Vol.2012-HPC-133 No.17
2012/3/26

高速なストレージにおけるファイルシステムの調査と検討

鷹津冬将[†1]　建部修見[†2,†3]

本稿では，SSD などの高速なストレージにおける効率的なファイルシステムの実現に向けて，Log Structured File System をベースにストレージへの書込が逐次書込となるよう mylfs の設計を行い、プロトタイプ実装を用い様々なアクセスパターンによる評価を行った。逐次書込の評価では、mylfs は raw device の性能に対し、HDD で書込は 94%、読込は 97%の性能を達成し、SSD で書込は 79%、読込は 98%の性能を達成した。ファイル更新の評価では、逐次更新、ランダム更新ともに mylfs は他のファイルシステムに比べ高い性能を示し、もっとも性能の良かった ext3 と比較して、逐次更新では HDD で 138%、SSD で 121%、ランダム更新では HDD で 572%、SSD で 135%の性能を達成した。

Study of a file system for high-speed storage

Fuyumasa Takatsu[†1] and Osamu Tatebe[†2,†3]

This paper designed a mylfs that writes data sequentially to a storage to study an efficient file system for high-speed storage, and evaluated the prototype implementation by various access patterns. Mylfs achieved 94% and 97% of the raw device performance for sequential writes and reads, respectively, using an HDD, and 79% and 98% of the raw device performance for sequential writes and reads, respectively, using an SSD, respectively. Mylfs outperformed other file systems for a file update benchmark, and achieved 138% and 121% of the ext3 performance for sequential updates using HDD and SSD, respectively, and 572% and 135% of the ext3 performance for random updates using HDD and SSD, respectively.

†1 筑波大学情報学群情報科学類
College of Information Science, University of Tsukuba
†2 筑波大学システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba
†3 独立行政法人科学技術振興機構 CREST
JST CREST

1. はじめに

近年，様々な分野の計算機で使用されるストレージデバイスがハードディスクから SSD や RAM DISK といった読み書きが高速なストレージデバイスに移行しつつある．SSD はハードディスクと違い記憶媒体としてフラッシュメモリを用いるストレージデバイスである．フラッシュメモリを用いていることからシークタイムがほぼないためランダムアクセスにおける性能がハードディスクよりも優れているが，書き換えるたびにフラッシュメモリのトンネル酸化膜が劣化するため，常時書き込みを行うなど同じセルを何度も書き換えるとハードディスクよりもハードウェアの寿命が短くなる場合がある．また，一般に利用されているハードディスクについても記録密度の向上により，旧来のハードディスクに比べると格段に読み書きの性能が向上した．しかし，依然としてハードディスクでは読み書きの際に起こるシークにかかる時間が SSD などに比べて著しく長いなど構造上の制約がある．

ストレージの処理能力が向上した一方で SSD には様々な制約があるため，SFS[1)] の様な様々な研究があり研究開発が活発に行われている．

書き込み時のシークの回数を大幅に削減したり，ファイルの書き換え時にストレージ上の同じ場所を書き換えないファイルシステムの１つとして Log Structured File System[2)] がある．

本研究では，HDD，SSD などにおいて効率的なファイルシステムの実現に向けて，ストレージへの書込が逐次書込となるよう設計を行った mylfs のプロトタイプ実装を行い，様々なアクセスパターンによる評価を行う．

本稿は６章で構成される．第２章では，mylfs の設計について詳述する．第３章では，本研究で行ったファイルシステムの比較のために実装した mylfs のプロトタイプ実装について詳述する．第４章では，ファイルシステムの比較を行う実験とその評価を示す．第５章では，関連研究について述べる．第６章で結論と今後の課題を示す．

2. mylfs の設計

Log Structured File System は Mendel Rosenblum らによって設計されたファイルシステムである．Linux では NILFS2 などの実装がある．本章ではこの Log Structured File System をベースとした mylfs の設計について詳述する．

Ext3 など現在広く使われているファイルシステムは書き換え時にシークが大量に発生する事からランダムライトの性能が低下している．また，現在のファイルシステムではファイ

