システムソフトウェアと
オペレーティング・システム 79－2
（1998．8．6）

メモリベース通信を用いたRPCの実装

亀沢寛之　松本　尚　平木　敬

当研究室で開発されている汎用超並列オペレーティングシステム(OS) SSS-CORE [6][松本,'94]の基本通信システムであるメモリベース通信を用いて遠隔手続き呼び出し(RPC)を実装した。RPCは既に基本技術となっているが、メモリベース通信は(1)通信相手のプロセス空間に直接データを書き込む事が出来る。(2)セグメント単位で通信を保証する。(3)非同期的な動作に対応するように設計されている。といった点でRPCとも相性が良く、「コピー回数が少なく、戻り値付き非同期型に対応し、必ず一回だけ実行する」RPCを自然な形でImplementできる。本論文では、メモリベース通信を用いたのClient−Server applicationの実装方法について考察した後、SUNRPC4.0のを基にしたRPCのSSS-CORE上への実装を行なう。実験としてSunOS UDP ,SSS-CORE UDP/MBCFで性能の測定／比較を行った。

RPC implemented on Memory Based Control Facility

HIROYUKI KAMESAWA , TAKASHI MATSUMOTO and KEI HIRAKI

We implemented Remote Procedure Call (RPC) library based on Memory Based Control Facility (MBCF). MBCF is a principal comunication system of our massively parallel operationg system(OS) SSS-CORE [6]Matsumoto,'94. Today,implementaion of RPC is not a hot topic. But MBCF has strong features (1)MBCF enables one process to write data directly into another process's memory space. (2)guarantees transaction of data segments. (3)is designed to work well with asynchronous data transaction. These features are useful to implement "reduced copy", "work asynchronously with returning results","exactly once execution" RPC.In this paper,we discuss implimentation technique of Client−Server application using MBCF and implement RPC library on SUNRPC4.0 on SSS-CORE. we compare performance of RPC on SunOS UDP, SSS-CORE UDP, SSS-CORE MBCF.

1. はじめに

Birrel と Nelson によるによる最初の遠隔手続き呼出し(RPC)の提案 [7][1984] 以来、80年代から90年代にかけてさかんにRPCの提案、実装が行われた。RPCはリモートにメッセージを送信して計算を依頼する際にメッセージの作成、送信、受信、結果の解釈をユーザから隠蔽して意識させないようにする事で、あたかもローカルの手続き呼び出しを行うようにリモート計算を行えるようにするソフトウェアである。実装されたRPCのうち多くのものはUDP/IPベースであり、専用の通信プロトコルを用いた実装も存在した。LANでの実装を考える場合、パケットの紛失はほとんどないと考えられるため、コネクションレスの通信プロトコル(UDP etc..)を用いるものがほとんどであった。IP(UDP)を基本プロトコルとする実現には

- 既に実現されているプロトコルを用いて実装する事が出来、作業を軽減できる。
- ほとんどのUNIXシステムで送受信できる
- 既存のネットワークを介してパケットを送信できる。

といった利点があったが、コストが大きくなるという欠点があった。このコストはIP(UDP)がLAN向けの軽い通信プロトコルとして設計／実装されているもので無いという点が原因であり、性能面から見たUDP/IPベースのRPC実装の主眼は、ヘッダの生成コストの軽減、、不用なコピーの減少などにあった。専用プロトコルを用いた実装では上記の欠点は解消され、IPを使用する際の不都合からも逃れられるが、利点も同時に失う事になり、実装コストもかかるので広く使用されたものは少なかった。結局、現在最も広く用いられているRPC protocolはNFS/NISを使用するのに用いられるUDP/IPベースのものとなっているが、高速、低コストで汎用性のあるRPCの実装は強く望まれている。

我々の研究室では汎用超並列オペレーティングシステムSSS-COREを開発している。メモリベース通信は低コストでのリモートメモリアクセスを可能にする

SSS-COREの通信機構である。メモリベース通信は低コストで信頼性のある通信レイヤーであり、その大きな特徴として完全にソフトウェアによる実装がなされており、専用ハードウェアを必要としない点があげられる。

