セキュアな RPC のためのコールバック型コネクション確立方式

平山秀昭[†]　本多弘樹[†]　弓場敏嗣[†]

WWW(World Wide Web) が開発されて以来、インターネットの利用者数は増大の一途を辿っている。現在、その用途の大半は単純な情報検索であるが、 EC(Electronic Commerce) への期待が高まる中、次第に WWW での処理が複雑化してきている。本論文では、グループウェア等のインターネット対話型処理を対象に、それを効率よく実行する仕組みを検討する。特に不正侵入等のセキュリティ問題に着目し、セキュアな RPC(Remote Procedure Call) を実現するためのコールバック型コネクション確立方式を提案する。この方式ではファイアウォールが、外部へのアウトバウンドコネクション要求は許可するが、内部へのインバウンドコネクション要求は許可しないことを前提に、サーバからクライアントにコネクション要求を発行する。このコネクション要求はサーバからクライアントに対するものなので、サーバ側ファイアウォールは通過するが、クライアント側ファイアウォールは通過しない。この問題を解決するためにクライアント側ファイアウォールに、ポート指定でインバウンドコネクション要求を 1 回のみ許可する機能を追加する。このコールバック型コネクション確立方式に基づく RPC の仕組みを示し、CORBA や Java RMI と比較する。

Callback Connection Scheme for Secure RPC

Hideaki Hirayama,[†] Hiroki Honda[†] and Toshitsugu Yuba[†]

Since WWW(World Wide Web) was developed, the number of the users of the Internet is continued to be increasing. Currently, most of the use of the Internet is information retrieval, but it is being complex while the service of EC (Electronic Commerce) is expected. In this paper, we discus the mechanism for effectively executing the Internet interactive systems. Particular, we pay attention to security problems such as illegal accesses. And we propose the Callback Connection Scheme for secure RPC (Remote Procedure Call). In this scheme, we assume firewall permits only out-bound connections and server programs request to connect to client programs. Because this connection request is from server programs to client programs, server-side firewall permits it but client-side firewall does not permit it. To solve this problem, we add a new mechanism for permitting the in-bound connection request with specified port to the client-side firewall. We describe the mechanism of the RPC with the Callback Connection Scheme and compare it with CORBA (Common Object Request Broker Architecture) and JAVA RMI (Remote Method Invocation).

1. はじめに

WWW(World Wide Web) が開発されて以来、インターネットの利用者数は増大の一途を辿っている。現在、その用途の大半は単純な情報検索であるが、EC(Electronic Commerce) への期待が高まる中、次第に WWW での処理が複雑化してきている。

WWW は元来、ホームページへのアクセスという単純な情報検索を指向したものである。そのため、そこで使用される HTTP(Hyper Text Transfer Protocol) は、 1 リクエスト / 1 リプライの動作を基本としていて、複雑な処理を実行するのに適していない。一方、一般的な C/S(Client and Server) 型アプリケーションは、複雑な処理を実行するのに適しているが、最もプリミティブな記述方法であるため開発効率が悪い。

本論文では、グループウェア等のインターネット対話型処理を対象に、それを効率よく実行する仕組みを検討する。基本的には RPC(Remote Procedure Call) を利用するが、 RPC には不正侵入等のセキュリティ問題があるため、セキュアな RPC を実現するためのコールバック型コネクション確立方式を提案する。

この方式ではファイアウォールが、外部へのアウトバウンドコネクション要求は許可するが、内部へのインバウンドコネクション要求は許可しないことを前提に、サーバからクライアントにコネクション要求を発行する。このコネクション要求はサーバからクライアントに対するものなので、サーバ側ファイアウォールは通過す

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

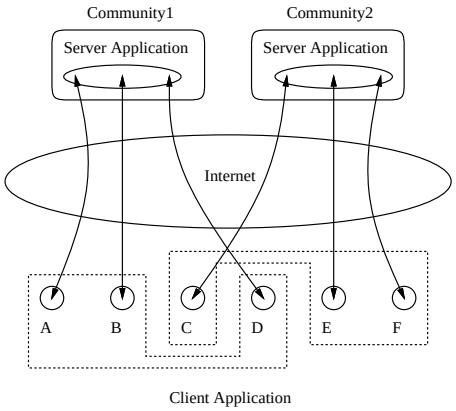


図 1 インターネット対話型処理

るが、クライアント側ファイアウォールは通過しない。この問題を解決するために、クライアント側ファイアウォールにポート指定でインバウンドコネクション要求を 1 回のみ許可する機能を追加する。

本論文では、このコールバック型コネクション確立方式に基づく RPC[1] の仕組みを示し、 CORBA(Common Object Request Broker Architecture)[2] および Java RMI(Remote Method Invocation)[3] と比較する。

2. 想定するアプリケーションモデル

本論文では、インターネット上で不特定多数のユーザが対話型処理を行うグループウェア等のアプリケーションをインターネット対話型処理と呼ぶ。図 1はインターネット対話型処理を示すもので、これをアプリケーションモデルとして想定する。このアプリケーションは、WWW のホームページのように、新規開発のプログラムが新たにサービスを開始したり、それまで利用されていたプログラムがサービスを中止することが、日常的に行われるものとする。

