リンク構造ファイルシステムの効率化および従来のファイルシステムとの互換操作の設計・実装

10G470 小林 憲弘 (最所研究室)

更新操作の記録から履歴管理を行えるリンク構造ファイルシステムにおける，アクセスの効率化および従来のファイルシステムとの互換操作について述べる．

1 はじめに

我々の研究室では，すべての更新操作を記録することによりファイルの履歴を取得するファイルシステムであるリンク構造ファイルシステムの開発を行っている [1]．本システムは可変長のブロックをリンク構造で構成することで，従来とは異なるデータ操作である，ブロック単位での挿入，削除操作を実現している．先行研究では，複数ファイルの同時書き込みなどによりデータが分散されて配置される問題に対し，ファイル毎に予約領域であるリザーブ領域を導入することで解決を図った [2]．しかし，リザーブ領域では履歴の管理までは考慮していなかったため，履歴情報がファイルから離れた位置に配置されてしまいアクセス効率が悪くなる．また，本システムでは挿入，削除といった特有の操作を用いてファイル操作を行うため，従来のファイルシステムで動作するアプリケーションから利用することができない．

本研究ではこれらの問題を解決するために, 履歴情報へのアクセスを考慮したディスク領域管理および従来のアプリケーションから利用できるように互換操作の設計・実装を行った．

2 ファイルシステム概要

リンク構造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つが，それらはリンクで連結された形となっている．ファイルは可変長ブロックの集合で，これらのブロックもリンクで連結された形になっている．可変長ブロックはテキストファイルの段落や行，データベースのレコードなどのデータの塊を意味し，ファイル操作の単位である．

リンク構造ファイルシステムではブロックの挿入と削除で操作を行う．図 1 に示すように，挿入操作では，指定したブロックの次の位置に新しいブロックを挿入し，削除操作では，リンクの繋ぎかえによって指定したブロックを削除する．これらの操作が行われるとき，操作前のリンクの状態を LinkHistory 構造 (以降 LHist) に保持することで，操作毎の履歴を残すことができる．また，LHist をたどることで過去の状態を取り出すことができる．

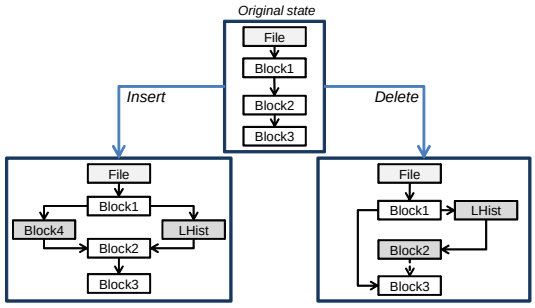

図 1: ファイル操作と履歴

3 アクセスの効率化

リンク構造ファイルシステムでは，リザーブ領域 [2] を用いた領域管理を行っていた．リザーブ領域ではファイルの実データであるブロックを近い位置に配置することによりシークを減らしアクセス効率の向上を図っている．しかし，LHist はファイルシステムで管理されており，リザーブ領域から離れた位置に配置される．そのため，挿入や削除，履歴の読み出しにおいてリザーブ領域と LHist の両方にアクセスする必要があり効率が悪くなる．そこで本研究では，リザーブ領域においてブロックと LHist の両方を管理することにし，その構造を新たにアロケーションブロックと定義することにした．

アロケーションブロックの構造を図 2 に示す．ファイルは複数のアロケーションブロックを持ち，データはすべてアロケーションブロック内で管理する．アロケーションブロックはアロケーションブロックを管理するヘッダとデータを割り当てる領域から構成される．ブロックは固定長のヘッダ (DBHeader) と可変長の実データ (DataBlock) に分けることにした．ブロックヘッダと履歴情報を同一のサイズにしており，領域の先頭から割り当てを行う．実データは領域の末尾から割り当てを行う．この構造により，ブロックヘッダと履歴情報といったメタデータと実際のデータを分けて管理でき，ブロックヘッダや履歴のみ更新する場合でも効率よくアクセスできるようになる．また，アロケーションブロックはファイルのバッファリングの単

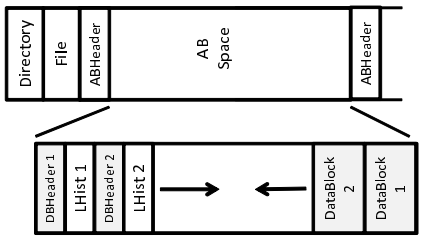

図 2: アロケーションブロックの概要

位として利用することによりアクセス効率を向上させる.