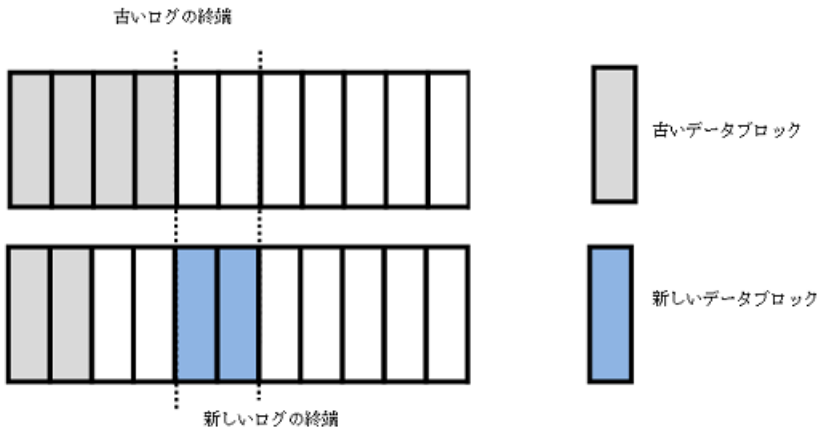

図 1 Log-Structured File System アーキテクチャ

ルの属性と実際のデータが分けられて保存されている．そのため読み込み時にもシークが大量に発生する．これらの問題を解消するファイルシステムの 1 つに Log-Structured File System などがある．Log-Structured File System のアーキテクチャを**図 1** に示す．

Log-Structured File System はすべてのデータを 1 つの構造化順次ログとし，書き換え時においても実際にブロックの書き換えを行わずログに追記する形でシークの回数を減らしている．Log-Structured File System にファイルを書き込む様子を**図 2** に示す． 図 2 では Dir1/File1 というファイルがすでにあるなか Dir1/File2 というファイルを書き込む様子を表している．この図で表している Inode はファイルの属性である．元のログの末端にファイルのデータと Inode，ファイルが含まれるディレクトリエントリとその Inode を連続して書き込み，すべての Inode の座標を含んだ Inode Map を書き込む．ファイルの作成に関わるすべての書き込みを 1 つの流れとして書き込むため，シークは最初にログの終端に移動するときにのみ発生する．そのためファイルの書き込み速度が向上する．読み込み時においては旧来のファイルシステムと同じように読み込むためシークの回数は変わらない．また 2 の例ではディレクトリエントリや，Inode マップについて書き換えが発生しているが，ストレージ上で同じ場所が書き換えられるといったことは発生していない．

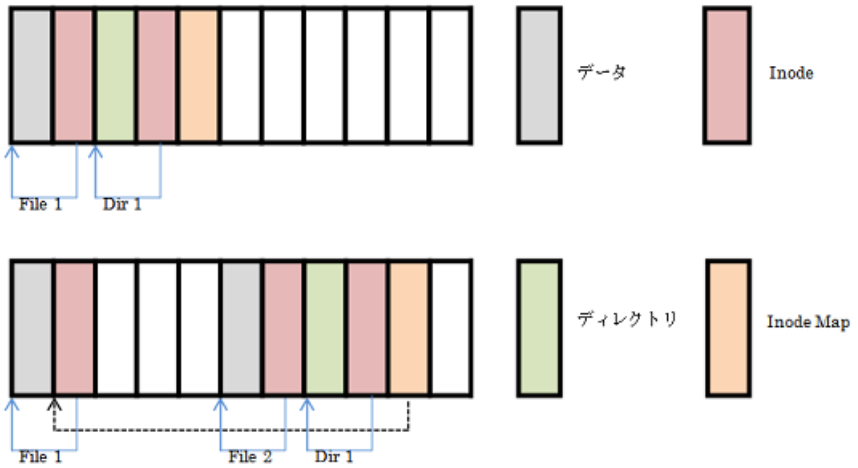

図 2 Log-Structured File System のファイルの書き込みの様子

3. mylfs のプロトタイプ実装

3.1 File System in Usersystem について

File System in Usersystem (FUSE) [3] は一般ユーザーがカーネルを書き換えることなくファイルシステムを作成することのできる機能を提供するカーネルモジュールであり，Linux 2.6.14 以降のカーネルで利用できる．FUSE を利用することにより POSIX に定められたファイルスシステムに関するすべての機能を実装することなくファイルシステムを提供することができる．また非特権ユーザーであってもファイルシステムを提供することができる．

FUSE によってファイルシステムを提供し ls コマンドを実行した際の動作を**図 3** に示す．ls コマンドを実行した場合，glibc によってカーネル空間に移り，VFS と FUSE を経由して再度ユーザー空間に戻り，ユーザーが実装したファイルシステムにリクエストが届く．そしてユーザー空間においてユーザーが実装したファイルシステムがリクエストの処理すると，またカーネル空間を経由し，ユーザー空間の任意のプログラムに結果が返る．そのため FUSE を通過したアクセスは一度ユーザー空間に戻るため，ほかの通常のファイルシステムに比べるとオーバーヘッドが発生する場合がある．