図 1の Community1 および Community2 は、ここで想定する C/S 型アプリケーションのサーバアプリケーションを示す。一方 A から F は、想定する C/S 型アプリケーションのクライアントアプリケーションを示す。状況に応じて、クライアントアプリケーション A から F は、 Community1 に接続する場合もあるし、 Community2 に接続する場合もある。図 1では、Community1 に A、 B、 D が接続し、一つのグループを構成している。また Community2 には、 C、 E、 F が接続し、別のグループを構成している。そして、これらが、グループウェアあるいは、チャット、共同文書作成、オンライン教育、対戦ゲーム等の対話型処理を行っている。

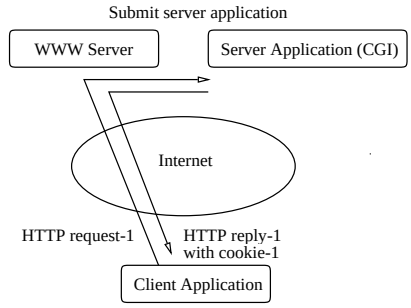
例えば A が、サーバアプリケーションを介して B や D に情報を送る。その一方で A は、サーバアプリケーションを介して B や D から情報を受け取る。グループウェア等の対話型処理では、ユーザ間で情報がリアルタイムに受け渡される必要があり、 A が B に情報を送った場合、それが B に即時に通知されなければならない。そのため、クライアントアプリケーションがサーバアプリケーションに処理を依頼するだけでなく、サーバアプリケーションからもクライアントアプリケーションに処理を依頼できなければならない。

3. インターネット対話型処理を WWW および C/S 型アプリケーションとして実現する場合の問題点

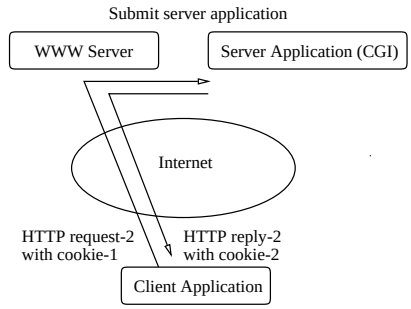
3.1 インターネット対話型処理を WWW で実現する場合の問題点

WWW はインターネット上での情報検索を可能にする仕組みであり、 CGI(Common Gateway Interface) プログラムを簡便に指定する方法として URL(Universal Resource Locator) を持つ。しかし、 WWW と CGI を用いた場合には、グループウェア等のインターネット対話型処理を効率的には実行できない。これは WWW で使用されている HTTP が、 1 リクエスト /1 リプライを基本としているためである。 HTTP は複数のリクエスト / リプライを繰り返すような処理には適さない。現状のシステムでは、このような問題に対応するために、クッキー (cookie) と呼ばれる手法を用いている。

クッキーとは、 WWW ブラウザとサーバアプリケーションの間で受け渡されるデータのことである。 HTTP は 1 リクエスト /1 リプライを基本としているが、このクッキーを受け渡すことで、いわゆるセッション管理を可能にしている。セッション管理とは、クライアントアプリケーションとサーバアプリケーションが、継続して処理を実行するための情報を制御する仕組みである。図 2は、 WWW と CGI におけるクッキーを用いたセッション管理を示す。 (a) では、クライアントアプリケーションが WWW サーバを介してサーバアプリケーションを CGI として起動している。サーバアプリケーションは処理結果をクライアントアプリケーションに戻す。この時、サーバアプリケーションがこの処理の続きを実行するための情報を、 cookie-1 によってクライアントアプリケーションに渡す。 (b) では、クライアントアプリケーションが cookie-1 を添付することで、この処理の続きをサーバアプリケーションに依頼している。サーバアプリケーションは CGI として新たにプロセスあるいはスレッドが生成されるが、クライアントアプリケー



(a) First request from client application to server application



(b) Second request from client application to server application

図 2 WWW+CGI の動作

ションから渡される情報 cookie-1 によって、(a) の処理の続きを行える。

このクッキーという手法により、WWW と CGI でもセッション管理が可能になる。しかし、これで従来の C/S 型アプリケーションと同等の柔軟な処理を行えるわけではない。また、CGI ではリクエストを受け取る毎にプロセスあるいはスレッドを生成するので、オーバヘッドも大きい。しかも、クッキーとして渡される情報を用いてセッション管理を行うので、プログラム開発が容易でないし、機能的な制約も受ける。

さらに WWW では HTTP を WWW サーバに送る時に TCP/IP のコネクションを張る。そのため、図 2に示すように、リクエストは常に WWW サーバを経由して送られる。サーバアプリケーションを WWW サーバとは異なるノードで起動したとしても、リプライは必ず WWW サーバを経由して戻す必要があり、WWW サーバがボトルネックとなり易い。

その一方、WWW では全ての通信に HTTP を用いているので、ファイアウォールを設定し、HTTP 以外のプロトコルは通さないようにすることで、サーバ側における不正侵入等のセキュリティ問題に対応することができる。

以上より、インターネット対話型処理を WWW で実現する場合の問題点をまとめると以下のようになる。

(1) クライアントアプリケーションとサーバアプリケーションの協調動作を効率的に実行できない。
(2) リクエストを受け取る毎にプロセスあるいはスレッドを生成するのでオーバヘッドが大きい。
(3) 全てのリクエスト / リプライが WWW サーバを経由するためボトルネックを生じ易い。

ただし、2 番目の問題点に関しては、Fast CGI[4] を用いれば、プロセスあるいはスレッドが常にリクエストを待つようになる。そのため、リクエストを受け取るたびにプロセスあるいはスレッドを生成する必要はなくなる。

