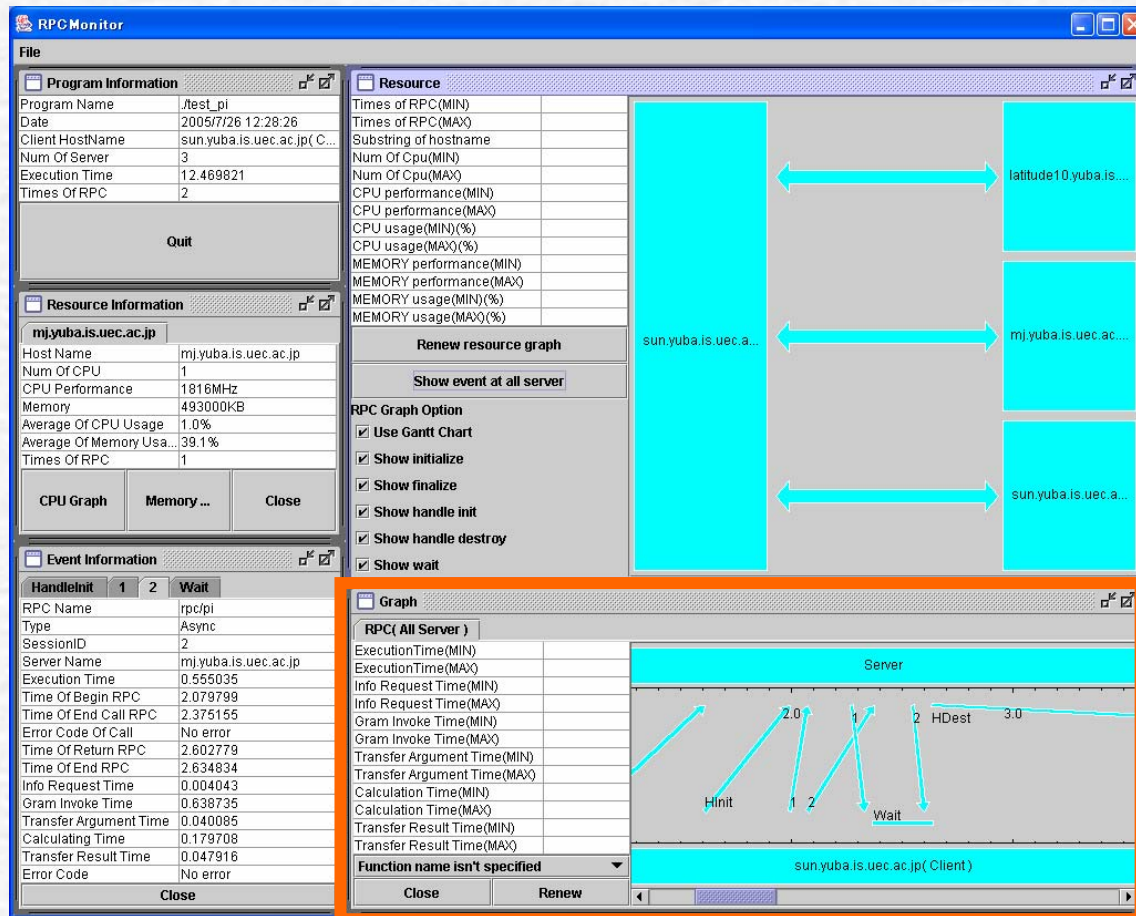


実行環境の略図

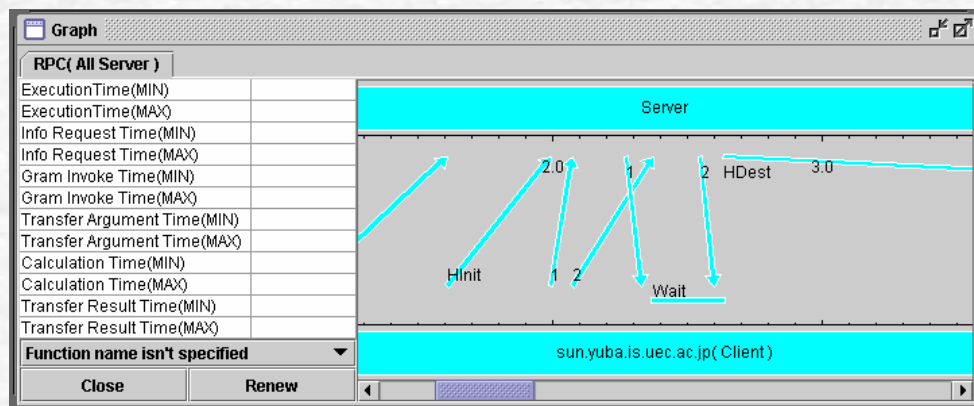


- 利用した計算資源の表示
- 高負荷な計算資源の特定
- エラーの起きた計算資源の特定
- フィルタ機能

RPC実行状況のグラフ