FUSE は開発者に対して主に 2 種類の API を提供している．高レベル API では実装が簡

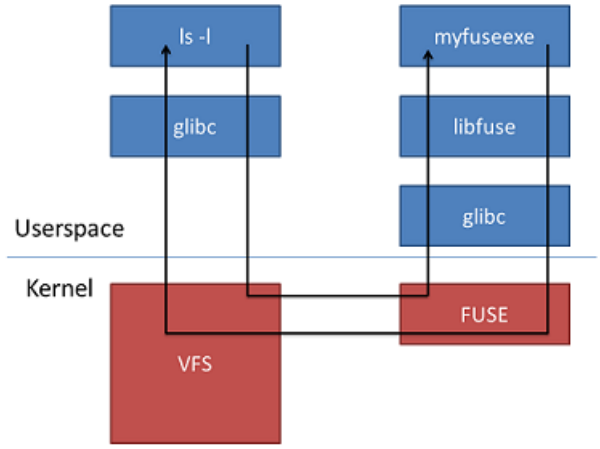

図 3 FUSE で実装したファイルシステムに対し ls コマンドを実行した際のアクセス

単だが，各コールバック関数には引数として操作するファイルのパス名が渡される．一方，低レベル API では実装が複雑だが，各コールバック関数には引数として操作するファイルの inode 番号が渡される．今回は低レベル API で実装した．

3.2 mylfs のプロトタイプ実装

第 1 章でも述べたとおり，SSD には書き換え回数の制限があり，HDD にはシークの発生による性能低下がみられる．しかしながら第 2 章で述べたように Log Structured File System ではファイルの書き換え等を行ってもストレージ上の同じ場所を書き換えることはなく，書き換えに伴うシークも 1 度のみしか発生しない．

そこで評価のために FUSE を用いて mylfs のプロトタイプを実装した．実装を簡単にするためにファイルの属性や，ディレクトリエントリの情報，InodeMap などはメモリ上に線形リストとして実装し，ストレージにはファイルのデータのみ書き込むように実装した．ストレージに二つのファイルが書き込まれるデータの様子を**図 4** に示す．

図 4 では赤いファイルと青いファイルの 2 つが連続に書き込まれている．図 4 に示されているようにストレージにはファイルのデータ以外は書き込まれていない．

そのためファイルやディレクトリの作成時にはディレクトリエントリや，ファイルの stat

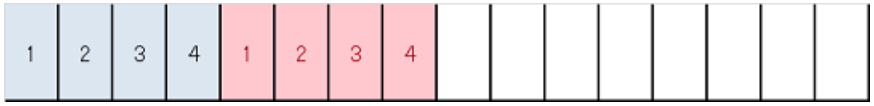

図 4 mylfs で実際にストレージに書き込まれる様子

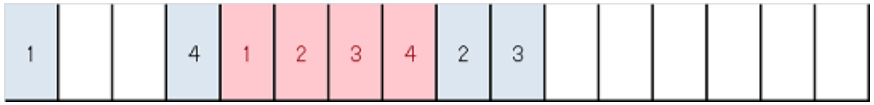

図 5 mylfs 上のファイルの一部を書き換えた様子

構造体の中身を線形リストに追加するように実装した．

また，Inode Map 上の各 Inode ごとに保持しているデータを格納する構造体を my_inode と定義した．my_inode 構造体はファイル，ディレクトリともに共通のものを使っている．ファイルやディレクトリの属性などの情報は stat 構造体で定義される．特定の inode 番号のものがディレクトリである場合は，そのディレクトリ以下のファイルの inode 番号とファイル名の組み合わせの一覧を持つ線形リストをから結果を返す．逆に特定の inode 番号のものがファイルである場合は，my_sector 構造体で定義されている線形リストにストレージ上でのアドレスが記述されているため，それを用いて結果を返す．ファイルにデータを書き込み際にはその my_sector 構造体で定義されている線形リストを操作するように実装した．ファイルのデータを読み込む際には my_sector 構造体による線形リストからファイルの要求されている場所に該当するストレージ上のアドレスを探し出し，そのアドレスにシークを行って要求されたデータを返すように実装した．ファイルの書き込まれ方によってはデータがストレージ上に散布されている場合があるが，その際は要求されたサイズのデータを読み込めるまで何度もシークを行い，要求されたデータを返すように実装した．この実装の結果，ファイルの後ろの方のデータに対して読み書きを行う際には処理に時間がかかる．それを避けるために my_inode 構造体ではストレージ上でのファイルの末尾の位置に関する情報を持たせている．また，ファイル名の変更や，stat 構造体の変更時には線形リスト内の該当する箇所を更新するように実装した．

図 4 の青いファイルの一部を書き換えた際の様子を**図 5** に示す．

図 5 で示されているように，ファイルの一部を書き換えた場合はストレージ上の同じ箇所を上書きするのではなく，ストレージ上のログの末尾に書き込むようになっている．その

CPU	Intel Core i7 3.5GHz (4 プロセッサ)
メモリ	1GiB
OS	Debian GNU/Linux (2.6.32)
HDD	Western Digital WD20EARS

