マルチメディア通信と分散処理 51－1
（１９９１．７．５）

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

藤長 昌彦　　加藤 聰彦　　鈴木 健二
KDD研究所

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)が、分散処理システムの開発技法として広く利用されている。本稿では、RPCの高速化を目的とする適応的データ表現変換について述べ、それに基づく異機種間RPCシステム"KoKo"の実装と評価について報告する。適応的データ表現変換は、クライアントとサーバが使用する内部データ表現をRPCに先立ち通知しあい、RPCごとに最適なデータ表現を決定することにより、不要なデータ表現変換を取り除きRPCメッセージの組立て／分解に伴うオーバヘッドを最小化するものである。実装したKoKoの性能評価を行ない、既存RPCシステムと比較することにより適応的データ表現変換の有効性を示した。

A Heterogeneous Remote Procedure Call System based on Adaptive Data Representation Conversion

Masahiko FUJINAGA, Toshihiko KATO and Kenji SUZUKI
KDD R&D Laboratories
Ohara 2-1-5, Kamifukuoka, Saitama 356, JAPAN

Remote procedure call (RPC) based on the client-server model has significant influence on the overall performance of distributed systems. In the high speed network environment, the cost of RPC stub level needs to be reduced in order to increase RPC performance. This paper describes an RPC system called " KoKo" based on the adaptive data representation conversion, which avoids unnecessary data representation conversion as much as possible. The data representation is negotiated, and if no conversion is needed, the arguments of RPC are conveyed using the internal data representation. The evaluation shows that the performance of KoKo is better than those of other RPC systems.

1. はじめに

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)が、ネットワークを介して複数の計算機を有機的に結合する分散処理システムの有効な開発技法として広く採用されている。RPCは、クライアントとサーバのプログラマに対し、通信手順の詳細を隠蔽し、手続き呼出しの形式によるプログラミングを可能とすることから、分散処理システムの開発を容易にすることができる。このため、RPCは、ネットワークファイルシステムやウィンドウシステム、分散オペレーティングシステム(OS)自身などの開発に用いられている。RPCが、分散処理システム開発の基本的な技法として使用されることから、分散処理システムの性能を向上させるためにはRPCを高速化することが重要であると考えられる。

RPCは、スタブレベルとトランスポートレベルによりモデル化することができる。スタブレベルは、スタブと呼ばれる通信用の特別なルーチンを含んでいる。スタブは、RPCメッセージの組立て/分解と、異なる内部データ表現を持つ計算機上で稼働するクライアントとサーバ間で通信を行なうためのデータ表現変換を行なう。トランスポートレベルは、計算機内通信とネットワーク通信との差異を意識させないプロセス間通信(IPC)機能をスタブレベルに対して提供する。

RPCを高速化するためには、スタブレベルにおけるコストとトランスポートレベルにおけるコストを低減する必要がある。従来は、トランスポートレベルのコストに比較して、スタブレベルにおけるコストは無視できると考えられてきた。このため、多くのRPCシステムが、RPCのための標準のデータ表現を定め、計算機の内部データ表現を画一的に標準データ表現に変換する方法を採っている[1-3]。

しかしながら、このようなデータ表現変換のコストを含むスタブレベルのコストは、現在一般的に使用されている10Mbpsイーサネット上のTCP/IP環境においても、トランスポートレベルのコストと比較して必ずしも無視できるものではない[4]。今後、100Mbpsの伝送速度を提供するFDDIや、VMTPあるいはXTPなどの高速トランスポートプロトコル[5,6]が普及するにつれ、むしろスタブレベルのコストがRPCの性能を決定する要因になることが考えられる。

このような背景から、スタブレベルにおけるコストを低減するための工夫を行なったRPCシステムも既に提案されている。NCSでは、送信側は内部データ表現を用いてRPCメッセージを組み立て、受信側において必要な場合にのみデータ表現の変換を行なう[7]。同様に、Amoeba分散OSでは、クライアントは内部データ表現を用いてRPCメッセージを送受し、サーバが必要なデータ表現変換を行なっている[8]。しかし、これらのRPCシステムでは、整数型や実数型などの基本データ型における不要なデータ表現変換を省いているのみであり、構造体や配列などの複合データ型における要素配置については標準のデータ表現を導入しているため、充分な効果を得るに至っていない。

