Network Enabled Server System の設計

中田 秀基[†1] 松岡 聡[†2,†3]
佐藤 三久[†4] 関口 智嗣[†1]

Network Enabled Server はグローバルコンピューティングのミドルウェアとして有望な形態である。本稿では、Network Enabled Server システムの設計において設計上の選択点を整理し、それぞれの設計方針の得失を検討する。選択点としては、クライアント/サーバ間の接続の方式、プロトコルの方式、セキュリティ機構の方式が挙げられる。われわれはこの検討を踏まえて、Network Enabled Server システム Ninf を新たに設計、実装した。本稿では、この Ninf 新システムの設計に関して、セキュリティ機構に重点をおいて紹介する。

Design issues for Network Enabled Server Systems

Hidemoto Nakada,[†1] Satoshi Matsuoka,[†2,†3] Mitsuhisa Sato[†4]
and Satoshi Sekiguchi [†1]

Network Enabled Server is considered to be a good candidate for global computing middleware. This paper clarifies design issues for Network Enabled Server systems and discusses alternatives on each issue. Issues are connection methods, protocol command representation, security methods.

We implemented new Ninf system considering with the issues. We also show the design of the system focusing on the security facility.

1. はじめに

Network Enabled Server システムは、クライアントからサーバに対してタスクの実行を依頼する、いわゆる RPC 型のグローバルコンピューティングシステムである。このような型の計算システムとしては、われわれの Ninf システムの他に Netsolve[1] がある。

Network Enabled Server(以下 NES) は、非常に直感的で理解しやすいユーザインタフェースを提供しており、アプリケーションプログラマにとって負担が少ない。また、Nimrod[2] などのパラメータサーベイを行うシステムの下位レイヤとしても使用できる。このため、Globus[3] などの、より低レベルの機構を提供するグローバルコンピューティングシステムとアプリケーションを橋渡しする、ミドルウェアとして有望な形態である。

われわれは、95 年以来 NES システムとして、Ninf システムを設計、実装し [4),5)]、これを用いた実験を通して NES の有用性を確認してきた [6)]。この実装経験により、NES システムを実装する上でのトレードオフについて知見を得た。

本稿では、NES システムの設計において問題となる点を整理し、それぞれの設計方針の得失を検討する。さらに、この検討を踏まえて、新たに設計した Ninf 新システムの概要をセキュリティ機構に重点をおいて紹介する。

以下、2 章では NES システムの一般形を示し、システム設計上の選択点を指摘する。さらに各選択点における選択肢を示し、トレードオフを議論する。3 章では新 Ninf システムの実装について示す。4 章はまとめである。

2. NES システム

一般に、NES システムは以下のコンポーネントから構成される (図 1)。

- クライアント: サーバに計算ルーチンの実行を依頼する。
- サーバ: 計算ルーチン実行依頼を受け取り、計算ルーチンを起動する。
- スケジューラ: 複数のサーバが存在する場合、どのサーバを使用するかを、後述の資源データベースから取得した情報に基いて決定する。

†1 電子技術総合研究所 Electrotechnical Laboratory
†2 東京工業大学 Tokyo Institute of Technology
†3 科学技術振興事業団
Japan Science and Technology Corporation
†4 新情報処理開発機構 Real World Computing Partnership

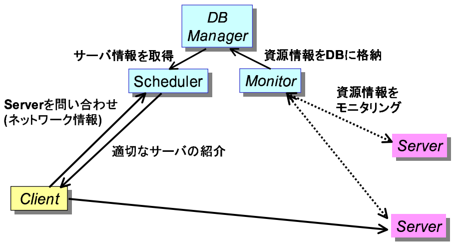

図 1　一般的な NES システムの構成

- モニタ: 計算資源、通信資源の状況を監視して、資源データベースに収める。
- 資源データベース: モニタが監視した資源の状況を保持する。

モニタは常に、サーバ、ネットワークを監視しており、その情報をデータベースに登録しておく。ユーザはクライアントから、計算の依頼を行う。クライアントはスケジューラに適切なサーバを問い合わせる。スケジューラはクライアントからの問い合わせを受けるとデータベースからサーバ情報を取得して、適切なサーバを選択する。クライアントはサーバの紹介を受け取ると、サーバに接続して計算を依頼する。さらに計算に必要なデータを送信する。計算が終了するとサーバは結果をクライアントに送信する。