表 1 HDD でのパフォーマンスを計測する環境

CPU	Intel Xeon 3.0GHz (4 プロセッサ)
メモリ	1GiB
OS	Fedora 13 (2.6.34.7)
SSD	Wildfire PW120GS25SSDR

表 2 SSD でのパフォーマンスを計測する環境

ため書き込む際のシークは 1 度ですむ．しかしながら，青いファイル全体を読み込む際には，シークが 3 回行われる．

また，ストレージ上でログの末尾となっているアドレスはメモリ上に保持し，ファイルにデータを書き込む際は常にログの末尾となっているアドレスに 1 度シークを行った後データを書き込み，書き込み後にメモリ上のログの末尾のアドレスを更新するように実装した．

4. 評　　価

4.1 評価環境

評価環境として HDD でのパフォーマンスを計測する環境と SSD でのパフォーマンスを計測する環境の二つを用意した．それぞれの環境は表**表 1** と**表 2** に示す．

4.2 比較対象

mylfs と比較する対象として ext3, NILFS2, btrfs, XFS, fuseext2 の 5 種類を定めた．

(1) ext3

third extended file system (Ext3)，Linux で主に使われているジャーナリングファイルシステムで，Ext2 の拡張として実装され，2.6.14 以降のカーネルで利用できる．Ext3 は 2 つの方針で設計されており，一つはジャーナリングファイルシステムであること，もう一つは可能な限り Ext2 の上位互換を維持することである．ジャーナリングファイルシステムであることからファイルシステムの一貫性を保証しているが，Ext3 においてその保証はシステムコール単位でのみ保証しているため大きなファイルの複製処理など複数の write() システムコールが行われる場合に何らかのエラーにより途中で複製処理が中断された場合に複製先のファイルは複製元のファイルに比べ小さくなる場合がある．

(2) NILFS2

New Implementation of a Log-structured File System (NILFS2) [4)5)] は NTT サイバースペース研究所によって開発された連続スナップショットをサポートする Linux 向けの Log Structured File System で，2.6.30 以降のカーネルで利用できる．NILFS2 はサービスを停止することなくスナップショットを自動かつ連続的に取得する機能などを提供しており，利用者の誤操作により消されてしまったファイルを簡単に復活できるなど，利用者の利便性を向上させている．また，他のファイルシステムのようにシステムクラッシュや正常でないシャットダウンが起きた場合にもファイルシステムの一貫性を保証しており，クラッシュ後でも短時間で普及可能などシステムの信頼性も向上させている．

inode やファイルの管理には B-tree をベースに実装しているほか，多数・巨大なファイルやディスクをサポートするために 64 ビットのデータ構造，２０３８年以降の対応を可能にする 64 ビットのタイムスタンプで実装されている．

(3) Btrfs

B-tree file system (Btrfs) [6)] は Linux 向けのコピーオンライトのファイルシステムで 2007 年に発表され GPL の元で公開されているが，未だ開発中で安定版はリリースされていない．Btrfs はすでに動的な inode の割り当てや，スナップショット機能，増分バックアップ，SSD 最適化モードなどを実装している．

(4) XFS

XFS はジャーナリング機能を持つファイルシステムで，2.4.21 以降のカーネルで利用できる．XFS は 64bit ファイルシステムとして実装されており，理論上 900 万 TiB のファイルを操作できる．

巨大なファイルシステムを効率的に操作するために，ブロックデバイスをアロケーショングループと呼ばれる領域に分割する．アロケーショングループでは inode と空き領域をグループ化して管理している．アロケーショングループを用いることによって空き領域の管理と inode の割り当てを平行して行えるようになった．また，各アロケーショングループは独立したエンティティとして扱われ，カーネルは複数のアロケーションに対して同時にアクセスできるため，同一ファイルに対して複数のプロセスからのアクセスを可能とした．

動的 inode の管理や空き領域の管理には B+-Tree を用いて実装されている．

(5) fuseext2

mylfs は FUSE を用いて実装しているため，FUSE によるオーバーヘッド等の影響が予測される．そこで ext2/ext3 によりフォーマットされたデバイスを FUSE によってマウントすることのできるプログラムである fuseext2[7] を使い，FUSE によるオーバーヘッドの割合を参考のため計測する．

