Ninf システムにおけるフォールトトレランス

白砂 哲[†] 中田 秀基[††,†] 松岡 聡[†,†††]

グリッドの使用が広く広まるにつれ、グリッドにおけるフォールトトレランスは重要な研究テーマになりつつある。グリッドでは計算資源は豊富であるが、不安定であり、専用の資源ではないため、すべての段階での障害をユーザ透過に扱わなければならない。GridRPC は、グリッド環境におけるプログラミングモデルの一つである。本研究では、GridRPC における計算の過程において、フォールトトラランスのさまざまな側面に対し、それぞれを別々に対処する必要があることを示す。今回、計算時におけるフォールトトレランスを実現するために GridRPC システムである Ninf を Condor と統合した。この統合はユーザ透過であり、粒度の大きい計算に対しオーバヘッドが比較的小さいことが分かった。しかし、粒度の小さい計算に対しては、計算の起動に対してのチェックポイントライブラリ転送のコスト以外に、変則的なオーバヘッドが生じる。

Fault Tolerance on the Ninf System

Satoshi Shirasuna,[†] Hidemoto Nakada[††,†]
and Satoshi Matsuoka[†,†††]

Fault Tolerance is becoming an increasingly important research topic in the Grid as it gains widespread use. The availability of abundance of albeit unstable resources in non-dedicated environments mandate that all faults in the stages of user computation be handled in a transparent and graceful fashion. Our analysis shows that, in GridRPC, which is one of the viable programming models and systems for the Grid, variable stages during the computation exhibits various facets of fault tolerance, and as such they must be handled in a stage-by-stage basis. An experiment in integrating Ninf, a GridRPC system, with the Condor system for checkpointing to enable fault tolerance for computation shows that the integration is largely transparent to the user, and for large-grained computations, the overhead is relatively small. On the other hand, overhead for smaller-grained computations exhibits anomolous and spurious overhead, in addition to overhead incurred for transfer of the checkpointing library on each invocation, and we are conducting further investigation on its viability.

1. はじめに

近年、インターネットの普及、ネットワーク基盤の整備にともない、広域ネットワーク上にある計算資源を用いた大規模計算などが注目されている。グローバルコンピューティングと称されるこれらの試みは、ネットワークの加速、インターネットの浸透によって、ますます盛んになるものと思われる。グローバルコンピューティングで行なわれる計算は、概して長時間に渡り、多くのユーザが有限の計算資源を共有するのが一般的である。そのため、予期しないソフトウェア、ハードウェア的な原因により、計算の実行が途中で停止するなどといった障害が起こる。またさらに、物理的、ネットワーク的に離れた多数の計算機を使う場合には、障害が起きる頻度は高くなる。逐次的な実行と異なり、グローバルコンピューティングにおいては、計算の一部が異常終了により計算全体が無意味となってしまうといった場合が多々ある。従って、ユーザによりよい環境を提供するにために、グローバルコンピューティング環境を構築するシステムがフォールトトレラントであることは重要である。

Ninf[1)] は、サーバに対してタスクの実行を依頼する GridRPC システムである。API が直観的で理解しやすく、アプリケーションプログラマにとって負担が少ないなどの利点がある。しかし、現在、有効なフォールトトレランス機能が提供されておらず、ユーザが個々に対応しなければならない。

本研究では、フォールトトレラントな GridRPC を実現するため、現在の Ninf における障害を分類し、対処法を述べる。また、計算時におけるフォールトトレランスを備えたシステムとして、多数の計算機上で

† 東京工業大学
Tokyo Institute of Technology
†† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
††† 科学技術振興事業団
Japan Science and Technology Corporation

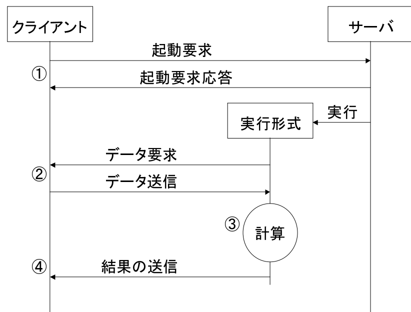

図 1　GridRPC の実行概要

のジョブのスケジューリングとチェックポイントによるフォールトトレランスを提供する Condor[2] を Ninf の計算プールとして利用できるように Ninf を拡張した。その結果、粒度の大きい計算に対し、オーバヘッドは比較的小さいことが分かった。しかし、粒度の小さい計算に対しては、チェックポイントライブラリの転送コストの他に、変則的なオーバヘッドが生じている。そのため、本システムは粒度の大きい計算に関して有効であるが、粒度の小さい計算に関してはさらなる研究が必要である。