筆者らは、データ表現変換を可能な限り避けることによりスタブレベルのコストを抑える「適応的データ表現変換」を提案し、これに基づく異機種間RPCシステム "KoKo" を開発してきた[4, 9, 10]。適応的データ表現変換では、RPCに先立って、クライアントとサーバの使用する内部データ表現が通知され、RPCごとに最適なデータ表現が用いられる。双方が同一の内部データ表現を持つ場合にはデータ表現変換を一切行なわずにRPCメッセージを送受する。たとえクライアントとサーバが異なる内部データ表現を使用する計算機上で稼働する場合であっても、その差異がRPCの引数に影響しない限り、内部データ表現を用いて通信を行なう。データ表現変換が必要な場合には、クライアントとサーバが稼働する計算機の処理能力等を考慮し、変換コストを最小とする変換法を決定する。

本稿では、適応的データ表現変換に基づくRPCシステム"KoKo"の実装と評価について報告する。まず、適応的データ表現変換について述べる。次に、KoKoの実装に関して、その構成要素ごとに詳細に述べる。更に、システム全体のパフォーマンスとRPCメッセージの組立て時間についてKoKoの性能評価を行ない、他のRPCシステムと比較する。最後に、適応的データ表現変換について考察する。

2. 適応的データ表現変換

適応的データ表現変換においては、RPCに先立ち、サーバとクライアント間で内部データ表現のネゴシエーションを行なってデータ表現変換の要否を判定し、必要な場合にのみ最小のコストで変換を行なう。以下に、データ表現のネゴシエーションとRPCごとのデータ表現変換の詳細を述べ

る。

2.1 データ表現のネゴシエーション方式

(1) ネームサーバによるネゴシエーション

クライアントサーバ・モデルに基づくRPCシステムにおいては、サーバが、その論理名と識別子とをネームサーバに登録し、クライアントがサーバの論理名をキーとしてサーバの識別子をネームサーバに問い合わせることで、クライアントとサーバ間の論理的な結合が行なわれる[11]。適応的データ表現変換では、この結合機構を拡張して、次のような手順で内部データ表現に関するネゴシエーションを行なう。

①サーバが論理名と識別子をネームサーバに登録する際、サーバの使用する内部データ表現を同時に登録する。

②クライアントによるサーバの検索時に、サーバの論理名とともにクライアントの使用する内部データ表現をネームサーバに通知する。

③ネームサーバでは、両者の内部データ表現と、各々が稼働する計算機の実効処理能力とを考慮して通信に使用するデータ表現を決定し、サーバ識別子とともにクライアントへ通知する。サーバ側でデータ表現の変換を行なう場合、最初のRPCの直前に、使用するデータ表現とクライアントの識別子とをサーバに通知する。

(2) ネゴシエーションで使用するデータ表現のパラメータ

適応的データ表現変換では、文字列、16/32ビット整数、32/64ビット実数の基本データ型と、配列と構造体の複合データ型を扱うこととする。これらのデータ型の内部データ表現は、表1に示す6つのパラメータで識別することができる。最初の4つのパラメータで基本データ型の表現法が規定される。例えば、16/32ビット整数の内部データ表現は計算機のバイトオーダによって定まり、Big-Endianを採用する計算機では上位バイトが低位アドレスに置かれ、Little-Endianの計算機では高位アドレスに配置される。また、32/64ビット実数の内部データ表現は、多くの計算機で用いられるIEEE-754フォーマットやDEC独自のフォーマットなどの形式とバイトオーダによって定められる。

最後の2つのパラメータと以下に示す規則により複合データ型における要素の配置が決定される。

① 各要素の開始アドレスは、その要素のアライメント値の倍数である。

表1 データ表現のパラメータ

パラメータ	値
文字列	ASCII, EBCDIC.
バイトオーダ	Big-Endian, Little-Endian.
32ビット実数の形式	IEEE-754, DEC-F.
64ビット実数の形式	IEEE-754, DEC-D, DEC-G.
アライメント要求	1, 2, 4あるいは8.
構造体における最小アライメント要求	1あるいは2.

