オペレーティング・システム 45－10
（1989. 11. 11）

広域分散環境における RPC

尾上 淳　光澤 敦　中島 達夫　所 真理雄
慶應義塾大学理工学部

概　要

多くの分散アプリケーションは、クライアント/サーバモデルに基づいて構築されている。その際、クライアント/サーバ間の交信のため、Remote Procedure Call (RPC) が用いられる。ネットワークの発達とともに、広域環境における RPC の重要性が高まりつつある。ここでは、様々なネットワークでのスループットや遅延を考慮しなければいけない。また、システムのスケールの拡大にともない、どのサーバを選択すれば良いか、ネットワークの輻輳をどのように回避するかなどについて考える必要がある。

本研究では、クライアントの要求に対し適切なサーバを選択するメカニズムと広域環境に適したトランスポートメカニズムを持つ RPC を提案し、その設計について議論する。

RPC in a Wide Area Distributed Environment

Atsushi Onoe　Atsushi Mitsuzawa　Tatsuo Nakajima　Mario Tokoro
Department of Computer Science, Keio University

Abstract

Many distributed applications are constructed based on a client-server model. These applications use a remote procedure call to communicate between a client and a server. Current network technologies make RPC suitable in a large-scaled distributed environment more important. Thus, we must deal with throughput and delay in various kinds of networks. Also, we must consider which server is appropriate and how congestion in networks and server must be avoided.

In this paper, we propose a design of RPC for large-scaled distributed environment, which have a mechanism selecting a suitable server and a transport protocol which is appropriate in large-scaled distributed environment.

1 はじめに

大規模なシステムは、多くのサービスを提供できる可能性を持っている。そのとき、アプリケーション間の交信のための通信をどのように扱うかが重要となってくる。つまり、柔軟が高く、効率の良い通信プリミティブを、アプリケーションに提供しなければいけない。そのようなプリミティブとして、Remote Procedure Call (RPC) が存在する。RPC は、分散アプリケーションを構築するために多く用いられているクライアント/サーバモデルに、非常に適している。そのため、RPC を用いることにより、様々な分散アプリケーションの構築が容易になると考えられる。

従来の RPC はローカルネットワーク上のアプリケーションをターゲットとしていたため、広域なインタネット環境に対しては、あまり考慮されていなかった。しかし、現在、日本においても多くの組織間にまたがるインタネットワークの整備が進められつつある [Mur89]。そのような環境での RPC は、従来の RPC では考慮されていな3つの点について考慮する必要がある。

- 複数のサーバのうち適切なサーバを選択する。
- マルチキャストなどを考慮した RPC に適した経路制御を行なう。
- ネットワークでの輻輳の回避を考慮する。

これを実現するため、我々の RPC は、以下の2つのコンポーネントから構成される。

- 名前サーバ
- トランスポート

クライアントは、サーバをアクセスするための情報を名前サーバに問い合わせる。その時、名前サーバは、クライアントの要求に適したサーバのアドレスを返す。また、要求や返答を転送する時、トランスポートレベルのゲートウエイを用いて、その性質に適したデータの転送を行なうことができるようにする。つまり、要求や返答に適した経路制御やネットワークの輻輳の制御をトランスポートレベルで、Point-to-Point に制御することを可能とする。

以下、第2節では、広域環境に適した RPC への要求について述べる。次の第3節では、RPC のモデルについて議論する。第4節では、サーバの選択について述べ、第5節では、トランスポートプロトコルについて述べる。そして、第6節では、我々の RPC のアーキテクチャについて解説する。

2 広域環境に適した RPC への要求

この節では、広域環境に適した RPC が持つべき性質について述べる。はじめに、翻訳サーバの例を通して、広域環境における RPC の性質に関して考える。そして、そのとき、現在の RPC を用いた時、何が問題になるかについて議論する。

