# ディレクトリ毎にジャーナリングモードを設定可能なファイルシステム

青山 航　岩崎 英哉

ジャーナリングとは，ファイルの作成・更新などファイルシステムを変更する際に，ログにその変更を記録することで，ファイルシステムの信頼性を向上させる手法である．ジャーナリングの信頼性とファイルシステムの速度はトレードオフの関係にある．そのため，既存のシステムは，ジャーナリングの方法 (モード) を，信頼性を重視したもの，速度を重視したものなど，何種類か提供している．しかし，ジャーナリングモードはファイルシステムに一つしか設定できないため，ファイルシステム内のファイル毎に異なる特性に合わせることができない．この問題を解決するため，本研究は，ジャーナリングモードをディレクトリ毎に設定できるファイルシステムを提案し，Linux の ext3 ファイルシステムを拡張して実現する．このことにより，ジャーナリングモードの設定単位を従来より細かくすることができ，ファイルの特性に合ったモードを設定することが可能となる．

## 1 はじめに

ファイルシステムとは，ディスク上のデータを管理するオペレーティング・システム (OS) の一機構である．一般的なファイルシステムは，ディスクを一定長のブロックに分割し，実際のデータが格納されているブロック (実データブロック) と，i ノードやブロックの空き状況を示すビットマップが格納されているブロック (メタデータブロック) の両方を用いて管理する．

ユーザプログラムがファイルに書き込みを行うと，ファイルシステムは対応する複数のブロックに書き込みを行う．このとき，OS の異常終了などにより実データやメタデータの一部が失われてしまうと，ファイルシステムの管理状態が中間的な不完全状態に陥り (この状態を，整合性が損なわれていると呼ぶ)，ファイルシステムが破損してしまう可能性がある．ファイルシステムの整合性を保証するために，ファイルシステムの整合性検査 (fsck) [3] やジャーナリング [1]，soft-updates [4]，コピーオンライト [7] などの技術が提案されている．中でも，ジャーナリングは ext3 ファイルシステム (以下，ext3 と呼ぶ)，JFS，XFS，ReiserFS，NTFS など様々なファイルシステムで用いられている．

ジャーナリングとは，ファイルの作成・更新などファイルシステムに対して変更を施す前に，変更履歴を記録する手法である．図 1 に，ジャーナリングを用いてファイルへ書き込みを行う様子を示す．ここで，$D$ は実データ，$M$ はメタデータ，$T_B$ はトランザクションの開始，$T_E$ はトランザクションの終了を表す．ジャーナリングでは，ジャーナルという特殊なログファイルにファイルの変更内容を一旦記録し (図 1 (b))，記録の終了を確認した後に，保存領域の対象のファイルに書き込みを行い (図 1 (c))，ジャーナルに書き込みが終了したことを記録する (図 1 (d))．OS が異常終了した場合，ジャーナルへの書き込みの状態を調べることによって，処理がどこまで進行したかがわかる．書き込みが完了していなければ，ファイルシステム変更が行われる前の状態に保たれ，ジャーナルへの書き込みが完了していれば，更新後の状態にし，ファイルシステムの整合性が損なわれないようにする．

Wataru Aoyama, Hideya Iwasaki, 電気通信大学大学院情報理工学研究科, Dept. of Communication Engineering and Informatics, University of Electro-Communications.

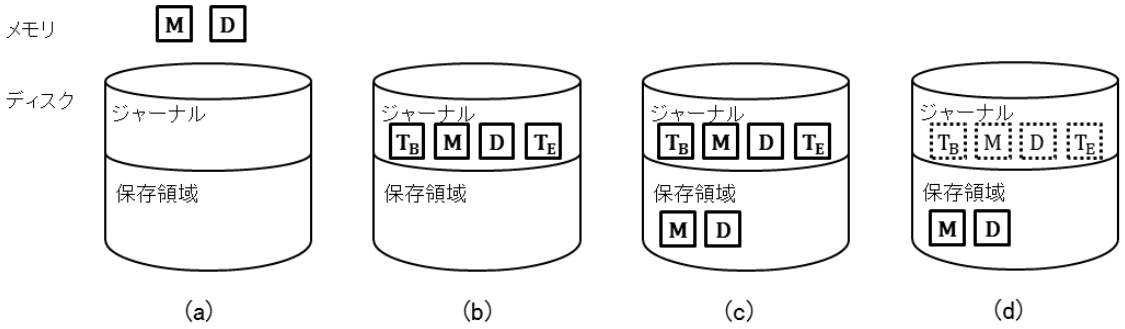


**図 1　ジャーナリングを用いたファイル書き込み．(a) 初期状態，(b) コミット：トランザクションをジャーナルに書き込み，(c) チェックポイント：データを保存領域に書き込み，(d) クリーンアップ：トランザクションが終了したことを記録．**

ext3 では，変更履歴に保存する対象や書き込みの順番を，ジャーナリングモードにより設定することができる．ジャーナリングモードとしては，信頼性を重視したもの，速度を重視したものなど，何種類か提供しており，重要なファイルには信頼性の高いモード，重要度があまり高くないファイルには速度が速いモードというように，ファイルの特性 (重要度) に合ったモードの設定することが望まれる [5] [6]．しかし，既存の Linux では，ファイルシステムに一つのジャーナリングモードしか設定できないため，ファイルシステム内のファイル毎に異なる特性に合わせることができない．