② 基本データ型のアライメント値は、そのデータ型の大きさと「アライメント要求」の小さい方である。

③ 構造体のアライメント値は、その構造体に含まれる各要素のアライメント値の最大値と「構造体における最小アライメント要求」の大きい方である。構造体における最小アライメント要求とは構造体全体のアライメントを定めるもので、例えば、SUN3では2であるため、"struct {char a;}" のアライメント値は2となる。

④ 構造体の末尾には、大きさがその構造体のアライメント値の倍数になるようパディングが行なわれる。

⑤ 配列のアライメント値は、その配列の要素のアライメント値である。

2.2 RPCごとのデータ表現変換の実現法

(1) 三種類の形態のデータ表現変換

ネゴシエーションにより決定されたデータ表現に基づき、スタブにおいて以下の三種類の形態のデータ表現変換を行なう。

① 変換なし。

② クライアントあるいはサーバの一方による、相手側内部データ表現への一段階の変換。

③ クライアントおよびサーバの両方による、ネゴシエーションにより決定されたデータ表現への変換。

①の形態は、RPCの引数に関して、クライアントとサーバが同じ内部データ表現を用いている場合に発生する。

RPCの引数の内部データ表現が、クライアントとサーバとで異なる場合、データ表現変換は避けられない。この時、可能な限り②の形態を選択し、変換回数を最小の1回としている。計算機の内部データ表現が極端に異なり、一方が他方の内部データ表現に変換するのが得策でない場合には③の形態を選択する。

(2) RPCの引数のデータ型を考慮した変換

クライアントとサーバとが異なる内部データ表

現を使用する場合でも、その差異がRPCの引数に影響しない限り、内部データ表現を用いて通信を行なう。例えば、"struct {char a; long b;}"の場合には、バイトオーダとアライメント要求を考慮する必要があるが、"struct {char a, reserved[3]; long b;}"の場合には、いかなる内部データ表現を持つ計算機においてもパディングを生じないため、アライメント要求を考慮する必要がない。後者のデータ型に関する限り、SUN3とSUN4あるいはVAXと386マシンは、同一の内部データ表現を持つと考えてよい。適応的データ表現変換では、RPCの引数のデータ型を解析し、必要最小限の変換のみを行なうスタブを生成する。

3. 適応的データ表現変換に基づくRPCシステム "KoKo" の実装

適応的データ表現変換に基づく、高速な異機種間RPCシステム "KoKo" の構成を図1に示す。システムは、スタブ、スタブジェネレータ(KiG: KoKo Interface Generator)、異機種間通信ライブラリ(HCLib: Heterogeneous Communication Library)、ネームサーバ(KNS: KoKo Name Server)から構成される。以下に、KoKoの各構成要素について述べる。

3.1 スタブジェネレータKiGとスタブ

KiG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。サーバのインタフェース仕様は、RPCの引数に使用されるデータ構造を定義する「型定義ファイル」と、サーバの提供する手続きを定義する「インタフェース定義ファイル」からなる。本システムでは、C言語による開発環境を想定しているため、型定義ファイルはC言語により記述される。一方、インタフェース定義ファイルでは、引数の入出力関係を明示するため、Pascal風の記法を採用している。

スタブの生成において、KiGは、RPCの引数のデータ構造を解析し、データ表現変換において考慮すべきデータ表現のパラメータを決定する。そのパラメータに対して、内部データ表現とネゴシエーションにより決定されたデータ表現とを比較するif()文を生成する。そのif()文は、変換が不要な場合、引数全体をそのままRPCメッセージにコピーし、変換が必要な場合には、個々の要素を変換しつつRPCメッセージを組み立てる。図2に、"struct {char a[2]; long b;}"を組み立てるスタブの例を示す。この例では、クライアントとサーバが同一の内部データ表現を持つ場合だけでなく、バイトオーダが同一で要素のアライメントがともに2以