4.3 評価方法

4.3.1 dd による評価

dd は入力から出力へデータをコピーするプログラムである．オプションを指定しなければ入力には標準入力を，出力には標準出力を使うが，if オプションや of オプションを用いることでファイルシステム上の特定のファイルから入力することや，特定のファイルに出力することができる．bs オプションを使うことで入出力のブロックサイズを指定することができ，count オプションによってコピーするブロック数を指定できるため，実際にそれらのオプションを使用した場合,「ブロックサイズ」×「ブロック数」分のデータをコピーすることができる．dd はデータのコピーが終わると，標準エラー出力にコピーしたサイズ，コピーにかかった時間，単位時間あたりのコピーしたサイズを出力する．これを用いて 2 GiB のデータを書き込み，書き込んだデータすべてを読み込む処理を１０回行うシェルスクリプトを作成した．書き込むデータの生成は/dev/zero を用い，生成がボトルネックとならないようにし，読み込んだデータは/dev/null にはき出し，読み込んだ後の処理で性能が落ちるようなことがないようにしている．このシェルスクリプトでは書き込みの後，ページキャッシュと Slab キャッシュによって読み込みの性能評価が正常に行われないようなことがないようにページキャッシュと Slab キャッシュを解放している．このスクリプトを用いて HDD，SSD ともに各ファイルシステムについて単位時間あたりの書き込み，読み込みを行ったサイズを比較することで評価する．

4.3.2 書き換え性能の評価

一般にファイルは同じファイルを何度も書き換えられる．そこで同じファイルを何度も書き換えるプログラムを作成し，その実行時間を計測した．プログラムは，引数で渡されたファイルに対し特定の場所を 4KiB ずつ書き換える処理を行う．書き込み後は毎回 fsync() を呼び出す．最後にそれぞれの処理に要した時間を出力する．今回はこのプログラムに引数で渡すファイルはあらかじめ dd コマンドで 160MiB 分のデータを生成したものを使用し，このプログラムを HDD，SSD ともに 10 回ずつ実行し平均を算出することで各ファイルシステムの性能を評価する．

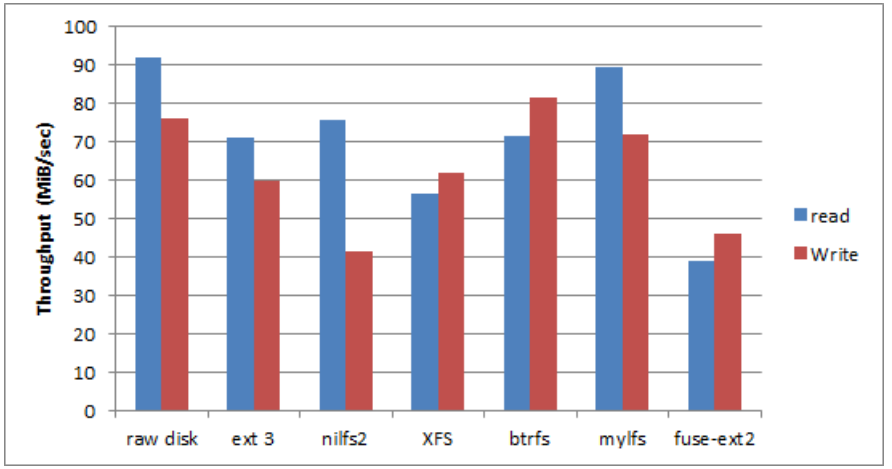

図 6 HDD 上の各ファイルシステムに対する dd を用いて評価した結果

評価は書き換える場所を下記のそれぞれの方式で選び行った．

(1) ファイルの先頭から逐次的に書き換える

(2) ファイルの中からランダムに書き換える

4.4 評価結果

4.4.1 dd による評価結果

dd によって各ファイルシステムを評価するにあたり，まずストレージデバイスに対し直接 dd を実行し評価した．ストレージの書き込み性能を評価する際には of オプションに，読み込み性能を評価する際には if オプションにそれぞれ各ストレージを指定し，各ファイルシステムを評価する際に設定したサイズで用いた．各ファイルシステムを評価する際と同様に 10 回計測し，その平均値を求めた．

dd によって各ファイルシステムを評価した結果を HDD については**図 6** に，SSD については**図 7** に示す．図 6 と図 7 にはストレージデバイスに対し直接 dd を実行した結果も併せて表示している．図は各ファイルシステムについて計測した値の平均値を示しており，各回の左側が読み込み時の転送速度を，右側が書き込み時の転送速度を示しているため，数値が高い方が高い性能を示している．

HDD での結果では Ext3 や，NILFS2 ，XFS に比べ読み込み・書き込みともに mylfs が高い性能を示しているが，Btrfs と比較すると読み込みは高い性能を示したものの書き込