2. GridRPC におけるフォールトトレランス

GridRPC システムにおけるフォールトトレランスを実現するため、GridRPC システムにおける障害を時系列に従い分類する。ここでは、Ninf システムの実行の仕組みにそって説明するが、多少の違いはあるが他の GridRPC システムにも当てはめられる。Ninf の実行は、**図 2** に示す大きく分けて 4 つの段階にそって行なわれる。

1. **起動**　まず、クライアントはサーバに対し計算の起動を行なう。
2. **データ通信**　その後、サーバ間とクライアント間でデータの交換を行なう。
3. **計算**　サーバによって起動されたプロセスで実際の計算を行なう。
4. **結果返送**　計算終了後、結果がクライアントに返送される。

次に、この 4 つの段階における障害の分類と対処法について述べる。

1. **起動時**　起動時に起こる障害としては、サーバが動作していない、クライアント・サーバ間のネットワークに障害がある、などが考えられる。対処法としては、サーバに正しく計算の起動を行なえなかったことを感知したクライアントが他のサーバを試みる、または一定時間待機した後にもう一度試みる、という方法がある。この機能は、クライアントに利用可能なサーバのリストを与えることにより、実現できる。クライアントは、計算の起動が失敗した場合には、次のサーバを試みる。この場合、サーバがひとつでも動いていれば、計算が実行できる。
   Netsolve[3] は、基本的にこの仕組みを用いて起動時のフォールトトレランスを実現している。利用可能なサーバのリストは、エージェントによって管理されており、クライアントはそのサーバの中から計算を投入するものを選ぶ。障害が起き失敗した場合は、リストの中から他のサーバを選び計算を再投入する。
2. **データ通信時**　データ通信時に起こる障害としては、まず、サーバからデータ要求が送られて来ないことが考えられる。これには、サーバ側の障害、ネットワークの障害が考えられる。しかし、データの要求は、計算の起動から一定時間内に来ることが仮定できるので、一定時間内にデータ要求を受け取らなかったクライアントは、障害が起きたことを検知できる。障害を検知したクライアントは、サーバへの計算の起動からやり直すことで対処する。クライアントからのデータ送信の際に起こる障害も同じく、サーバ側の障害、ネットワークの障害が考えられる。クライアントは、サーバへの計算の起動からやり直すことで対処する。
3. **計算時**　計算時には、計算プロセスの障害、計算ホストの障害が考えられる。GridRPC システムにおける計算は、長時間に渡ることが多い。よって、ここで障害がおこる可能性は高い。また、障害の起こった計算を始めからやり直すとすると、それまでの計算がすべて無駄になってしまい、損害が大きい。この障害に対処するため、チェックポイントを取ることにより計算途中の状態を保存し、障害が起こった場合にはそこから計算を再実行することが考えられる。本稿では、ここで起きる障害に注目しシステムを開発した。詳細は、次章以降で述べる。
4. **結果返送時**　結果返送時に起きる障害としては、ネットワークの障害が考えられる。ここで問題となるのは、クライアント側から、サーバがまだ計算を終了していないのか、それとも障害が起こっているのかを容易に識別できないことである。また、GridRPC による計算は長時間に渡るため、タイムアウトなどの方式は使用しづらい。適切に対処するためには、まず障害の識別が必要となる。ここで起こる障害としては、サーバ計算機自体の障害、計算プロセスの障害、ネットワークの障害がある。まず、計算プロセスが正常に動作しているかを確かめるために、サーバから定期的にプロセスの状態を報告する。ここでは、プロセスは終了するものと仮定している。プロセスの終了性は理論的に決定不可能であり、プロセスが終了することは、計算ライブラリ作成者の責任である。ク

ライアントは、サーバからの定期的な報告により、プロセスが動作しているか否かを知ることができる。サーバからの定期的な報告が来ない場合には、クライアントは、それがサーバ計算機の障害なのか、ネットワークの障害なのかを識別する。プロセスの異常終了に関しては、計算時の障害として、本稿のシステムをもちいて自動的に対処される。サーバの異常の場合には、計算の起動をし直すことで対処する。ネットワークの異常に対しては、しばらくネットワークが回復するのを待ち、回復しないようなら、計算の起動をし直すことで対処する。