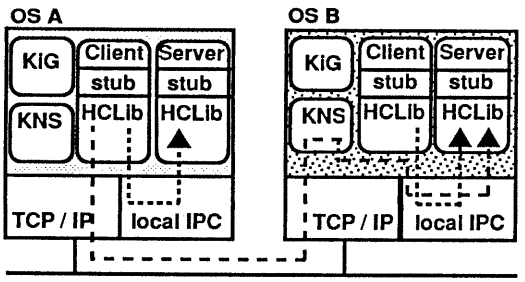


図1 システム構成

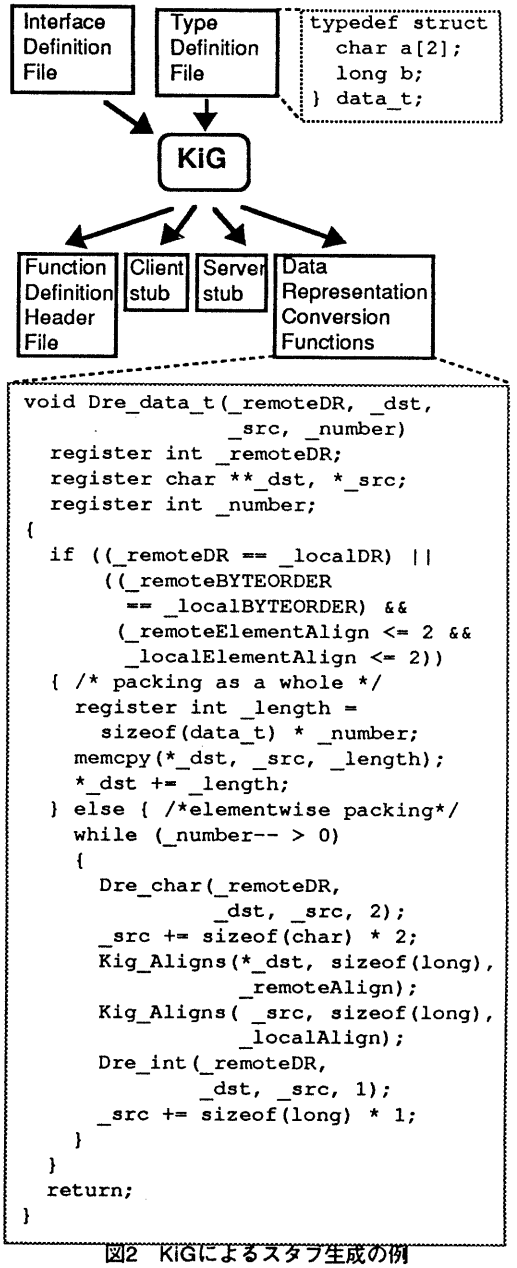


図2 KiGによるスタブ生成の例

下の場合にもデータ表現変換を行なわない。

3.2 異機種間通信ライブラリHCLib

HCLibは、計算機内通信とネットワーク通信の差異や、OSによる通信機構の違いを意識させない統一的なプログラミングインタフェースによるIPC機能を、スタブに対して提供する。また、データ表現のネゴシエーションをスタブから隠蔽している。提供関数としては、クライアントとサーバとの論理的な結合のために、サーバを登録する"Hc_CheckIn()"、その登録を削除する"Hc_CheckOut()" およびサーバを検索する "Hc_LookUp()" を準備している。また、RPCメッセージの転送用として、RPCメッセージの送受信を行なう"Hc_Send()", "Hc_Receive()" およびクライアント側において応答を持つRPCで使用される"Hc_Rpc()" を用意している。

表1に示した内部データ表現のパラメータは、ひとつの16ビット整数に符号化される。"Hc_CheckIn()" は、サーバの論理名等とともにサーバの内部データ表現をKNSへ登録する。"Hc_LookUp()" は、サーバの論理名とクライアントの内部データ表現をKNSへ通知し、ネゴシエーションの結果のデータ表現を受け取る。サーバ識別子、サーバ計算機のIPアドレスおよびネゴシエーションの結果のデータ表現は、HCLibで保持され、スタブは必要に応じて "Hc_GetRemoteDR()" によりそれを参照する。

