トランザクション機構を持つ Log-Structured File System の設計と実装

4M－3

大軒 淳志　猪原 茂和　益田 隆司
東京大学 大学院 理学系研究科 情報科学専攻

1　はじめに

分散協調作業などの新たなアプリケーションでは、並行したファイルアクセスやフォールトに対して、ファイル内のデータの一貫性を保つためにファイルシステムレベルにおいてもトランザクション機構 [1] が重要である。本論文では Log-Structured File System (LFS)[2, 3] に、トランザクション機構を付加した新たなファイルシステムを提案、実装する。このため、従来の LFS に加え、(1) ユーザープロセスからのトランザクション要求の管理、(2) トランザクション内外での一貫したファイルアクセス、の二つの機能を提供する。

従来の Unix File System (UFS) に代表されるファイルシステムでは、少量の変更であってもファイルを定められたブロックに記録していた。この非効率性を指摘し、LFS ではファイルへの変更点をディスクの連続した空き領域に記録することにより性能向上を計っている。

本システムでは、LFS に内包される木構造に複数のビューを持たせることによってトランザクションを実現する。これらの実装にともなう動作中のオーバーヘッドの増加は小さく、トランザクションが実装された新しいファイルシステムは従来の LFS に上位互換なファイルシステムとして振舞う。

2　従来のファイルシステムへの変更:LFS の利用

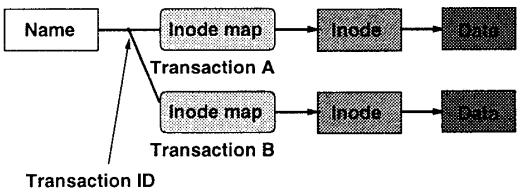
ファイルシステムレベルでのトランザクション支援機構に必要なのは何であろうか。ここではファイルシステムを多重化し、各ユーザーが異なったビューを持てるようにすることでトランザクション支援をすることを考える。

UNIX のファイルシステムにおいて、内部的にはファイルは木構造で管理されている。UFS においては **inode** と呼ばれる固有の構造体 (inode 番号によって識別)、実際に内容を記したディスクブロックの番号、の二層である。これに機能付加して複数のビューを持たせる場合、外部からの引数であるファイル名から inode への変換を多重化する方法と、inode からブロック番号への変換を多重化する方法という二つの実現法がある。だが、これらの変換は元々単一であることを大前提として設計されているので、単純な構造体やそのポインタ参照を多用している。これを多重化することは元のファイルシステムに大改造を要するばかりか、得られる性能もかなり従来より低下することが予想される。

一方、UFS が read access に最適化されたアクセスをしていたのに対し、LFS は write access 時のパフォーマンスが最適になるような方針でアクセスを行なっている。具体的には書き込みを一括して連続なブロックに行ない、既存のファイルへの変更においては上書きをしない。ファイルシステム全体は使用するにつれて断片化し、ファイルシステムに対する garbage collection(GC) によってこの断片化を解消する。

UFS が二層の構造でディスクのブロックを同定していたのに対し、LFS は GC のため、**inode map** と呼ばれるファイル名と inode の中間に位置する構造を一層余計に持っている。Inode map はディスク上のどこのブロックに記録された inode を実際に使ってよいか、ということをシステムに教える構造である。Inode map から inode、さらにディスクのブロックという順にポインタをたどり、どこからもたどれないブロックはすべて garbage として回収、再利用されることになる。

そこで、我々のファイルシステムでは inode map を複数持つことによってトランザクションに必要な複数のビューを効率的に実現する。Inode map を複数持つことにより、ファイル名から inode への変換の時点で多重化が可能である。すなわち、単一のファイル名に対して個別の inode を登録することができる。その個別の inode は当然別々のディスク上ブロックを指すことができるから、うまく求める多重ビューを提供することができる。Inode map は木構造の最上位にあるため、通常の全てのアクセスが通過する。これを多重化することで UFS の持っていた inode とブロックという内部構造全体をより下位の構造に手を加えることなく多重化できる。



Inode map は stable な状態ではディスク上に存在するものであるが、通常はキャッシュして参照される。複数の inode map を同時にキャッシュしておいて何らかの外部の要求に従って使用するものを切り換えるようにすれば、変更はどの inode map へのポインタを参照するかを変えるだけで済む。よって、この多重化にともなうオーバーヘッドはさほど大きくないと考えられる。