2.1　NES システム設計上のポイント

NES システム設計上のポイントとしては、クライアント/サーバ間の接続の方式、通信プロトコルの方式、セキュリティ機構の方式が挙げられる。さらに、システムの設計をどの程度オープンにするかも問題になる。

2.1.1　クライアント/サーバ間の接続の方式

接続方式には 1) 常時接続にするか、2) プロキシを用いるか、の 2 つのポイントが存在する。

常時接続/必要に応じた接続:
常時接続とは、サーバ側で計算を行っている間、クライアント/サーバ間のコネクションを維持する手法である。これに対する手法として、必要なときだけ接続を行う手法が考えられる。

常時接続は、実装も容易であるし、サーバ側に故障が生じた際には、接続が切断されるため、容易に検出ができるという特長がある。

短所としては、同時に起動できるタスクの数が制約されることが挙げられる。複数のサーバに対して同時にタスクの実行を依頼する場合、複数のサーバに対する接続を維持しなければならず、その数のファイルディスクリプタが必要となる。多くのオペレーティングシステムでは、プロセスあたりのファイルディスクリプタ数が制限されているため、ひとつのクライアントから実行できるタスクの数が制約されてしまう。

また常時接続では、クライアントの運用法が制約される。サーバ側で計算を実行している間、クライアントプロセスは常に起動していなければならず、さらにクライアントサーバ間の接続が切れてはならない。NES で行われる計算は、実行時間が数日以上に及ぶ場合もあるためこの制約は問題である。

これに対して、必要に応じて接続を行う手法がある。クライアントは、サーバにリクエストを発行し、必要なデータを送信したら接続を切断する。サーバ側で計算が終了した後、クライアントと再接続して結果を通信する。再接続の方法としては、サーバ側からクライアントにコールバックする方法とクライアントが定期的にポーリングする方法が考えられる。

この場合には、上述した制約はなくなるが、システムプロトコルが複雑になる、サーバのエラー検出に別の方法が必要になる、接続のためにコストがかかるなどの問題点がある。

直接接続/プロキシを用いた接続:
クライアント/サーバ間の通信を直接行うか、プロキシを用いて中継して行うか、という選択肢である。

旧 Ninf ではすべてのトラフィックがクライアントプロキシを経由してサーバに転送されていた。これは、サーバの選択に関するスケジューラとの通信等の動作をクライアントプロキシに集約することで、クライアントプログラムを単純化するためである。

反面、プロキシを用いる手法には、通信がプロキシを経由するためそのコストがかかる、クライアントプロキシに通信が集中してボトルネックになる可能性がある、といった問題点がある。

2.1.2　通信プロトコルコマンドの方式

プロトコルコマンドとは、通信時に通信内容を表現するために用いるコマンドである。プロトコルコマンドの方式としては、大別してバイナリ形式とテキスト形式が挙げられる。

バイナリ形式には、コマンドのパーズが容易でソフトウェアに負担がかからないという特徴がある。反面、構造化しにくく拡張性に乏しい、第三者が理解しにくいという問題点がある。

テキスト形式は、構造化が容易なため拡張性を確保することができるが、解釈にパーザが必要になり、ソフトウェアの負担が大きい。

テキスト形式の場合、その具体的な形式については選択肢がいくつか考えられる。S 式は Lisp で用いられてきた構造データ表現である。パーズが比較的容易であるという特徴があるが、データ構造表現として考えると、構造を束縛するスキーマを定義する表現がない、という短所がある。