2.1 広域環境での分散アプリケーション

まず翻訳を例にとり、広域環境で、複数の翻訳サーバがあるとき、翻訳を依頼するクライアントが、どのようにサーバと通信するのが好ましいかについて議論する。

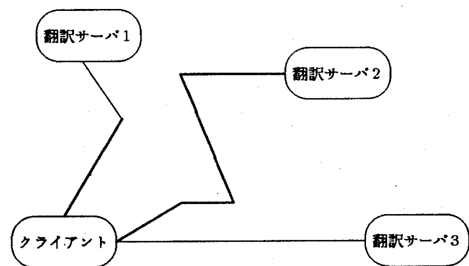

図 1: 翻訳サーバの例

クライアントから見た時、実際の翻訳において最も問題になるのは、翻訳を依頼してから結果が返るまでの応答時間である。つまり、可能な限り短い時間で結果を得ることが望まれる。

翻訳では、多くのデータ転送と計算量を必要とする。そのため、この両方を考慮しないと、十分短い応答時間は期待できない。ここでは、図1のように3つの翻訳サーバが利用可能な場合について考える。クライアントとサーバ1の間のリンクは、たえず混雑していて、データを送るために、多くの時間がかかる。そのため、サーバ1の要求を出すことは、リンクの混雑を増加させることになる。次の、サーバ2との間のリンクは、全く混雑していないので、十分なスループットがあり、遅延もほとんど生じない。しかし、サーバが過負荷となっているので、なかなか結果を返してくれない。そして、サーバ3とは、バンド幅が小さいリンクで接続されていて、その遅延が大きくなるため、早い応答は期待できない。

このとき、各クライアントが、できるだけ短い応答時間で結果を返すようにするための条件は、サーバ及びネットワークにかける負荷を最小限に抑えながら、できるだけ速いリンクにつながった、できるだけ速く処理することができるサーバに要求を送ることである。ここでは、各サーバ

のサービスの処理に影響を与える要因として、計算量と通信量という2つのパラメータを考慮して、適切なサーバを選び、適切な通信のためのポリシを選択する必要がある。我々の RPC ではこの2つのパラメータを考慮したサービスを提供する。

2.2 広域環境での RPC の問題点

ここでは、サーバの選択と大量データの転送の2点から、従来の RPC の問題点に関して検討する。

サーバ選択

従来の多くの RPC は、サーバを選択する際、次の2つの方法のいずれかを用いていた。

- 直接アクセスするサーバのアドレスを指定する。
- ブロードキャストを用いて、サーバのアドレスを検索する。

1番目の方式では、サーバを指定するために直接サーバが存在するホストを指定する。そのため、その時点で最適なサーバを動的に選択するということが困難である。従って、複数のサーバが利用可能なときでも、負荷の高いホストのサーバや遠距離にあるため通信に時間を要するホストのサーバを利用することがある。しかし、広域環境では、全体として同じサービスを提供する複数のサーバが存在するので、通信に要する時間もサーバの存在するホストによって大きく異なる。そのため、システムが、適当なサーバを選択することが重要な問題となる。

2番目の方式では、ブロードキャストを用いることにより、適当なサーバを選択する。つまり、要求を全てのサーバにブロードキャストして、最も早く返答を返したサーバを用いる。しかし、この方法を用いた時、やみくもにブロードキャストするのは、サーバの負荷を増加させるため、好ましくないと考えられる。そのため、一度ブロードキャストを用いてサーバを選択したら、その後はそのサーバをアクセスすることにすれば、負荷を増加させることはない。この方式は、ローカルエリアネットワーク上でのサーバの選択のため、YP (Yellow Page)[Sun86] で用いられている。しかし、この方法は、動的なサーバの負荷の変動に対処できないとか、インタネットワーク上でのブロードキャスト通信の実現は現実的であるとは考えられないなど、広域環境で用いるためには問題がある。

以上の議論より、ネットワークやサーバの負荷の動的な変化に対応できるような、サーバを選択する方法が必要となる。また、サービスの性質から、必要な計算量と通信量を見積もり、サーバ、及び、そこへの経路を選択しなければいけない。

大量データ転送

NFS[Sun89] などのように、大量のデータ転送が必要になる時や、サーバの負荷が重く、結果をなかなか返すことができない時は、要求メッセージの再送の制御が必要となる。例えば、Sun RPC[Sun88] では、要求が送られたかどうかは、返答が返るまで確認することができない。そのため、サーバの負荷が重い時は、何度も再送が繰り返されることになり、負荷をさらに向上させてしまうことになる。また、タイムアウトの間隔より計算時間が長い場合は、再送が循環してしまう可能性がある。そのため、要求メッセージが送られたかどうかは、それに対する確認を用いて調べる必要がある。