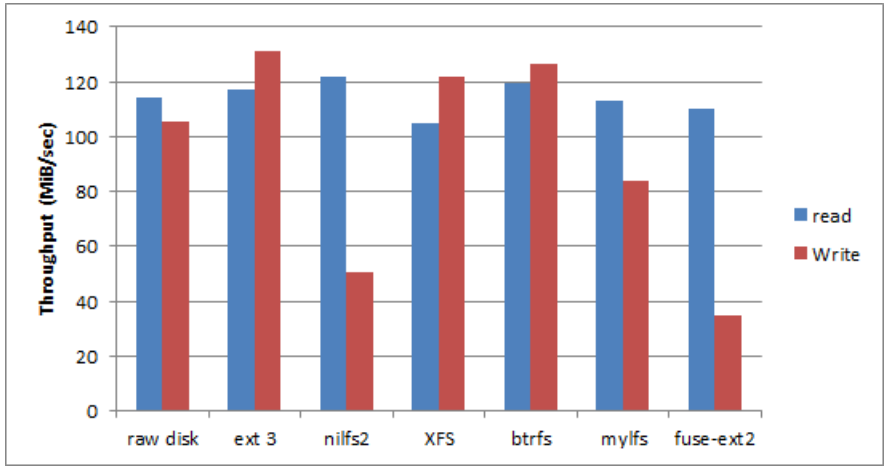

図 7 SSD 上の各ファイルシステムに対する dd を用いて評価した結果

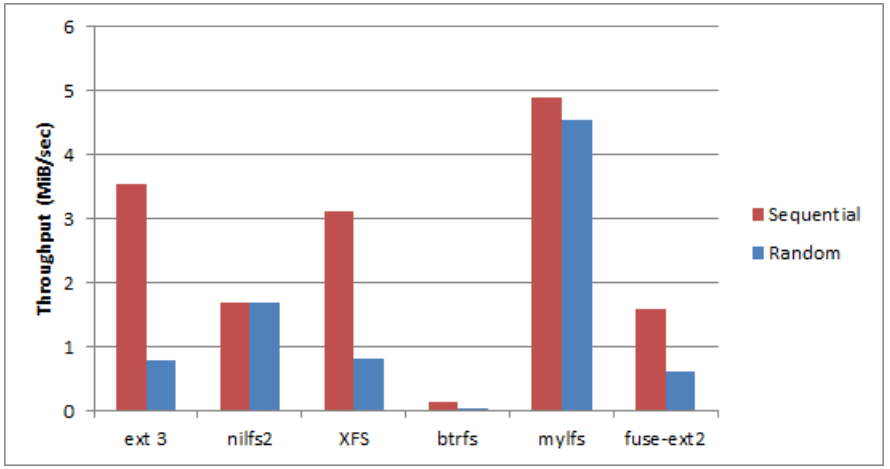

図 8 HDD 上の各ファイルシステム上に対する書き換え性能を評価した結果

みにおいては高い性能を示さなかった．また，Ext3 と fuse-ext2 を比較すると読み込みでは 4 割，書き込みにおいては 2 割以上もの性能低下がみられる．SSD においては mylfs は fuse-ext3 に比べ読み書きともに高い性能を示したが，Ext3 や XFS，Btrfs と比較すると読み書きともに高い性能を示さなかった．SSD においても Ext3 と fuse-ext2 を比較すると読み込みでは大きな性能低下は観られなかったが，書き込みでは 7 割以上もの性能低下がみられた．

dd を用いてファイルの書き込みを行う際は，ファイルの末尾にデータを追加していく形になっている．また，mylfs の書き込み時の操作は，インデックスを保持している線形リストの操作と，ストレージへの書き込みの操作の二つに分けられる．

HDD において mylfs が Ext3 に比べて高い書き込みの性能を示したのはストレージへの書き込みの操作がインデックスの操作に比べて時間がかかるためと思われる．一方，SSD において mylfs が Ext3 に比べ高い書き込みの性能を示さなかったのはストレージへの書き込みの操作が高速でインデックスの操作に比べて時間がかからなかったためと思われる．

読み込みの性能が HDD において Ext3 に比べると mylfs が高い性能を示したのはストレージへの書き込みが線形リストの操作に比べて時間がかかったためと思われる．また，SSD において Ext 3 に比べると mylfs の読み込みの性能が高い性能を示さなかったのは線形リストの操作に時間がかかったためと思われる．

しかしながら，HDD では raw disk の性能に対して書き込みが 94%，読み込みが 97%の性能を出した．また，SSD では raw disk の性能に対して書き込みが 79%，読み込みが 98%の性能を出した．

4.4.2 書き換え性能の評価結果

mybench によって逐次的に書き換えを行い性能を評価した結果を HDD については**図 8** に，SSD については**図 9** に示す．

SSD での評価において XFS,Btrfs,fuse-ext2 は評価に 3600 秒以上経過しても終了しなかったため評価を途中で打ち切った．

図は各ファイルシステムについて計測した値の平均値を示しており，各回の左側が読み込み時の転送速度を，右側が書き込み時の転送速度を示しているため，数値が高い方が高い性能を示している．赤いグラフが逐次的に書換を行った際の結果を，青いグラフがランダムに書き換えを行った際の結果を表している．