次章では、計算時に起こる障害に対処するためのシステム概要について述べる。

3. Condor を用いた Ninf

前章で述べた計算時におけるフォールトトラレンスを実現するため、Ninf を改良した。本システムでは、実際の計算は Condor[2] によって管理されている計算プールでフォールトトレラントに行なう。

3.1 Condor の概要

Condor は、計算機資源を無駄なく高い稼働率で運用することを目的としたジョブスケジューリングシステムである。Condor には、実際の計算を行なう Condor Pool と、それを管理しジョブの割当を行なう Central Manager がある。クライアントがジョブを Condor に投入する際は、Central Manager に対し、適したホストの情報を問い合わせる。それに従い、クライアントは Condor Pool 内の計算機にジョブを投入する。ジョブの終了時には、Condor Pool 内の計算機は、クライアントに対し、結果を送信する。

また、Condor は専用のチェックポイントライブラリをリンクさせせることにより、チェックポイント・マイグレーションを可能にする。チェックポイントライブラリがリンクされた Condor のジョブは、チェックポイントを定期的に取り、障害に備える。障害が起こった際は、チェックポイントから計算を再実行させせることにより、それまでの計算を無駄にすることなく計算を続行できる。また、他のジョブに計算機を空け渡す必要がある場合には、ジョブを他の計算機にマイグレーションさせ計算を継続する。チェックポイントライブラリをリンクしない場合にも、制限はあるがシステムを利用できる。その場合に障害が起こった場合は、ジョブは途中からではなく最初から再実行される。また、ファイルシステムはネットワーク的に共有されていなければならない。

3.2 システムの概要

図 2 に本システムの概要を示す。本システムの動作の流れを既存の Ninf と比較する。まず、既存の Ninf は以下のように動作する。

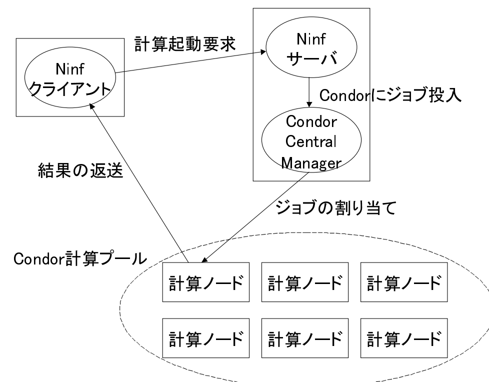

図 2 システムの概要

- Ninf クライアントは Ninf サーバに対して計算起動の要求を送る。
- 計算起動を受信した Ninf サーバは、その計算を行なうプロセスを起動する。
- 計算を終えたプロセスは、Ninf クライアントに結果を送信する。

それに対し、本システムの計算の実行は下記の過程をもって行なわれる。

- 既存の Ninf と同様に、Ninf クライアントは Ninf サーバに対して計算起動の要求を送る。
- 計算起動を受信した Ninf サーバは、その計算をジョブとして Condor に投入する。
- 計算を終えた Condor のジョブは、Ninf クライアントに結果を送信する。

Ninf クライアントにおいて、Ninf サーバに出す計算起動の要求と計算結果の受信の方法は、既存の Ninf と同一である。そのため、ユーザは既存のクライアントを作成し直すことなく、本システムを利用できる。

3.3 システムの実装

既存の Ninf に対して以下の変更を行なった。

まず、サーバ側を変更し、Condor にジョブを投入可能にした。現在の Ninf では、新しいプロセスを作成し計算を行なう。それをサーバの設定により、Condor へジョブを投入可能にした。

次に投入するジョブに Condor のチェックポイントライブラリをリンクできるように、計算の実行形式を作成するツールの変更を行なった。このツールにオプションをつけて利用することで Condor に対応した実行形式を作成できる。このチェックポイントライブラリをリンクした場合の計算には以下の性質がある。

- 計算中にチェックポイントが定期的に取られる。
- 障害が起こった場合は、計算は直前のチェックポイントより実行が続けられる。

しかし、何らかの理由でチェックポイントライブラリをリンクできない場合も考えられる。その場合、Ninf サーバはチェックポイントライブラリがリンクされていないことを自動的に判定し、計算を行なう。その場

合の計算には以下の性質がある。

- ファイルシステムがネットワーク的に共有されている必要がある。
- 計算中にチェックポイントは取られない。
- 障害が起こった場合は、計算は始めから再実行される。