また、3 番目の問題点に関しては、アクセスのたびにデータが変更されない単純なホームページ等のコンテンツに限定すれば、ラウンドロビン DNS(Domain Name System)[4] を用いることで WWW サーバのボトルネックを分散させることができる。しかし、この方法はカウンタ等、アクセスのたびにデータが変更されるコンテンツに関しては適用できない。

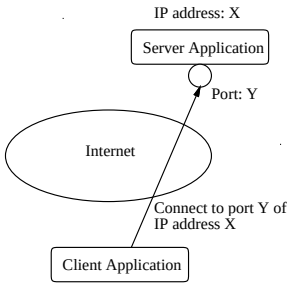
3.2 インターネット対話型処理を C/S 型アプリケーションとして実現する場合の問題点

C/S 型アプリケーション方式は、ユーザが直接アクセスするフロントエンド側と、データ処理を行うバックエンド側を、ネットワークを介して接続する場合の基本的な方法である。そのため、グループウェア等のインターネット対話型処理を柔軟に実現できる。しかし、その反面、プログラミングは容易でない。また、WWW で CGI プログラムを指定する URL のような、サーバアプリケーションを指定する簡便な方法を持たない。

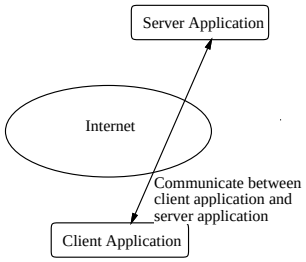
図 3は、一般的な C/S 型アプリケーション方式の動作を示す図である。(a) では、クライアントアプリケーションが、サーバアプリケーションの実行されるノードのアドレスとポート番号を指定して、コネクションを確立している。(b) では、クライアントアプリケーションとサーバアプリケーションが、(a) で確立されたコネクションを通して協調動作している。

また、クライアントアプリケーションとサーバアプリケーションの間の通信プロトコルを、WWW のように HTTP に限定することができないため、ファイアウォールでサーバ側における不正侵入等のセキュリティ問題に対応していても、特定ポートを開放し続けねばならない。そのためインターネットでは、ポートスキャン等により不正侵入を受け易くなる。

その一方で、サーバアプリケーションが実行される計算機を分散させれば、クライアントアプリケーションとサーバアプリケーションの間の通信路が分散される。



(a) Connection from client application to server application



(b) Communication between client application and server application

図 3 一般的な C/S 型アプリケーションの動作

従って WWW のように全ての通信が WWW サーバを経由するため、それがボトルネックになるようなことはない。

また、一旦コネクションを張ったら、全ての処理が完了するまでサーバアプリケーションは生存し続けるので、リクエストを受け取るたびにプロセスあるいはスレッドを生成する必要がない。

以上より、インターネット対話型処理を C/S 型アプリケーションとして実現する場合の問題点をまとめると以下のようになる。

(1) クライアントアプリケーションとサーバアプリケーションの協調動作を効率的に実行できるが、プログラミングが容易でない。
(2) WWW で CGI プログラムを指定する URL のような、サーバアプリケーションの簡便な指定方法を持たない。
(3) 特定ポートを開放し続けるので、ポートスキャンによる不正侵入を受け易い。

ただし、問題点 2 に関しては、サーバアプリケーションを簡便な名前にマッピングするために LDAP[5] 等のディレクトリサービスによって、アプリケーションが実行されるノードのアドレスとポート番号を一元的に管理することができる。

4. コールバック型コネクション確立方式に基づいた RPC

4.1 インターネット対話型処理の RPC による実現

RPC は、グループウェア等のインターネット対話型処理の開発に適していて、前節で示した WWW や C/S 型アプリケーションの多くの問題点を解決する。 RPC は基本的に C/S 型アプリケーションにプログラミング容易性を提供するものであり、 WWW で実現する場合の問題点 (1) と、 C/S 型アプリケーションとして実現する場合の問題点 (1) を解決する。 RPC の本質的な動作は C/S 型アプリケーションと同じなので、 WWW で実現する場合の問題点 (2) と (3) も解決する。 C/S 型アプリケーションとして実現する場合の問題点 (2) に関しても、 LDAP 等のディレクトリサービスを用いることで解決される。しかし、特定ポートを開放し続けるので、ポートスキャン等による不正侵入を受け易いという C/S 型アプリケーションとして実現する場合の問題点 (3) は解決できない。我々は、この問題を解決するために、コールバック型コネクション確立方式を提案する。

4.2 コールバック型コネクション確立方式

コールバック型コネクション確立方式は、従来の C/S 型アプリケーションあるいは RPC によるインターネット対話型処理に実現方法と異なり、サーバからクライアントに対してコネクション要求を行う。この方式においては、ファイアウォールは以下の設定を仮定している。

- HTTP は許可
- TCP/IP は外部へのコネクション (アウトバウンドコネクション) 要求は許可、内部へのコネクション (インバウンドコネクション) 要求は不許可

図 4に従い、本方式の動作を説明する。

(1) クライアントはサーバからのコネクション要求を受け付けるポートを開設する。
(2) クライアントは、このポートに対するインバウンドコネクション要求を許可するように、ファイアウォールに指示を出す。
(3) クライアントは、 HTTP でこのポート番号をサーバに通知し、このポートでコネクション要求を待つ。
(4) クライアントからサーバへのポート番号通知は HTTP なので、クライアント側およびサーバ側のファイアウォールを通過する。
(5) サーバは HTTP で受けたクライアントのポートに対してコネクション要求を出す。
(6) このコネクション要求はサーバ側ではアウトバウンド要求なので、サーバ側ファイアウォールを通