XML(Extensible Markup Language) は、最近広く用いられるようになった構造データを表現する手法である。S 式と比較するとパーズの負荷は大きいが、データ構造のスキーマを定義する DTD という機構が存在する点が特長である。

2.1.3 セキュリティ機構の方式

セキュリティに関しては、使用環境に応じてさまざまな選択肢がある。

運用環境がローカルな範囲に限られ、参加者が完全に信頼できる環境であるならば、セキュリティ機構は不要であるし、キャンパス内などの閉じた環境であるならば、クライアントの IP アドレスに対して制約することで十分である。また、グローバルな環境で運用するならば、悪意のあるユーザの存在を想定しなければならず、暗号化技術に基いた認証機構が必要になる。

暗号化の手法としては、対象鍵暗号を用いる Kerberos、公開鍵暗号を用いる SSL などが候補となる。

2.1.4 システムのオープン性

システムをどの程度オープンに設計するかは、大きなデザインチョイスである。グローバルコンピューティングシステムの一部として用いることのできる、汎用サブシステムがいくつか提案、実装されている。Globus はグローバルコンピューティングのツールキットで、通信レイヤ、セキュリティレイヤ、ディレクトリサービス、モニタシステムなどを提供する。

NWS(Network Weather service[7]) は、通信資源、計算資源をモニタするシステムであるが、高度な補完機能を持ち、ネットワークの細動に左右されない安定した計測と予測を提供する。

LDAP はディレクトリサービスを行う標準的なインターフェイスで、資源データベースとして使用することができる。LDAP をアクセスするライブラリは、C 言語、Java 言語ともに整備されているため、実装は比較的容易である。Globus では MDS(Metacomputing Directory Service) を LDAP サーバで実装している。

これらの汎用サブシステムを使用し、いわばオープンにシステムを設計することで、高度な機能が容易に実現できる。反面、これらの汎用サブシステムを使用することによって生じる制約もある。これらの汎用サブシステムは、比較的大きなシステムでありインストールにも時間がかかる上、ディスク上のフットプリントも大きい。また、サポートしているアーキテクチャや使用言語も限定されている。汎用サブシステムを用いることは、これらの制約を NES システムにも持ち込むことになる。

3. 新システムの実装

3.1 新システムのコンセプト

前節で述べた論点に留意して、Ninf 新システムを設計した。新システムを設計する上では、柔軟性と拡張性に重点をおいた。NES システムの実行時間はサーバでの計算時間がドミナントになることが予想される。このため、実行時間に悪影響を及ぼす可能性がある選択肢であっても、あえて選択している。

3.1.1 クライアント/サーバ間の接続の方式

NES システムの運用において、複数サーバへの同時リクエスト発行は本質的であるという観点から、常時接続を放棄し、必要に応じて接続することとした。

またプロキシに関しては、クライアントの構造を単純化するために採用した。しかし、プロキシの役割は起動リクエストの中継のみにとどめ、データ転送はクライアント/サーバ間で直接行うこととし、データ転送コストの増加は避けている。

3.1.2 通信プロトコルコマンドの方式

柔軟性、拡張性を重視し、コマンドプロトコルとしては XML を用いたテキストベースとした。XML が事実上の業界標準となっていること、DTD でスキーマが明確に定義できること、フリーのパーザが C 言語、Java 言語双方に存在し、システム実装上の負担が小さいことが理由である。

3.1.3 セキュリティ機構の方式

グローバル環境を前提とし、SSL(Secure Socket Layer) をベースにした Authentication, Authorization を行うこととした。

暗号手法にはほかにも Kerberos が選択肢として存在するが、SSL が業界標準になりつつあること、フリーのライブラリが C 言語、Java 言語で共に入手可能であることを考慮して SSL を用いることとした。

3.1.4 システムのオープン性

汎用サブシステムを用いることによる制約を避け、柔軟性を重視するため、基本的なコンポーネントはすべて独自で実装した。ただし、機能単位でコンポーネントを細分することで、汎用サブシステムの条件が整った場合にはそちらへ移行することができるよう、配慮してある。