"Hc_Send()", "Hc_Receive()" および "Hc_Rpc()" は、OSが提供する通信機構を用いて実現する。計算機内通信については、個々のOSに最適化された通信機構により実現している。SUN OSの場合はUNIXドメインのデータグラムソケットと共有メモリを組み合せた方法を用いた。また、ネットワーク通信については、広く利用されているTCP/IPを採用している。

3.3 ネームサーバKNS

KNSは、各計算機にひとつずつユーザプロセスとして走行し、クライアントとサーバの結合、データ表現のネゴシエーション、およびネットワーク通信のRPCメッセージの中継を行なう。

KNSは、その計算機上で動作するサーバの論理名、サーバ識別子、内部データ表現を管理している。クライアントは、ローカル計算機上のKNSに対してサーバの検索要求を行なう。サーバが、他の計算機上で稼働している場合、クライアント側計算機のKNSからサーバ側計算機のKNSに対して検索要求を転送し、サーバ側KNSにおいてクライアントとサーバの論理的結合並びにデータ表現のネゴシエーションを行なう。

ネットワーク通信のRPCメッセージは、サーバ側計算機上のKNSにより中継される。KNSによる中継は付加的なオーバヘッドを伴うが、サーバでは計算機内通信のみによりRPC要求を待つことができるためHCLibの実装を簡略化することができる。

3.4 RPC手順の例

図3に、サーバスタブでデータ表現変換を行なうことを想定した場合の、RPC手順の例を示す。

① サーバが、論理名、サーバ識別子(SID)、内部データ表現(SDR)を、ローカル計算機上のKNSへ登録する。

② クライアントが、サーバの論理名、ホスト名、クライアント識別子(CID)およびクライアントの内部データ表現(CDR)を用いてSIDを検索する。

③ クライアント側KNSが、サーバ側KNSへTCPコネクションを確立し、サーバの論理名、CID、CDRおよびクライアント側計算機の実効処理能力(CMIPS)を通知する。

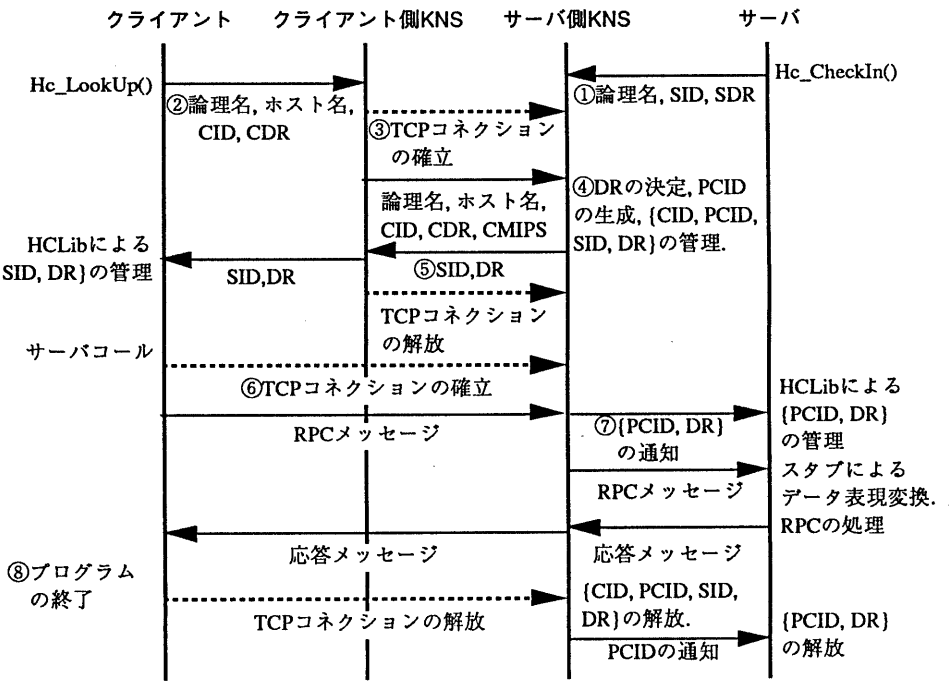
④ サーバ側KNSは、サーバ識別子を検索し、そのRPCにおいて使用するデータ表現(DR)を決定する。サーバ側KNSは、クライアントの"proxy"としてクライアントとサーバとの通信を中継するため、proxyの識別子(PCID)を割り当て、CID、PCID、SID、DRの対応を記録する。

