リモートエディタのプロトコルとその有効性

新垣 将史† 河野 真治††
† 琉球大学理工学研究科情報工学専攻 †† 琉球大学情報工学科

概要

ネットワークを介した異なるマシン上に保存されているファイルを編集する既存の方法として、リモートログインや、NFS などを利用する方法が考えられる。

本研究で作成するリモートエディタでは、ファイル編集を行うエディタの機能を、クライアントとサーバの 2 つに分割する。そして、両方にバッファを持ち、そのバッファ間で編集内容の通信を行う。

本研究では、リモートプロシジャコール (RPC) を用いて、リモートエディタの通信プロトコルの提案、実装を行い、その有効性を評価するのがこの研究の目的である。またリモートエディタでは、サーバによって全てのユーザのファイルを管理することになるので、そのための拡張についても考察する。

Remote Editing Protocol

†Masashi Arakaki ††Shinji Kono
†Specialty of Information Engineering, University of the Ryukyus.
††,Faculty of Information Engineering, University of the Ryukyus.

Abstract

Generally, remote-login or NFS can be used to access the network computer to edit the files.

In this research, remote-editor with client function and server fuunction is to be built. It can access both the buffer of client and the buffer of server. The communication of the buffer between the client and the server can be implemented by the remote editor.

The purpose of this research is to verify the effectiveness of the remote-editor using RPC.

1 始めに

リモートエディタとは、サーバ ·アプリケーションによってファイル管理を行い、クライアントであるエディタを用いてサーバに存在するファイルに接続、編集を行うエディタである。

ネットワークを介した異なるマシン上にあるファイルを編集するとき、一般的には、リモートログインで接続し、接続先のマシン上にあるエディタを用いて編集を行うことになる。また、NFS などを用いてファイルサーバをマウントし、ファイルにアクセスするという方法もある。

本研究で目標とするリモートエディタにおいて、編集作業時のデータ送受信は、サーバ側で管理するファイル全体のバッファと、クライアントで管理するウインドウ (画面) にファイル内容を表示するためのバッファ間で行われる。

また、リモートエディタでは、すべてのファイルをサーバで管理することにより、他のクライアントが編集したファイルに対して追加編集を行うといったことができる。そのため、多人数でのプログラム作成や、書類の編集などが容易になるこ

とが期待できる。

2 クライアント・サーバ化による利点

ここでは、リモートエディタが、他の既存のプロトコルと比較して有効と思われる点を挙げる。

2.1 リモートログインを用いたファイル編集

リモートログインを用いたファイル編集では、ローカルマシンはキーボード入力のたびに、そのデータをリモートホストに対してを入力を送る。リモートホストは受け取ったキー入力に対する処理を行い、その結果をクライアントの端末に返す。現在のネットワークの速度では、その遅延を感じることはまれである。しかし、ネットワークの速度が極端に低下したときは、キー入力の度に大きな遅延が起こる。また、完全にネットワークが切断されたときは、編集作業は完全に中断されてしまう。

2.2 NFS を用いたファイル編集

NFS を介した編集では、NFS クライアントはファイルの内容を一度全て受け取り、その受け取ったデータに対して編集作業を行う。そして、保存するときはまたそのデータを NFS サーバに渡し、サーバは受け取ったデータをファイルとして書き込む。この場合、ファイルサイズが小さいときは問題ないが、大きなファイルを編集するときには、そのデータを全て転送することになり、ネットワークやクライアントに対する負荷が大きくなることになる。

2.3 リモートエディタの場合

本研究で目標とするリモートエディタにおいて通信は、サーバ側で管理するファイル全体のバッファと、クライアントで管理するウインドウ (画面) にファイル内容を表示するためのバッファの間で行われる。

開いたファイルのバッファをサーバ側で管理することにより、NFS のように場合によっては大きなデータを送らなければならない、といったことがなくなる。また、クライアントにバッファを持つことにより、ネットワークの切断が起きても、クライアントのバッファを用いて編集作業を行い、ネットワークが回復したところで変更箇所をサーバに送るようにすることで、編集作業の中断を最小限におさえることができる。