資源データベースとしては、LDAP(Light weight directory service) サーバを採用した。資源データベースはシステムの一箇所に存在すればよく、アーキテクチャに対して制約となりにくいことが理由である。

3.2 新システムの概要

新システムは以下のコンポーネントから構成される(図 2)。

- クライアント
  計算を依頼するコンポーネント。
- サーバ
  クライアントからの計算依頼を受け、エグゼキュータブルを実行する。
- プロキシ
  クライアントに代わってスケジューラとの通信を行い、計算をリクエストするべきサーバを決定し、リクエストをサーバにフォワードする。
- エグゼキュータブル
  実際の計算を行うライブラリルーチンが組みこまれたコンポーネント。クライアントと通信して、計算を行う。

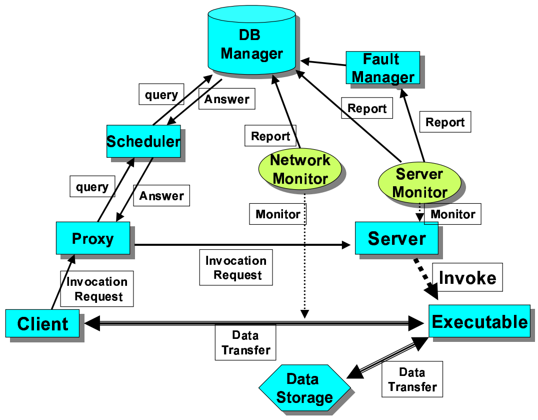

図 2　新システムの概要

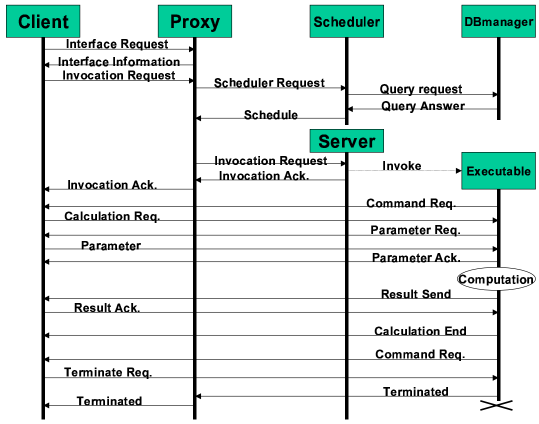

図 3　実行時の通信

- データ・ストレージ
  複数のサーバにまたがる計算を行う場合などに、計算結果を保存しておくのに用いる。
- スケジューラ
  プロキシからのリクエストに応じて、適切なサーバを紹介するスケジューラ。この際データベースマネージャと通信して、各サーバに関する情報を取り出し、その結果に基いて、適切なサーバを決定する。
- データベースマネージャ
  資源データベースを管理する。資源データベース自身は他のデータベース機構 (この場合は LDAP サーバ) で実装されており、他のコンポーネントからの要求はこのデータベースマネージャが中継する。これによって、データベースの実装がシステムから切り離されるので、任意のデータベースを使用することができる。
- ネットワークモニタ/サーバモニタ
  ネットワーク/サーバの状態を監視し、データベースマネージャに報告する。
- フォルトマネージャ
  何らかのエラーが検知された場合に、そのエラーに対する対応を行う。例えば、サーバが稼動していないことが発見された場合、そのサーバをデータベースから削除する。

3.3　クライアント/サーバ接続方式

新システムによる典型的な実行の際の通信を図 3に示す。

クライアントは、まずプロキシに対してルーチンのインターフェイス情報を要求する。引き続いてエグゼキュータブルの起動を要求する。クライアントは起動要求を出すと、プロキシへの接続を切断し、コールバック接続を待つ状態に入る。この時点でクライアントは他のコンポーネントと接続していない状態になるので、大量のリクエストを同時に発行することが可能になる。

