Emacs上のリモートエディタ

新垣 将史† 河野 真治†
†琉球大学理工学研究科情報工学専攻

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が、大きなファイルを扱う場合の通信時間や応答性に問題があった。
本研究では、ファイル編集を行うエディタの機能を分割、双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、リモートエディタを提案した。そしてこれまで、簡単な実装による通信時間の評価や、編集における応用などについて考案してきた。
本稿では、エディタEmacsを用いて実装を行い、また、1サーバ型の接続だけでなく、クライアントエディタ同士による相互編集などについても考察する。

Remote Editor on Emacs

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

Abstract

Remote login or NFS can be used to edit files on network computers. But in case of large file, these methods suffers from response time or large file transfer time. To solve these problems, a remote editor using client server model is shown. The remote editor coordinates both the buffers of the clients and the buffer of the server. We developed prototype and measured the effectiveness of the remote editor.
We apply remote editing protocol to Emacs and build remote editor on it. And we examine the extension of file management for multi user case.

リモートエディタとは、サーバ・エディタによってファイル管理を行い、クライアント・エディタを用いてサーバに存在するファイルに接続し編集を行うエディタである。

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。

これまで、本研究では小型エディタをベースにリモートエディタの試作を行い、既存のプロトコルとのネットワークトラフィックの比較評価をおこなった。

本稿では、より高度なエディタを目指すために、ベースとなるエディタにEmacsを選択し、これを用いた実装を行った。また、これまでに提案したプロトコルの問題点に関して考察し、よりネットワーク効率のよいプロトコルについて考える。

1　リモートエディタについて

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。

1.1　リモートログインを介したファイル編集

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。

1.2　NFS を介したファイル編集

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、保存するときは再びそのデータを NFS を介しサーバに渡され、サーバ上のディスクへファイルを書き込む。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。

1.3　リモートエディタ

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、クライアントエディタのバッファ間で行われる。

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。

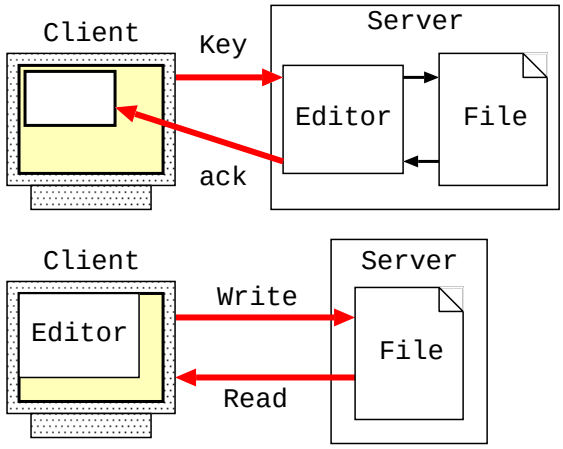

図 1: リモートログインおよび NFS を介したファイル編集

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。

2　ファイル編集に関する考察

2.1　テキストエディタにおける編集処理の流れ

ここでは、テキストエディタにおける、ファイル編集の流れについて述べる。

一般的なテキストエディタでは、読み込んだファイルの内容を、バッファを用意しその中へ格納する。編集による変更は、バッファに読み込まれたデータに対して行われる。そしてエディタの終了時に編集内容の保存は、バッファをファイルに書き出すことで行われる。

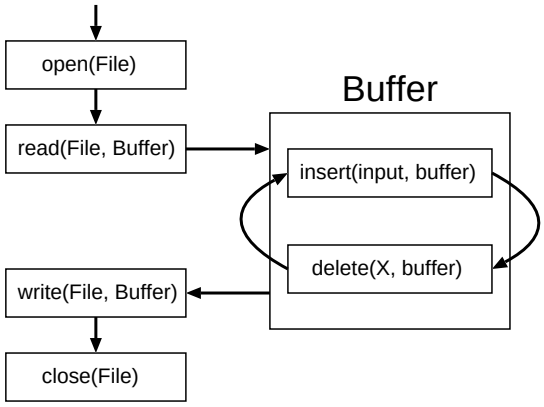

図 2: 編集の流れ

2.2 編集に必要な要素

テキストエディタでファイルを開いた場合、ファイルの内容は全て、メモリ上にバッファリングされる。このうち、ユーザが編集の際に必要な情報は、画面に表示する分のデータである。

