可変長ブロック単位で操作を行うファイルシステムの開発

05G467 津 紀孝 （最所研究室）

本研究ではファイルを可変長ブロックの集合として考え、可変長ブロック単位で操作を行うファイルシステムを提案する。この提案するファイルシステムは可変長ブロック単位で挿入や削除といった操作を行うことができる。ファイルシステムの設計を行い、そのプロトタイプを実装しその性能を計測した。

1 はじめに

従来のファイルシステムはファイルを単純なバイトの集合として管理してきた。ファイルに対する操作は元のデータに上書きを行うか、ファイルの最後に新しいデータの追加を行うか、ファイルを任意の大きさに切り詰めを行うことしかできなかった。

本ファイルシステムではファイルを可変長ブロックの集合として扱う。可変長ブロックをリンクでつなぎ、リンクを操作することで、従来のファイルシステムではできなかったファイルの途中にデータを挿入する操作や、ファイルの途中にあるデータを削除する操作を行うことができるようになる。このファイルシステムはシステムログやブログなどに応用することができ、システムログの中から不要な部分だけを削除したり、ブログの中のコメントを挿入することができるようになる。

卒業研究 [1] においても可変長ブロックとリンク構造を利用したファイルシステムの研究を行ってきたが、このファイルシステムではファイル操作単位でバージョン管理を行うためにリンク構造と可変長ブロックを利用していたのであり、可変長ブロックのリンクを操作して挿入や削除といった操作を行うものではなかった。本研究は可変長ブロックのリンクを操作して挿入や削除といった操作をおこなう。

2 概要

提案するファイルシステムは従来のファイルシステムと同様にディレクトリやファイルを持ち、それぞれ作成時間や更新時間やファイルサイズなどの情報を持っている。ファイルは可変長ブロックの集合で構成され、可変長ブロックはリンクによってつながっている。各可変長ブロックはリンク構造を作成するために自身の前になる可変長ブロックへのリンクと自身の後になる可変長ブロックへのリンクを持っている。

ファイルに対する操作は、可変長ブロック単位で行う。図 1 に示すように、ファイルの末尾に新しい可変長ブロックを追加する操作と、指定した可変長ブロックの前に新しい可変長ブロックを挿入する操作と、指定した可変長ブロックを削除する操作ができる。

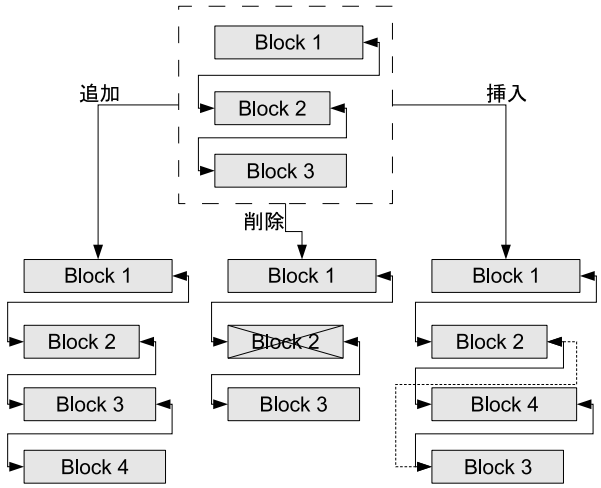

図 1: ファイルの操作の概念図

3 設計

データ構造として、ファイルシステムの管理情報を保存するためのファイルシステム構造と、ディレクトリの管理情報を保存するためのディレクトリ構造と、ファイルの管理情報を保存するためのファイル構造とファイルの可変長ブロックを保存するためのライン構造を設計した。

ファイルを操作する機能として、ファイルを作成する操作とファイルを開く操作とファイルを閉じる操作を設計した。また、ファイルの可変長ブロックを操作する機能として、可変長ブロックを追加する操作と可変長ブロックを挿入する操作と可変長ブロックを削除する操作と読み出し操作とシーク操作を設計した。

追加操作はファイルの最後の可変長ブロックから新しく追加する可変長ブロックへリンクをつなげ、ディスクに可変長ブロックを書き込む。挿入操作は現在位置の可変長ブロックの一つ前の可変長ブロックから新しく追加する可変長ブロックへリンクをつなぎ、新しく追加する可変長ブロックから現在位置の可変長ブロックへリンクをつなげ、ディスクに可変長ブロックを書き込む。削除操作は現在の可変長ブロックに削除したという印をつける。読み出し操作は、指定したサ