本論文ではLANによるワークステーションクラスタ上に実装されたSSS-CORE [8]上でメモリベース通信を用いたRPCの実装／実験を示す。並びにSSS-CORE上でのメモリベース通信を用いたClient−Server型アプリケーションの実装方針を示す。メモリベース通信を利用し、特徴を生かした実装を行う事により、低コスト、高信頼性を持つRPCを完全にソフトウェアで実現できる。本論文では、SUNRPC4.0[Sun Microsystems]の移植をベースに、メモリベース通信を用いたRPCとUDP/TCP上のRPCの実装を同時に行った。これらの実装に対して性能の測定、他プロトコルによるRPCとの性能の比較を行った。

2. メモリベース通信

まず始めにメモリベース通信の特徴について述べた後、メモリベース通信の実際について述べる。最後に測定されているメモリベース通信のRound-trip LatencyとPeak Band widthを示す。

2.1 メモリベース通信の特徴

メモリベース通信は低コストのリモートメモリアクセス機構であり、SSS-COREのワークステーションクラスタ上で完全にソフトウェアによって実装されている。メモリベース通信の特徴としては以下のようなものが挙げられる。

- リモートメモリへの直接アクセス
- 軽い保護
- 完全なソフトウェアによる実装
- 低コストな呼び出し
- リモートメモリの高機能操作
- 順序序管理／到着保証

メモリベース通信ではこれらを、

- 論理アドレス空間、MMU,TLB利用した軽い保護
- メモリベース通信専用の軽いシステムコール
- ユーザ空間への直接書き込みによるコピー回数の減少
- シーケンスナンバーの管理と効率的なAck回収

によって実現している。☆

2.2 メモリベース通信の実際

メモリベース通信のコネクション、仮想化、送受信、高機能関数について述べる。まず、いくつかの用語について説明を行う。

ノードID SSS-COREの各計算ノードを特定するID number。

タスクID SSS-COREでは、一台のプロセッサに割りあてられる実行の軌跡(命令流)をスレッドと呼び、複数のスレッドで一つのプロセスを形成する。このプロセスを形成するスレッドのうち論理的にも物理的にも完全にメモリ空間を共有するスレッドがタスクを構成する。タスクのIDはノード内で一意であるので、ノードID + タスクIDでメモリベース通信を用いて操作するアドレスを指定できる。

Accesskey TASK毎に設定されるメモリ操作許可チェックのためのkey number。メモリベース通信における論理アドレス⇔物理アドレス変換による保護に加えての2重のチェックとして作用する。

Permission Accesskeyを設定する際に同時に指定するリモートTASKの自TASKに対するリモートメモリ操作に対するPermission。現在、read write read,write。

TASKQUE 各TASK毎に存在するユニークな通信キュー。TASKQUEの使用、不使用はそのownerが宣言できる。TASKQUEの使用を宣言した場合は、他のTASKはTASKQUEに対して自由にメッセージを送信できる。このTASKQUEでの受信と同時に送信元との間にコネクションが張られる。

論理コネクション 2つのTASK間で互いに張られるメモリベース通信用のコネクション。TASK側からは論理TASK番号という形で見える事になり、各TASKは論理タスク番号を介してメモリベース通信を発行する。

論理タスク番号 論理コネクションを指定するID番号。ノードID＋タスクIDに対する仮想化であり、この仮想化により相手タスクのマイグレーションにも対応する。

ACKの確認 それぞれのメモリベース通信を行う際に、ACK Fieldをユーザが指定する事が出来る。このFieldを指示する事で、リモートに対して明示的にACKを要求できる。(メモリベース通信はパケットの順序管理を行うので通常は1パケット毎にACKが帰ることはない。)

論理コネクションの確立手順は以下の2通りの方法が用意されている。

− 通信するタスク同士があらかじめ通信相手のノードID, タスクID、Accesskeyを知って

☆ 詳細に関しては参考文献 [6] [9] [10]