3.4 システムの利点

Condor に管理されている計算プールを利用する本システムには以下の利点があげられる。

- チェックポイント、マイグレーションを利用することにより、計算途中で障害が起こった場合にも直前のチェックポイントより計算を続行できる。これにより、計算時におけるフォールトトレランスが実現される。
- 現在の Ninf では、クライアントが計算を複数の計算機に割り振りたい場合には、クライアント側で指定する必要がある。しかし、本システムでは、計算要求を受け取った Condor の Central Manager が、計算プールの中から適したホストに計算を割り当てる。計算が非同期であった場合には、同時に複数の計算機に割り当てる。よって、ユーザはすべての計算要求を単一の Ninf サーバに対して出すだけで、自動的に計算の効率かを計ることができる。これにより、プログラマの負担が軽減される。
- Ninf において計算を行なう計算プールは、研究機関のクラスタなどの何人かで共有する環境であることが考えられる。このような計算資源の場合、他の優先度の高いユーザが使う必要がある場合、計算機資源をを明け渡す必要がある。この場合には、チェックポイントを行ない計算の途中経過を保存しておき、後で計算を再開できる。また、例えばクラスタの半数を空け渡す必要がある場合には、マイグレーションを用い、計算の実行をクラスタの半分に移動させ、計算を続行できる。

4. 性能評価

既存の Ninf に比べてのオーバヘッドを測定するために、以下の性能評価を行なった。

- チェックポイントライブラリのリンクによる速度低下
- チェックポイントを取るのにかかる時間
- 既存 Ninf システムとの実行時間の比較
- 粒度が低い計算の実行時間

評価環境は、100BASE-T Ethernet でスイッチ接続された PC クラスタで、単体ノードは CPU Pentium III 600MHz、メモリ 128MB である。

4.1 チェックポイントライブラリのリンクによる速度低下

チェックポイントライブラリをリンクによる実行速度低下の有無を調べるため、NAS Parallel Benchmarks 2-serial によるベンチマークを行なった (**表 1**)。また、その際のチェックポイントライブラリののリンクによる実行ファイルのサイズの変化を調べた。

この結果によるとチェックポイントライブラリをリンクした実行ファイルに実行速度の低下は見られず、チェックポイントライブラリのリンクによる性能低下はないと言える。しかし、ライブラリをリンクする前の実行ファイルの大きさに対し、リンク後のファイルの大きさが 2MB 以上増大している。この実験自体では、影響が現れなかったが、実際の利用では、この実行ファイルはネットワークを通して Ninf サーバから Condor 計算プールへと送信されるので、その際の通信のオーバヘッドが心配される。この結果については、後述する。

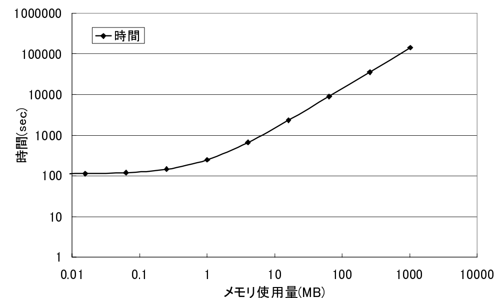

図 3 チェックポイントにかかる時間

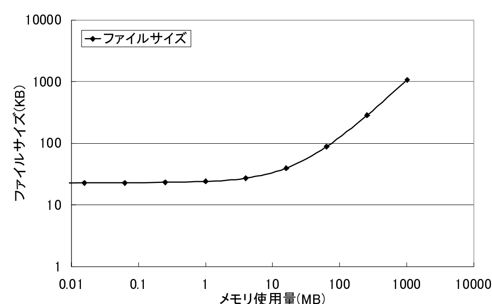

図 4 チェックポイントファイルのサイズ

4.2 チェックポイントを取るのにかかる時間

次に本システムがチェックポイントを定期的に取るオーバヘッドを見積もるため、プログラムの使用メモリサイズとチェックポイントにかかる時間を測定した (**図 3**)。また、その際のチェックポイントファイルサイズも調べた (**図 4**)。

この結果、使用しているメモリサイズが増加するにつれて、チェックポイントにかかる時間およびチェックポイントのファイルサイズが増大していることが分かる。このチェックポイントにかかる時間が本システムに影響するかの測定は、次の性能評価で行なう。