逐次的に書換を行った場合は dd による書き込みと違って write() を呼び出すごとに fsync() を行っているため，シーク回数が少ないものの性能はが低下したと考えられる．しかしながら比較した中では mylfs は他のファイルシステムと比較して HDD，SSD ともに最も高い性能を示している．

また，ランダムに書換を行った場合は逐次的に書き換える場合と違ってシーク回数が増え

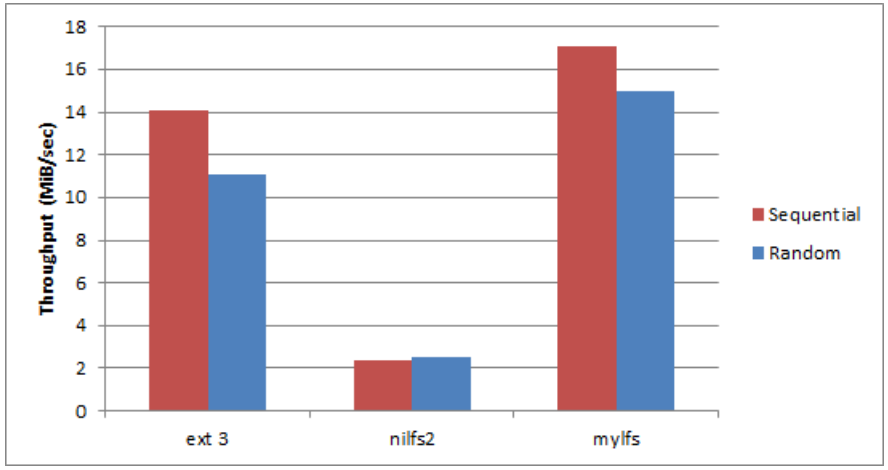

図 9 SSD 上の各ファイルシステムに対する書き換え性能を評価した結果

るため，逐次的に書き換える場合と比べると性能は低下している．しかしながら mylfs や NILFS はランダムに書き換える場合においてもシーク回数は少ないので，性能低下は比較的起きていない．一方，ext3 などはシーク時間の長い HDD において大きく性能が低下している．その結果，mylfs は HDD，SSD ともに他のファイルシステムに比べて高い性能を示した．特に HDD においては ext3 の 570%もの性能を出している．

5. 関 連 研 究

Log Structured File System として実装されているファイルシステムには 第 3 章で比較に用いた NILFS2 がある．NILFS2 は LFS の特徴のを利用して連続スナップショット機能を提供している．数秒おき，あるいは，同期書き込みを行ったタイミングで NILFS2 はチェックポイントを生成する．チェックポイントには寿命が定められており，寿命を迎えるとそのチェックポイントはガーベージコレクション機能によって回収され，チェックポイントに利用されていたストレージが利用できるようになる．ユーザーは任意のチェックポイントをスナップショットとして変更できる．スナップショットはガーベージコレクション機能によって回収されることはない．またスナップショット数に制約はなく，スナップショットはストレージがいっぱいになるまで生成できる．各スナップショットはファイルシステムがマウントされたまま，オンラインでリードオンリーのファイルシステムとしてマウントできるためオンラインバックアップに利用できるなどの特徴を備えている．

NILFS2 のほかにも，Log Structured File System として実装されているものとして JFFS[8] や YAFFS[9]，UBIFS[10] がある．これらのファイルシステムは SSD などフラッシュメモリ式ストレージデバイス向けとして実装されている．フラッシュメモリ式のストレージデバイスでは第 1 章で述べたような制約のほかに，書き込む前に大きな領域を消去する必要がある．これらのファイルシステムでは本研究と同様に Log Structured File System を使うことでウェアレベリングを行っている．

また，本稿では高速なストレージデバイスとして SSD をターゲットにしたが，ほかの高速なストレージデバイスとして Storage Class Memory があり，Storage Class Memory 向けのファイルシステムとしては SCMFS[11] がある．

最後に，Log Structured File System ではないが，現在も開発が続けられている Btrfs には SSD 向けに最適化するオプションが存在する．このオプションを有効にした場合，Btrfs は書き込みの要求があった際に，ストレージ上のなかで未だ書き込みが行われていない箇所を探す．そして書き込みをその中から要求に適した箇所にデータを書き込む．このように書き込みを分散させることで本研究とは別の方法でウェアレベリングを行い SSD の寿命低下を防止している．