また、通常のエディタでは、編集されたファイルを他のクライアントが編集を行うには、そのファイルのアクセス権限の変更や、一旦自分のカレントディレクトリにコピーして、そのファイルを編集するなどの作業が必要になる。

リモートエディタでは、すべてのファイルをサーバで管理することにより、他のクライアントが編集したファイルに対して追加編集を行うといったことができる。そのため、多人数でのプログラム作成や、書類の編集などが容易になることが期待できる。

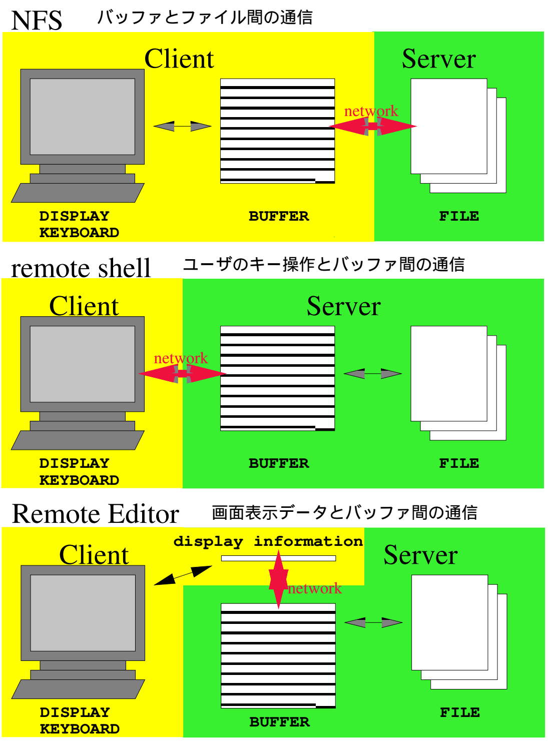

図 1: NFS、リモートログイン、リモートエディタによるファイル編集

3 Remote Procedure Call

Remote Procedure Call(RPC) は、Sun Microsystems によって開発された通信プロトコルである。RPC は、ローカルプログラムから、ネットワークでつながっている別なマシン上のプロシジャにデータを渡して、計算結果を得ることをできるようにするための機能である。

クライアントからリモートプロシジャが呼び出されると、そのデータはスタブによってフィルタに通され、XDR 形式に変換されてネットワークに送られる。ネットワークを介して送られたデータは、サーバ側のスタブによってフィルタに通され、サーバ・マシンの処理できる形に変換される。そして変換されたデータを元にサービスプロシジャが実行される。結果もまた同様に、サーバスタブがフィルタを元に変換し、そのデータがネットワークを通り、クライアントのスタブに渡される。クライアントは結果を受け取り処理を続ける。XDR フィルタを利用することでネットワーク間のデータ形式が統一され、異なるマシン間でも通信が可能となる。(図１)

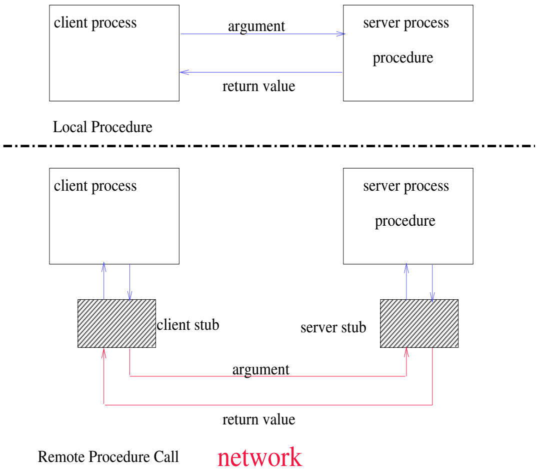

図 2: ローカルなプロシジャコールと RPC における通信

4 PICO について

本稿では元となるエディタとしてワシントン大学により開発された PICO(Pine male Composer) を用いた。リモートエディタの作成は、PICO のソースコードを２つ用意し、一方をクライアント、一方をサーバとして、以下の関数に RPC による拡張を加えることから始めた。(表１)