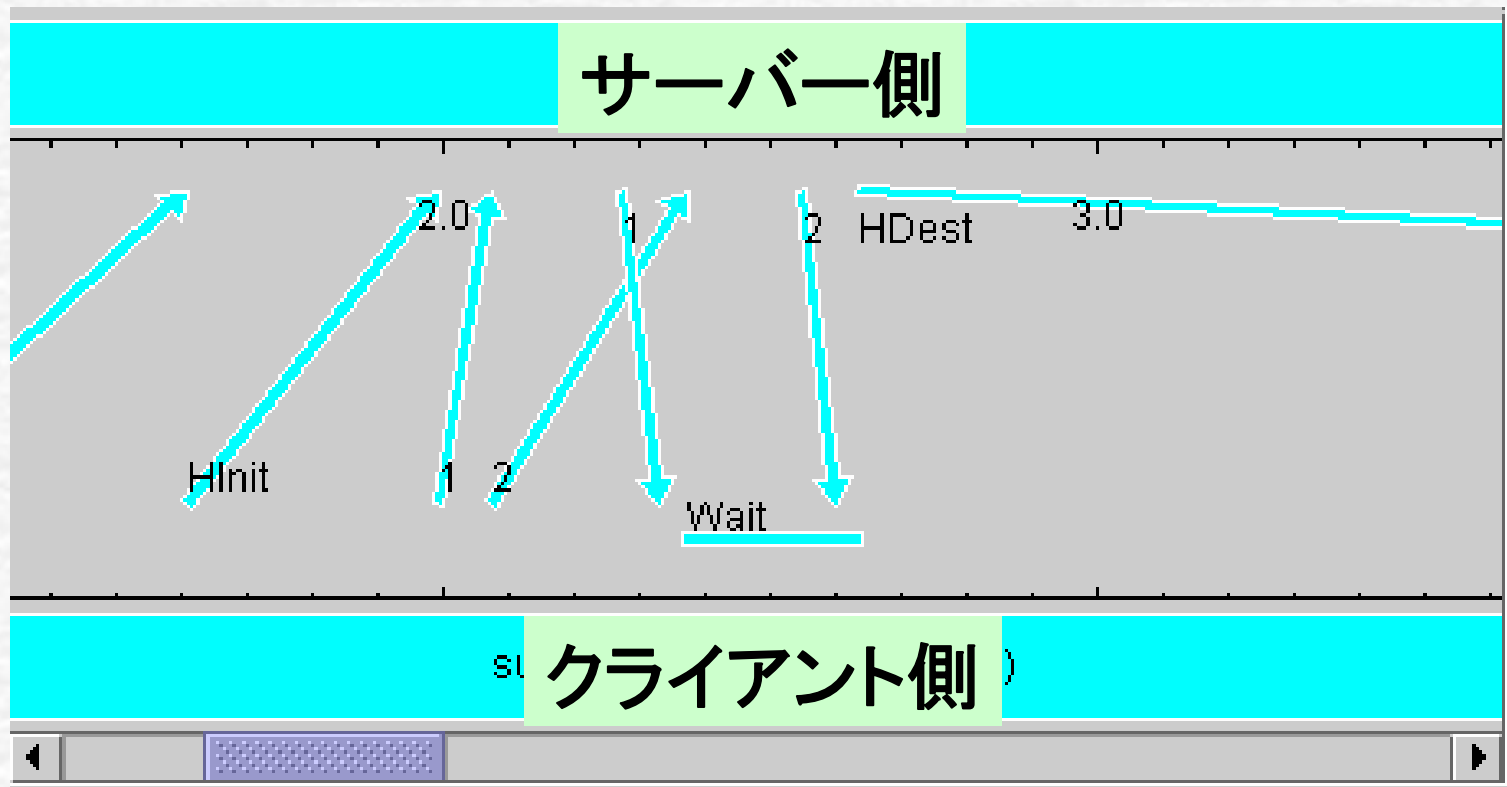


RPC実行状況のグラフ



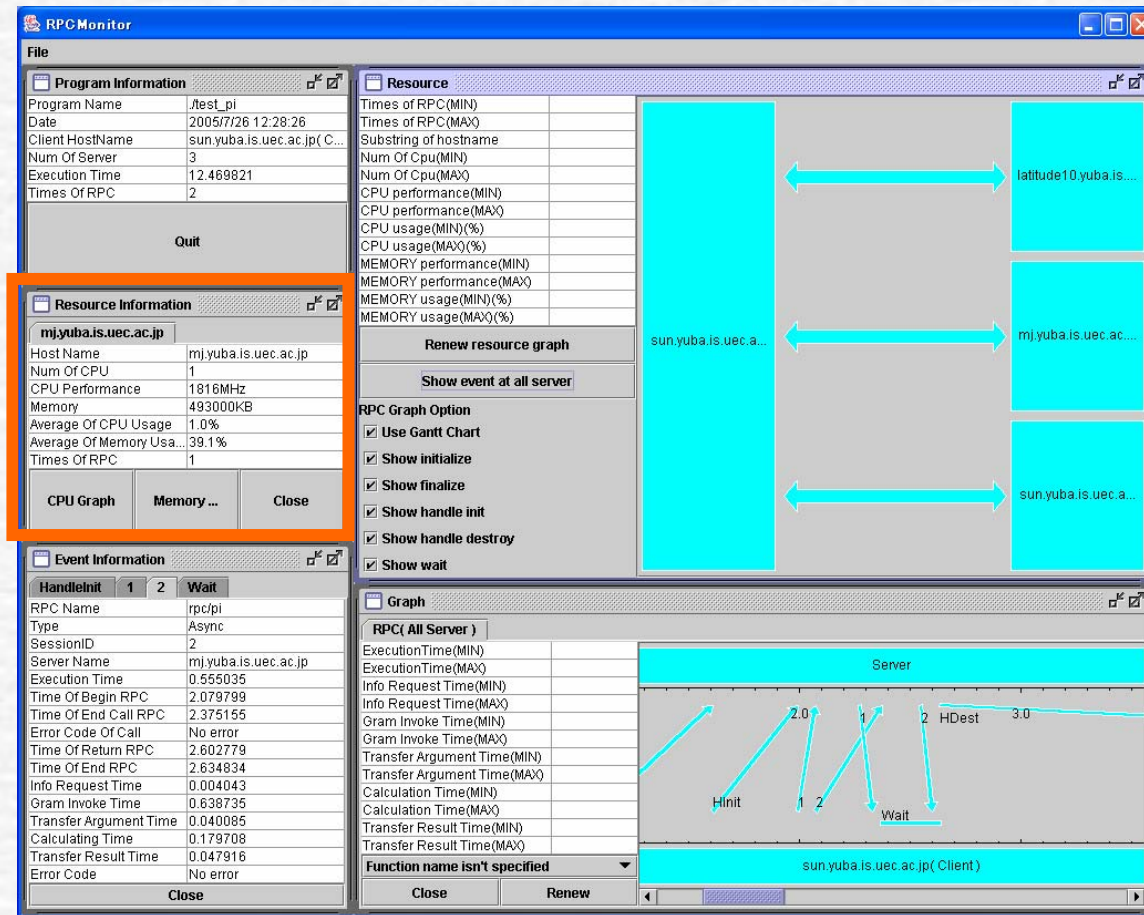
- 計算資源ごとの処理状況の比較
- 長時間を要した処理の特定
- プロセス間通信の表示
- フィルタ機能

