Remote Procedure Calls in Accelerator Controls

Kazuro FURUKAWA and Norihiko KAMIKUBOTA

National Laboratory for High Energy Physics (KEK)

Abstract

Application of Remote Procedure Calls (RPC) in accelerator controls is investigated. RPC will enable transparent communication between application programs on the control computer network, and will help implementing intelligent functions at the remote control computers.

加速器の制御におけるリモートプロシージャコールの利用

1．はじめに

加速器が大型化するにつれ，加速器の制御においては複数の計算機を用いることは避けられないこととなっている。そのため，制御系内の通信方法の確立が重要になってきている。実際には計算機ネットワーク技術の進展により，基本的な部分では問題は解消されてきている。しかし，制御系全体をひとつに扱えるような環境があれば，効率のよい制御系を構築できると思われる。ここではこのような分散処理の問題の一部を解決しようとするリモートプロシージャコール（以後RPCと呼ぶ）について，加速器の制御系での応用を考えてみる。

2．分散処理

現在の計算機上の分散処理は，ほとんどの部分が個々のアプリケーション内の専用のプログラムで実現されている。現在1台の計算機の上，またはクラスタなどと呼ばれる密結合の計算機群の上で動いているオペレーティングシステムの動作環境が，将来的には，より一般的な計算機ネットワークの上で動作するようになると思われる。その時には，制御やモデリングのプログラムは，全く複数の計算機を意識せずに書くことができるようになる。しかし，それまでの間は我々の手で分散処理を実現しなくてはならない。

計算機ネットワークシステムの標準化とその実現が行われる以前は，制御系内のプログラムのかなり大きな部分がネットワークのために書かれていた。最近ではそのの標準化とTCP/IPなどの上の汎用プロトコルによってプログラムの負担は下がってきている[1)・2)]。しかし，ネットワークを利用するプログラムのデザインの仕方が，1台の計算機上で実現できるプログラムとは大きく異なるために，プログラム開発の効率は高くならない。

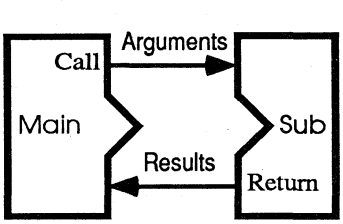
分散処理の支援としては，いくつかの試みが既に行われている。ひとつは，プログラム間の同期や通信のためのシステムサービスのネットワーク上での一般化で，例えばISIS[3)]と呼ばれるネットワーク上でのタスクグループ管理サービスなどが実用化されている。加速器の制御系でも，現在は個々のタスクごとにこのような管理が行われており，これらの利用価値は高い。もうひとつが，ネットワークの処理をアプリケーションプログラムから隠そうとするRPCである。

3．RPC (Remote Procedure Calls)

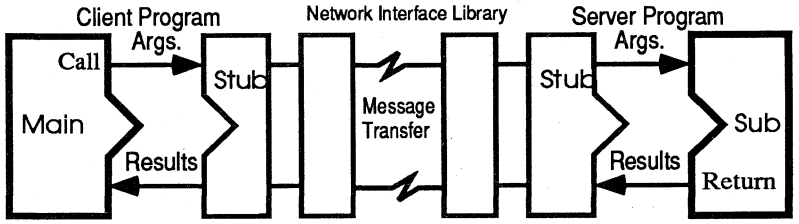
RPCの主な目的は，通常ひとつのプログラム内で行われるサブルーチンの呼び出しをネットワーク上に一般化することである。これによって，複数台の計算機上のプログラムがひとつのプログラムを書くことと全く同様に行うことができ，ローカルには存在しない資源の利用が容易に行えるようになる。RPCを利用することによって，例えば，均質な計算機ネットワーク上で大規模なプログラム群を並列実行し，負荷の均一化を計ることが可能になり，高エネルギー実験などで1台の計算機では扱えない大量のデータを処理するために使われる。また，制御システムでは個々の計算機に実現されている様々な制御のサービスサブルーチ