4 互換操作

リンク構造ファイルシステムではブロックの挿入, 削除操作といった操作しか行えないため, ストリーム形式 (バイト単位) で操作を行う従来のファイルシステムで動作するアプリケーションから利用できない. そこで, 従来のアプリケーションにおいて利用できる互換 API を提供することにした. 以下に互換 API のデータの書き出し (write), データの読み出し (read), 読み書き位置の変更 (seek) の実現方法について述べる.

seek seek はバイト単位で指定した位置に読み書き位置を変更できる必要がある.. 互換 API においては, ブロック単位の情報に加えてバイト単位の情報を保持することで操作を実現する.

read read では, ブロックの途中から必要なだけデータを読み出せる必要がある. 互換 API では, seek 同様にバイト単位の情報を保持することでブロック途中からの読み出しを実現する.

write write は新しいデータ書き込みのたびにブロックを作成し, ブロックの挿入削除を行うことで実現する. write による既存のブロックを更新を図 3, 複数のブロックを更新する場合を図 4 に示す. 既存のブロックを更新する場合, 更新されたブロックを挿入し, 元のブロックを削除することによって実現する. 複数のブロックを更新する場合, 更新された複数のブロックを一つのブロックとし挿入し, 元のブロックすべてを削除することによって実現する.

5 性能に関する考察

5.1 アロケーションブロック

アロケーションブロックはリザーブ領域に比べ, 履歴を用いた書き込み, 読み込み共にブロックと履歴情報との距離が短くなり, アクセス効率が向上する. また, ディスクバッファを用いることにより, さらにディスクへのアクセスを減らすことができる. アロケーショ

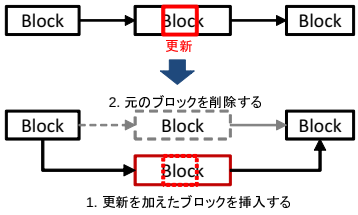

図 3: write によるブロック更新

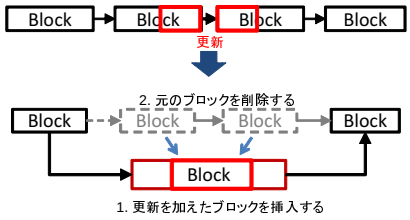

図 4: write による複数のブロック更新

ンブロックではファイル単位での管理が行えるため, 複数のファイルを同時に扱った場合でもファイルシステムで履歴管理を行っているリザーブ領域に比べて他のファイルからの影響を受けにくい.

5.2 互換 API

互換 API と一般的なファイルシステムの操作とを比較する. 互換 API で最もアクセス性能の差が考えられるのは write である. write では前後のブロックヘッダと LHist の書き込み, 元のブロックの削除が余分に発生するため, 従来の write と比べて時間がかかる. この性能差を軽減するために, ブロックの更新に対応する操作やディスクバッファといった対策が必要となる.

6 おわりに

本研究では, リンク構造ファイルシステムのアクセスの効率化のためにアロケーションブロックを導入し, 履歴をより効率的に利用できるようにした. また, 従来のアプリケーションへの互換 API を提供し, 本システムの履歴を利用できるようにした.

今後の課題としてディスクバッファの設計や互換操作の効率化, 未実装部分の実装, 評価と考察内容の検証が必要である.

参考文献

[1] 津紀孝, 最所圭三, “行指向ファイルシステムについて”, 平成 18 年度 電気関係学会四国支部連合大会,15-37, p.242, 2006.

[2] 小林憲弘, 大橋洸一, 最所圭三, “リンク構造ファイルシステムにおける領域管理”, 平成 22 年度 電気関係学会四国支部連合大会, 17-25, p.297, 2010.