6. お わ り に

本研究では，HDD，SSD などにおいて効率的なファイルシステムの実現に向けて，ストレージへの書込が逐次書込となるよう設計を行った mylfs のプロトタイプ実装を行い，様々なアクセスパターンによる評価を SSD，HDD ともに行った．

dd によるシーケンシャルリード，シーケンシャルライトの性能評価では，raw device と比較した場合において，HDD では性能に対して書き込みが 94%，読み込みが 97%の性能を出した．また，SSD では書き込みが 79%，読み込みが 98%の性能を出した．他のファイルシステムと比較すると HDD，SSD ともに fuse-ext2 に比べると高い性能を示したものの，他のファイルシステムに対しては必ずといって高い性能を示すとは限らなかった．しかしながら，fuse-ext2 と Ext3 の性能を比較すると fuse-ext2 は Ext3 に比べて 2 〜 7 割の性能低下が確認されたことから，FUSE で実装したプログラムがユーザー空間で動作することに起因すると考えられる．

次にファイルを書き換え性能の評価は，逐次的，ランダムともに mylfs が他のファイルシステムに比べて高い性能を示した．特にランダムに書き換える評価では広く使われている

ext3 と比較すると HDD では 570%，SSD では 135%の性能を示した．

今後の課題として，以下のことが挙げられる．

(1) FUSE のオプションの差による入出力性能の測定
今回は FUSE を用いて Log Structured File System を実装し，実行するさいには特にオプションを指定せずに実行した．そのため，実際の書き込みは 4KiB ごとに行われた．しかしながら，FUSE には読み書きに行うサイズなどのオプションが存在する．メタデータの更新はデータの書き込みごとに行われるため，それらのオプションを変えた場合における入出力性能の変化を測定する必要があると考えられる．

(2) メタ情報などのストレージへの書き込み
今回実装した Log Structured File System はディレクトリエントリや，メタ情報などをメモリ上の線形リストに保持しているため，ファイルシステムをアンマウントするとストレージ上のデータにアクセスできなくなってしまう．また，メタ情報等を書き込んでいないため，ストレージに実際に書き込まれるデータのサイズが小さくなっているので性能評価でより高速になっていると思われる．そのため，メタ情報などや，ログの末端のアドレスなどをストレージに書き込む必要がある．

(3) 空き領域マネジメントの実装
今回実装した Log Structured File System では，実装を簡単にするためデータを書き換えた際に発生する空き領域の管理を行っていない．しかしながら，現在多く利用されている SSD はストレージの容量が HDD に比べると著しく少ないため，実際に運用するとストレージをすぐに使い果たすと考えられる．そのため空き領域を管理する機能が必要になると考えられる．

(4) 分散ファイルシステムへの応用
今回は単独のストレージに対して Log Structured File System を実装したが，SSD など高速なストレージは分散ファイルシステムのデータノードのストレージとしても利用されており，Gfarm[12] など分散ファイルシステムに対して応用ができないか検討し，応用していきたいと考える．

(5) 不揮発性メモリにおけるファイルシステムの設計
今回は高速なストレージとしてフラッシュメモリを使用している SSD をターゲットに評価を行った．今後は SSD 以外の高速なストレージとして不揮発性メモリについてもターゲットにしていきたい．

謝辞 本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」および文科省次世代 IT 基盤構築のための研究開発「研究コミュニティ形成のための資源連携技術に関する研究」(データ共有技術に関する研究) による．

参 考 文 献

1) Changwoo Mina, Kangnyeon Kimb, Hyunjin Choc, Sang-Won Leed, Young Ik Eome.; SFS: Random Write Considered Harmful in Solid State Drives, Proceedings of the 10th USENIX Conference on File and Storage Technologies, pages 1-16, 2012.
2) Mendel Rosenblum and John K. Ousterhout; The Design and Implementation of a Log-Structured File System, Proceedings of the 13th Symposium on Operating System Principles, pages 1-15, October 1991.
3) FUSE, http://fuse.sourceforge.net/
4) NILFS, http://www.nilfs.org/ja/
5) 佐藤ほか, ログ構造化ファイルシステム NILFS の設計と実装, 情報処理学会 論文誌コンピューティングシステム (ACS), Vol.2, No.1, pp.110-122, 2009.
6) Btrfs, https://btrfs.wiki.kernel.org/
7) fuse-ext2, http://sourceforge.net/projects/fuse-ext2/
8) D. Woodhouse. Jffs: The journalling flash file system. In The Ottawa Linux Symposium, RedHat Inc, 2001.
9) Yaffs. Available on July 2011 from http://www.yaffs.net/.
10) The design of UBIFS, http://osl.sed.hu/~havasi/ubifs/
11) Xiaojian Wu, Narasimha Reddy ; SCMFS : A File System for Storage Class Memory, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pages 39:1–39:11, 2011.
12) Osamu Tatebe, Kohei Hiraga, Noriyuki Soda, "Gfarm Grid File System", New Generation Computing, Ohmsha, Ltd. and Springer, Vol. 28, No. 3, pp.257-275, 2010.