マルチメディア通信と分散処理56－10
（１９９２．　７．　10）

# RPCプロトコルのエミュレーション方式による分散処理システムの相互接続

藤長　昌彦　　加藤　聰彦　　鈴木健二
KDD研究所

異なる分散処理システムを組み合わせ、より高度な分散処理を行なうためには、分散処理システムの相互接続を行なうことが必要となる。現在の分散処理システムの多くがクライアントサーバ・モデルに基づくRPCをプロセス間通信の基本技法として採用していることから、異なる分散処理システム上で動作するクライアントとサーバとの間にRPCの機能を提供することにより、分散処理システムの相互接続を行なうことができる。本稿では、クライアントがサーバ側のRPCプロトコルをエミュレートすることにより、分散処理システムの相互接続を容易に実現する手法を提案し、既存の分散処理システムに適用・実装した結果について報告する。

# Interconnection of Distributed Systems using Emulation of RPC Protocol

Masahiko FUJINAGA, Toshihiko KATO and Kenji SUZUKI
KDD R & D Laboratories
Ohara 2-1-15, Kamifukuoka, Saitama 356, Japan
{fuji, kato}@osi.kddlabs.co.jp

The interconnection of existing distributed systems enables more flexible, versatile distributed processing. Such interconnection can be achieved by providing remote procedure call (RPC) facility to a client and a server that reside on different distributed systems, since the most existing distributed systems employ RPC based on the client-server model as the basic scheme for inter-process communication. This paper proposes a method to implement the inter-distributed systems RPC easily, using the emulation of RPC protocol of the servers. It also describes the application of the proposed method to the existing distributed systems, SUN/ONC and NCS.

## 1. はじめに

複数の計算機を有機的に結合する分散処理システムの開発が活発に行なわれており、今後、複数の分散処理システムが共存しつつ普及してゆくものと考えられる。このような環境において、異なる分散処理システムを組み合わせ、個々のシステムの特質を活かしながら、より高度な分散処理を行なうためには、分散処理システムの相互接続を行なうことが必要となる。現在の分散処理システムの多くは、クライアントサーバ・モデルに基づく遠隔手続き呼出し(RPC)をプロセス間通信の基本技法として採用している。従って、異なる分散処理システム上で動作するクライアントとサーバとの間にRPCの機能を提供することにより、分散処理システムの相互接続を行なうことができる。

一般にRPCの通信は、①クライアントとサーバとの論理的結合、②RPCメッセージの組立／分解、③クライアントとサーバ間でのRPCメッセージの転送により実現される。異なる分散処理システムにまたがるRPCの提供に当たっては、独自の手法により実現されているこれらの機能を変換する必要がある。

本稿では、データ転送機能については標準プロトコルを採用し、クライアントとサーバの論理的結合ならびにRPCメッセージの組立／分解についてはサーバ側の分散処理システムのプロトコルをエミュレートすることにより、異なる分散処理システムにまたがるRPCを容易に実現する手法を提案する。更に、現在広く普及している "SUN/ONC" (Open Network Computing)と、OSFで採用された "NCS" (Network Computing System)を対象として、提案する手法を適用した結果について述べる。

## 2. RPCに基づく分散処理システムの相互接続の課題

一般に、クライアントサーバ・モデルに基づくRPCは、図1に示す要素により実現される。

① クライアントとサーバの論理的結合：サーバは自分自身の論理名と、通信に使用する実行時識別子とをネームサーバに登録する。クライアントはサーバの論理名を用いてネームサーバを検索し、サーバの実行時識別子を得る。分散処理システムによっては、この時点でクライアントとサーバとの間に通信路が設定される。

② RPCメッセージの組立／分解：サーバは、サーバの論理名、手続き名とその引数を記述するインタフェース仕様を公開する。スタブジェネレータはサーバのインタフェース仕様からスタブを自動生成する。スタブは、サーバおよびクライアントプログラムに組み込まれ、RPCメッセージの組立／分解を行なう

③ データ転送機能によるRPCメッセージの転送：スタブは、その分散処理システムによって提供されるデータ転送機能を使用し、クライアントとサーバ間でRPCメッセージを転送する。