表 1: PICO における主要な編集操作関数

カーソル操作関数	
forwchar(int f, int n)	カーソルを n 文字進める
backchar(int f, int n)	カーソルを n 文字戻す
forwline(int f, int n)	カーソルを次の行へ移動
backline(int f, int n)	カーソルを前の行へ移動
gotobol(int f, int n)	カーソルを行の先頭へ移動
gotoeol(int f, int n)	カーソルを行の最後へ移動
scrollforw(int n, int movedot)	PageDown キーによる移動
scrollback(int n, int movedot)	PageUp キーによる移動
文字入力関数	
linsert(int n, int c)	カーソル位置に文字 c を挿入
lnewline()	改行時に新しい行を追加する
ldelete(int n, int (*preserve))	カーソル位置から n 文字削除
ファイル入出力関数	
readin(char fname[], int lockfl)	ファイル名 fname でファイルを開く
writeout(char *fn, int readonly)	ファイル名 fn でファイルを書き出す

5 実装

実装は、クライアントのバッファとサーバのバッファが、同じ編集結果を得るようにすることで行った。

実装法の一つとして、クライアントで呼び出された関数と同じサーバ側の関数を、クライアントと同じ引数を与えて実行させることで、クライアントとサーバの編集結果が同じになるようにする方法が考えられる。

この方法では、各操作ごとにクライアント・サーバ間の通信を行うことになる。これは、リモートシェル (telnet、rlogin など) を介して別のマシンのファイルを編集するように、キー操作一つ一つに対して通信を行っているのと似たようなものである。また、リモートシェル同様の問題が発生する恐れがある。そこでまず、編集操作をある程度まとめてサーバへ転送することで、ネットワークの負荷を減らす方法について考えた。

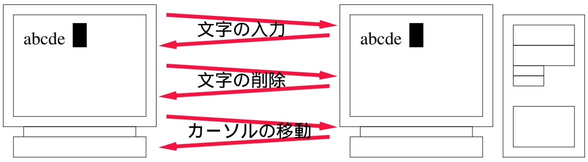

図 3: 同じ関数を呼び出す場合の通信

本稿で実装を行った方法は、まず関数に id を割り振り、クライアントでは関数が実行されるたびにその id および関数への引数を配列に格納する。そしてその配列をサーバに転送し、配列順に id の対応した関数を実行することで、クライアントで

の操作をサーバ側に反映させる方法である。(図２)

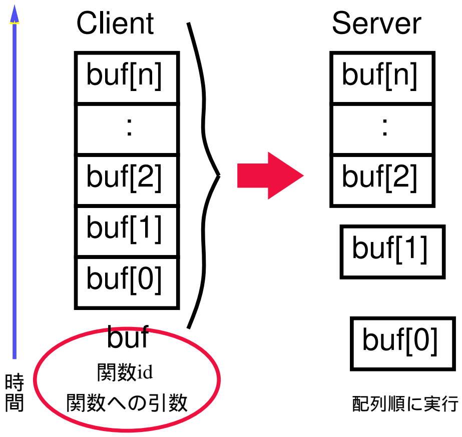

図 4: クライアント・サーバ間の通信プロトコル

関数を id 化してサーバに転送しているのは、異なるエディタ間でも通信を行うことを想定した設計である。

6 評価

次の表は NFS とリモートエディタのファイル保存にかかる時間を測定したものである。

表 2: 100Kbyte のファイル保存にかかる時間 (msec)

	local	NFS	r_edit
1 回目	31	568	97
2 回目	33	477	88
3 回目	37	489	97
4 回目	29	603	88
5 回目	30	531	87
平均	32	534	91

表 3: 1.2Mbyte のファイル書き込みにかかる時間 (msec)

	local	NFS	r_edit
1 回目	341	5770	868
2 回目	351	5724	876
3 回目	344	6896	853
4 回目	347	5944	886
5 回目	346	6608	876
平均	345	6188	872

リモートエディタは、サーバ側で保持されているバッファに対してファイル保存を要求するだけなので、NFS よりも高速になっていることが分かる。

次に、表４はサーバに配列を渡してから、クライアントに処理が返ってくるまでの時間を測定したものである。この時間がユーザの感じる遅延時間となる。

表 4: 配列長による処理時間の変化 (msec)