テキストエディタでの編集において、ユーザの直接入力による変更は、画面に表示されている部分にしか行われない。また、テキストエディタには、入力文字列の検索・置換機能が備わっていることが多い。文字列の置換も文書編集機能の一部ではあるが、これはユーザ側には見えない部分での編集操作である。

つまり、ネットワークを介した、エディタによる編集では、クライアント側で行うバッファリングは、画面表示分だけ行うのが最適であると考えられる。

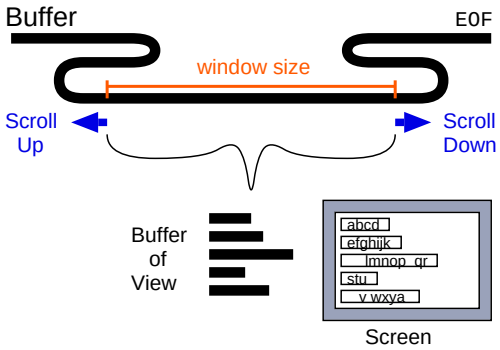

図 3: 本稿で提案するデータ転送

3 設計、実装

ここでは、まずこれまで行った実装例を紹介し、次に本稿における実装で用いた、Emacs エディタと、それを用いた設計および実装例について述べる。

3.1 クライアント・サーバ型リモートエディタ

ここで紹介する実装例は、一つのエディタをサーバとし、そこへ複数のクライアントエディタを接続する。ファイルの入出力および管理はサーバ側で行う。

3.1.1 これまでの実装例

本研究では、PICO というエディタをクライアントとサーバの両方に用い、1 対 1 の通信を行うエディタを試作した。そして、クライアントで行われた編集結果を、サーバのバッファに適用する編集プロトコルとして、行単位による通信と、クライアントで実行された編集操作を、サーバ側のバッファに対して同様に適用する方式を提案した。

前者は、クライアントで編集を行い、行間移動が行われる際に、移動前の行に変更があれば、その行をまとめて転送し、サーバは送られてきた行を、バッファにおいて対応する個所へ置換する方式である。

後者は、あらかじめ編集操作を行う関数に対し ID を割り振り、クライアントでの編集操作のログをキューに格納していく。そして、そのキューがある程度溜ったところでサーバへ送り、サーバはそのキューに格納された ID に対応した関数を順に実行していく方法である。

また、前者の問題点として、行移動については、サーバがクライアントのカーソル位置を常に追従しなければいけないことがある。後者の問題点は、実装時に、用いるエディタの編集操作関数をすべてリストアップし、それらに ID を割り振る手間と、カーソルの移動の連続や、テキストの挿入の直後にその領域を再び削除、など、バッファに直接影響のない処理についても、キューが溜った時点で通信を行わなければいけないことがある。

3.1.2 Emacs による実装

ここでは、本稿の実装で用いた、Emacs エディタと、設計方針および実装例について述べる。

Emacs は、Emacs-Lisp という言語処理系を内蔵しており、Emacs-Lisp を用いて、Emacs 自身から別プロセスを起動し、そのプロセスと通信を行うことができる。

リモートエディタを作成するにあたり、クライアント・サーバともに Emacs を用い、TCP/IP を用いたクライアント・サーバ型のメッセージ送受信プログラムを作成し、それを Emacs 内部で実行することで、エディタ間の通信を行う。

また、Emacs は標準でマルチバッファの機能を備えている。各々のクライアントが個々のバッファを持つようにすることで、ネットワーク切断時の再接続においても、クライアントは切断前の編集状態を維持することができる。そこで、1 つのバッファを 1 クライアントに対応させる形で実装を行った。

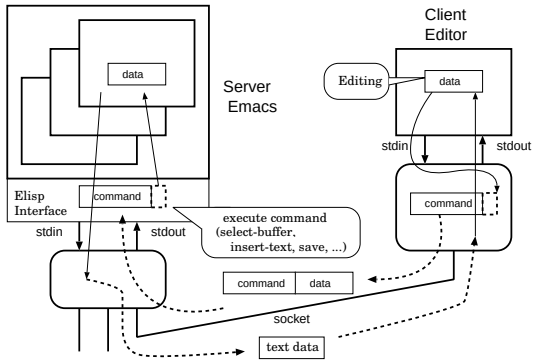

図 4: クライアント・サーバ型リモートエディタ

サーバエディタでは、クライアントによって開かれるファイル全体のバッファとバッファ名、及びクライアントへ送信したバッファ領域へのポインタが管理される。

サーバ通信インタフェースは、クライアントの接続の受け付けおよびソケットの管理と、クライアント側で起動される、クライアント通信インタフェースとの間に、エディタ間のデータストリームを提供する。