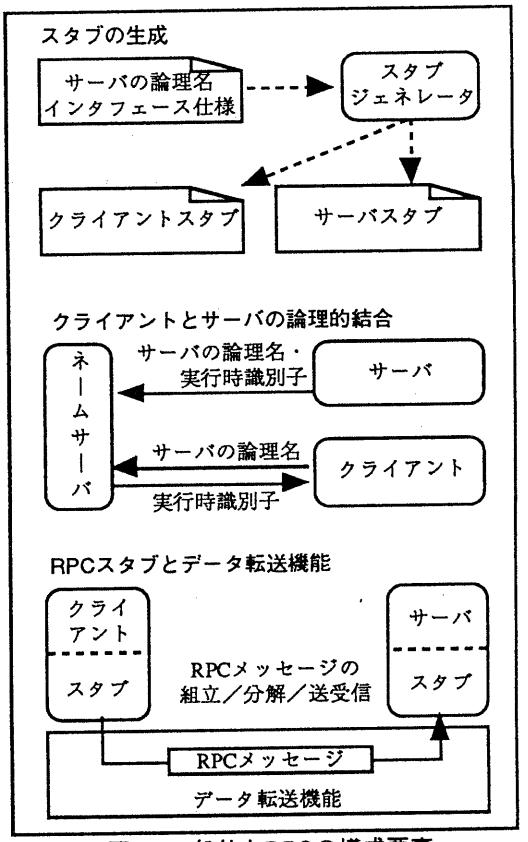


図1　一般的なRPCの構成要素

本論文では、①と②の実行手順を「RPCプロトコル」と呼び、③におけるデータ転送手順をトランスポートプロトコルと呼ぶこととする。即

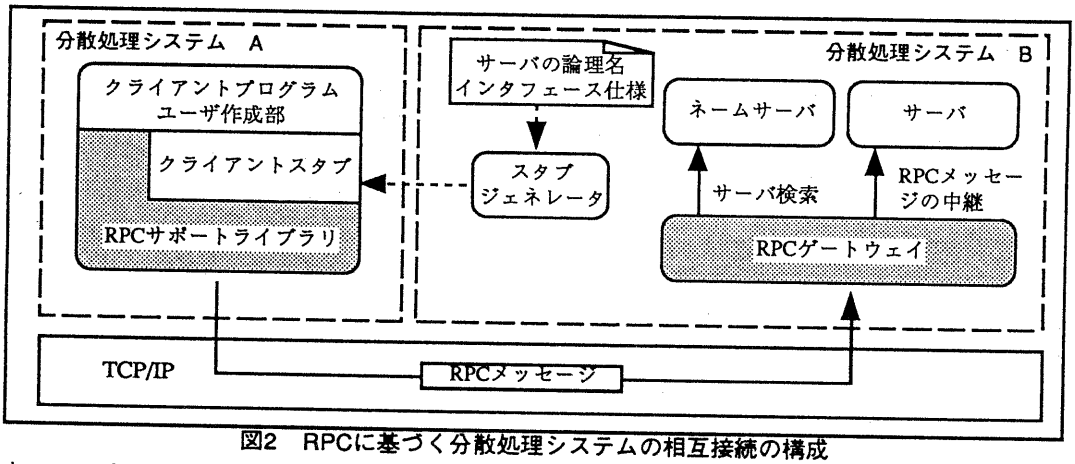


図2 RPCに基づく分散処理システムの相互接続の構成

ち、RPCプロトコルは、サーバの論理名の形式、インタフェース仕様の記述法、RPCメッセージの形式を含む。

RPCに基づく分散処理システムを相互接続するためには、以下の課題がある。

(1) 相互接続される分散処理システムの間で、RPCプロトコルとトランスポートプロトコルの変換を行なう必要がある。各プロトコルに関して、標準プロトコルを導入するか、あるいは一方が他方のプロトコルをエミュレートするかを検討する必要がある[1, 2]。標準プロトコルを導入する方式では、相手側分散処理システムのプロトコルを意識する必要がなく、変換機能をひとつ準備するだけで良いという利点がある。一方、エミュレーション方式は、プロトコル変換の回数が1回ですむため実行効率の点で有利である。

(2) プロトコル変換のためのゲートウェイを準備するか、クライアントあるいはサーバが直接プロトコル変換を行なうかを選択する必要がある。更に、ゲートウェイによりプロトコル変換を実現する形態には、特定のサーバあるいはクライアントに専用のゲートウェイを設ける形態と、すべてのサーバあるいはクライアントに対するアクセスを扱う汎用のゲートウェイを設ける形態とがある。