配列長	1	10	50	100
1 回目	18	32	48	72
2 回目	18	30	48	69
3 回目	24	33	52	73
4 回目	32	29	47	77
5 回目	33	29	46	75
平均	25	31	48	72

100 個の関数の実行にかかる時間でも 0.1 秒以下となっている。これは、サーバへ送信する配列サイズやそのタイミングの工夫により、より良い結果が得られることが期待できる。

7 これからの課題

7.1 複数のユーザの接続

今回はクライアント、サーバ共に PICO を用いた。しかし、PICO には複数のファイルを開くようなマルチバッファ機能はないため、複数のユーザが同時に接続編集するには拡張が必要である。

エディタ Emacs では、マルチバッファを標準で実現している。Emacs のウインドウに表示されているバッファと、クライアントの画面に対するバッファを接続するようにすれば、複数のユーザの接続に対して複数のバッファへの編集を提供することができるであろう。

また、複数のクライアントがサーバに接続されると、あるユーザが作成したファイルを別のユーザが追加編集を行った場合、元のファイルを削除してはいけない。元のファイルはあるユーザにとっては編集中のファイルであり、それをそのまま上書きするのは好ましくはない。これらの問題を解決するようなファイル管理機能を作る必要がある。

今考えているサーバで複数のクライアントのファイルを管理する方法として、ファイル保存の際にディレクトリを作成して、そのディレクトリの中へファイルを保存するという方法を考えている。

作成したディレクトリ名には、保存するファイル名をつけるなどユニークな名前にする。そしてそのディレクトリに保存されたファイルに対して追加編集を行った場合、新しいファイルの保存は、前のファイルのカレントディレクトリに新たにディレクトリを作成し、その中へ保存するようにする。そしてそのディレクトリ名には、バージョン番号を

付加するなどして他のバージョンのファイル (ディレクトリ) と区別する。

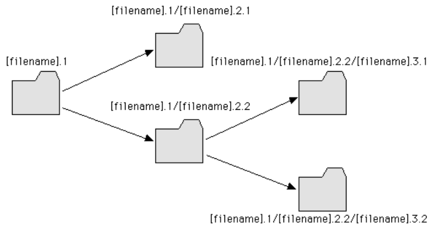

図 5: ディレクトリを用いたバージョン管理

このようにすることで、ファイルのバージョンを木構造で管理することができる。

7.2 異なるエディタ同士の接続

クライアントで用いるエディタが選択できれば、ユーザに対して様々な画面、操作を提供できる。またクライアントは、ユーザに対して画面とコマンド (キー操作) を提供するものであり、サーバで行われている処理は気にしなくてもよい。

異なるエディタ間で接続した場合、クライアントで行われた編集操作をサーバで実行するためには、エディタの編集操作をクライアント・サーバ間で統一する方法が考えられる。これは今回実装を行った、関数を id に変換し、サーバ側で対応した id を持つ関数を実行する方法が有効であると思われる。

7.3 転送データの効率化

クライアント側には画面に表示している分だけのファイルデータがあれば良く、画面に見えていない部分まで受ける必要はない。クライアントは、カーソル移動によるスクロールなど必要に応じてサーバにデータを要求するようにすれば、通信を行うデータをより少なくできる。

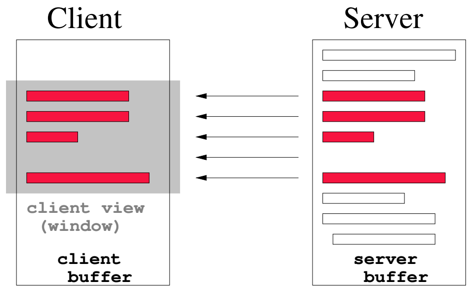

図 6: より効率的な転送

以上の問題を解決することが今後の課題である。

参考文献

[1] John Bloomer 著、森島晃年訳 “RPC プログラミング” アスキー 1995.

[2] Hal Stern 著、倉骨彰訳、砂原秀樹監訳 “NFS & NIS” アスキー 1992.

[3] David A.Curry 著、アスキー書籍編集部監訳 “UNIX C プログラミング” アスキー 1991.

[4] W.Richard Stevens 著、篠田陽一訳 “UNIX ネットワーキングプログラミング” トッパン 1992.