Sprite RPC[Wel86] ではこの点は改良されており、要求に対してまず確認応答を返し、それから処理を行う。これによって、クライアント側で要求が受理されたことを、ネットワークのラウンドトリップタイムを利用することにより、確認することが可能になる。また、再送された要求を、再度実行しないように、返答したメッセージをとっておいて、2回以上、要求が実行されることを防ぐことができるようになる。

しかし、以上のような従来の RPC では、ネットワークの輻輳の制御、転送するメッセージの性質の利用などを行なうことができない。特に、メッセージの性質を利用することにより、アプリケーション毎のきめ細かい制御を行なうことが可能となる。

3 RPC のモデル

我々は、RPC における計算をモデル化するため、サービスのモデル化、クライアント/サーバのモデル化、及び、通信のモデル化について検討する必要がある。ここでは、これらについて検討し、我々の RPC の特徴について述べる。

3.1 サービスのモデル化

RPC を実行する際、静的に決定されるパラメータは、各サービスの性質により決定される。ここでは、これを計算量と通信量の2つのパラメータとして定義する。計算量とは、処理の複雑さから推定される値であり、各サービスの処理時間をとして与えられる。つまり、この値にサーバの性能と現在の負荷を加えたものが実際の計算所要時間となる。また、通信量とは、各サービスを実行するために必要なデータの量から推定される値であり、通信コストとして与えられる。ここで、この通信量とネットワークの経路、

及び、負荷から通信時間が決定する。

計算量

計算量は、それぞれのサービスを、同一の計算機上で実行した処理時間として定義する。これは、そのサービスの実行時間の最悪の場合を意味する。

実際の計算時間はサービスの計算量と、それを実行するサーバを実行する計算機の性能、及び、その計算機の負荷により決定される。このうち、最も短い時間で返答を返せると考えられるサーバが選択される。

計算量は、システムの状態により変化しない静的な量である。つまり、各サービスを実現するプログラムの時間複雑度の上限として定義される。

通信量

通信量は、サーバへメッセージを送るのに必要なデータ量とデータタイプにより定義される。実際の転送時間は、通信量、サーバまでの経路、及び、ネットワークの負荷により決定される。このうち、最も早い通信時間のリンクを持つサーバを選択すればよい。実際には、計算時間と通信時間の組み合わせにより、サーバが選択される。

通信量も、システムの状態に依存しない静的な量となる。つまり、転送に必要な、メッセージ量の上限として定義される。

3.2 通信のモデル化

クライアントは、サーバとの通信により用いられる引数のデータタイプの性質を利用して、通信の効率を向上させる。つまり、データタイプの性質から、データを送るのに必要な信頼性の程度を決定し、信頼性が必要ないデータに関しては、メッセージが失われた時の再送やメッセージのシークエンス制御を行なわない。実際には、各要求/返答の転送のため、以下のようなパラメータを設定する。

Best Effort 通信の信頼性に関しては、ネットワーク層に任せる。

Reliable ホストがクラッシュしなければ、メッセージが失われたり、メッセージの順序が入れ替わることがないことを保証する。

Atomic 要求/返答をアトミックに実行する。

また、クライアントの要求により、応答時間に関する制約が存在する時は、指定された時間内に、メッセージが転送されることを保証する。これも、以下の3つのパラメータが存在する。

Best Effort 通信時間に関しては、ネットワーク層に任せる。

Soft Real Time できるだけ指定された時間内に送れるようにする。

Hard Real Time 指定された時間内に送れることを確実に保証する。

これらのパラメータはシステムの動的な状態とは無関係に、RPC の引数と、クライアントの要求のみから選択される。

3.3 ネットワーク、サーバのモデル化

システムの状態は、時間と共に動的に変化する。ここでは、これを、ネットワークとサーバの動的なパラメータとしてモデル化する。ネットワークのパラメータは、サーバまでの経路と各リンクのスループット/遅延により決定される。これから、各サーバまでのスループットと遅延が見積もられる。また、サーバのパラメータは、現在のサーバの負荷から決定される。