プロキシは、スケジューラに適切なサーバの紹介を依頼する。スケジューラはデータベースマネージャにサーバのリストを要求し、クライアントサーバ間のスループット、サーバの負荷を考慮して、適切なサーバをプロキシに紹介する。

プロキシは、紹介されたサーバに対して、エグゼキュータブルの起動依頼をフォワードする。サーバはこれに応じてエグゼキュータブルを起動する。

起動されたエグゼキュータブルは、指定されたクライアントにコールバックで接続して、コマンド要求を行う。これに応じてクライアントは計算の開始を指示する。エグゼキュータブルは、計算に必要な引数をクライアントに要求する。必要な引数を受領すると、エグゼキュータブルはその旨をクライアントに通知し、一旦接続を切断する。そののち、要求された計算を開始する。クライアントは再びコールバック接続を待つ状態に入る。

計算が終了すると、エグゼキュータブルは再びクライアントに接続し、計算結果を送信した後、計算の終了を通知する。次にエグゼキュータブルはコマンド要求を行う。これに対して、クライアントは終了コマンドを送信する。

エグゼキュータブルはプロキシに終了を報告する。プロキシはこれをクライアントにフォワードする。プロキシに終了を報告するのは、プロキシが計算の管理を担当しており、資源データベースへの登録などを行うので、計算が終了したことを知っている必要があるからである。

3.4　通信プロトコルコマンド方式

通信プロトコルコマンドの例として、クライアントからサーバ側のルーチンを起動する際に用いるプロトコルコマンドの DTD を図 4に示す。

この DTD に基いた、起動コマンド例を図 5に示す。

起動コマンドには、client と observer という二つのアドレスが含まれている。client は、クライアントへコールバック接続するアドレスである。observer はプ

```
<!ELEMENT invoke_executable
    (issuer ,function_name, client, observer)>
<!ELEMENT issuer EMPTY>
<!ATTLIST issuer process CDATA #REQUIRED>
<!ATTLIST issuer host CDATA #REQUIRED>
<!ATTLIST issuer port CDATA #REQUIRED>
<!ATTLIST issuer session_key CDATA #REQUIRED>
<!ELEMENT function_name EMPTY>
<!ATTLIST function_name module CDATA #REQUIRED>
<!ATTLIST function_name entry CDATA #REQUIRED>
<!ELEMENT client (peer)>
<!ELEMENT observer (peer)>
<!ELEMENT peer EMPTY>
<!ATTLIST peer host CDATA #REQUIRED>
<!ATTLIST peer port CDATA #REQUIRED>
```

図 4　起動コマンド DTD

```
<invoke_executable>
<issuer process="nsserver" host="hpc.etl.go.jp"
    port="30000" session_key="12345" />
<function_name module="test" entry="mmul" />
<client>
   <peer host="hpc.etl.go.jp" port="30000" />
</client>
<observer>
    <peer host="hpc.etl.go.jp" port="30001" />
</observer>
</invoke_executable>
```

図 5　起動コマンド例

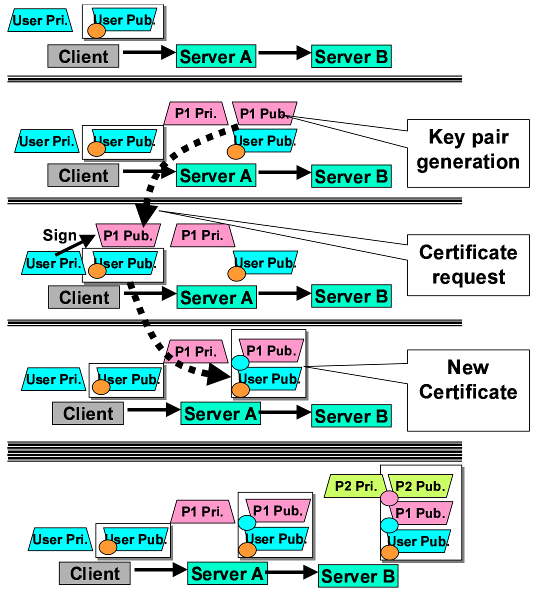

図 6　アイデンティティの委譲

ロキシに終了を通知する際にコールバック接続するアドレスである。

3.5　セキュリティレイヤの実装

NES システムにおけるセキュリティには、Authentication, Authorization, Privacy の 3 つの要素がある。Authentication とは、接続してきたユーザが誰なのかを知ること、Authorization とは、そのユーザに何を許すのかを決定すること、Privacy とは、通信を秘匿化することである。