⑤ SIDとDRがクライアント側KNSを通してクライアントに通知される。クライアントのHCLibが両者の対応を保持する。

⑥ サーバコールが発行されると、クライアントとサーバ側KNSとの間にTCPコネクションが張られる。この例では、DRはCDRと等しいため、クライアントのスタブは内部データ表現を用いてRPCメッセージを組み立てる。

⑦ サーバ側KNSはPCIDとDRとをサーバに通知し、両者の対応をサーバのHCLibが保持する。RPCの引数の変換は、サーバスタブでなされる。

⑧ クライアントが実行を終了すると、TCPコネクションが切断される。KNSは、このイベントをサーバに通知し、KNSおよびサーバで保持されていたクライアントに関する情報が解放される。



(注) SID, CID, PCID: サーバ、クライアント、proxyの識別子
SDR, CDR, DR: サーバ、クライアント、ネゴシエートされたデータ表現
CMIPS: クライアント計算機の実行処理能力

図3 RPC手順の例

4. KoKoの性能評価

本章では、KoKoの性能評価を示すことを目的として、システム全体のパフォーマンスとRPCメッセージの組立て時間を測定した結果について述べる。他のRPCシステムと比較するため、SUN RPC (SUN OS 4.0)とNCS (SUN OS用リリース1.5)を取り上げ、同様の測定を行なった結果についても、併せて述べる。

以下の評価では、バイト、整数および "struct {char a[4]; long b;}" の配列を入力引数とする三種類のRPCを提供するサーバを使用した。いずれのRPCも出力引数としては、ひとつの32ビット整数を持っている。サーバは、RPC要求を受け付けると直ちに応答を返す。すべての測定は、10Mbpsイーサネットに接続された2台のSUN 3/260（主記憶24Mバイト）を使用して行なった。

4.1 システムパフォーマンス

システム全体のパフォーマンスの指標として、クライアント側での実時間によりRPCの応答時間を測定した。計算機内通信のRPCと、ネットワーク通信のRPCの場合について、RPCの引数の大きさを変化させ、実測した結果を表2に示す。

KoKoにおいては、同機種間通信では内部データ表現のまま通信を行なうため、引数のデータ型に関わりなく同一の応答時間を示している。これは、構造体などの複雑なデータ型に対しても高速なRPCを実現できることを示している。

SUN RPCにおいては、標準のデータ表現であるXDRが、SUNのアーキテクチャと同一のBig-Endianを採用しているため、実際のバイトオーダの変換は行なわれていない。整数と構造体に見られる応答時間の増加は、引数の各要素をたどる処理と、RPCメッセージの組立て関数を呼び出すオーバヘッドによるものと考えられる。

NCSにおいても、同機種間通信における基本データ型の不要なデータ表現変換は省かれるため、バイトオーダの変換は行なわれていない。また、基本データ型の配列については各要素をたどらず、一括してRPCメッセージにコピーしているため、整数配列の転送でのオーバヘッドは見られない。しかし、構造体については要素の標準的な配置法を導入しているため、オーバヘッドが生じている。

また、KoKoでは、計算機内通信に個々のOSに最適化した方法を用いているため、ネットワーク

表2　RPCの応答時間(単位ミリ秒)

		計算機内通信			ネットワーク通信(同機種)		
		2KB	8KB	16KB	2KB	8KB	16KB
KoKo	バイト列/整数/構造体はすべて同一	14.8	18.7	25.3	29.7	54.8	85.3
SUN RPC	バイト列	10.2	25.9	46.9	12.8	30.1	53.4
	整数	30.6	101.6	204.4	30.6	79.9	147.4
	構造体	95.6	400.2	737.8	73.8	262.9	519.0
NCS RPC	バイト列	23.4	50.2	87.8	20.5	44.9	81.9
	整数	26.0	55.8	96.6	21.6	50.6	79.4
	構造体	35.3	92.9	170.6	29.4	79.4	148.8