表 1　チェックポイントライブラリによるオーバーヘッド

	CG	EP	IS	LU	SP
実行時間 (チェックポイントライブラリなし) (sec)	36.42	223.55	13.64	1571.45	1390.85
ファイルサイズ (KB)	88.9	60.6	30.6	146.4	152.4
実行時間 (チェックポイントライブラリあり) (sec)	36.89	230.42	13.58	1262.10	1355.22
ファイルサイズ (KB)	2834.8	2809.1	2765.7	2887.5	2893.5
実行時間のオーバーヘッド (%)	1.29	3.07	-0.44	-19.69	-2.56

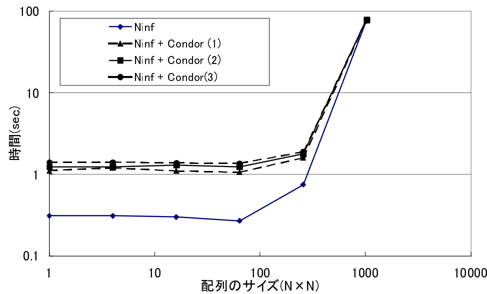

図 5　Linpack による性能比較

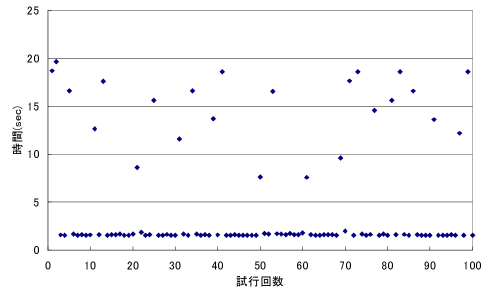

図 6　Linpack による所用時間の測定

4.3　既存 Ninf システムとの実行時間の比較

この実験では、既存の Ninf と以下の 3 種類の本システムの性能を比較する。

Ninf + Condor (1)　純粋に計算のみ行なう。

Ninf + Condor (2)　計算途中でチェックポイントを 1 回取る。

Ninf + Condor(3)　計算途中でチェックポイントを取った後、ジョブは一旦終了する。その後即座にチェックポイントより計算を再実行する。

測定プログラムには、Ninf に対応させた Linpack を用いた (**図 5**)。参考として、チェックポイントライブラリをリンクしていない場合の実行ファイルのサイズは 1531KB、リンクした場合の容量は 2224KB である。

この結果によると、データ量が少なく計算時間が短い場合には、本システムは既存の Ninf に比べて、3 倍以上のオーバヘッドがあることが分かる。しかし、データ量が多くなり計算時間が増大するにつれて、オーバヘッドの割合が減少する。配列サイズが 1024×1024 の場合には、まったく問題とならない。また、チェックポイントを取るオーバヘッドとチェックポイントより計算を再開するオーバヘッドは比較的小さいことが分かる。この実験の結果により、本システムのオーバヘッドは、実行中のものではなく、Condor の Central Manager によるジョブの割り当てと実行ジョブの転送によるものだと言える。

4.4　粒度が小さい計算の実行時間

本システムでは、まれに予想以上に実行時間に時間がかかることがある。その原因を追求するため、粒度が小さい計算の実行時間の測定を行なった。 測定プログラムには Ninf 対応させた Linpack を用い、データサイズは 128×128 である。100 回連続して測定した値を**図 6** に示す。

この結果によると、5 回に 1 度の割合で極端に実行時間が長いものが存在することが分かる。ログ解析の結果、Condor の Central Manager によって計算が計算機に割り振られるまでに時間であることが分かった。これ以上の解析は、Condor のソースが非公開であるために解析できていない。

5.　考察・今後の課題

前章での評価によると、粒度の大きい計算においては、オーバヘッドは比較的小さい。しかし、粒度の小さい計算においては、オーバヘッドが大きく、3 倍以上の速度低下が見られる。チェックポイントとチェックポイントからの再実行のコストは比較的低いので、オーバヘッドは、Condor の Central Manager によるジョブの割り振りと、Condor のライブラリがリンクされた実行ファイルの転送によるコストと考えられる。さらに、Central Manager によるジョブの割り振りにかかる時間には、原因不明のばらつきがある。理由として、Condor は本来、計算機を空き時間に利用するためのシステムであるので、インタラクティブに計算を行なうことに最適化されていない可能性がある。しかし、この現象の根本的な解決には、Condor チームとの協力が必要である。

その他にもいくつかシステムを改善すべき点をあげる。

- ネットワーク上を流れるファイルサイズを減らし、ファイル転送によるオーバヘッドを削減する。
  現在の Linux 版の Condor では、ダイナミックリンクライブラリを使用できないため、チェックポイントライブラリを静的にリンクしている。Condor でダイナミックリンクライブラリをサポートする