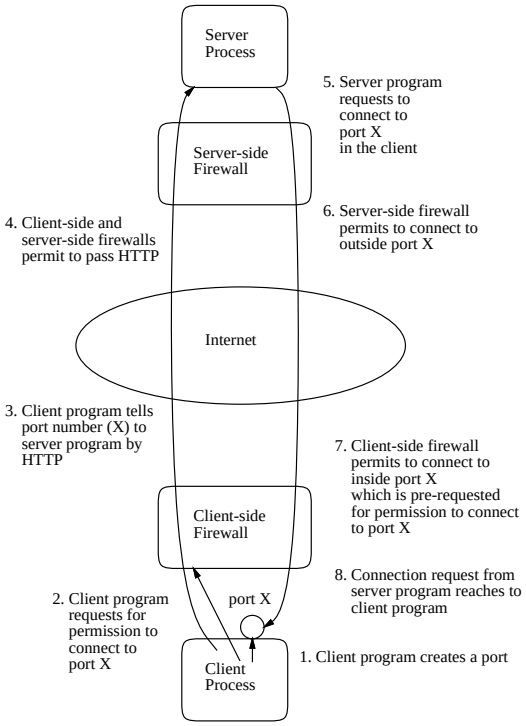


図 4 コールバック型コネクション確立方式の動作

過する。

(7) このコネクション要求はクライアント側ではインバウンド要求だが、2で許可することを指示されたポート番号に対するものなので許可する。

(8) サーバからのコネクション要求が、クライアントが待ちうけているポートに到達する。

コールバック型コネクション確立方式では、クライアントプログラムからの指示により、特定ポートに対するインバウンドコネクション要求を一時的に許可する機能をファイアウォールに組み込む。ファイアウォールは指定されたポートに対するインバウンドコネクション要求を1回のみ許可する。すなわち1回コネクション要求を許可したら、再びそのポートへのコネクションは拒絶する。これにより、クライアントがコネクション要求を期待している僅かな時間しかインバウンドコネクション要求を許可しない。しかもクライアントはコネクションを受け付けるポートを固定する必要がないので、外部からの不正侵入を受ける確率は極めて低くなる。当然、サーバアプリケーションの不具合等により、コネクション要求を出したポートに対して、コネクション要求がこない可能性もある。そのため、コネクション要求の許可にはタイムアウト時間を設定し、一定時間を経過してもコネクション要求がこない場合には、許可を取り消す。

なお、ここまでの説明では、クライアント側ファイアウォールに対し、外部からのコネクション要求を一時的に許可するポートと記述してきたが、正確には特定計算機 (IP アドレス) のポートを示している。

4.3 コールバック型コネクション確立方式に基づく RPC

コールバック型コネクション確立方式に基づく RPC について考える。ここではコールバック型コネクション確立方式をベースに、以下の機能を持った RPC を設計する。

- URL によるサーバアプリケーション指定
- サーバからクライアントへのコネクション
- 同期 / 非同期 RPC

コールバック型コネクション確立方式に基づく RPC の基本的な動作は、以下のようになる。

(1) 初期化関数 callback_rpc_init() を実行し、サーバアプリケーションを起動する。この時、サーバアプリケーションを URL で指定する (図 6(a))。

(2) 起動されたサーバアプリケーションは、クライアントアプリケーションが用意しているポートに対してコネクションを張る (図 6(b))。

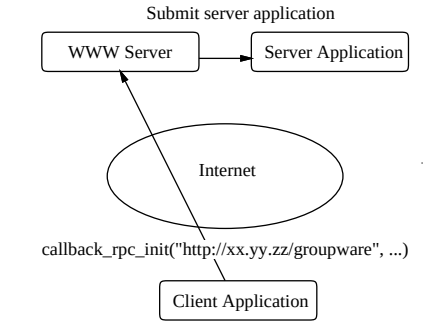
(3) 一旦、サーバアプリケーションからクライアントアプリケーションに対してコネクションが張られたら、以降の処理は全てこのコネクションを利用して、同期 / 非同期 RPC として実行する (図 6(c))。

以降、これらの機能について述べる。

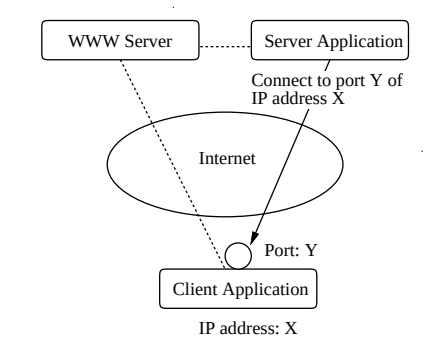
4.3.1 URL によるサーバアプリケーション指定

コールバック型コネクション確立方式に基づく RPC では、 URL でサーバアプリケーションを指定する。コールバック型コネクション確立方式に基づく RPC で提供する同期 / 非同期 RPC の実行に先立ち、クライアントアプリケーションは初期化関数 callback_rpc_init() を実行する。この初期化関数の引数として、サーバアプリケーションを指定する URL を渡す。 URL は、アプリケーションを起動するノードの WWW サーバとアプリケーション自身を名前で指定する。