RPC実行状況のグラフ

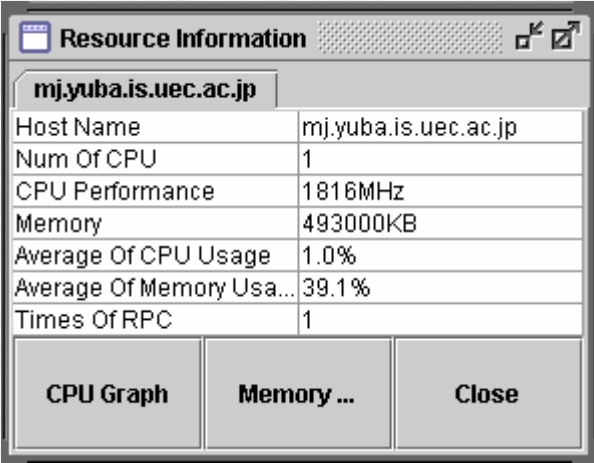


時間

計算資源の静的情報



計算資源の静的情報



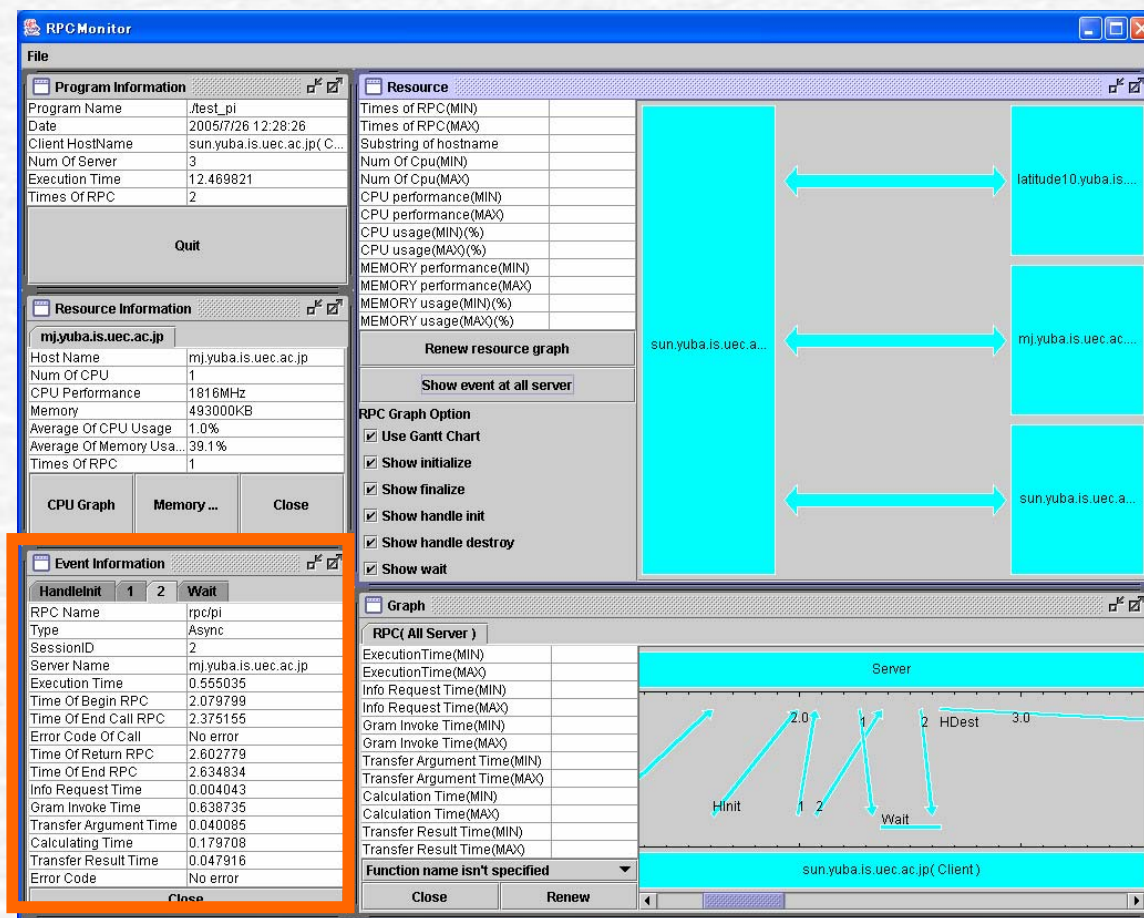
The screenshot shows a window titled 'Resource Information' with a tab for 'mj.yuba.is.uec.ac.jp'. It contains a table of system metrics and three buttons at the bottom: 'CPU Graph', 'Memory ...', and 'Close'.

mj.yuba.is.uec.ac.jp	
Host Name	mj.yuba.is.uec.ac.jp
Num Of CPU	1
CPU Performance	1816MHz
Memory	493000KB
Average Of CPU Usage	1.0%
Average Of Memory Usa...	39.1%
Times Of RPC	1

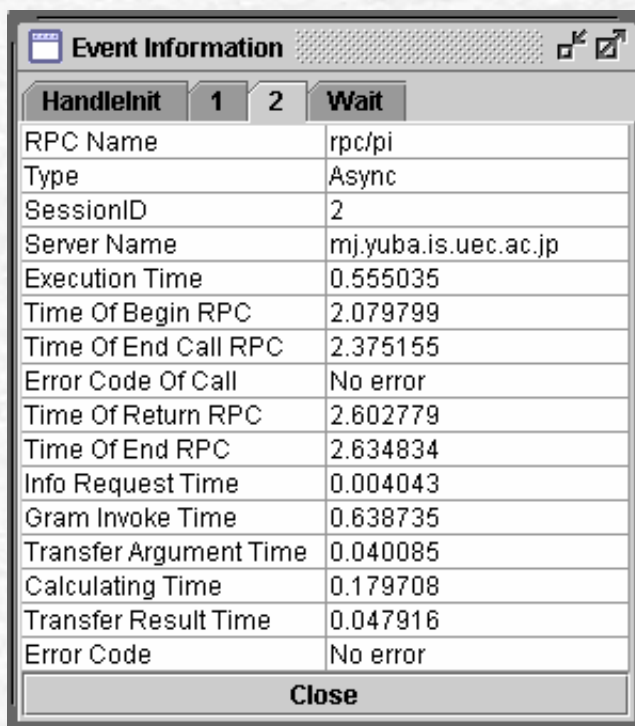
CPU Graph Memory ... Close

- 性能情報
- 平均負荷情報
- RPC実行回数
- etc...