ンを別計算機から呼びだすために利用することができる。ユーザインターフェースと実際の処理を分離するためにも有用である。

RPC の実際の実行の方法を下の図に示す。左側が通常のひとつのプログラム内でのファンクションコールの際のデータのやり取りを示している。メインプログラムとサブプログラムの間で制御が移動すると同時に引き数や返り値の伝達も行われる。右側がそれをRPC の仕組みを通して遠隔実行するときの様子である。RPC のライブラリとユーザプログラムの間の接続を行うための通常スタブと呼ばれるモジュールは，RPC システムのツールによってインターフェース記述ファイルから自動的に生成される。



(A) Local Procedure Call



(B) Remote Procedure Call

現在の形の RPC は Zerox で 1981 年に実現され，後には BSD-Unix にも移植された Courier と呼ばれるシステムが最初のものである。これを原形として，さまざまな用途に RPC が拡張されてきており，例えば最もよく知られているのが，NFS (Network File System) の基礎となっている Sun-RPC である（現在は ONC-RPC）。その他にも現在の OSF の NCS-RPC の中核となっている Apollo の Broker や，OS/2 の RPC がある。また我々に関連する高エネルギー分野では，実験データ収集用の CERN-RPC やイベントデータ解析用の Fermilab-CPS が利用されている。加速器の制御用には CERN-RPC をさらに使いやすくした CERN-NC/RPC が SPS/LEP で開発されている[4]。

上に挙げたものはコンパイラ言語のシステムであるが，インタプリタ言語の場合は遠隔実行するサブプログラム自体を実行時にサーバ側に送り，解釈実行させることができる。従って，計算機依存性が少なくなり，利用しやすい形にすることができる。この例としては，KEK-Tristan の制御用言語である NODAL の EXEC-REMIT がある。また，Tk/Tcl と呼ばれる，ユーザインターフェース構築用のツールの Send もこれにあたる[5]。これらは開発効率はよいものであるが，サブプログラムが大きくなったときには実行効率が悪くなる。

RPC の条件としては以下のようなものが挙げられる。（a）透過性。RPC 対応でないプログラムを RPC 対応にする際に，できるだけ原形のまま利用できることが必要である。書き換えの量やルールが多いのであればソフトウェアの開発や維持の効率化にはつながらない。（b）引き数や返り値の変換。計算機によっては整数値のバイトオーダ，浮動小数点実数の表現，構造体中のパディングなどが異なるために転送の際に正しい変換を自動的に行わなくてはならない。場合によってはポインタで示された変数の実体の転送も必要となる。（c）速度。従来の方法による通信に比べて，同程度の応答速度が得られるものでなければ実用にはならない。（d）同期処理と非同期処理。直列処理を行うか並列処理を行うかによって，同期処理と特に制御では非同期処理を使い分ける必要がある場合が多い。また同報処理が必要な場合もある。（e）サービスの名前付け。処理を呼びだす際に，相手の計算機の名前とプログラムの名前または番号を指定するのではなく，サービス名を指定するだけで自動的に接続を行う機構が大規模なシステムでは必要となる。（f）時刻同期。個々の計算機で動作するリアルタイム制御プログラムは，その計算機上で時刻または時間の管理が正しいことを前提としている場合が多い。RPC を利用した際にもその前提を崩さないようにしたい。（g）信頼性。様々なエラーやタイムアウトについて耐性がなくてはならない。（h）セキュリティ。誤操作が事故につながるようなサービスについては，特定のプログラムからしか呼び出しを受け付けないような機構が必要となる。

上のような条件の大部分を満たすものとして OSF の NCS-RPC が挙げられるが，機構が複雑であり，残念ながらまだその環境を利用できるシステムがまだ多くない。NFS が普及しているために最も移植性が