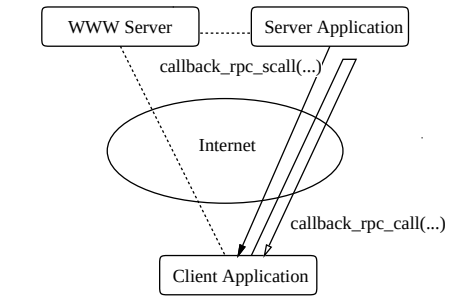
図 5(a) は、クライアントアプリケーションが、 callback_rpc_init() でサーバアプリケーションを指定する様子を示す図である。 callback_rpc_init() の最初の引数である URL によって、サーバアプリケーションを指定する。正確にはサーバアプリケーションを起動する WWW サーバと、そこで起動すべきアプリケーションの名前である。ただし、一旦サーバアプリケーションが起動されると、それ以降の処理は、サーバアプリケーションからクライアントアプリケーションに張られたコネクションを介して行われるので、 WWW サーバがボ



(a) Server application specified by URL



(b) Connection from server application to client application



(c) Synchronous and asynchronous RPC

図 5 コールバック型コネクション確立方式に基づく RPC の動作

トルネックになることはない。

この URL によるサーバアプリケーション指定機能によって、一般的な C/S 型アプリケーションとして実現する場合の問題点 (2) が解決される。

4.3.2 サーバからクライアントへのコネクション

クライアントアプリケーションが、コールバック型コネクション確立方式に基づく RPC の初期化関数 callback_rpc_init() を実行することで、クライアントアプリケーションとサーバアプリケーションの間に、TCP/IP コネクションを確立する。コールバック型コネクション確立方式に基づく RPC では、最初にサーバアプリケーションを指定するためだけにクライアントアプリケーションと WWW サーバの間で HTTP を利用する。それ以降の RPC の実行は、クライアントアプリケーションとサーバアプリケーションの間で直接接続された経路を用い、 WWW サーバを経由しない。

サーバアプリケーションからクライアントアプリケーションにコネクションを張るため、クライアントアプリケーションが実行する RPC の初期化関数では、まずソケットを作成し、そのソケットのポート番号を IP アドレスと共に HTTP でサーバアプリケーションに渡す。サーバアプリケーションは受け取った IP アドレスの受け取ったポート番号に対してコネクションを張る。

図 5(b) は、サーバアプリケーションからクライアントアプリケーションに、コネクションを張る様子を示す図である。 callback_rpc_init() 実行時に、その HTTP リクエストを受け取った WWW サーバは、指定されたサーバアプリケーションが起動されていなければ、まずそれを起動する。この後 WWW サーバは、サーバアプリケーションに対し、クライアントアプリケーションから渡されたクライアントの IP アドレスとポート番号を伝える。これによりサーバアプリケーションは受け取った IP アドレスの受け取ったポートに対してコネクションを張る。

このサーバからクライアントへのコネクション機能によって、 WWW で実現する場合の問題点 (2) と (3)、一般的な C/S 型アプリケーションとして実現する場合の問題点 (3) が解決される。

4.3.3 同期 / 非同期 RPC

コールバック型コネクション確立方式に基づく RPC は、プログラミングの容易さを考慮して、クライアントアプリケーションからサーバアプリケーションへの RPC として実現している。これを同期 RPC と呼ぶ。また、対話型処理を想定しているので、同期 RPC に加えて、サーバアプリケーションからクライアントアプリケーションに対して非同期に処理を依頼するための非同期 RPC を提供する。非同期 RPC は、グループウェア等で他のクライアントから、情報が転送される場合等に用いる。

図 5(c) は、クライアントアプリケーションが実行する同期 RPC である callback_rpc_call() と、サーバアプリケーションが実行する非同期 RPC である callback_rpc_acall() の様子を示す図である。

この同期 / 非同期 RPC 機能によって、 WWW で実現する場合の問題点 (1) と、一般的な C/S 型アプリケーションとして実現する場合の問題点 (1) が解決され、合わせて WWW および一般的な C/S 型アプリケーションによる実現方式の全ての問題点が解決される。

5. CORBA および Java RMI との比較

5.1 CORBA との比較

CORBA を用いても、グループウェア等のインターネット対話型処理を、効率的かつ容易に実現でき、WWW で実現する場合の問題点 (1) と C/S 型アプリケーションとして実現する場合の問題点 (1) を解決できる。一旦コネクションを確立してしまうと、その経路を用いて RPC の処理を行うため、 WWW で実現する場合の問題点 (2) と (3) も解決できる。しかし、不正侵入等のセキュリティ問題は解決できず、サーバ側の特定ポートに対するインバウンドコネクションを許可するように、ファイアウォールを設定するしかなく、ポートスキャン等による不正侵入を受け易い。我々が提案しているコールバック型コネクション確立方式は、 CORBA に対して適用することも可能で、不正侵入等のセキュリティ問題を解決することができる。

5.2 Java RMI との比較

Java RMI を用いた場合、 Java という特定の言語のみに限定した場合ではあるが、グループウェア等のインターネット対話型処理を、効率的かつ容易に実現でき、WWW で実現する場合の問題点 (1) と C/S 型アプリケーションとして実現する場合の問題点 (1) を解決できる。そして当然、一旦コネクションを確立してしまうと、その経路を用いて RMI の処理を行うため、 WWW で実現する場合の問題点 (2) と (3) も解決できる。 Java RMI ではファイアウォールがある場合にも、 HTTP を用いてサーバ (リモートオブジェクト) への接続を行うことができる。しかし、これは Java RMI をファイアウォール上で通過させるための方法でしかなく、ポートスキャン等による不正侵入を防ぐことはできない。我々が提案しているコールバック型コネクション確立方式は、 Java RMI に対して適用することも可能で、不正侵入等のセキュリティ問題を解決することができる。