いる場合。この場合はあらかじめコネクションが張られた状態で各 TASK がスタートする。

- 通信相手の Accesskey を知らない場合。Client–Server タイプのアプリケーションに代表されるが、この場合は通信を開始する側が相手の TASKQUE に通信要求を書き込む。(便宜上、書き込んだ側を Client, 書き込まれた側を Server とする) この Server 側の TASKQUE への書きこみが成立した時点で Client と Server の間に論理コネクションが成立する。このコネクション間での全ての問題について、Client 側が一方的に全責任を負う。

既に存在しない TASK とのコネクションは自動的に破棄される。また、Server が TASKQUE によって自動的に登録された論理コネクションのみユーザから破棄する事が可能となっている。

メモリベース通信でサポートされている機能のうち代表的なものについて説明しておく。

WRITE TASKQUE 前述した リモートタスクの TASKQUE に対する書き込み。書き込む内容はユーザが自由に作成できるため、ある程度の情報をコネクション成立時点で引き渡す事が出来る。

WRITE 書き込み操作。相手先のアドレス、書き込むデータ、データの長さを指定する。

READ 読み出し操作。相手先のアドレス、読み取り用バッファ、データの長さを指定する。

メモリベース FIFO ユーザが自らのアドレス空間内に用意した FIFO に対してメモリベース通信による書き込み操作を行う事が出来る。データはセグメント毎に管理される。自らのアドレス空間内に用意した FIFO であるためデータ到着の確認、覗き見などの操作も容易である。

メモリベース SIGNAL あらかじめユーザが指定したトラップハンドラをユーザが指定した QUEUE にデータを渡しながら起動させる。(いくつか直接引数を渡す事もできる) トラップハンドラはユーザレベルで起動する。ユーザはトラップハンドラ内で QUEUE を確認しながら継続的に要求を処理できる。

不可分メモリ操作 swap,compare and swap, fetch and add などの不可分操作を行う事が出来る。

2.3 メモリベース通信の性能

メモリベース通信の性能が以下の条件で測定されている。

100Base-TX によって接続されたワークステーションクラスタでの測定である。[9]

- Axil 320 model 8.1.1 (Sun SPARCstation 20 compatible,85MHz,SuperSPARC × 1)
- Sun Microsystems Fast Ethernet SBus Adapter 2.0
- 3Com Hub/TP100

以上のハードウェアを用いて round trip latency と peak band width の測定を行った。

Round-trip latency の測定は Remote Write 操作とその ACK の受信を用いて行なった。**表1**

Peak bandwidth は Remote Write を連続発行して行った。**表2**

これらの性能測定の結果から MBCF の性能は Network の性能に非常に接近していると判る。

3. 遠隔手続き呼び出し (RPC)

3.1 同期型 RPC と非同期型 RPC

様々な RPC がこれまで実装されて来たが、プロトコルコンパイラや使用形態ではなく、クライアント／サーバスタブの通信方式に関して分類するとこれらは一般に (1) 同期型 (2) 非同期型の 2 種類に分ける事が出来る。

ここで言う同期型 RPC とは Remote Procedure Call の発行後、返り値を受けとるまで実行を Block するような物の事を指す。同期型 RPC を用いて並列動作するプログラムを実装する場合、RPC を発行する度に新たな実行スレッドを生成するようにしてそれぞれの RPC を別々のスレッドで行う形を取らねばならない。信頼性のない通信プロトコル (UDP など直接の下位レイヤとして使用する際ユーザレベルでリトライやタイムアウト、バッファ管理を行わなければならないためであり、RPC を UNIX 上でライブラリとして実装する時は同期型 RPC として実装する場合が多い。

非同期型 RPC とは RPC 発行後すぐにユーザに制御を戻し、返り値をなんらかの形で後から受けとるようなものを指す。ただし、非同期型 RPC もさらに大きく 2 つに分けられる。返り値無し RPC と返り値付き RPC である。