クライアントエディタでは、編集に必要な、画面表示分のバッファを持ち、また、その領域のサーバ側に開かれたファイルにおけるオフセット値を保持する。そして、テキスト内での移動などで、画面の更新が行なわれる際には、再びサーバへ表示に必要なテキストデータを要求する。

ファイル保存の際には、編集によって変更の行われた領域のみを、サーバへ送信する。

3.2 プロトコル

クライアントからのパケットは、1 バイトのコマンド識別子に、引数が続く形式とした。

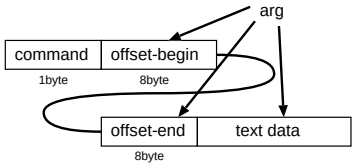

図 5: パケット形式

次に編集コマンドを示す。ここでのサーバとは、編集対象となるバッファを持つエディタ、クライアントとは、そのバッファへ接続し編集を行うエディタとする。

テキストの編集は、基本的に、open、read、write、close、の 4 つから成る。

- open：サーバ側のファイルの読み込みを行う。引数としてファイル名を与える。サーバ側でファイルが開かれるとクライアントへ、そのバッファ名を、コマンド識別子を付加して返す。また、同時に次の read を行う。

- read：クライアントで open コマンドが実行されたとき、またはエディタの行移動などで、画面が更新される際に、必要なテキストデータをサーバへ要求する。引数として、要求するテキストのオフセット値を与える。サーバはこのコマンドを受けると、オフセット値から適当な長さのテキストを、コマンド識別子を付加してクライアントへ返す。同時にサーバでは、クライアントへ送ったテキストに対応するバッファのオフセット値を記憶する。

- write：サーバのバッファに対し、クライアント側で行った変更を加える。引数として、テキストデータおよびそのオフセット値を与える。サーバはこのコマンドを受けると、サー

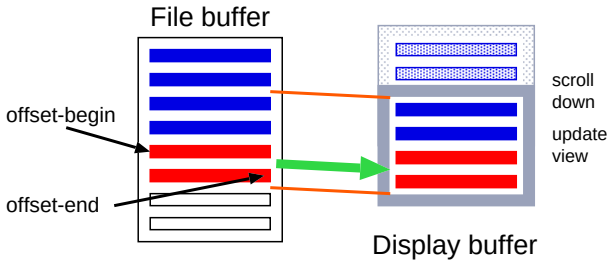

図 6: read コマンド

バ側のバッファにおけるオフセット値と、受け取ったデータに含まれる引数のオフセット値を元に、現在のバッファ領域のテキストを受け取ったデータに置換する。そしてサーバは、クライアントに対して、新しいオフセット値を返し、そのオフセット値を記憶する。本実装において、write コマンドは、クライアントの画面更新の際に、クライアントビューから外れた領域において更新があった場合、その領域に対して実行される。