Ninf はクライアントがプロキシに接続し、プロキシがサーバに接続する構造となっている。サーバの Authorization はクライアントのアイデンティティに基づいて行わなければならない。このため、プロキシからサーバへの接続時に、クライアントのアイデンティティによって接続しなければならない。これがアイデンティティの委譲である。

新システムでは、Authentication と Privacy を実現する NAA(NES Authentication Authorization) と呼ぶモジュールを実装した。NAA は下位レイヤとして SSL を採用し、その上にアイデンティティの委譲と Authorization を実現している。

NAA は、C 言語と Java 言語でそれぞれ実装されている。これは新 Ninf システムから独立したレイヤとなっているので、他のシステムで使用することもできる。アイデンティティの委譲は、NAA によって自動的に行われるので、プログラマは、証明書の設定などをするだけでよく通常のソケットライブラリと同様に使用することができる。

3.5.1　アイデンティティの委譲

SSL におけるアイデンティティ は CA によって保証された証明書である。CA は階層化することが可能で、ある証明書を用いて、他の証明書にサインすることが可能である。NAA では、ユーザのアイデンティティをユーザの証明書と、その証明書を用いて署名されたすべての証明書に広げることでアイデンティティの委譲を実現した。

SSL は、対象鍵暗号を用いている。SSL における証明書の本質は、ユーザの公開鍵を CA の秘密鍵で暗号化したものである。さらに、新たに公開鍵、秘密鍵のペアを作成し、ユーザの秘密鍵を用いて暗号化することで、新たな証明書を作ることができる。これが証明書チェーンである。

NAA では、この証明書チェーンを用いて、チェーンにあるユーザの証明書が含まれていれば、そのユーザのアイデンティティであるとみなす。この手法は、Globus のセキュリティレイヤでも用いられている [8]。

クライアントがサーバ A に接続し、サーバ A がサーバ B に接続する場合を考える。サーバ A はクライアントからの接続を受けると、新たに鍵ペアを作成する。サーバ A はこの鍵ペアの公開鍵を証明書リクエストとしてクライアントに返送する。クライアントは、この公開鍵に対してクライアント自身の秘密鍵を用いて署名し、新たな証明書を作成し、サーバ A に返送する。サーバ A がサーバ B に接続する際にはこの新たな証明書とクライアントの証明書の双方を用いて接続する。この様子を図 6に示す。

3.5.2　NAA ポリシ

NAA における Authorization は、Java 1.2 の Policy クラスに類似したポリシと呼ぶ構造をもとに行う。ポリシは、グラントと呼ぶ構造の集合である。グラントはあるユーザに対して許されるパーミッションの集合から構成される。ポリシは NAA のユーザが独自に

```
<!ELEMENT policy (grant)*>
<!ELEMENT grant (permission)*>
<!ATTLIST grant userid CDATA #REQUIRED>
<!ELEMENT permission EMPTY>
<!ATTLIST permission class CDATA #REQUIRED>
<!ATTLIST permission target CDATA #REQUIRED>
<!ATTLIST permission action CDATA #REQUIRED>
```

図 7　ポリシファイル DTD

```
<policy>
<grant userid="c=jp,o=etl">
  <permission class = "stubexec"
        target = "test/entry0" action="100 20"/>
  <permission class = "stubexec"
        target = "test/entry1" action="100 20"/>
</grant>
<grant userid="c=jp,o=etl,CN=nakada">
  <permission class = "stubexec"
        target = "test/entry3" action="100 20"/>
</grant>
</policy>
```