ローカルエリアネットワーク上では、通常、スループットが十分大きく、高速であるので、無駄な再送や輻輳が生じる可能性は、非常に小さい。しかし、広域環境を考慮する時は、これらについて無視することは不可能である。つまり、どんなに、サーバの負荷が軽く、そのホストの計算スピードが十分あっても、そこへのリンクのバンド幅が小さいなら、結局、応答時間は悪くなってしまうかも知れない。また、輻輳がおきないように、フローコントロールを用いて、ゲートウェイにおける混雑がおきないようにパケットの量を制限する必要がある。

4 サーバの選択

第2節で述べたようにクライアントが RPC によって要求を出す際に複数のサーバの中から最も適当なものを選択するということが考えられる。ここではその方法について考察する。

4.1 応答時間

同じサービスを提供するサーバが複数あった場合、通常最も速く応答が得られるようなサーバを選択することが適当であると考えられる。また要求を出す際に、ある時間以内に応答が得られないのなら最初から要求を出すことなくエラーを返して欲しいという要求もあり得る。これらの要求を満たすためには応答を得るまでの時間を予測することが必要である。この時間はそのサービスの計算時間及び通信時間の和として与えられる。このうち計算時間は、サービスの計算量・サーバの性能・サーバの負荷に依存し

ており、通信時間はサービスの通信量・ネットワークのスループット・ネットワークの輻輳に依存している。

これらは以下の3つのタイプに分類することができる。

1. サービスに固有の量
   実際にどの程度の量の計算が必要であってそのためにどの程度の量のデータを通信しなければならないかは、サービスごとに特定できる。すなわちこれらはサーバの種類や状態とは無関係にクライアントの要求によって定まる量である。

2. サーバによって定まる量
   サーバの性能はサービスの種類との関係はなく、サーバが存在するホストに固有の一定量である。またネットワークのスループットはクライアントとサーバの間の経路が定まればそれに応じて定まる一定量である。

3. 時間と共に変化する量
   サーバの負荷やネットワークの輻輳は他のソフトウェアや通信に大きく影響を受け、時間と共に変化する可能性のある量である。

各サーバについてこれらの量をなんらかの方法で知ることができれば、それに基づいて要求に対して応答を得るまでの時間をそれぞれのサーバに対して予測することができる。その結果最も適当と思われるサーバを選択することができるのである。つまり、問題はこれらの量をいかにして得るかということになる。

4.2 情報の収集

選択に必要なデータを得るための方法として、2つの方法が考えられる。

1. 実際に選択が必要な時点で集計する方法
2. 必要なデータを別の手段を用いてあらかじめ参照できるようにしておく方法

前者はYPなどのようにブロードキャストを用いてサーバを選択するものである。要求に先だってクライアントから送られたサービスの種類などのデータから各サーバが応答時間を見積り、クライアントはその結果を見て実際の要求を出すサーバを選択する。

この方式の利点はサービスの瞬間のデータを知ることができるので、かなり正確に応答時間を見積ることができることである。また欠点はあるサービスを実行するために、必ず情報収集が必要となるので、そのためのオーバヘッドが生じることである。

このオーバヘッドは特に広域環境ではかなり大きな問題となる。実際にこの様な情報を得るためにはマルチキャストを用いて複数のサーバから情報を得ることになるが、場合によっては実際の要求による通信よりも通信量が多くなることもあり得る。また広域全体にブロードキャストを行うことを避けるためには、どのサーバがサービスを行っているかということはあらかじめ別の手段で知っていなければならない。

後者は分散あるいは集中されたデータベースを検索することによって必要な情報を得るものである。この方式では各サーバに関する情報をある程度まとめて入手することができるのが特徴である。また欠点としては、あらかじめ登録されたデータを参照するため、必ずしも現在の状況を反映していないという点があげられる。