API呼び出しの詳細情報



API呼び出しの詳細情報



The screenshot shows a dialog box titled 'Event Information'. It has a tabbed interface with 'HandleInit', '1', '2', and 'Wait' tabs. The 'Wait' tab is selected. The dialog contains a table with various RPC-related fields and their values. At the bottom, there is a 'Close' button.

HandleInit	1	2	Wait
RPC Name	rpc/pi		
Type	Async		
SessionID	2		
Server Name	mj.yuba.is.uec.ac.jp		
Execution Time	0.555035		
Time Of Begin RPC	2.079799		
Time Of End Call RPC	2.375155		
Error Code Of Call	No error		
Time Of Return RPC	2.602779		
Time Of End RPC	2.634834		
Info Request Time	0.004043		
Gram Invoke Time	0.638735		
Transfer Argument Time	0.040085		
Calculating Time	0.179708		
Transfer Result Time	0.047916		
Error Code	No error		

Close

- 処理開始時刻
- 所要時間
- エラーコード
- etc...

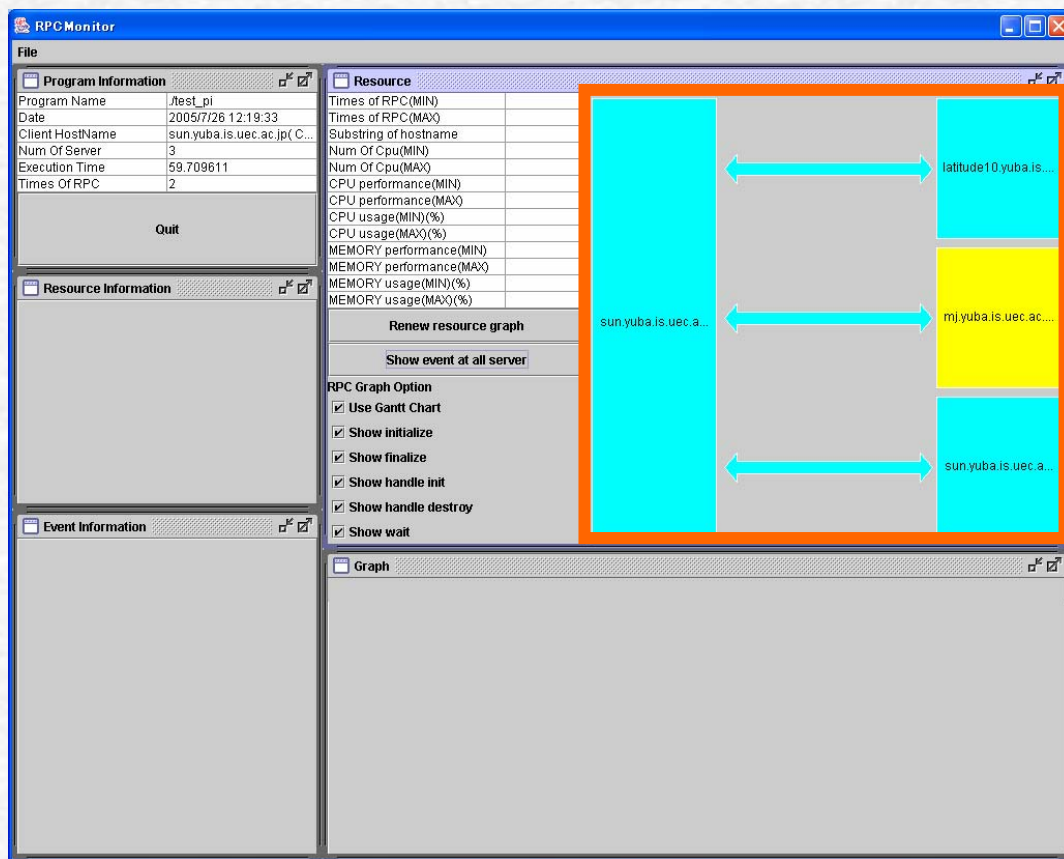
ツールの利用手順

1. RPCアプリケーションのプログラムを記述
2. 本ツールヘッダファイルをインクルード
3. コンパイルおよび実行
4. 生成されたログファイルをGUIツールにより可視化

ツールの有用性の検証

- モンテカルロ法を用いて円周率を求めるアプリケーションを利用
 - クライアントマシン1台, サーバマシン3台を利用
 - 各サーバに一回ずつRPCを実行
 - 通常なら10秒程度で終了する
 - 1つのサーバの負荷を意図的に高くして実行
 - 実行に60秒程度かかってしまった
- ⇒性能低下の原因を特定できるかを検証