返り値なし非同期 RPC はリクエストに対してのリプライが無いタイプのもので、リプライが存在しないので信頼性のない下位レイヤー上に構築された場合、リモートでリクエストが実行されたか知る手段が無くなる。このタイプで下位レイヤーに TCP を用いる Sun BatchedRPC ではリクエストを連続発行した後、最後のリクエストのみリプライを要求する。

表1　100Base-TXにおけるリモートライトのRound-trip latency
Table 1　Round-trip latencies of MBCF's Remote Write with 100Base-TX

data size (byte)	4	16	64	256	1024
round-trip latency (μs)	49	54	60.5	88	200

表2　100Base-TXにおけるリモートライトの peak band width
Table 2　Peak bandwidths of MBCF's Remote Write with 100Base-TX

data size (byte)	4	16	64	256	1024	1408
peak band width (Mbyte/s)	0.29	1.06	4.03	8.28	10.86	11.24

返り値付き非同期RPCはリクエストに対するリプライを要求しつつ、非同期にRPCを発行できるようなものを指す。

これらを実装する場合、下位レイヤの信頼性が問題になってくる。UDP上で直接実装しようとした場合、返り値なしならば「実行されたかどうか」を確認できず、返り値つきならば、通信中にリクエストが紛失した場合／リクエストの順序が入れ換わった場合に対処するコードを追加する必要がある。UDP/IP上での専用プロトコルやTCPを使用して信頼性のある下位レイヤを仮定すれば信頼性のある非同期型RPCの実装は可能になるが、多くのRPCの使用形態がLAN上での通信である事を考えるとパケットの紛失、順序の入れ替えが起こる確率はかなり低く、これらに対処するリトライやタイムアウトといったプロトコル処理のコストがその効果に対して比較的大きなものとなってしまうのは否めない。結果、実装の手間というものもあるが、広く使用されているRPCは信頼性の無いプロトコル(UDP)上に構築される同期型RPCという事になる。

3.2　RPCのセマンティクス

同期型／非同期型にも関連してくるが、RPCのセマンティクスは大きく3つに分かれる。

(1)　最低一回実行
(2)　最大一回実行
(3)　一回実行 (一回だけ確実に実行される)

の3つである。

これらの区分が生まれる理由は下位通信レイヤの問題に帰着するが、「信頼性の無い」通信レイヤ上に構築される場合、RPCのスタブ部分でリトライ、タイムアウトなどを実装する事になる。

最大一回実行

クライアントからのRPCリクエストに対してACkを返さない場合、クライアントはServerからのリプライをタイムアウトしながら待つ事になる(「待たない」選択肢もある)。タイムアウト後、リトライをかけたとする。サーバが既に同じRPCメッセージを受けていてこのRPCリクエストに対しては「取っておいた(キャッシュしておいた)以前の実行結果をリプライとして返す」もしくは「既に実行された旨をクライアントに通知してサーバには何も渡さない」場合、「最大一回実行」となる。(クライアントのリトライ回数には上限があるとする。)

しかし、例えば、リクエストの到着よりもサーバのリクエストキャッシュのクリアが早い場合、(UDP/IPを使用する場合、方々を廻ったパケットがある時点で届く事は 考えられる)これを実行してしまう可能性がある。こうなるとキャッシュを用いた最大一回実行を完全に行うのは難しくなる。

最低一回実行クライアントのリトライに対して必ずサーバに引数を渡して実行するようにすれば、「最低一回実行」になる。クライアントはリプライを入手するまでリトライを続ける。

一回実行信頼性のあるプロトコルを用いてリクエストに対するACK, リプライに対するACKを確実に取るようにすれば「一回実行」である。「一回実行」では失敗した際に「一回も実行されなかった」事をユーザが察知できるものとする。1

「一回実行」が理想ではあるが、これには信頼性のある下位レイヤを使用する必要があり、専用プロトコルを実装するにしても、TCPを利用するにしてもそのコストの大きさは無視できない程大きい。