4.3 名前サーバの利用

あるサービスを提供するサーバを知るための方法としては、静的にプログラム中で定める方法、ユーザに明示的に指定させる方法、名前サーバなどのデータベースを検索する方法などがある。これらの中で最も汎用性に富んでいるのは、明らかに名前サーバを用いた方法である。

この方式ではあるサービスに対して、それを処理可能なサーバをデータベースに登録しておき、クライアントからの要求に対し名前サーバがこのデータベースを用いて答える。このデータベースにサーバのホスト名やアドレスだけでなく、サーバの性能などのデータも登録しておけば適当なサーバの選択が可能となる。

その際問題となるのは名前サーバを検索することによるコストの増加である。しかし広域環境では一般にホスト名からそのアドレスを得るためにも名前サーバが利用されているので[Dun86]、名前サーバの検索はいずれにしても必要となるため、コストの増加は生じない。

ここで、静的な情報だけでなく動的な情報も扱えるような名前サーバを利用することを考える[Ish+89]。このような名前サーバにサーバの負荷などの情報を登録しておけばサーバの検索の際にサーバの種類(計算機の power)・その時点での負荷という必要な情報が一度に入手できるため、効率の低下が最小限に抑えられる。

4.4 ネットワーク情報の取得

サーバ選択のとき必要な情報のうち、ネットワークの状態に関する情報は名前サーバにより扱うのは問題である。そのために、各ホストの持つネットワークに対する情報を管理する routing table を参照する。

現在広く使用されている RIP[Hed88] ではネットワークの情報としてサイト間のゲートウェイの数の情報しか伝達

されないのでサーバの選択に用いるためには不十分である。そこでスループット/遅延/信頼性の3次元の情報を伝達する OSPF[Moy89] などのプロトコルを用いてサーバが存在するサイトへのリンクを知れば、それから情報を得ることができる。

ここで問題になるのは経路が複数存在するときである。その場合、経路の選択権はネットワーク層にありどの経路が選択されるかをレイヤリングモデルを侵害することなく知ることはできない。従ってそのような場合は最適と思われるものを選択し、その根拠となった通信量・応答時間などの要求をネットワーク層まで伝達する必要がある。

4.5 キャッシュによる高速化

サーバを連続して使用するとき、毎回名前サーバ及び routing table の検索を行っていたのではそのオーバヘッドが無視できなくなる可能性がある。もちろん名前サーバ自身もキャッシュを行うためその影響は最小限であると考えられるが、短時間に同じ要求を出す場合には同じサーバを選択することによる害は少ない。従って RPC のレベルでサービス毎にキャッシュを行うことが有効となる。ただしその時間をあまり長くとると、ダイナミックに変化するデータであるサーバの負荷・ネットワークの輻輳の変動が大きくなる可能性があるので、それらの情報の更新間隔に依存した適当な時間に抑える必要がある。

5 トランスポート層での RPC

ここではトランスポート層までを含めた RPC について考察する。

5.1 Flow Control

フロー制御に使われる手法としては、

1. パケット毎の確認応答を返す方法
2. 選択的な確認応答を返す方法
3. rate base flow control
4. window base flow control

などの方法があげられる。

それぞれ LAN あるいは広域に適した部分があり、サーバの位置によって適当な方法を使い分けることが必要となる。その際サーバの位置を知る必要があるが、これはサーバのアドレスと routing table から判断することができる。

Local Area Network

LAN では通信にかかる時間はほぼハードウェアに依存した一定値であり、しかも短時間である。したがってある程度の間隔をおいて送信すればハードウェア的にはほぼ確実に受け取られることが期待できる。そのため rate base のフロー制御が有効であり、しかもこの場合はその rate が一定の値となる。

また通信そのものの信頼性もかなり高いと考えることができるので、ack についてはまとめて、あるいは nack を返すことで行えば十分であると考えられる。

Wide Area Network

ゲートウェイを介してつながれたインタネットワークでは、通信時間はネットワークやゲートウェイの混雑度によって大きく変化する。したがって一定の間隔で送っても受信できる保証はなく、いたずらに輻輳を発生させてしまう恐れがある。

これを避けるためには window base のフロー制御がある程度有効となる。これはネットワーク上に流れ出すパケットの量そのものを制限することができるため、輻輳制御には重要な手法となる。