ゲートウェイによる実現では、既存のサーバ、クライアントをそのまま利用できる反面、プロセス間通信の回数が増えるため実行効率を損なう恐れがある。逆に、サーバあるいはクライアントに直接プロトコル変換を行なわせる場合には、既存のサーバあるいはクライアントに改修を加える必要がある。

## 3. 実現方法

### 3.1 基本設計

ここでは、分散処理システムの相互接続を容易に実現する一手法を提案する。本手法においては、個々の分散処理システムの特質を活かした相互接続を実現するために、既存のサーバに改修を加えることなく、他の分散処理システムからのアクセスを可能とすることを目的とする。ただし、クライアントについては新規に作成されるものを対象とした。

提案する実現法を以下に示す(図2参照)。

- RPCプロトコルについては、サーバ側の分散処理システムで用いられるものをエミュレートする。即ち、クライアントは、サーバの論理名と実行時識別子およびインタフェース仕様をそのまま使用し、クライアントスタブについてもサーバ側分散処理システムのスタブジェネレータにより生成されたものをそのまま利用する。
- トランスポートプロトコルには、信頼性の高いコネクション型通信を提供し、広く利用可能なTCP/IPを標準プロトコルとして採用する。
- サーバ側分散処理システムに「RPCゲートウェイ」を設け、サーバの検索とRPCメッセージの中継を行なう。
- サーバ側分散処理システムで生成されるクラ

イアントスタブがクライアント側分散処理システムで動作できるよう、「RPCサポートライブラリ」を準備する。

RPCメッセージの基本的な構造は、分散処理システムごとに定められるが、個々のRPCメッセージはサーバの手続きごとに異なる。また、一般には、RPCメッセージを解析しただけでは、それが運ぶRPC引数のデータ型を判定できない。従って、汎用のゲートウェイを用いてRPCメッセージを変換するためには、ゲートウェイがすべてのサーバのインタフェース仕様を管理する必要があり、実行効率の点から現実的でない。一方、新規にクライアントを作成する場合、プログラマは、サーバの論理名とインタフェース仕様について知識を有していると仮定できる。そこで、本方式では、サーバ側分散処理システムで規定されるサーバの論理名、インタフェース仕様、スタブをクライアントにおいてそのまま利用し、サーバの理解できるRPCメッセージを直接組み立てることとした。

逆に、トランスポートプロトコルについては、サーバ側分散処理システムに合わせて新たに開発することは容易でなく、また現実には多くの分散処理システムでTCP/IPをサポートしていることから、これを標準プロトコルとして採用することとした。

クライアントに組み込まれたスタブとRPCサポートライブラリ、およびRPCゲートウェイの組み合わせは、サーバの観点からはサーバ側分散処理システム上のクライアントとして認識される。このため、サーバおよびサーバ側分散処理システム上のネームサーバを何ら改修することなく、他の分散処理システムからRPCを発行することができる。

## 3.2 RPCサポートライブラリとRPCゲートウェイ

サーバ側分散処理システムで生成されたスタブは、その分散処理システムの機能を用いてRPCメッセージの転送を行なう。RPCサポートライブラリは、このスタブが異なる分散処理システム上で動作できるようにするものである。

RPCサポートライブラリの提供関数はスタブから呼び出される関数であり、

- サーバ検索のための関数
- RPCの発行関数
- 基本データ型の表現変換関数

からなる。RPCサポートライブラリは、サーバ側で生成されたスタブに対してこれらの関数を提供するものであるため、基本的には、クライアント側の分散処理システムに依存しない。

サーバ検索のための関数とRPCの発行関数については、実際の処理をRPCゲートウェイに依頼する形で実現する。このため、RPCサポートライブラリ自体で実装すべきものは、RPCメッセージの組立／分解時に呼び出される基本データ型の表現変換関数と、RPCゲートウェイとの間の通信機能のみである。