6. 実　　装

コールバック型コネクション確立方式に基づく RPC のプロトタイプを、 Solaris 上に C 言語を用いて、ライブラリとして実装した。 WWW サーバには、 Apache[6] を用いた。ただし、指定したポートに対するインバウンドコネクション要求を一時的に許可するというファイアウォールの機能拡張は行っていない。

コールバック型コネクション確立方式に基づく RPC では、クライアントアプリケーション用に、表 1に示す API を提供している。

また、サーバアプリケーション用には、表 2に示す API を提供している。なお表中の C.A. はクライアントアプリケーションを示す。

表 1 コールバック型コネクション確立方式に基づく RPC のクライアント API

API 名	機能
callback_rpc_regist()	非同期実行される関数を登録
callback_rpc_init()	サーバと接続
callback_rpc_call()	サーバ関数の同期実行
callback_rpc_acall_invoked()	非同期実行要求を処理
callback_rpc_close()	サーバとの接続解消

表 2 コールバック型コネクション確立方式に基づく RPC のサーバ API

API 名	機能
callback_rpc_regist()	同期実行される関数を登録
callback_rpc_connect()	クライアントの要求に従い接続
callback_rpc_acall()	クライアント関数の非同期実行
callback_rpc_call_invoked()	同期実行要求を処理
callback_rpc_close()	クライアントとの接続解消

図 6、図 7は、各々簡単なインターネット対話型処理のクライアントアプリケーションとサーバアプリケーションの擬似コードである。以降、これを用いて、コールバック型コネクション確立方式に基づく RPC のプログラミング環境について説明する。

図 6において、 (1) はクライアントアプリケーションのメイン関数である。このメイン関数は callback_rpc_regist() により、サーバアプリケーションから非同期 RPC で実行される関数 recv_info() を登録する (2)。次に callback_rpc_init() により、サーバアプリケーションと接続する (3)。サーバアプリケーションは、 URL("http://xx.yy.zz/groupware") で指定する。この後、 thr_create() (thr_create() は Solaris のスレッド生成システムコール) により、サーバアプリケーションから受けた非同期 RPC 要求を実行するためのスレッド (実行関数は async()) を生成する (4)。最後に、メイン関数は while ブロックに入る (5)。この while ブロックの中の callback_rpc_call() で、サーバアプリケーションの関数 send_info() を同期実行する (6)。send_info() は情報を他のクライアントに送る関数とする。

(7) はサーバアプリケーションから受けた非同期 RPC 要求を実行するためのスレッドの実行関数 async() である。この関数は、 while ブロックに入る (8)。この while ブロックの中の callback_rpc_acall_invoked() で、サーバアプリケーションから受けた非同期 RPC 要求を実行する (9)。

(10) はサーバアプリケーションから非同期 RPC によ

```
/*
 * Client Application
 */
main(int argc, char **argv)                                          (1)
{
    callback_rpc_regist(..., recv_info, ...);                        (2)
    s = callback_rpc_init(..., "http://xx.yy.zz/groupware", ...);    (3)
    thr_create(..., async, s, ...);                                  (4)
    while (1) {                                                      (5)
      ...
      callback_rpc_call(..., s, "send_info", ...);                   (6)
      ...
    }
}

async(int s)                                                         (7)
{
    while (1) {                                                      (8)
      ...
      callback_rpc_acall_invoked(s);                                 (9)
      ...
    }
}

recv_info(...)                                                       (10)
{
    ...
}
```

図 6 インターネット対話型処理のクライアントアプリケーションの擬似コード

```
/*
 * Server Application
 */
main(int argc, char **argv)                                          (1)
{
    callback_rpc_regist(..., send_info, ...);                        (2)
    while (1) {                                                      (3)
      callback_rpc_listen(..., &addr, &port, ...);                   (4)
      s = callback_rpc_connect(addr, port);                          (5)
      thr_create(..., child, s, ...);                                (6)
    }
}
child(int s)                                                         (7)
{
    while (1) {                                                      (8)
      ...
      callback_rpc_call_invoked(s);                                  (9)
      ...
    }
}
send_info(...)                                                       (10)
{
    ...
    callback_rpc_acall(..., s, "recv_info", ...);                    (11)
    ...
}
```

図 7 インターネット対話型処理のサーバアプリケーションの擬似コード

り実行される関数 recv_info() である。 recv_info() は、他のクライアントから送られた情報を受ける関数とする。

図7において、サーバアプリケーションは、アプリケーション名 (”groupware”) と関連付けられていて、クライアントアプリケーションが実行する callback_rpc_init() に応じて、 WWW サーバから起動される。 (1) はサーバアプリケーションのメイン関数である。このメイン関数は callback_rpc_regist() により、クライアントアプリケーションから同期 RPC で実行される関数 send_info() を登録する (2)。次にメイン関数は while ブロックに入る (3)。この while ブロックの中の callback_rpc_listen() により、クライアントアプリケーションからの接続要求を受ける (4)。