5.2 信頼性

保証すべき信頼性の程度によって ack の制御、再送制御、イベントスケジューリング、ネットワーク層などが影響を受ける。

確認応答の制御

rate base のフロー制御を行っている場合、この rate を変化させる必要がなければ ack を返すかどうかは、単純に信頼性を要求するかどうかに依存する。ただし広域環境などで rate を動的に変化させる必要がある場合や、window base のフロー制御を行っている場合には、rate や window size の制御のために確認応答が必要となる。

再送制御

再送制御が最も信頼性に影響を与える部分である。具体的には何度まで retry するかということがパラメータとなる。

5.3 パケットの処理

event scheduling

信頼性が要求されている場合には他に優先して送られることが望ましい。トランスポート層は一般的に入力・出力・タイマの部分からなるが、タイマによってキューの処理を行う際、その順序を入れ換えることによって優先的に処理することができる。

ネットワーク層

ゲートウェイなどでキューが溢れた場合にはパケットが捨てられる可能性がある。その場合信頼性を要求しているパケットは優先的に送り、極力捨てられないようにするすることが考えられる。そのためには優先パケットのために一定量のバッファを常に残しておくことが必要である。また、どのパケットが優先されるべきなのかを示すために、要求される信頼性の程度をネットワーク層まで伝達するような機構も必要となる。

6 設計および実装

全体の構成を図2に示す。

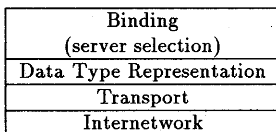

図 2: 全体の構成

ここでデータ表現にはXDR[Sun87]を用い、ネットワーク層はIP(UDP)を用いることとし、サーバの選択、トランスポート層について設計を行なった。

6.1 サーバの選択

サーバの名前

名前サーバを用いてサーバを選択するためには、あるサービスを提供するサーバ群に名前をつけなければならない。名前サーバはドメイン毎の資源の分散管理を行なうので、サーバ群の名前もドメイン情報を含むものとなる。すなわち、名前はドメイン名とサービスの種類を結合したものとする。

ただしこの方法ではサーバの属するドメイン名を指定することになるため、同一ドメインに複数のサーバが存在しないかぎり、ユーザにとってはサーバの存在するホスト名を指定する方法と同様となってしまう。そこで、指定されたドメインのサブドメインをも検索する機能を付加することによって、サーバ選択の範囲を広げる。

名前サーバのもつ情報

選択のために名前サーバが保持する情報は、サーバの存在するホスト毎のアドレス、計算能力、そして負荷である。このうちアドレスと計算能力は静的な情報であるが、負荷は動的に変化するため、これを更新するプロトコルが必要である。そのためサーバの存在する各ホストにデーモンをおき、一定間隔毎に負荷を名前サーバに伝達する。

ユーザがあるサービスを利用するときには、名前サーバはサービス名を含んだサーバの名前からサーバの存在するホストのリストを生成し、それを返す。ホストのリストにはホストのアドレス、計算能力、負荷という情報が入っていて、その中からどれを選択するかはクライアント側に任される。

ネットワークの情報

ネットワークについてはクライアントとサーバの間のスループットおよび遅延に関する情報が必要となる。これらはOSPFなどのような、ネットワークの情報を伝搬するプロトコルにより、ネットワーク情報として各ホストに格納される。これはネットワーク単位の情報であり、名前サーバから返された、サーバのアドレス情報から必要なエントリを参照することができる。

6.2 トランスポート層

トランスポート層は、実験的にデーモンとして実装し、後にカーネルに組み込むものとした。従って各ホストにRPCデーモンをおき、このデーモン同士が直接の通信を行なう。(図3)