ツールによる可視化

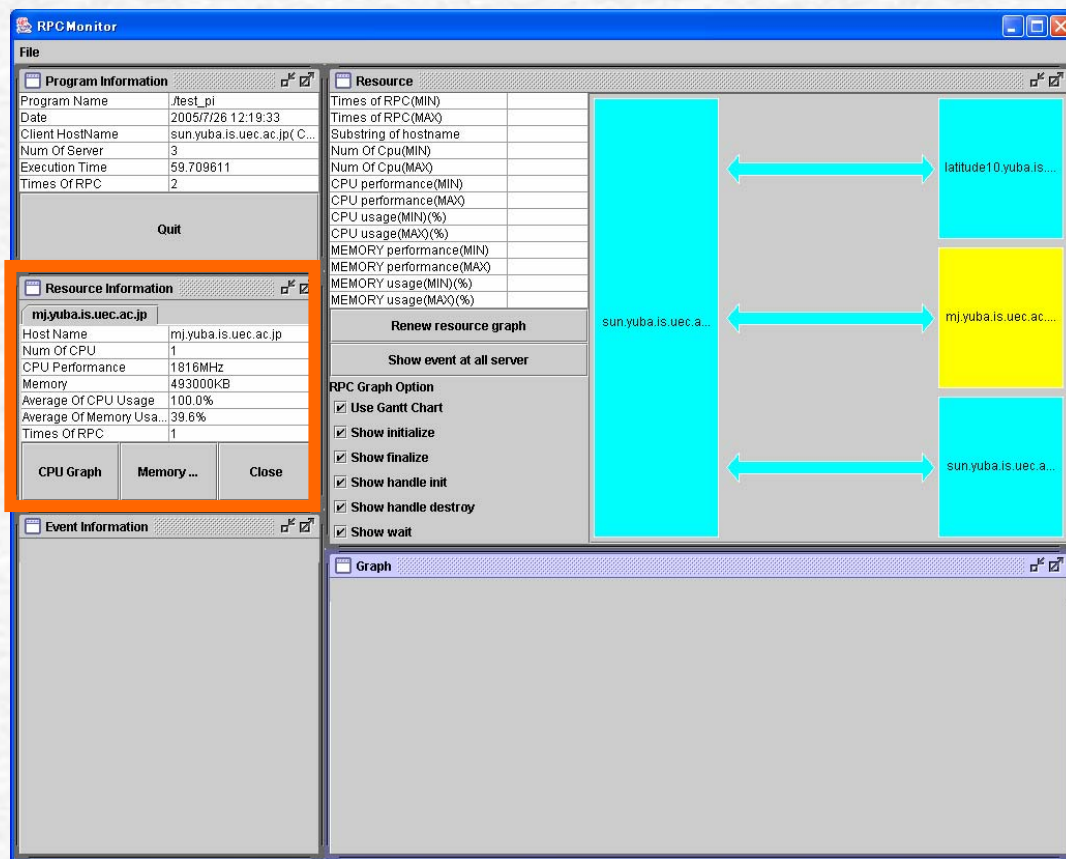


実行環境の状態
を可視化する

負荷の高い計算
資源が見つかる

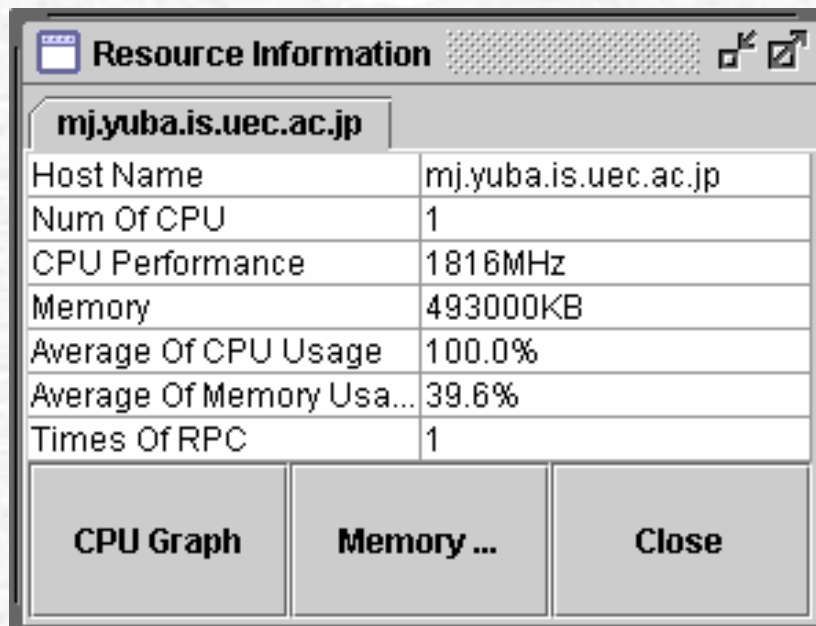
⇒これが性能低下
の原因ではない
か？

ツールによる可視化



計算資源の状態を調べる

ツールによる可視化

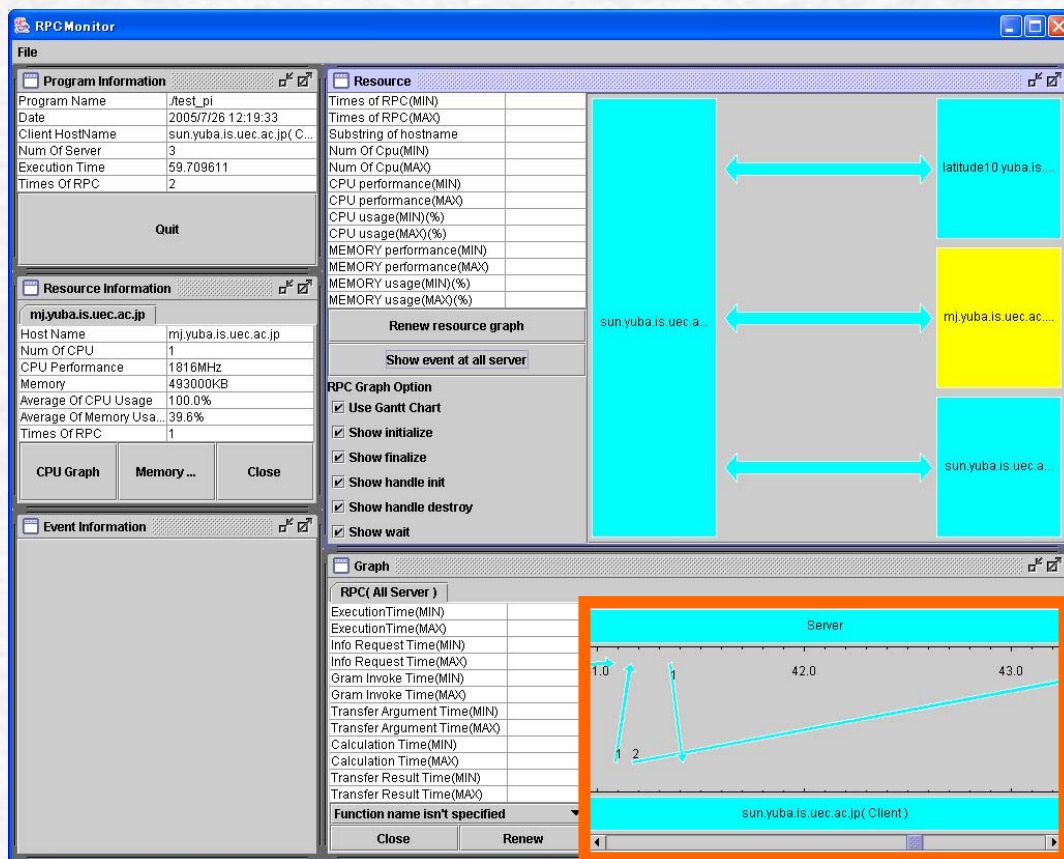


Resource Information	
mj.yuba.is.uec.ac.jp	
Host Name	mj.yuba.is.uec.ac.jp
Num Of CPU	1
CPU Performance	1816MHz
Memory	493000KB
Average Of CPU Usage	100.0%
Average Of Memory Usa...	39.6%
Times Of RPC	1
CPU Graph Memory ... Close	