正確には、 WWW サーバが、クライアントアプリケーションからの接続要求を受け、このサーバアプリケーションを起動し (既に起動されている場合には起動しない)、クライアントアプリケーションからの接続要求を、サーバアプリケーションに渡す。接続要求したクライアントアプリケーションの IP アドレスとポート番号は、各々、変数 addr と変数 port に入る。続いてサーバアプリケーションは、 callback_rpc_connect() により、クライアントアプリケーションに接続する (5)。接続が行えたところで、クライアントアプリケーションから受けた同期 RPC 要求を実行するスレッド (実行関数は child()) を生成する (6)。

(7) はクライアントアプリケーションから受けた同期 RPC 要求を実行するためのスレッドの実行関数 child() である。この関数は、 while ブロックに入る (8)。この while ブロックの中の callback_rpc_call_invoked() で、クライアントアプリケーションから受けた同期 RPC 要求を実行する (9)。

(10) はクライアントアプリケーションから同期 RPC により実行される関数 send_info() である。 send_info() は、あるクライアントから他のクライアントに送られる情報を、送り元から受ける。 (11) の callback_rpc_acall() は、他のクライアントアプリケーションの関数 recv_info() を非同期実行する。 recv_info() は、あるクライアントから他のクライアントに送られる情報を、送り先に渡す。

7. 評　　価

コールバック型コネクション確立方式に基づく RPC は、 WWW サーバを用いてサーバアプリケーションを起動するが、一旦サーバアプリケーションを起動すると、以降の処理はサーバアプリケーションとクライアントアプリケーションの間に直接張られたコネクションを利用して行うので、 WWW で CGI アプリケーションを起動するよりも高速である。

そこでコールバック型コネクション確立方式に基づく

RPC および WWW と CGI で実現んしたアプリケーションの性能比較を行った。評価用ベンチマークプログラムは、以下の仕様とした。

- クライアントアプリケーションが、サーバアプリケーションに一度に送るリクエスト数は、平均L[個]のポアソン分布に従う。
- クライアントアプリケーションが、サーバアプリケーションにリクエストを送る間隔は、平均 M[秒]のポアソン分布に従う。
- サーバアプリケーションが、１つのリクエストを処理する時間は、平均N[秒] のポアソン分布に従う。

評価環境は、以下に示すクライアント計算機とサーバ計算機を 10Mbps のイーサネットで接続した。

- クライアント計算機:
  UltraSPARC II(296MHz) × 4
- サーバ計算機:
  SuperSPARC(50MHz) × 2

クライアント計算機をサーバ計算機より高速にしたのは、コールバック型コネクション確立方式に基づく RPC が想定する環境では、サーバ計算機に多数のクライアント計算機からコネクションが張られ、サーバ計算機でボトルネックが生じるのが一般的だからである。この測定でも、サーバ計算機でなく、クライアント計算機がボトルネックになってしまうような本来と異なる状況が生じないように、クライアント計算機を高速にした。

上記のベンチマークプログラムの処理を、 WWW と CGI による実現 (WWW+CGI) およびコールバック型コネクション確立方式に基づく RPC による実現 (CALLBACK_RPC) で実行した。表 3は、クライアントアプリケーションが、サーバアプリケーションに送る平均リクエスト数 (L) が 5 の場合の応答時間の測定結果である。全般に、コールバック型コネクション確立方式に基づく RPC の方が、 WWW+CGI よりも高速である。特にクライアントアプリケーション数 (PROC) が 20 を越すと、その差は大きくなっている。

クライアントアプリケーション数が少ない (1 および 10) 時の性能差は、 WWW サーバ経由でプロセスを生成するオーバヘッドに起因すると考えられる。コールバック型コネクション確立方式に基づく RPC では、最初にセッションを張る時に、 WWW サーバ経由でプロセスを生成する。それ以降はクライアントアプリケーションとサーバアプリケーションの間で張られたコネクションを介して直接通信する。それに対し、 WWW と CGI による実現では、毎回 WWW サーバ経由でプロセスを生成している。そのため WWW サーバ経由でプロセスを生成する回数が、１回の処理あたり、コールバック型コネクション確立方式に基づく RPC の約 L 倍となる。

表 3 応答時間 (L=5, M=1, N=1)

クライアント プロセス数	WWW+CGI [秒]	CALLBACK_RPC [秒]
1	11.3	10.4
10	12.1	11.0
20	15.0	11.8
40	32.5	15.5

表 4 応答時間 (L=10, M=1, N=1)

クライアント プロセス数	WWW+CGI [秒]	CALLBACK_RPC [秒]
1	22.5	20.7
10	24.2	21.5
20	30.3	22.7
40	64.6	30.5

クライアントアプリケーション数が多い (20 および 40) 時に性能差が広がっているのは、 WWW サーバ経由でプロセスを生成する処理がボトルネックになっているためと考えられる。 WWW と CGI による実現では、処理要求毎に WWW サーバを介してプロセス生成を行うと、クライアントアプリケーション数が多くなった場合に WWW サーバがボトルネックとなる。

表 4は、クライアントアプリケーションが、サーバアプリケーションに送る平均リクエスト数 (L) が 10 の場合の応答時間の測定結果である。この場合も、表 3と同様の傾向が見られる。

8. 関 連 研 究

我々が提案しているコールバック型コネクション確立方式と同様に、サーバからクライアントに対してコネクションを確立する例として FTP(File Transfer Protocol) がある。我々の提案する方式は、 FTP のように、単にサーバからクライアントにコネクションを確立するというわけではなく、セキュアな RPC を実現するためのファイアウォールの制御方法まで含めたものである。