RPCゲートウェイは、サーバ側分散処理システムごとにひとつのプロセスとして設けられ、他の分散処理システム上で動作するクライアントからの要求に応じて子プロセスを生成する。生成された子プロセスはクライアントと1対1に対応し、サーバ側分散処理システムにおけるクライアントのproxyとして、サーバとの論理的結合とRPCメッセージの中継を行なう。サーバの手続きごとに異なるRPCメッセージの引数部分については、クライアント内のスタブにより組み立てられるため、RPCゲートウェイではその内容に関与することなく中継することができる。

RPCサポートライブラリとRPCゲートウェイ間の通信は、サーバの検索とRPCメッセージの中継という処理の依頼であるため、これ自体、RPCと捉えることができる。そこで、筆者らが開発したRPCシステム[3]のスタブジェネレータを利用し、RPCサポートライブラリとRPCゲートウェイ間の通信機能を実現することとした。これにより、RPCサポートライブラリにおけるサーバの検索のための関数とRPCの発行関数は、サーバとしてのRPCゲートウェイに対するクライアントスタブとなる。RPCゲートウェイにおける処理の実現には、サーバ側の分散処理システムの機能が利用できる。RPCゲートウェイは、RPCサポートライブラリからの要求に応じてサー

バ検索関数あるいはRPC発行関数を呼び出すという形で、容易に実現することができる。

## 4. 既存分散処理システムへの適用

本章では、前章で提案した手法を、現在広く普及しているSUN/ONCと、OSFで採用されたNCSに適用し、RPCサポートライブラリとRPCゲートウェイを試作した結果について述べる。

### 4.1 SUN/ONCへの適用

SUN/ONC[4]では、サーバの論理名として{ホスト名, プログラム番号, バージョン番号}の組を用い、サーバの実行時識別子としてソケットアドレスを使用している。両者の対応は、各計算機上でひとつ動作する「ポートマッパ」と呼ばれるネームサーバが管理している。クライアントがサーバとの論理的結合を要求すると、論理名で指定された計算機上のポートマッパへアクセスし、プログラム番号とバージョン番号で識別されるサーバのソケットアドレスを問い合わせる。サーバが稼働している場合には、ソケットアドレスやソケットのディスクリプタ、RPC-メッセージのためのバッファなどを含む構造体がクライアント内に生成され、この構造体を指すポインタ(「クライアントハンドル」と呼ばれる)がその結果として返される。クライアントスタブは、このクライアントハンドルを指定して、RPCの発行要求を行なう。

RPCメッセージは、通信相手の内部データ表現にかかわりなく、すべてSUN/ONCの定める標準データ表現形式であるXDR (External Data Representation)により組み立てられる。

トランスポートプロトコルは、UDP/IPとTCP/IPが準備されており、サーバとの論理的結合時に選択することができる。トランスポートプロトコルとしてUDP/IPを使用する場合、ひとつのRPCメッセージはひとつのデータグラムで転送される。TCP/IPを使用する場合には、レコードマーキングプロトコルが併用される。レコードマーキングプロトコルは、長いRPCメッセージをセグメンティングし、各セグメントの先頭にセグメント長と後続セグメントの有無を示すフラグをセットする。

表1　SUN/ONC用サポートライブラリの関数

| | |
|---|---|
| ●サーバ検索等<br>clnt_create(), clnt_control()等 | 4種 |
| ●RPC発行関数<br>clnt_call() | 1種 |
| ●基本データ型変換関数<br>xdr_char(), xdr_int(), xdr_array(), xdr_opaque()等 | 22種 |
| ●バッファ管理関数<br>xdr_create()等 | 11種 |

SUN/ONC用のRPCサポートライブラリでは、表1に示す38種の関数を作成した。

サーバのトランスポートプロトコルとしてUDP/IPを使用する場合の通信手順を図3に示す。クライアントが、サーバの論理名である{ホスト名, プログラム番号, バージョン番号}を指定して'clnt_create()'を呼び出すと、RPCサポートライブラリからRPCゲートウェイへTCPコネクションが設定され、サーバの論理名が転送される。RPCゲートウェイは、SUN/ONCの提供する'clnt_create()'を発行し、その内部にクライアントハンドルを生成・保持し、子プロセスを'fork()'する。子プロセスは、クライアントハンドルの識別子をクライアントへ返送し、クライアントからの次の要求を待つ。