「一回実行」を行なう低コストのRPCを実装するのは既存の通信レイヤを用いた実装では難しいと言える。

4.　メモリベース通信を用いたRPCの実装

2節で示したメモリベース通信を用いて低コストで、一回実行なRPCを実装する方法について考察する。メ

モリベース通信にはいくつかの高機能関数が存在するため。クライアントスタブ、サーバスタブの実装に関してもいくつかのバリエーションが考えられる。

4.1 メモリベース通信による Client−Server

RPC は典型的な Client−Server 型アプリケーションであるが、Server のメモリ空間に対するリモートメモリアクセスの許可をクライアントに出す事は安全上望ましく無い。

この節ではメモリベース通信を用いつつサーバ側の安全性を保証するクライアント／サーバ型通信について考察した後、メモリベース通信を用いた RPC の実装について述べる。

通常、メモリベース通信を用いて Client−Server 通信を開始する手順は以下のようになる。

(1) Client は自タスクに対して accesskey、パーミッションをセットする。

(2) Clinet は Server の TASKQUE に対して Request を書き込む。この書き込みが成功した時点で相手タスクに対する論理タスク番号が Client に与えられ、CLIENT, SERVER 間に論理コネクションが張られる。書き込みの内容はユーザが自由に決定できるのでこの時点で Server 側に必要な情報を与えられる。

(3) Server は TASKQUE を読み出す。Server は CLIENT に対する論理タスク番号と accesskey を手に入れる。この後、Client と Server は送受信を開始する。

Server が Accesskey を Client に教えるか教えないかによって通信の形態は変わって来る。ただ、いずれの場合もクライアントからの登録によって通信が開始されるので、2 節で述べたように通信に関しては Client 側が一切の責任を負う。

Accesskey の有無によるトランザクションの違いは以下のようになる。

Server が Client に accesskey を与える場合 この場合不適切な CLIENT を排除するため、accesskey を与える前に CLIENT をチェックしなければならない。

Server が書きこみ許可を出している場合 Client は Server の address 空間にデータを直接送り込む事ができる。Server は READ Buffer のアドレスを CLIENT に教えてやる。高機能関数を使用する場合、Server は FIFO、Trap Handler などを作成してそのアドレスを与えてやる事もできる。ただしこの場合、不適切なクライアントによって Server タスク自身を破壊される事も考えられるためそれに対する防御策を施しておく必要がある。

Server が読み取り許可を出している場合 Server は Client に読み出すべきメモリのアドレスを与える事が出来る。この場合も Client に与えたくない情報がある場合は (Client は Server タスクのメモリ空間内を自由に読み出せる) 防御策を講じる必要がある。

Server が Client に accesskey を与えない場合 TASKQUE から得られた情報を元に Server は仕事を行う。必要であれば、リモートメモリアクセスで Client タスクから情報を読み出し、Client タスク に情報を書き込む。

この後、トランザクションが終了した時点で Server 側が論理タスク番号を破棄する。

上記の事柄とメモリベース通信の特徴を踏まえると、以下のようなタイプの Server 実装の類型が示される。

1. **単発トランザクションタイプ** TASKQUE に書く際にパッキングされた引数を渡してやる。スタブからの関数呼び出し後、クライアント側にデータを引き渡してコネクションを切る。
   長所 (1)Server から Client に accesskey を引き渡す 事が無い必要が無く実装が容易 (2) コネクションと同時にリクエストを処理できる。
   問題点 (1) 要求がある度にコネクションを張る事になる。(2) クライアント側から送信されるデータのサイズに制限がかかる。

   これらの特徴から、散発的なアクセスがありリクエストデータのサイズがそれ程大きくならない場合に有効であると考えられる。また、リプライが期待されない通信を行う場合は ACK の確認だけ行えば通信の成功は確認できるので効率良く動作できる。この場合、UDP ベースのソフトウェアを移植する際にはコーディングがかなり似通う。IP address をノード ID に、port 番号をタスク ID に起き替えてリクエストを発行させ、リプライの書き込みを待つ事になる。