計算資源の状態を調べる

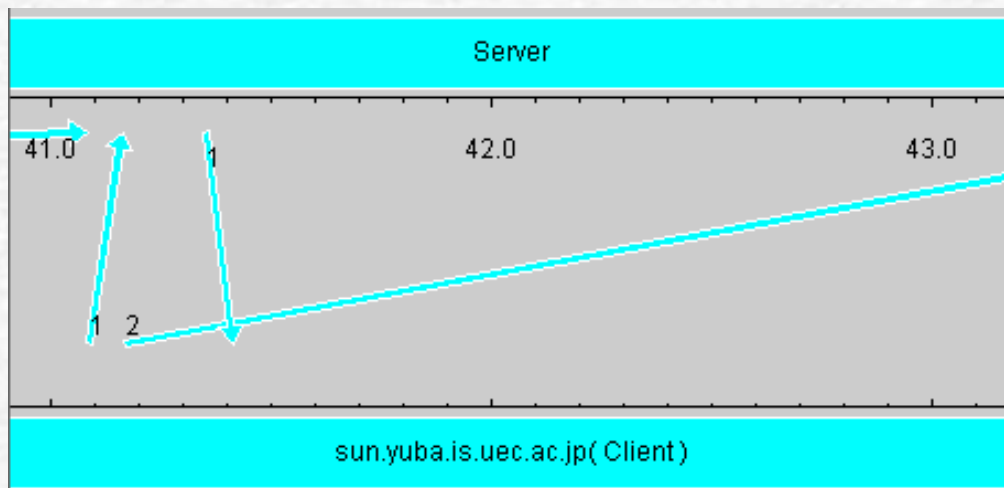
⇒CPU使用率が100%

ツールによる可視化



RPC実行状況を調べる

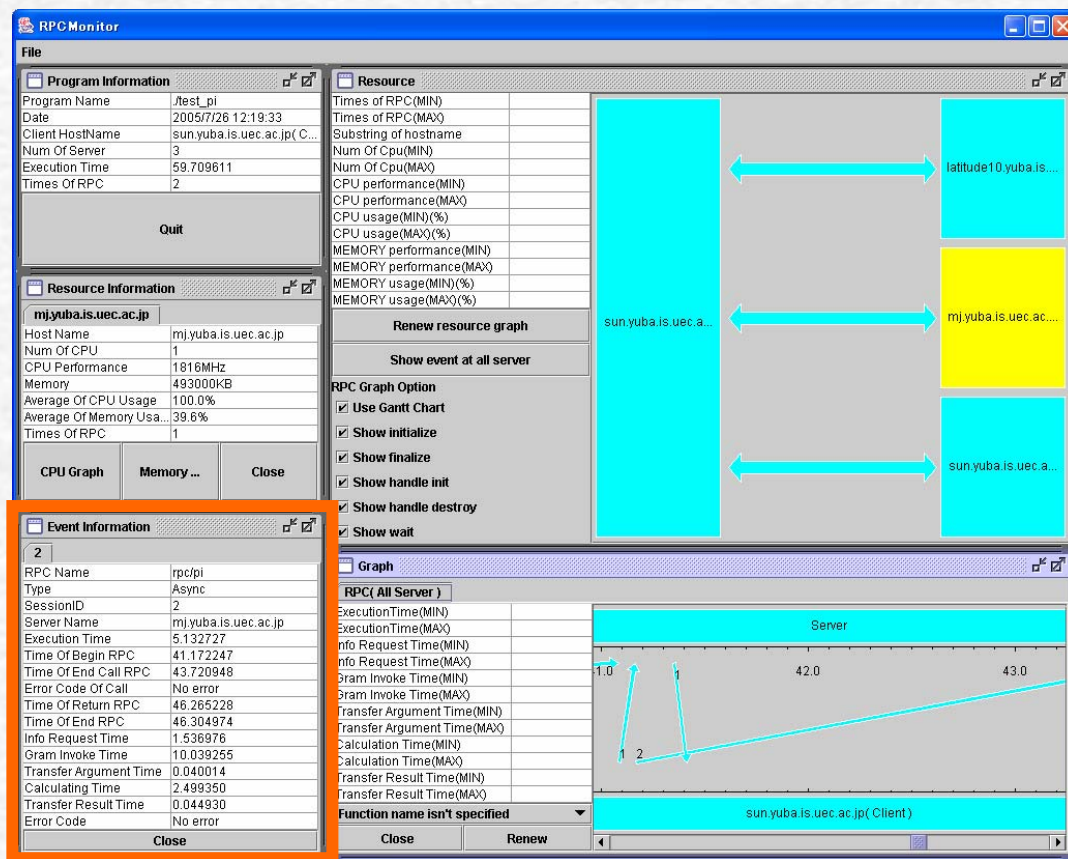
ツールによる可視化



RPC実行状況を調べる

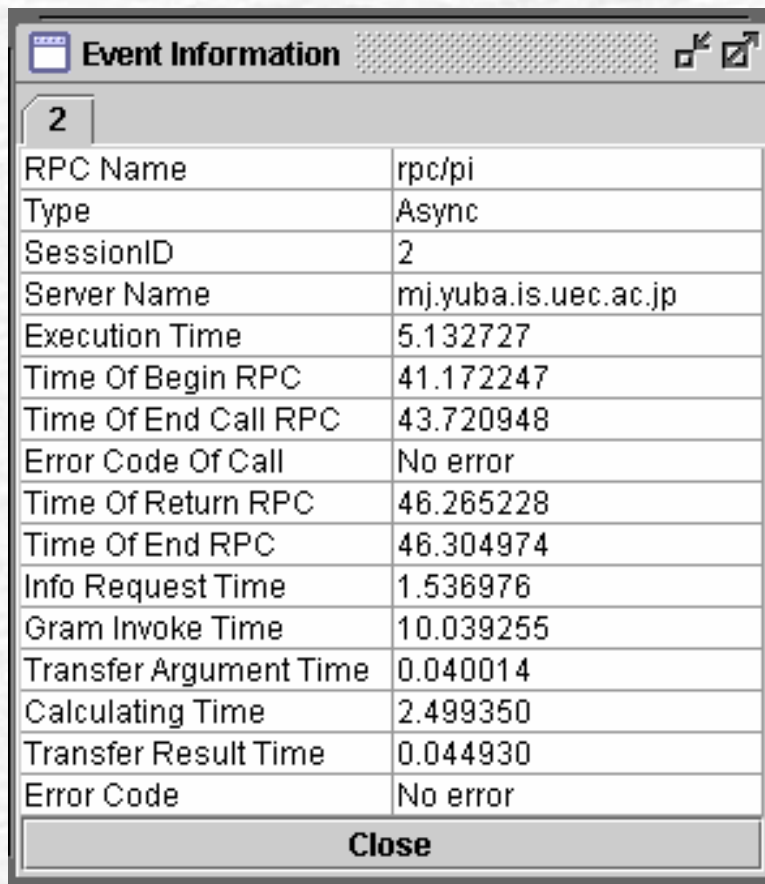
⇒長時間を要した処理が見つかる

ツールによる可視化



処理の詳細情報を調べる

ツールによる可視化



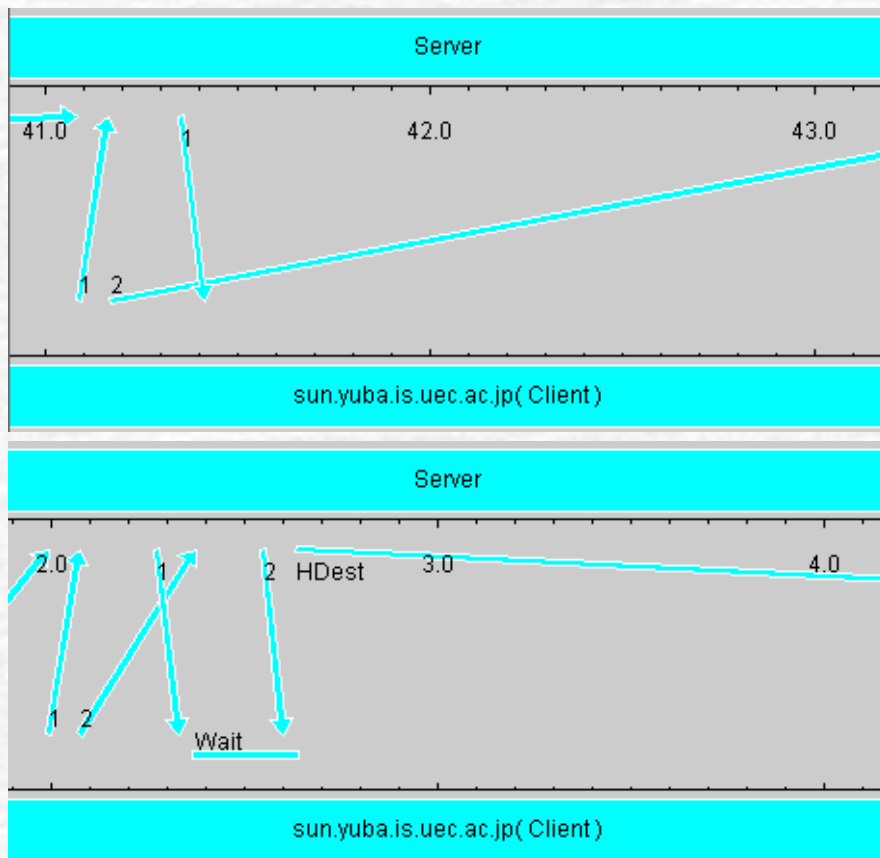
The screenshot shows a window titled 'Event Information' with a tab labeled '2'. It contains a table with 17 rows of RPC execution details. The table has two columns: the left column lists the field names, and the right column lists the corresponding values. At the bottom of the window is a 'Close' button.

RPC Name	rpc/pi
Type	Async
SessionID	2
Server Name	mj.yuba.is.uec.ac.jp
Execution Time	5.132727
Time Of Begin RPC	41.172247
Time Of End Call RPC	43.720948
Error Code Of Call	No error
Time Of Return RPC	46.265228
Time Of End RPC	46.304974
Info Request Time	1.536976
Gram Invoke Time	10.039255
Transfer Argument Time	0.040014
Calculating Time	2.499350
Transfer Result Time	0.044930
Error Code	No error

Close

処理の詳細情報を調べる
⇒高負荷な計算資源上での処理

ツールによる可視化



正常に終了した場合と比較する

⇒どの程度の性能低下が起きているかが分かる

⇒原因究明完了

性能改善へのツール利用

● RPCにタイムアウトを設定したい

⇒ 正常な場合とそうでない場合の動作内容の比較が必要

- API呼出しごとに処理時間などを出力するためのコードを挿入

- 得られた情報から動作内容を把握

⇒ 本ツールのRPC実行状況のグラフ化機能を用いれば容易に行うことが可能

結論

- GridRPCアプリケーションのデバッグおよび性能改善を支援するツールを開発した
 - RPC実行情報の収集
 - 計算資源情報の収集
 - 収集した情報の可視化
- ツール利用によってグリッド特有の再現性のない問題の原因究明も行なえることを示した
 - 高負荷な計算資源の特定