イズを可変長ブロックのリンクをたどってディスクからデータを読み出す。シーク操作は、指定した位置へ可変長ブロックのリンクをたどって移動する。

4 実装

設計したファイルシステムのプロトタイプを実装した。通常のファイルシステム上のファイルを擬似的なストレージデバイスとみなし、その擬似的なストレージの操作を行うようにして、設計したデータ構造と機能をそれぞれ C 言語で実装した。

5 計測

作成したプロトタイプの性能を計測した。性能計測は表 1 に示す環境で行い、Linux に標準として存在する time コマンドを使ってファイル操作群の経過時間を計測した。

表 1: 実験環境

CPU	Pentium4 2.53GHz
Memory	512MB
HDD	16MB Cache
Interface	ATA UDMA 100
FileSystem	ext3
OS	Fedora Core 4

作成したプロトタイプとの比較対象として、従来のファイルシステムの性能を計測した。通常の書き込みを行った normal write と、書き込み同期を行った sync write と、カーネルバッファリングを利用しないダイレクト IO を利用し書き込み同期を行った direct write をそれぞれ計測した。1 回に書き込むサイズを変えて 2MB のデータを書き込んだ結果を図 2 に示す。

この図から、書き込み同期を取ると通常の書き込みに比べ非常に時間が掛かることがわかる。また、カーネルバッファリングを利用しないものと書き込み同期だけを取ったものを比べると 3 秒程度の差が出ることがわかる。

次に、作成したプロトタイプの追加操作と挿入操作の性能を計測した。書き込み同期を取った sync. 追加と sync. 挿入と、カーネルバッファリングを利用しないダイレクト IO を利用し書き込み同期を取った direct. 追加と direct. 挿入をそれぞれ計測した。可変長ブロックのサイズを変えて 2MB データを書き込んだ結果を図 2 に示す。

図 2 の direct. 追加と direct. 挿入の比較からバッファリングを利用しない場合では挿入操作のほうが時間が掛かることがわかる。しかし、sync. 追加と sync. 挿入の比較からバッファリングを利用することで追加操作と挿入操作の差がほとんどなくなることがわかる。また、バッファリングを利用することで操作に掛かる時間が半分になることがわかる。

図 2 より、バッファリングを利用しない場合、追加操作はダイレクト書き込みに比べて 2.5 倍程度の時間がかかり、挿入操作はダイレクト書き込みに比べて 3 倍程度の時間が掛かる。バッファリングを利用した場合では、追加操作も挿入操作も同期書き込みに比べて 2 倍程度の時間が掛かる。

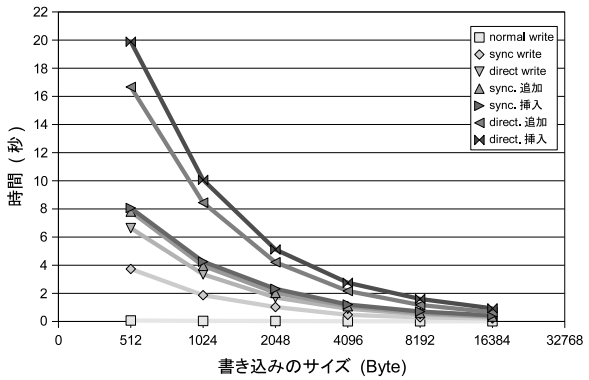

図 2: 性能計測の結果

6 まとめ

本研究では可変長ブロック単位で操作を行うファイルシステムの開発を行った。ファイルシステムの設計とそのプロトタイプを実装し、その性能を計測した。現在のプロトタイプからは従来のファイルシステムに比べ 2 倍以上性能が悪くなっている。しかし、従来のファイルシステムではできなかった操作を行っておりこの程度の性能の変化は当然であると思う。また、リンク構造を改良したり、バッファリング方法を改良することで性能を改善することができると考えている。

今後の課題は、提案するファイルシステムはリンクが線形リストであるため、ランダムアクセスの性能が悪いと考えられるので、リンク構造を変更し、B 木のような構造を利用してランダムアクセスの性能を向上させることと、現在のところ削除したファイルや可変長ブロックを未使用領域として回収していないので、削除した領域を回収するガベージコレクションの実装を行うことと、このファイルシステムに適したバッファリングを考えることである。

参考文献

[1] 津 紀孝,「システムコールレベルでのバージョン管理機能を持つファイルシステムの研究」香川大学工学卒業論文 2005