注)　2KB, 8KB, 16KBはRPCの引数の大きさ

通信のRPCに比較して高速になっている。SUN RPCとNCSでは、計算機内通信とネットワーク通信とで同一の通信機構を採用しているためほぼ同一の応答速度を示している。ネットワークRPCが若干高速なのは、RPCメッセージの組立て／分解がクライアント計算機とサーバ計算機とに分散されるためと考えられる。

データ表現変換のコストを評価するため、KoKoにおいて、クライアント側でバイトオーダの変換を行なった場合の応答時間を表3に示す。表2との値と比較して、構造体における応答時間の増加が大きく、データ表現変換のコストが無視できないものであることが分かる。

4.2　RPCメッセージの組立て時間

適応的データ表現変換の効果をより詳細に調べるため、クライアントスタブにおけるRPCメッセージの組立て時間を実測した。RPCの引数の大きさを変化させた場合の測定結果を表4に示す。この結果が示すように、一切のデータ表現変換が不要なバイト列の場合には、KoKo, SUN RPC, NCSのいずれにおいても、スタブレベルにおけるコストは極めて低く、トランスポートレベルのコストに比較して無視できる。一方、構造体をRPCメッセージに組み立てる場合は、基本データ型の場合と比較してスタブレベルのコストが大きい。このことから、高速なRPCを実現するためには、構造体を可能な限りそのまま扱うことが有効であることが分かる。

ここでは、クライアントスタブにおけるRPCメッセージの組立て時間のみを比較したが、SUN RPC およびNCSについてはサーバスタブにおいても同等の分解時間が必要となる。適応的データ表現変換では、たとえ変換が必要な場合においても、クライアントスタブがサーバの内部データ表現に変換することにより、サーバスタブにおいては変換を必要としない。

表3　KoKoにおいてデータ表現変換を行なった場合の応答時間（単位ミリ秒）

	2KB	8KB	16KB
整数	30.0	62.7	102.0
構造体	43.4	109.4	201.6

注)　2KB, 8KB, 16KBはRPCの引数の大きさ

表4　スタブにおけるRPCメッセージの組立て時間（単位ミリ秒）

		2KB	8KB	16KB
KoKo	バイト列/整数/構造体はすべて同一	0.3	1.2	2.4
SUN RPC	バイト列	0.5	1.8	3.8
	整数	6.5	25.8	51.9
	構造体	23.4	93.8	187.4
NCS	バイト列	0.9	1.8	3.0
	整数	0.9	1.8	3.0
	構造体	4.5	16.4	32.2

注)　2KB, 8KB, 16KBはRPCの引数の大きさ

5.　考察

(1)　スタブレベルにおけるコスト

上述の実測結果から、RPCの引数が複雑な構造を持つ場合、スタブレベルにおけるデータ表現変換のコストが、RPC全体の性能に対して支配的な影響を及ぼすことが分かる。RPCシステムが、標準のデータ表現を採用する場合、高速なトランスポートプロトコルや高速なネットワークの普及につれ、スタブレベルがRPCシステムの性能を決定するボトルネックになると考えられる。

(2)　内部データ表現によるデータ転送の高速性

RPCの引数がバイト列の場合には、RPCメッセージの組立て／分解に要するコストは充分に小さい。適応的データ表現変換では、データ表現変換のコストが大きい構造体などの複雑なデータ型を、内部データ表現のままでRPCメッセージに組み立てることにより、任意のデータ型の引数に対するスタブレベルのコストをバイト列の場合のコ

ストに抑えている。

NCSのように、基本データ型における不要なデータ表現変換を取り除くシステムであっても、複合データ型における要素の配置に標準の形式を導入したものでは、個々の要素にアクセスするコストが高く、充分な効果が得られない。

(3) データ表現に関するネゴシエーションの利点

データ表現に関するネゴシエーションを行なうことの利点のひとつは、変換が必要な場合にも、処理能力の高い側で変換を行なうことによりコストを最小化できることである。NCSのようにRPCメッセージの送信側、あるいはAmoebaのようにサーバ側という形で、データ表現変換を行なう側を固定する方法は必ずしも最適でない。