クライアントがサーバの手続きを呼び出し、スタブが'clnt_call()'を呼び出すと、既に設定されているTCPコネクションを通して、RPCサポートライブラリからRPCゲートウェイに、クライアントハンドルの識別子とRPCメッセージが渡される。RPCゲートウェイは、クライアントハンドルの識別子から保持していたクライアントハンドルを取り出し、SUN/ONCの提供する'clnt_call()'を発行してRPCメッセージをサーバへ転送する。

サーバがTCP/IPを使用する場合には、RPCメッセージの中継時におけるRPCゲートウェイでのプロトコル変換は実質的に不要となる。このため、クライアントからサーバへ直接コネクションを確立することとし、実行効率の向上を図った。その通信手順を図4に示す。クライアントが'clnt_create()'によりサーバとの論理的結合を要求すると、RPCゲートウェイとの間にTCPコネ

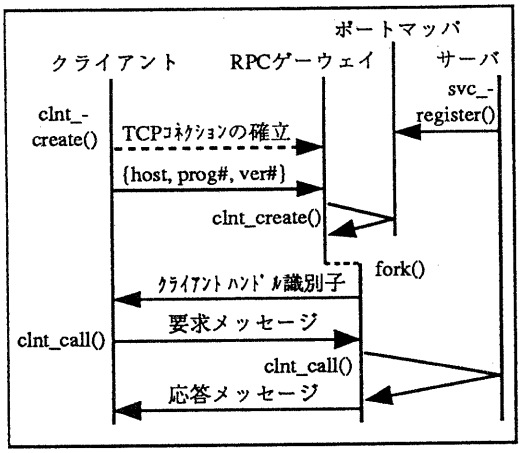


図3　SUN/ONCでの通信手順
(サーバがUDP/IPを使用する場合)

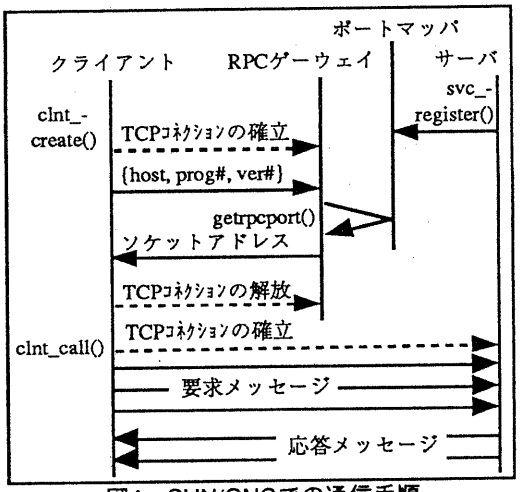


図4　SUN/ONCでの通信手順
(サーバがTCP/IPを使用する場合)

クションが設定され、サーバの論理名が通知される。RPCゲートウェイはSUN/ONCのライブラリ関数である'getrpcport()'を用いてポートマッパへアクセスし、サーバのソケットアドレスをクライアントへ通知する。この時、RPCゲートウェイ内部には、クライアントハンドルは生成されない。RPCサポートライブラリは、サーバのソケットアドレスを受け取ると、RPCゲートウェイとの間のTCPコネクションを解放し、クライアントハンドルを生成する。このクライアントハンドルは、SUN/ONCの'clnt_create()'により生成されるものを簡略化したものであり、スタブがRPCメッセージを組み立て、RPCゲートウェイに転送するための最小限の情報のみを保持している。

RPCサポートライブラリは、最初の'clnt_call()'が呼び出された時点で、サーバのソケットアドレスに対してTCPコネクションを確立する。その後、レコードマーキングプロトコルに従ってセグメンティング／リアセンブリングを行ないつつ、RPCメッセージの送受信を行なう。

## 4.2 NCSへの適用

NCS[5]では、サーバのインタフェース、サーバの管理するデータオブジェクト、オブジェクトのタイプの各々に16バイト長のUUID (Universal Unique Identifier)を割り当て、三種類のUUIDの組み合わせを論理名としている。サーバの実行時識別子としてはSUN/ONCと同様、ソケットアドレスを使用している。ネームサーバは「ロケーションブローカ」と呼ばれ、複数の計算機上のサーバを管理する集中型ネームサーバである。クライアントは、サーバとの論理的結合をクライアント内部に作成する「RPCハンドル」によって管理している。RPCハンドルは、サーバのUUIDやソケットアドレス、サーバとの結合状態等を保持する構造体へのポインタであり、これを指定してRPCの発行要求等を行なう。