2. **READ 駆動タイプ** TASKQUE に書き込む際に Client 側リクエストのデータ部のアドレスを渡してやる。Server 側はリモートメモリ読み出しを発行してデータを入手し、処理を行なった後リモートに書き出してやる。
   長所 (1) Server が Clienlt に Accesskey を渡す必

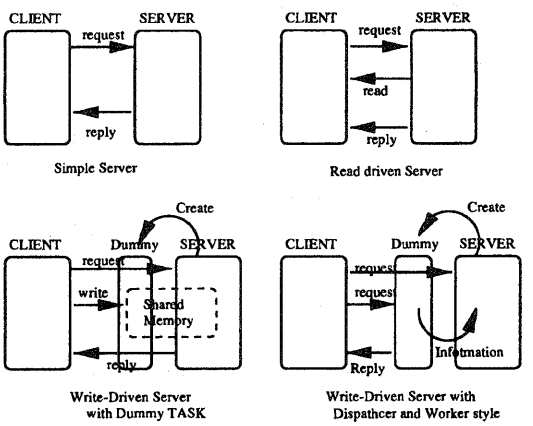

図1 メモリベース通信を用いたClient−Serverアプリケーション

Fig. 1 Client - Server application with MBCF

要が無く、実装が容易である。(2)「1」のタイプと比較して自由な大きさのリクエストデータを扱える。

短所 (1)READ駆動であるので要求到着時点でServerが処理を開始できない

何らかの形で追加リクエストの有無を確認できれば連続的な処理が可能になるため、連続して要求発行される場合、Server側で次のリクエストを予想可能な場合に向いている方式と言える。また、「1」と組み合わせる事で「1」のリクエストデータの大きさの制限を回避するような実装を可能とするオプションになる。

3 **WRITE駆動タイプ** TAKSQUEによるコネクション成立の後、Client側からServerのバッファにリクエストを書き込む。これ以後自由に処理を行い、処理終了後にコネクションを破棄する。この場合ClientにWrite Permissionを出す必要があるためにServerタスクのアドレス空間を守るための対策を講じる必要がある。

3 a **CSMタイプ** この対策の一例として、SSS-COREの同一ノード内のタスク間共有メモリであるクラスタ共有メモリ(CSM:Cluster Shared Memory)を利用する方法が考えられる。CSMはページ管理機構を活用したページ単位のノード内共有メモリであり、低コストで利用可能な共有メモリである。

Serverは共有メモリ用にダミータスク(アドレス空間)を用意してクライアントタスクにはそのダミータスクの名前を返答する。クライアントはもう一つダミータスクとコネクションを張る。

ダミータスクとServerタスクの間で要求受けつけ用データエリアのみをCSMでメモリ共有しておく。こうする事で、本来のメモリベース通信を用いてクライアントから安全にRemote Writeを受けつけることが可能になる。このようにした場合には、ServerはClientから直接干渉を受ける事がなくなる。

長所 (1)ClientはServerの許す範囲である程度eagerにリクエストを送信できる。(2)Client側もほぼ全てのメモリベース通信機能を利用できる。

短所 (1)共有メモリ、Server側バッファの管理などある程度繁雑な管理が必要になる。(2)コネクションの確立の手間が大きい。

という点になる。1、2よりもかなり強力にメモリベース通信を利用したClient−Serverが実現できるのでコネクションを維持して使用されるならばメモリベース通信の基本性能の恩恵を充分に受けられる。

このタイプのServer実装は従来のTCPによるServer実装にかなり近くなる。異なるのはread systemcallの発行ではなく、ユーザレベルでバッファを覗き見しながらServerの動作を制御できる点である。

3 b **独立ダミータスクタイプ** CSMを張らずに生成したダミータスク内でリクエストを処理してしまう方

式も考えられる。この場合、メモリベースシグナルを用いた実装が可能かつ効果的である。

Server は TASKQUE の読み出しを行い、必要ならばダミータスクを生成して Client にソノ名前を与える。ダミータスクはメモリベースシグナル用のトラップハンドラを設定する。