図 8　ポリシファイルの例

拡張可能な構造となっている。

NAA ライブラリはポリシと現在のユーザのアイデンティティを保持している。プログラム中からは、特定のパーミッションが現在許可されているかを問い合わせる。ライブラリは、現在のユーザアイデンティティに対するパーミッションがポリシに含まれているかどうかを調べて判定する。

NAA におけるユーザの名前空間は、X.509 のコンベンションに従ったツリー構造となっている。NAA はこの名前の階層構造を用いて、階層的なアクセス制御を行う。

パーミッションには、クラス、ターゲット、アクションの 3 つの属性がある。クラスはそのパーミッションが許可する操作を表す。ターゲット、アクションは、操作の対象と操作の種類をそれぞれ示す。

ポリシは、XML によるポリシファイルで設定する。ポリシファイルの DTD と記述例を図 7と図 8に示す。

図 8の例では、2 つの grant が定義されている。上の grant は、ユーザ ID に `c=jp,o=etl` が含まれているユーザは、`test/entry0` と `test/entry1` を実行することができることを意味する。下の grant は、ユーザ ID が `c=jp,o=etl,CN=nakada` であるユーザは `test/entry3` を実行することができるという意味である。

`c=jp,o=etl,CN=nakada` には `c=jp,o=etl` が含まれているので、`c=jp,o=etl,CN=nakada` は `test/entry0` と `test/entry1` と `test/entry3` が実行できる。このように、名前空間の階層構造を利用することによってポリシ記述が容易になっている。

4. おわりに

本稿では、NES システムの設計上の選択点を整理し、それぞれの点に関してトレードオフを示した。また、これらの点に留意して設計した Ninf の新システムを紹介した。

今後、新システムで採用した設計ポリシの有効性を検証するべく、実際に運用を行っていく予定である。また、各デザインチョイスが性能に与えた影響を調査するべく、包括的な性能測定を行う必要がある。

謝　　辞

本研究の一部は情報処理振興事業協会（IPA）の次世代アプリケーション開発事業「広域分散コンピューティングアプリケーションの開発」の一環として行われた。また、設計の詳細について議論いただいた NTT ソフトウエアの岩崎佳克氏、大久光崇氏日本コンピュータ研究所の相良博氏、川口正壽氏に感謝する。

参 考 文 献

1) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proceedings of Super Computing '96* (1996).

2) Abramson, D. and Giddy, J.: Scheduling Large Parametric Modeling Experiments on a Distributed Meta-computer, *Proceedings of PCW'97* (1997).

3) Foster, I. and Kesselman, C.: Globus: A metacomputing infrastructure toolkit., *Proc. of Workshop on Environments and Tools, SIAM.* (1996).

4) 中田秀基, 佐藤三久, 関口智嗣: ネットワーク数値情報ライブラリ Ninf のための RPC システムの概要, TR 95-28, 電子技術総合研究所 (1995).

5) 中田秀基, 竹房あつ子, 松岡聡, 佐藤三久, 関口智嗣: グローバルコンピューティングのための スケジューリングフレームワーク, 並列処理シンポジウム JSPP'99 論文集, pp. 277–284 (1999).

6) Takefusa, A., Matsuoka, S., Ogawa, H., Nakada, H., Takagi, H., Sato, M., Sekiguchi, S. and Nagashima., U.: Multi-client LAN/WAN Performance Analysis of Ninf: a High-Performance Global Computing System, *Supercomputing '97* (1997).

7) Wolski, R., Spring, N. and Peterson, C.: Implementing a Performance Forecasting System for Metacomputing: The Network Weather service, *Proceedings of the 1997 ACM/IEEE Supercomputing Conference* (1997).

8) Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S.: A Security Architecture for Computational Grids, *Proc. 5th ACM Conference on Computer and Communication Security, toappear* (1998).