GridRPCシステムへの期待

- 自動的な資源割り当て・障害回復

- RPCの再スケジュール・再実行
- 負荷分散

- GridRPCの実行履歴にアクセスするための標準インタフェース

- 現状: 独自形式のログファイル





以下予備資料

RPC実行情報収集機能の実装

- 情報収集のためのヘッダファイルを提供
 - インクルードすることでGridRPCのAPI利用時に自動で情報を収集
- 収集した情報はログファイルへ出力
- プログラマは3行程度のコードを追加するだけでRPC実行情報収集機能を利用可能

計算資源情報収集機能の実装

- 情報収集のためのヘッダファイルを提供
- MDSから計算資源情報を取得
- アプリケーションの開始時および終了時に自動で情報を収集
 - ⇒本ツールのAPIを利用することで任意のタイミングでも収集可能
- 収集した情報はログファイルへ出力

可視化機能の実装

- Javaで実装したGUIツールで情報を可視化
 - アプリケーション全体の情報の表示
 - 実行環境の略図の表示
 - RPC実行状況のグラフの表示
 - 計算資源の静的情報の表示
 - API呼び出しの詳細情報の表示

評価項目

実行時オーバヘッドの計測

- RPC実行情報収集処理
- 計算資源情報収集処理

ツールの有用性の検証

- 性能低下の原因特定
- 性能改善におけるツール利用

評価環境

	CPU	ネットワーク
ホストA	Pentium4M 1794(MHz)	100 BASE-T
ホストB	Pentium4 1816(MHz)	100 BASE-T
ホストC	Pentium4 1816(MHz)	100 BASE-T

OS:

Redhat Linux9

グリッドミドルウェア:

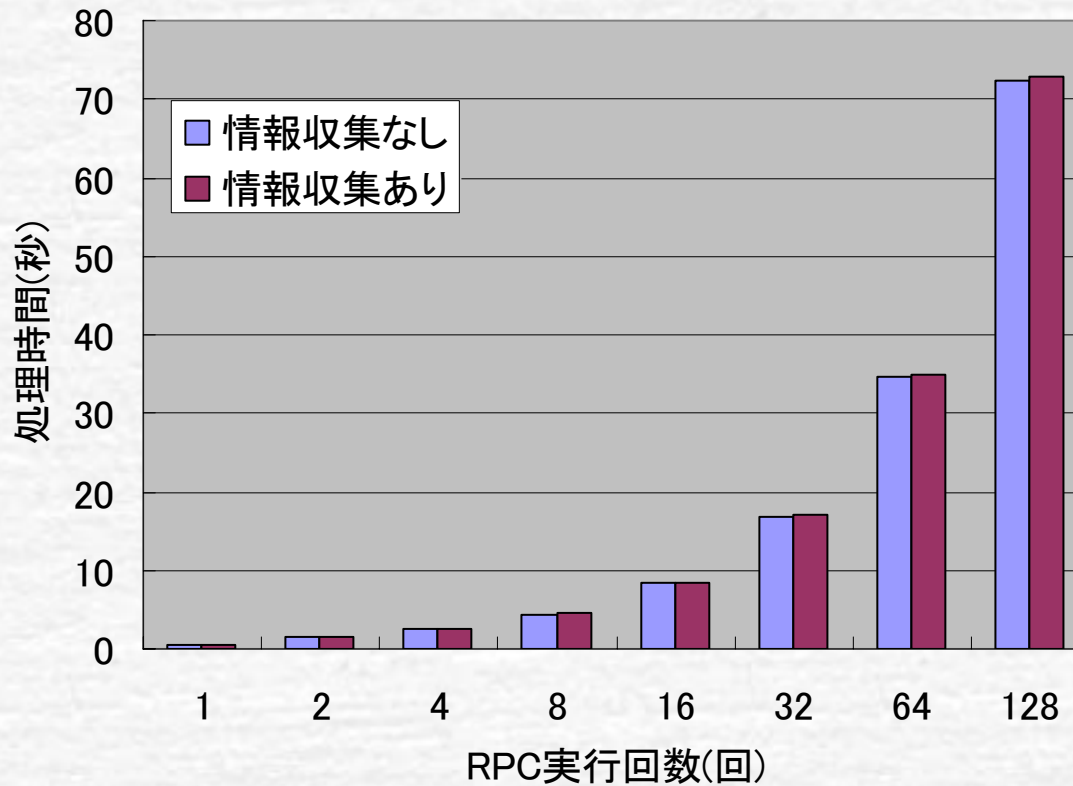
Globus Toolkit 2.4.3

Ninf-G 2.3

RPC実行情報収集による 実行時オーバヘッドの計測

- モンテカルロ法を用いて円周率を求めるアプリケーションを利用
- RPCの実行回数を変えながら実行時間を計測
- ホスト1をクライアント, ホスト2をサーバとして利用

評価結果



⇒最大で実行時間の2.4%程度

計算資源情報収集による 実行時オーバヘッドの計測

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

⇒1回の計測につき約4ミリ秒必要

関連研究

並列プログラムの可視化ツール[3][4][5]

⇒以下の点が考慮されていない

- 資源が不均一で変動する
- 規模が大きくなると情報量が膨大になる
- 実行環境の構成が動的に変化する

⇒グリッド特有の問題点の解決が困難

- 実行環境の状態によるアプリケーション性能の低下
- 大規模環境利用による情報量の肥大化

[3] 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SACSYS2005, pp61-70(2005).

[4] 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SACSYS2005, pp53-60(2005).

[5] Trace Analyzer: <http://www.intel.com/cd/software/products/asmo-na/eng/cluster/tanalyzer/index.htm>

関連研究

Grid Explorer[6]

- グリッド運用のためのツール
 - 分散環境において一括してコマンド実行が可能
 - ⇒psコマンドを実行することで
各資源の負荷情報が分かる
 - コマンド実行結果はテキストで出力
 - ⇒情報量が増えるとテキストでは把握が困難
- ⇒情報のフィルタ機能や可視化機能が必要となる

[6] Kenjiro Taura: Grid Explorer: A Tool for Discovering, Selecting, and Using Distributed Resources Efficiently, SWoPP2004, pp235-240.

今後の課題

- 大規模な環境における有効性の検証
- 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
- 現在対応していないNinf-GのAPIへの対応

ツール公開

以下のウェブページにてダウンロード可能

<http://www.yuba.is.uec.ac.jp/~kobayashi/>