インターネットレベルで、グローバルなアプリケーションサービスを提供するグローバルコンピューティングシステムとして、 Ninf[7)8)]、 Globus[9)]、 Legion[10)] 等がある。これらのシステムでは不正侵入等のセキュリティ問題を検討していない。我々が提案しているコールバック型コネクション確立方式は、このようなシステムに対して適用しても有効である。

論文[11)] では、グローバルコンピューティングシステム Globus が、ファイアウォールの環境では利用できないという問題を解決する方式を提案している。この方式では、サーバ側ファイアウォールの外側にプロキシを置

き、コネクション要求はプロキシに対して発行するようにする。このプロキシに対するコネクションがきたら、ファイアウォールの内側から、このプロキシにコネクションを張らせることで、ファイアウォール環境でもGlobusを利用可能にしている。この方式は、サーバからクライアントに対してコネクションを張らせるという点で、我々が提案しているコールバック型コネクション確立方式と似ている。しかし、この方式では、結局バインドされているサーバ側のポートが、コネクション要求を常時受け付けるようになってしまい、Globusがファイアウォールの環境で使用できないという問題は解決しているが、ポートスキャン等による不正侵入を防止するわけではない。我々が提案しているコールバック型コネクション確立方式では、クライアント側で非固定的に割り当てられたポートを、コネクションを実施する間だけ一時的に開放することで、不正侵入を受ける確率を小さくしている。

RPCの機能を限定した方法として、HTTP上にRPCを作成したXML-RPC[12]がある。Globus等をファイアウォール環境下で利用可能にするには、このような方法も考えられる。

Windows2000のRAS(Remote Access Server)では、サーバアプリケーションへの接続を制御するために、最大接続時間等の細かな設定が可能だが、我々の提案するコールバック型コネクション確立方式のような方法は持たない。

インターネットにおけるセキュリティ問題には、我々が提案しているコールバック型コネクション確立方式で対応している不正侵入の他にもデータの漏洩、改竄等の問題があり、SSL(Secure Socket Layer)による暗号化等、種々の方法が検討されている[13]。我々が提案しているコールバック型コネクション確立方式は、不正侵入の問題に限定したセキュリティ対策であり、この問題に関しては他に例を見ない。

9. むすび

インターネット上で不特定多数のユーザが対話型処理を行うグループウェア等のアプリケーションを対象に、それを効率よく実行する仕組みを検討した。特に不正侵入等のセキュリティ問題に着目し、セキュアなRPCを実現するためのコールバック型コネクション確立方式を提案した。このコールバック型コネクション確立方式に基づくRPCの仕組みを示し、CORBAおよびJava RMIと比較した。

今回はコールバック型コネクション確立方式に基づくRPCのプロトタイプを実装した。ただし、指定したポートへのコネクション要求を一時的に許可するファイアウォールの拡張機能は実装していない。今後、ファイアウォールの拡張機能も実装し、実システムへの適用を試みる。また、本論文で示したような独自のRPCを考えるのではなく、コールバック型コネクション確立方式をJava RMIのような標準のRPCに適用することを検討する。

参考文献

1) 平山秀昭, 本多弘樹, 弓場敏嗣, "ワールドワイドなインターラクティブシステムのためのHTTPコネクション型RPC方式の検討,"情報処理学会研究報告 DSM-2000-DSM-18-1, pp.1–6, 2000年7月.
2) 小野沢博文著, "分散オブジェクト指向技術CORBA,"ソフト・リサーチ・センター, April 1996.
3) J. Schneider, R. Arora著, MidWatch訳, 種村明宏監修, "Enterprise Java デベロッパーズガイド,"ソフトバンク, 1998年4月.
4) R. Buyya, "High Performance Cluster Computing: Architecture and Systems, Volume 1," Prentice Hall, 1999.
5) T. Howes, M. Smith著, 松島栄樹, 岡薫訳, "LDAP インターネット ディレクトリ アプリケーション プログラミング,"プレンティスホール出版, 1997年11月.
6) B. Laurie, P. Laurie, "Apache: The Definitive Guide," O'Reilly & Associates, Inc., February 1999.
7) 小川宏高, 松岡聡, 中田秀基, 佐藤三久, 関口智嗣, "分散メモリ計算機用Ninf APIの実現に向けて,"情報処理学会研究報告 96-HPC-81, 1996年8月.
8) 鈴村豊太郎, 中川貴之, 松岡聡, 中田秀基, "クライアント・サーバ型のグローバルコンピューティングシステムの比較 - Ninf, NetSolve, CORBA, Ninf-on-Globusの性能評価 -,"情報処理学会研究報告 99-HPC-34, 1999年8月.
9) I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastracture Toolkit," International Journal of Supercomputer Applications, 11(2), pp. 115-128, 1997.
10) A. Grimshaw, W. Wolf, "Legion - a view from 50,000 feet," Proceedings of 5th IEEE Symp. on High Performance Computing, pp. 89–99, 1996.
11) 田中良夫, 平野基孝, 佐藤三久, 中田秀基, 関口智嗣, "Globusを用いたグローバルコンピューティング環境の構築とその評価,"インターネットコンファレンス'99(IC'99), 1999年12月.
12) "http://www.xml-rpc.com"
13) R. Oppliger, "Security at the Internet Layer," IEEE Computer, pp.43–47, September 1998.