NCSでは、送信側計算機のスタブは自分自身の内部データ表現によりRPCメッセージの引数部分を組み立て、RPC発行関数である'rpc_$sar()'を呼び出す。'rpc_$sar()'は、RPCメッセージのヘッダ部を組み立て、計算機内部のデータ表現形式(バイトオーダ、実数の表現形式、文字コード)を示すフラグを付加して送信する。受信側計算機のスタブは、このフラグを参照して必要なデータ表現変換を行なう。

トランスポートプロトコルは、UDP/IP等のコネクションレス型のプロトコルと、その上位のRPCメッセージのセグメンティング機能や再送機能の組み合わせとして実現されている。

NCSでは、三種類のUUIDの組み合わせにより柔軟なサーバ検索機能を提供しており、クライアントとサーバの論理的結合を管理する関数や通信モードの設定を行なう関数など、豊富なライブラリ関数を提供している。これらのライブ

表2　NCS用サポートライブラリの関数

| | |
|---|---|
| ● サーバ検索等<br>lb_$lookup_interface(), rpc_$bind()等 | 11種 |
| ● RPC発行関数<br>rpc_$sar() | 1種 |
| ● 基本データ型変換関数<br>rpc_$marshall_long_int(),<br>rpc_$cvt_long_float()等 | 49種 |
| ● バッファ管理関数<br>rpc_$alloc_pkt(), rpc_$free_pkt() | 2種 |
| ● 例外処理関数その他 | 16種 |

ラリ関数の中から、クライアントにおける使用頻度の高いもの81種を選び、RPCサポートライブラリとして作成した。これらを表2に示す。

NCSでの通信手順は、図5に示すように、SUN/ONCにおけるUDP/IPの場合と同様のものである。RPCハンドルに格納される情報は公開されていないため、RPCハンドルにかかわる実際の処理はすべてRPCゲートウェイで行なうこととした。このため、RPCハンドルの実体はRPC-ゲートウェイの内部に作成し、クライアントへはその識別子のみを返している。

RPCメッセージは、ライブラリ関数である'rpc_$sar()'によりヘッダ部分がセットされ、送信される。従って、RPCゲートウェイがRPCメッセージの中継処理において'rpc_$sar()'を単純に発行すると、RPCメッセージの引数部分がクライアント計算機の内部データ表現により組み立てられているにもかかわらず、ヘッダ部分にはRPCゲートウェイの内部データ表現を示すフラグがセットされる。クライアントとRPCゲートウェイの内部データ表現が異なる場合においても矛盾のないRPCメッセージの中継を実現するため、RPCゲートウェイでは、内部データ表現の形式を示す広域変数'rpc_$local_drep_packed'の値を、クライアント計算機の内部データ表現に合わせて操作している。

## 5. 考察

(1) RPCプロトコルのエミュレーションを行なうことにより、広く普及しているSUN/ONCとNCSにおいて、分散処理システムにまたがるRPCを容易に実現することができた。これは、以下の理由による。