Client とダミータスク間でコネクションを成立させた後、Client は駆動したいルーチンのアドレスを指定してメモリベースシグナルリクエストを送信する。この際に付加できるデータの長さはパケットの最大長の制限を受ける。長所 (1) ダミータスクと Client は互いに全てのメモリベース通信を使用できる。(2) ディスパッチサーバはリクエストの多寡に合わせて動的にダミータスクを増減できる。(ダミータスク⇔ディスパッチサーバ間での通信が必要) (3) 強力なメモリベースシグナルを使用できる。

短所 (1) コネクションの確立の手間が大きい。

(2) タスクの生成後、クライアントからのリクエスト要求が可能となる。

注意する点としては、ユーザがきちんとタスクの生成／除去を行う必要がある点があげられる。特に短期的に長時間のトランザクションが複数の Client から集中する場合を考慮するならばダミータスク数の上限を適切に決めておく必要がある。

4.2 メモリベース通信を用いた RPC の実装

これまでに述べたメモリベース通信の特徴を踏まえると、以下のような特徴を持つ RPC を実装するできる。

- 同期型／非同期型双方に対応する
- 一回実行型のセマンティクスを持つ

メモリベース通信の軽さを生かすような実装を行う事で、常に問題とされて来た「信頼性に伴うコスト」に対処する。メモリベース通信の「リモートメモリアクセスであり明示的なレシーブの発行が必要とされない」性格が非同期型の実装をさらに容易になっている。

前節を踏まえると、「低コスト、戻り値付き非同期型に対応し、必ず一回だけ実行する」RPC の実装に最も適応するのは「3 b 独立ダミータスク型」で、メモリベースシグナルを利用する実装であると考えられる。しかし、今回 RPC を実装するにあたって、SUNRPC4.0 をまず SSS-CORE 上の UDP/TCP で実装し、これの UDP ルーチンを参考にメモリベース通信に移植を行った。この為、今回の実装では「3 Write 駆動タイプ」をメモリベース FIFO を使用して実装してある。

この RPC の特徴は以下のようになる。

- 低コストのリクエスト発行が可能でコネクションを維持しての連続リクエスト発行に向いている。
- 信頼できる下位レイヤ上に実装され、「一回実行」セマンティクスを持つ
- Request の発行と Reply の受理を分離する事で非同期的リクエストの発行が可能

また、この実装は SUNRPC4.0 の移植であるので XDR ルーチンを用いてデータのマーシャリング／アンマーシャリングを行っている。

5. 実　　験

ワークテーションクラスタを用いて RPC の Round-trip Time(RTT) を計測をした。使用プログラムはクライアントがあるサイズの String データを送信し、サーバは同じサイズの String データで返信するという単純な物を用いた。コード自体は SUNRPC4.0 のプロトコル・コンパイラに吐かせたコードを SSS-CORE で使用できるよう手を加えて使用している。参考までに 10Base-T を使用した際の SSS-CORE UDP のデータも掲載する。

実験環境は以下の通りである。

- Axil 320 model 8.1.1 (Sun SPARCstation 20 compatible,85MHz,SuperSPARC × 1)
- Sun Microsystems Fast Ethernet SBus Adapter 2.0
- Bay Networks BayStack 350T half-duplex mode **100Base-TX**
- CenterCom Hub/3016TR **10Base-TX**

この実験では 0 byte から 1024 byte の Client の Request に対して同じ長さのデータを持つ Server の Reply を返している。Client と Server が互いに送信している。データ長がゼロの場合でも RPC header(Request 48 byte, Reply 28byte) が転送されている事に注意されたい。10Base-T での実験結果**表3**を見ると、UDP だけを比較しても SSS-CORE の UDP の方が良い値となっているがこれは SSS-CORE の UDP routine 用 system call のいくつかをメモリベース通信で使用されている軽い system call を使用する事によって達成されている。ここで注意を促しておきたい事はメモリベース通信は信頼性のあるプロトコルであり、ACK パケットを飛ばしている (User に見える Ack を要求するように RPC library は書かれている) という事であり、それでも尚、メモリベース通信を使用したリモートメモリアクセスの方が良い値を残しているという事である。