The Design and Implementation of a Transactional Log-Structured File System
A. Ohnoki, S. Inohara, and T. Masuda
University of Tokyo
7-3-1 Hongo,Bunkyo-ku,Tokyo 113,Japan

3 方針

ここでは、既にLFSがある4.4BSDベースのNetBSD 1.0上にトランザクション支援機構を持つLFSを実装する。この変更は従来のLFSに互換性を保ったまま行なうことができるので、従来のLFS用のファイルシステムも利用可能なシステムを作成することができた。

実際には単純に多重化するだけではトランザクションの内外でのファイルの共有に関する問題が残る。LFSの性質によって改変されたファイルの改変された部分は自動的に複製されるが、それがどのトランザクションによって作成されたもので、どのビューから有効であるかが不明瞭になってしまう。これを解決するため、ファイルがどのトランザクションからアクセス可能か、という情報をinode構造体に付加する。

さらに、実際にトランザクション機構を利用できるようにシステムコールのインターフェースも合わせて必要である。必要なシステムコールはトランザクションの開始を指示するtransaction_start()、トランザクションのcommitに相当するtransaction_commit()、abortに相当するtransaction_abort()の三種である。また、それらのユーザーから要求を受けたトランザクションを管理するために各プロセスに関連付けられたトランザクションID、そしてファイルシステム自体が実際にどれだけのビューを持っているか、といった情報を内部構造として保持しておく必要がある。当然、トップレベル以外のトランザクション下のみからアクセス可能なファイルは、通常のgarbage collectorにはgarbageとして見えてしまうので、garbage collectorもこの情報を参照してファイルシステムのGCが正しく行われるように変更する。

4 関連研究

LFSにトランザクション支援機構を導入する試みはSeltzerによって既に行なわれている[4]。この研究は次の点において本研究と異なる。

- Lock managerが存在し、本来のデータベース的用途を主眼としている。
- Inodeは単一で、各々がトランザクション外のものとは別にトランザクション専用のディスクブロックへのリストを保持している。
- トランザクション下でのdirty pageはメモリ上に保持され、commitされるまで書き込まれない。

これに対し、我々は通常のファイルシステムと同様の形態での利用を目的としたシステムを設計した。

5 現状とまとめ

現在、NetBSD上で稼働しているLFSのシミュレータにトランザクション支援機構を付加したLFSの一部の機能が実装されている。次の機能を既に実装した。

- Inode mapの多重化
- トランザクション支援のための多重ビューの提供
- システムコールのインターフェース

	拡張前	拡張後	
トランザクション	なし	未使用	使用
transaction_start			112.2
read	10.3	10.3	10.3
write	11.7	11.7	11.7

表 1: 従来のLFSとの速度の比較 (単位:msec)

未実装の機能はgarbage collectorおよびcommit時の各ビューの融合に関する機構であるが、従来のLFSとの速度比較においてはこれらを加味しなくても同等であろう。

ここまで実装が終了した時点におけるトランザクション機構を付加したことによる各システムコールレベルでの従来のLFSからの性能低下は一度inode mapをディスクから読むオーバーヘッドのみで、これは測定誤差の範囲内である。表1はトランザクション機構を付加する前のLFSと付加したLFSとでシステムコールを1000回実行して平均した結果である。transaction_start()は内部的な16KB(inode mapの初期サイズ)のコピーに相当するもので、read(), write()はそれぞれ4KBづつのファイルの読み書きである。

複数のファイルシステムを同一のバッファで扱うことと同じことになるので、最終的な性能低下はキャッシュ効率の低下に還元されるものと思われる。トランザクション機構を付加することによるオーバーヘッドの増加は低くおさえられており、機能的なものを考えれば十分な速度ということができる。

参考文献

[1] J. Eliot B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, Cambridge, Massachusetts, 1985.

[2] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 1–15. Association for Computing Machinery SIGOPS, October 1991.

[3] M. Seltzer, K. Bostic M. K. McKusick, and C. Staelin. An implementation of a log-structured file system for UNIX. In *Proceedings of the Winter 1993 USENIX Technical Conference*, pages 307–326. USENIX Association, 1993.

[4] M. Seltzer. Transaction support in a log-structured file system. In *Proceedings of the 9th International Conference on Data Engineering*, pages 503–510, March 1993.