高いと思われるSun-RPCは，初期のRPCであるCourierをそのまま拡張しているため，プログラムの中でネットワークの存在をかなり意識させるものとなっており，多少透過性に問題がある。加速器の制御用のCERN-NC/RPCは最近拡張され[6]，これらの条件のほとんどを実現している。そこで，これをKEK-Linacでテストしてみた。

4．加速器制御でのRPCの応用

CERN-NC/RPCはTCP/IPを持つほとんどのUnix上やOS9やVMSで動作させることが可能で，まずDecStation-Ultrix，SparcStation-SunOS，NeXT，FujitsuA60-SXAに移植してみた。全て既定値のNetworkパラメタを利用した場合，サーバ（サブルーチン）側のプログラムは全く書き換えの必要がない。そのためまず1台の計算機用に制御サブルーチンを用意できれば，簡単な呼び出しインターフェース記述ファイル(IDF)を書くだけでネットワーク対応（RPC対応）のソフトウェアとすることができる。これまではソケット通信のライブラリ呼び出しなどが多く，またアプリケーションごとにプロトコルを決めなくてはならなかったので，RPCの利用はソフトウェアの負担の低減になる。クライアント（メインルーチン）側のプログラムには初期化のために1－3行のファンクションコールの追加が必要であるがサブルーチン呼び出しの部分については全くこれまでのプログラムと同等にすることができる。また，サーバ側の書き換えを行わずに，非同期処理や同報処理を行うことが可能である。これは特に2台以上の計算機と通信を行う場合の全体の並列処理のために有効である。なお，上の条件の（e）と（f）はCERN-NC/RPCでは解決されないので，（e）はネームサーバとNISで，（f）はNTPで解決している。これらによってネットワーク上での名前と時刻の同等性が管理される。

速度についてはアプリケーションによって，また低位のプロトコルの利用の仕方によって異なるが，10MIPSのUnix計算機間の約100バイトの引き数と返り値の転送を伴うRPCで6ミリ秒，1キロバイトの場合で12ミリ秒程度であった。これはソケット通信を直接利用した同じアプリケーションで，それぞれ3ミリ秒と6ミリ秒であったことを考えると，2倍程度の応答速度の悪化になっている。しかし，RPCでは内部機構でエラー処理，変数変換や信頼性の向上を行っていることを考えると特に悪いものではないと思われる。実際，ソケット通信をあからさまに利用した場合に比べ，RPCを利用した場合のソフトウェアの開発と将来の維持管理の負担の低減は大きなものがあると思われる。

実際の応用については現在のところ，現場の機器操作用のサブルーチンのネットワーク経由の呼び出しに利用することを考えていて，クライストロンのRF位相操作等の実現を行っている。今後は，ネットワークを利用しなければならないアプリケーション全てについて応用が可能と思われ，オペレータインターフェースと制御系の間の通信などでも利用価値は高いものと思われる。最初に述べたISISなどのタスクグループ管理のサービスなどと組み合わせれば，さらに高度な制御が行うことが可能になる。

5．まとめ

これまでの分散処理は計算機ネットワークの利用のために大きな負担を払ってきているが，RPCはこれをかなりの部分まで隠し，ネットワークを透過的に扱うことを可能にしている。これまでソフトウェアの負担のために機器の近くに様々な上位処理を置くことができずにいた場合が実際多かったが，今後はRPCを利用して本来の意味の分散処理が可能になると思われる。

参考文献

1) K. Furukawa et.al., Nucl. Instr. and Meth. **A293** (1990) 16.
2) K. Furukawa et.al., Proc. 14th Linear Accelerator Meeting in Japan (1989) 159.
3) K. P. Birman et.al., ACM Transactions on Computer Systems, **9**(1991) 3.
4) P. S. Anderssen et.al., Nucl. Instr. and Meth. **A293** (1990) 225.
5) K. Furukawa et.al., this meeting.
6) K. Kostro, CERN, private communication.