表3 10Base-T における RPC によるデータ転送の Round-trip latency

Table 3 Round-trip latencies of RPC with 10Base-T

data size (byte)	0	64	256	512	1024
SunOS UDP (μs)	778	808	1218	1697	2690
SSS-CORE UDP(μs)	542	679	1072	1574	2614
SSS-CORE MBCF(μs)	457	545	925	1446	2452

表4 100Base-TX における RPC によるデータ転送の Round-trip latency

Table 4 Round-trip latencies of RPC with 100Base-TX

data size (byte)	0	64	256	512	1024
SunOS UDP (μs)	695	764	821	931	1283
SSS-CORE MBCF(μs)	338	360	465	649	1020

100Base-TX のデータ表4と合わせて見ると、特にデータサイズの小さいところで SSS-CORE のの性能比の良い事から、メモリベース通信の軽さが有効に働いている事がわかる。また、計測条件が異なるが、最初に掲載したメモリベース通信の Round-Trip Latancy の表表1と比較すると、RPC の ルーチン自体の持つオーバーヘッドがかなり大きいとわかる。

6. おわりに

SSS-CORE に上でのメモリベース通信を用いた RPC の実装方法を検討し、メモリベース FIFO を用いた実装を行った。その上で実験として、RPC の Round-trip time の計測を行った。その結果、信頼性のある下位レイヤーを持ちつつも充分に軽量な RPC を実装できる事が示された。また、リモートメモリアクセスであるメモリベース通信を用いた安全な Client - Server の構築に関して考察を行った。

今回の実装ではダミータスクとの共有メモリ型の Server 実装となったためメモリベース FIFO を使用したが完全ダミータスク型のメモリベース Signal をを用いた実装も考えられる。今後、メモリベース Signal を用いた実装を行った後、実際に使用されているようなアプリケーションを実装した実験を行いたい。

謝辞 本研究の一部は情報処理振興事業協会 (IPA) が実施している独創的先進的情報技術に係わる研究開発の一環として行われた。

参考文献

1) Sun Microsystems, Inc: RFC 1050: RPC: Remote Procedure Call Protocol specification (1988). Obsoleted by RFC1057 ?][. Status: HISTORIC.

2) Srinivasan, R.: RFC 1831: RPC: Remote Procedure Call Protocol Specification Version 2 (1995). Status: PROPOSED STANDARD.

3) Bloomer, J.: *Power programming with RPC*, O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA (1992).

4) Tay, B. H. and Ananda, A. L.: A Survey of Remote Procedure Calls, *ACM Operating Systems Review*, Vol. 24, No. 3, pp. 68–79 (1990).

5) Ananda, A. L., Tay, B. H. and Koh, E. K.: A Survey of Asynchronous Remote Procedure Calls, *sigops*, Vol. 26, No. 2 (1992).

6) 松本尚, 古荘進一, 平木敬: 汎用超並列オペレーティングシステム SSS−CORE, 日本ソフトウェア科学会第 11 回大会論文集, pp. 13–16 (1994).

7) Birrell, A. and Nelson, B.: Implementing Remote Procedure Calls, *ACM Trans. Computer Systems*, Vol. 2, No. 1, pp. 39–59 (1984).

8) 松本尚, 駒嵐丈人, 渦原茂, 竹岡尚三, 平木敬: 汎用超並列オペレーティングシステム : SSS-CORE — ワークステーションクラスタにおける実現 —, 情報処理学会研究報告 96-OS-73, Vol. 96, No. 79, 情報処理学会, pp. 115–120 (1996).

9) 松本尚, 平木敬: 100BASE-TX によるメモリベース通信の性能評価, コンピュータシステムシンポジウム論文集, 情報処理学会, pp. 101–108 (1997).

10) 松本尚, 平木敬: MBCF:A Protected and Virtualized High-Speed User-Level Memory-Based Communication Facility, *Proc. of the 1998 ICS* (1998).