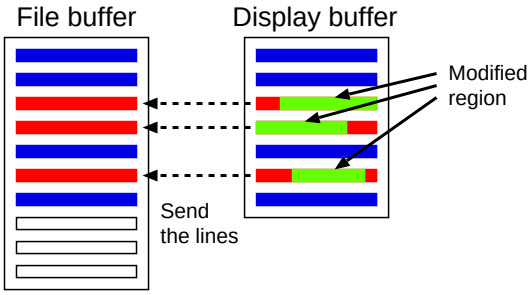

図 7: write コマンド

- close：編集バッファを破棄する。サーバはクライアントとのコネクションを切断する。

3.3 その他の実装法について

ここでは、エディタ同士で互いのバッファに対して相互に編集を行う実装を紹介する。

この実装では、サービスプログラムに接続されたエディタが、プログラムを介し、互いに通信することで編集を行う。

サービスプログラムでは、エディタ間の接続の

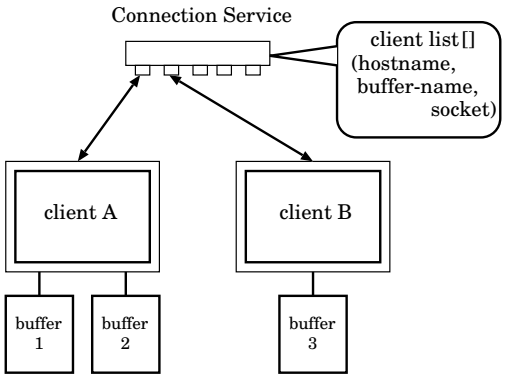

図 8: リモートエディット・サービス

仲介を行い、クライアント・エディタのホスト名や編集に用いられるバッファ名、クライアントソケットを管理する。クライアントは、編集コマンドを、編集を行うバッファ名をアドレスとして、サービスプログラムへパケットを送り、サービスプログラムは、バッファ名に対応したバッファを持つクライアントへ、コマンドを転送する。

この方法では、先にサービスプログラムへ接続されているエディタのバッファに対し、他のエディタがアクセスし、同時に編集を行う。この実装の利点は、あるユーザがローカルで編集中のファイルを、別なユーザが修正を施したいときなどに、ftp などを用いずに直接修正が行なえることなどである。

ただし、この実装は複数バッファを持つエディタでないとむずかしいと考えられる。

4 考察、課題

4.1 マルチユーザに関する考察

Emacs における 1 つのバッファを、1 クライアントに対応させることにより、複数のクライアントが同時に接続し、編集を行うことができる。しかし、同時に同じファイルへの複数ユーザによる編集が行われる可能性が発生する。

解決策の一つに、同じ名前のファイルを作成し、同時に各々ディレクトリを作成し、その中へ別々に保存、管理することで、文書をバージョン分岐させる方法が考えられる。

そのほか、あるユーザが、文書構造を破壊するような改変を行うと、他のユーザは、そのテキストに対しての追加編集が困難になることが予想される。

これは、あるユーザの変更が、他のユーザから見た場合に、文書としての意味を成さないということから発生する。

4.2 文書の構造について

何らかの文書を編集する場合を考えると、その文書の種類や構造から、編集単位を与えることができる。

例えば、本文書のようなテキストの構造は、節ごとに分けることができる。また、プログラムソースコード、例えば C 言語によるソースコードの場合でも、プリプロセッサ部、変数およびプロトタイプ宣言部、各関数部、のような文書構造を持つ。

よって、複数ユーザによる同じファイルへの変更を、文書構造単位により行うことで、より効率的な編集が出来ると考えている。

また、上で述べた、同じファイルをベースとした複数の変更についても、文書構造ごとの変更にすることで、複数のバージョンを差分形式で管理することができると思われる。

文書構造を与えるための技術に、マークアップ（タグ付け）言語 XML がある。XML では、タグを作成し、そのタグを用いて文書に構造を与えることができる。XML を用いテキストに構造を与えることで、複数ユーザにより編集されるファイルを効率的に管理できるのではないかと考えている [4]。

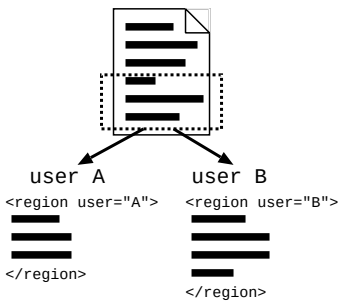

図 9: 文書のバージョン分岐

4.3 異なるエディタによる接続

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操作、環境を提供できる。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。

異なるエディタ間の通信では、編集を行う関数に対して与えるデータ形式を統一しなければならない。

本稿では、どのエディタでも最低持っているであろうテキストの挿入、削除コマンドと、テキストデータのオフセット値を用いた方法を紹介した。

本稿の実装では、クライアントエディタとクライアント通信インタフェース間の通信および、パケットの展開処理に、EmacsLisp を使用している。この部分に相当するプログラムを、エディタごとに作成することで、通信が可能になると考えられる。

参考文献

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

[2] 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.

[3] 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.

[4] 新垣将史、河野真治 ”リモートエディタのプロトコルとその XML への応用” 第１６回日本ソフトウェア科学会大会論文集 1999.