ことにより、ネットワーク上を流れるファイルサイズを減らし、ファイル転送によるオーバヘッドを削減することが可能になる。

- キャッシュによりファイル転送のオーバヘッドを削減する。
  科学計算の分野では、同じ関数をパラメータを変えて何度も呼び出すことが多い。現在は、計算毎に実行ファイルを計算スプールに転送しているが、計算スプール上でキャッシュとして実行ファイルを保存しておくことで、ファイル転送のオーバヘッドを削減することが可能になる。

また、本システムには、計算プロセスが Ninf クライアントとの通信中に障害が起こった場合は、計算を再開することができない。これは、Condor のチェックポイントライブラリの性質に起因する。Condor は通信中はチェックポイントを取らず、通信中に障害が起こった場合には、計算プロセスを通信前に取られたチェックポイントから再開する。しかし、Ninf クライアントは通信途中の状態であり、両者の間に不整合が生じる。これを解決するためには、通信に通し番号をつけ、Ninf クライアントを計算プロセスのチェックポイント時の状態に戻す必要がある。この機能を Ninf のライブラリに追加することで通信時におけるフォールトトレランスを実現する。

前述したように、このシステムによって実現できるフォールトトレランスは、サーバにおける計算時におけるものである。今回は、ここに重点をおいて行なったが、その他の部分のフォールトトラランスも Ninf に組み込むことが望ましい。

6. 関連研究

Ninf と同じような GridRPC システムとして、NetSolve[3) がある。NetSolve においてもフォールトトレランスを提供するフレームワークの研究が行なわれている[4)。NetSolve では、イントラサーバフォールトトレランスといわれるサーバ内で起こる障害と、インターサーバフォールトトレランスといわれるエージェントがサーバに計算要求を出す過程で起こる障害を扱っている。イントラサーバフォールトトレランスの一実装として、本研究と同様の Condor を利用したシステムも開発されている[5)。インターサーバフォールトトレランスとしては、エージェントを用いて定期的にサーバの状態を観測することによって行なっている。計算途中で障害が起こった場合には、計算を新しいサーバに投入し直す。

また、Globus[6) では、Heartbeat Monitor (HBM)[7) と呼ばれるプロセスの状態を監視するコンポーネントが存在する。HBM には、各計算機においてプロセスを監視する HBM Local Monitor (HBMLM) とそのデータを集める HBM Data Collector (HMBDC) がある。HBMLM は定期的に heatbeat メッセージを送りプロセスの状態を報告する。HBMDC は、メッセージの受信が中断したことでプロセスの異常を感知し、クライアントに通知する。このシステムは、2 章で述べた結果返送時のフォールトトレランスに適応できると考える。

7. おわりに

GridRPC 上のフォールトトレランスの実現について考察した。今回、GridRPC システムである Ninf を改良し、Condor の計算プールを用いるようにすることで、計算時におけるフォールトトレランスを実現した。評価の結果、粒度の大きい問題に対してはオーバヘッドが比較的小さく、本システムは十分実用できると言える。しかし、粒度の小さい問題に対してはオーバヘッドが大きい。この解決は今後の課題である。また、サーバ・クライアント間でのデータ通信を含めたトータル GridRPC システムのフォールトトレランスの実現も今後の研究課題である。

参考文献

1) 中田秀基, 松岡聡, 佐藤三久, 関口智嗣: Network Enabled Server System の設計, 情報処理学会システムハイパフォーマンスコンピューティング研究会, No. 2 (2000).

2) Basney, J., Livny, M. and Tannenbaum, T.: *Deploying a high throughput computing cluster*, Vol. 1, chapter 5 (1999).

3) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proc. of Supercomputing '96 Conference* (1996).

4) Plank, J. S., Casanova, H., Beck, M. and Dongarra, J.: Deploying Fault-Tolerance and Task Migration with NetSolve, *Lecture Notes in Computer Science*, Vol. 1541, pp. 418–432 (1998).

5) Casanova, H. and Dongarra, J.: Applying NetSolve's Network-Enabled Server, *IEEE Computational Science & Engineering*, Vol. 5, No. 3, pp. 57–67 (1998).

6) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, No. 2, pp. 115–128 (1997).

7) Stelling, P., Foster, I., Kesselman, C., C.Lee and von Laszewski, G.: A Fault Detection Service for Wide Area Distributed Computations, *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, pp. 268–278 (1998).