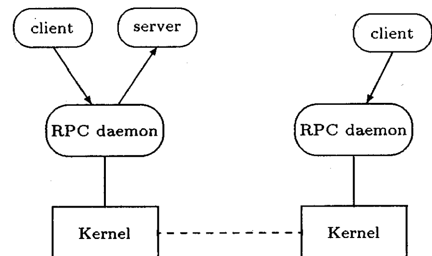

図 3: RPC daemon

フロー制御

フロー制御の方法はクライアントとサーバの存在するホスト間のネットワークによって変更する。まず、ゲートウェイを介さずに直接接続される場合、およびトランスポートゲートウェイとの通信の時は、Point-to-Point でrate base のフロー制御を行なう。その場合のrate は通信相手のネットワークの性能により、一定値を用いる。

次に、サーバの存在するホストが他のネットワーク上であった場合は、輻輳を避けるためにwindow base のフロー制御を行なう。window size はネットワーク情報のうち主にスループットから決定するが、要求に対する確認応答が返ってくるまでの時間の変化によって、これを修正しながら通信を行なう。つまり、確認応答が返るまでの時間が増えた場合は、輻輳が生じていると判断してwindow size を小さくしたり、タイムアウト時間を長くしたりする。

トランスポートゲートウェイ

Point-to-Point で信頼性を得たい場合や、マルチキャストルーティングを行なうためにトランスポート層でのゲートウェイ機能を設ける。

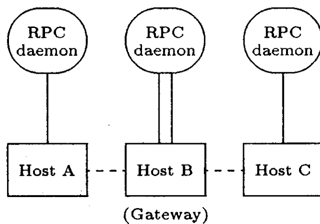

図 4: Transport Gateway

図4で、Host A のクライアントがHost C にあるサーバを利用するときにHost B のRPC デーモンがその中継を行なう。このゲートウェイでは要求や応答を一度スプールし、その時点で確認をとって再送などを行なう。特に信頼性が必要な場合はスケジューリングの際にキューの順番を入れ換えることによって確実に伝達を行なう。

また、マルチキャストを行なう時にはHost A からはHost B のみに送り、Host B のRPC デーモンがその後、Host C への中継などの処理を行なう。このことによってネットワーク内の通信量を減らすことができる。

7 おわりに

ネットワークの拡大にともなって様々なスループット/遅延を持つネットワーク考慮しなければならなくなった。そのため複数のサーバが存在する場合、その中からどれを選択するかという問題が重要となってくる。さらに通信そのもののコストを低く抑えるためにも可能な限り再送を抑え他の通信を妨げることのないようにしなければならない。

本研究ではサーバの選択については名前サーバを用いてあらかじめ必要な情報を得、さらにあらかじめ応答時間を予測することによってユーザが処理を変更できるような機能を用意した。また通信についてはクライアントとサーバの位置の関係から場合によって適当なアルゴリズムを選択することによって実用上の効率の低下を最低限に抑えつつ広域環境で必要となる congestion avoidance を実現する方法を示した。

参考文献

[Bir+84] A. Birrel and B. Nelson. Implementing Remote Procedure Calls. *ACM trans. on Computer Systems*, 2(1):39–59, February 1984.

[Dun86] K. J. Dunlap. *Name Server Operations Guide for BIND*. University of California Berkeley, April 1986. UNIX System Manager's Manual (SMM).

[Hed88] C. Hedrick. Routing Information Protocol, June 1988. RFC 1058.

[Ish+89] Y. Ishikawa, K. Saga, A. Onoe, and J. Murai. Resource Management Function of WIDE. In *Proc. of the 4th International Joint Workshop on Computer Communications*, July 1989.

[Moy89] J. Moy. Draft OSPF Specification, July 1989.

[Mur89] J. Murai. Construction of the Widely Integrated Distributed Environment. In *Proc. of the 4th International Joint Workshop on Computer Communications*, July 1989.

[Sun86] Sun Micorosystems, Inc. *Yellow Pages Protocol Specification*, 1986.

[Sun87] Sun Microsystems, Inc. *XDR: External Data Representation standard*, June 1987. RFC 1014.

[Sun88] Sun Microsystems, Inc. *RPC: Remote Procedure Call Protocol Specification*, June 1988. RFC 1057.

[Sun89] Sun Microsystems, Inc. *NFS: Network File System Protocol Specification*, March 1989. RFC 1094.

[Wel86] B. B. Welch. The Sprite Remote Procedure Call System. Technical report, University of California Berkeley, July 1986.