また、一方が他方の内部データ表現に直接変換することが困難な場合にも、標準のデータ表現を導入することができる。

さらに、変換ルーチンの増大を防ぐことが可能である。例えば、EBCDICやDECの実数の形式などのような特殊な内部データ表現の変換ルーチンを、それを採用する計算機のみに持たせることが可能となる。

(4) データ表現のパラメータ

現在のところ、表1に示したパラメータによりすべての計算機の内部データ表現を特定しているが、これらのパラメータだけでは識別できない内部データ表現を持つ計算機が出現した場合にも、いくつかのパラメータを追加することで対処可能と考えている。極めて特殊な内部データ表現を持つものに対しては、計算機システムの名前そのものをデータ表現ネゴシエーションのパラメータに加えることも可能である。

(5) スタブの構成法

SUN RPCにおけるスタブは、複合データ型の組立て／分解を行なう場合、プログラマが定義したデータ型ごとに組立て／分解のための関数を生成している。一方、NCSにおいては、ひとつのRPCの引数についてひとつの組立て／分解の関数が生成されるだけであり、関数呼出しのオーバヘッドを取り除いている。スタブレベルのコストを低減するためには、このようなスタブの構成法についても留意する必要がある。

6. むすび

本稿では、適応的データ表現変換に基づく、高速な異機種間RPCシステム"KoKo"の実装と評価について述べた。適応的データ表現変換は、RPCに先立ってデータ表現に関するネゴシエーションを行ない、不要なデータ表現変換を取り除くことにより、RPCメッセージの組立て／分解に伴うオーバヘッドを最小化し、RPCの高速化を図るものである。これまでに、KoKoをSUN3, SUN4ワークステーションとMacintosh上に実装しているが、その性能評価と、既存RPCシステムとの比較を行ない、適応的データ表現変換の有効性を示した。光ファイバを伝送路とし、高速トランスポートプロトコルを採用する次世代のネットワーク環境においては、本方式の有用性が一層高まるものと考えられる。

最後に、日頃ご指導戴くKDD研究所小野所長、浦野次長に感謝する。

参考文献

[1] "Networking Programming," SunMicrosystems Inc., 1988.

[2] Accetta, M., Baron, R., Golub, D., Rashid, R., Tevanian, A., and Young, M.: "Mach: A New Kernel Foundation for UNIX Development," In Proceedings of the Summer 1986 USENIX Conference, 1986.

[3] Liskov, B.: "Distributed Programming in Argus," Communications of the ACM Vol. 31, No. 3, Mar. 1988.

[4] 加藤, 藤長, 杉山: "適応的にデータ表現変換を行なうリモート・プロシジャ・コールの検討", 情報処理学会第40回全国大会, 5G-2, Mar. 1990.

[5] Cheriton, D. R. and Williamson, C.: "VMTP as the Transport Layer for High-Performance Distributed Systems," IEEE Communications Magazine Vol. .27, No. 6, June 1989.

[6] Doeringer, W. A., Dykeman, D., Kaiserswerth, M., Meister, B. W., Rudin, H., and Williamson, R.: "A Survey of Light-Weight Transport Protocols for Hight-Speed Networks," IEEE Transactions on Communications Vol. 38, No. 11, Nov. 1990.

[7] Kong, M.: "Network Computing System Reference Manual," Prentice Hall, 1990.

[8] van Rossum, G.: "AIL - A Class-Oriented Stub Generator for Amoeba," In Proceedings of the Workshop on Experience with Distributed Systmes, Springer Verlag, 1989.

[9] 藤長, 杉山, 加藤: "異機種間リモートプロシジャコールシステム'KoKo'の設計", 情報処理学会第41回全国大会, 7Q-1, Sept. 1990.

[10] 藤長, 加藤: "リモートプロシジャコールにおける適応的データ表現変換の効果", 情報処理学会第42回全国大会, 2S-7, Mar. 1991.

[11] Birrell, A. D. and Nelson, J. B.: "Implementing Remote Procedure Calls," ACM Transactions on Computer Systems Vol.2, No.1, Feb. 1984.