① スタブの生成をサーバ側分散処理システムに委ねることにより、多様なサーバのインタフェース仕様に対処することができる。

② RPCサポートライブラリでは、基本データ型の表現変換等の関数を実現するだけでよい。

③ 実際のサーバ検索やRPCの発行をRPCゲートウェイで実行するため、サーバ側分散処理システムの機能をそのまま利用することができる。

④ RPCサポートライブラリとRPCゲートウェイ間の通信をRPCとして捉えることができ、既存のスタブジェネレータを利用することにより実装が容易となる。

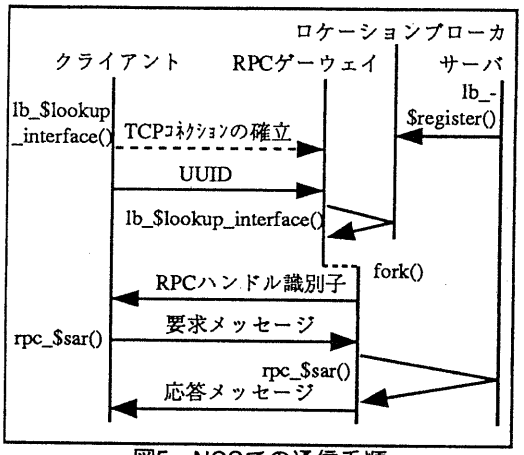

図5　NCSでの通信手順

(2) 分散処理システムの相互接続を実現するためには、RPCプロトコルとトランスポートプロトコルの双方に標準プロトコルを導入する方法が考えられる。この方法では本稿で提案する方法と異なり、相互接続される分散処理システムのサーバの論理名やインタフェース仕様を、標準方式を用いて規定する必要がある。しかし、サーバの論理名やインタフェース仕様の形式は、分散処理システムに依存して大きく異なっており、標準方式を導入することは、一般には困難であると考えられる。更に、標準方式により規定されたとしても、標準方式と個別の方式の整合を保つ必要があるという問題点がある。即ち、サーバの論理名と実行時識別子の対応については、個別の分散処理システムのネームサーバとは別に、複数の分散処理システムにまたがるネームサービスを構築する必要があり、また、イン

タフェース仕様については、サーバのプログラマが分散処理システム固有のインタフェース仕様と標準的なインタフェース仕様のふたつの整合を取る必要がある。

これに対し、提案する方法では、上記の問題点を回避するために、クライアントのプログラマに、アクセスするサーバが動作する分散処理システムで使用される論理名とインタフェース仕様の形式について知識を持つことを要求している。

(3) また、分散処理システムの相互接続に関する他の研究例として、ワシントン大学のHCSプロジェクトがある[6]。HCSでは、RPCメッセージの形式とトランスポートプロトコルについてはエミュレーションを行なっており、サーバの論理名、インタフェース仕様には標準方式を導入している。トランスポートプロトコルを含めエミュレーション方式を採用しているため、プロトコル変換のためのゲートウェイが存在せず、高い実行効率を達成している。しかし、相互接続の対象となる分散処理システムの数が増加した場合、種々のトランスポートプロトコルを実装することは必ずしも容易でない。例えば、NCSにおいては、サーバからクライアントへのコールバック機能など種々の機能がトランスポートプロトコルで実現されているが、そのソースコードは公開されていないためエミュレーションは困難である。

これに対し、提案する実現法では、RPCサポートライブラリとRPCゲートウェイ間の通信には広く利用可能なTCP/IPを採用し、RPCゲートウェイとサーバとの間の通信にはその分散処理システムが提供するトランスポートプロトコルを利用する。このため、相互接続の対象となる分散処理システムの数が増加した場合にも実装は容易である。

## 6. むすび

本稿では、異なる分散処理システムの相互接続を目的として、分散処理システムにまたがるRPCを提供する手法を提案するとともに、本手法をSUN/ONCおよびNCSに対して適用した結果について述べた。本手法は、既存のサーバに変更を加えることなく、他の分散処理システム上のクライアントからRPCの発行を可能とするために、トランスポートプロトコルについてはTCP/IPを標準プロトコルとして採用し、RPCプロトコルについてはサーバ側のプロトコルをエミュレートする方式を採用している。サーバ側分散処理システムで使用される論理名やインタフェース仕様をそのまま利用するため、多様なサーバインタフェースに対して自然に対応することができ、相互接続を容易に実現することが可能である。最後に、日頃御指導戴くKDD研究所小野所長、浦野次長に感謝する。

## 参考文献

[1] 藤長, 加藤: "RPCに基づく分散処理システムの相互接続に関する検討", 「1990年代の分散処理」シンポジウム, 情処シンポジウム論文集 Vol. 90, No. 5, pp. 21-30, Nov. 1990.

[2] 加藤, 藤長: "分散処理システムの相互接続のためのRPCプロトコル変換", 情報処理学会第42回全国大会, 2S-6, Mar. 1991.

[3] 藤長, 加藤, 鈴木: "適応的データ表現変換に基づく異機種間RPCシステムの実装と評価", 情報処理学会 マルチメディア通信と分散処理研究会, 51-1, pp. 1-8, Jul. 1991.

[4] Sun Microsystems: "Network Computing," 1988.

[5] M. Kong, et. al.: "Network Computing System Reference Manual," Prentice-Hall, 1990.

[6] B. M. Bershad, et. al.: "A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems," IEEE Trans. on Software Engineering, Vol. SE-13, No. 8, August 1987.