このような問題点を解決するため，本研究では，ジャーナリングモードをディレクトリ毎に設定することが可能なファイルシステムを提案する．このことにより，ジャーナリングモードの設定単位を従来より細かくすることができ，ファイルの特性に合ったモードを設定することが可能となる．

本稿の構成は次の通りである．2 節で，ext3 とジャーナリングの処理について述べる．3 節では，既存研究の特徴と問題点を述べる．4 節では，本研究で提案するシステムの概要と動作の流れについて述べ，5 節ではシステムの実装について述べる．6 節で，実験を行い，最後に 7 節で本稿をまとめ，今後の課題を述べる．

## 2　ext3 ファイルシステム

Linux の標準的なファイルシステムである ext2 ファイルシステム (以下，ext2 と呼ぶ) には，OS が異常終了後の復旧に時間がかかるという問題があった．この問題を解決するために，ext2 と互換性を保ちながらジャーナリング機能を導入した ext3 が開発された．

### 2.1　ディスクキャッシュ

Linux では，ディスク I/O を減らすために，ディスク上のデータをメモリにキャッシュする．これを，ディスクキャッシュと呼ぶ．ディスクキャッシュには，ファイルの実データをページ単位に分割したページキャッシュや，ファイルの i ノードをキャッシュした i ノードキャッシュなどが存在する．たとえば，ディスクへの読み込みが発生すると，カーネルはファイルの i ノードから i ノードキャッシュ，実データをページ単位に分割してページキャッシュを生成する．その後，同じファイルをアクセスすると，i ノードキャッシュとページキャッシュを用いてファイルを高速に読み込む．

Linux では，`O_SYNC` や `O_DIRECT` フラグを明示的に指定しない限り非同期 I/O となる．図 2 に，ファイルへの非同期書き込みを行う様子を示す．ここで，*D* は実データ，*M* はメタデータ，Dirty フラグが立っているデータを灰色で表す．非同期書き込みでは，ディスクキャッシュへの書き込みが完了すると，ユーザに書き込みが終了したことを通知する (図 2 (a))．この

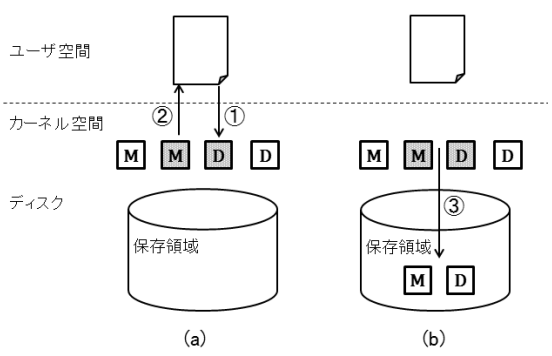


**図 2　非同期書き込み．(a) write システムコールによる処理 (b) カーネルスレッドによる処理**

時，作成したページキャッシュおよび i ノードキャッシュに変更が加えられたことを表す Dirty フラグを立てる．その後，カーネルスレッド pdflush が，Dirty フラグを立っているデータをディスクの保存領域に書き込む．(図 2 (b))

### 2.2　ジャーナリング

ジャーナリングでは，不可分なファイル操作処理を 1 つのトランザクションとし，以下の 3 つの操作によりファイルシステムの整合性を保証する．

**コミット**
トランザクション内のファイルシステムへの操作処理をジャーナルに書き込む処理．

**チェックポイント**
トランザクション内のファイルシステムへの操作処理を保存領域に書き込む処理．

**リカバリ**
コミットが完了した処理を，再度チェックポイントすることでファイルシステムを復元する処理．

ext3 では，ジャーナルをファイルシステム内の隠しファイル，またはファイルシステム外の別デバイスとして扱う．また，保存する変更履歴の種類をジャーナリングモードによって設定することができ，以下の 3 つのモードを提供している．

1. **data モード** メタデータと実データの両方をジャーナリングの対象とする．このモードでは，変更履歴を全て保存するため信頼度は最も高いが，同じデータをディスクに二回書き込む必要があるため，オーバーヘッドは大きい．
2. **ordered モード** メタデータのみをジャーナリングの対象とする．ただし，実データがディスクに書き込まれた後にメタデータの書き込みを行う．このモードでは，実データより前にメタデータがディスクに書き込まれることがないので，メタデータが不正なデータを指し示すことはない．
3. **writeback モード** メタデータのみをジャーナリングの対象とする．このモードでは，メタデータが不正なデータを指す可能性があり信頼性は低いが，メタデータの書き込み順番に制約はないので，ディスク I/O 待ち時間が少なく，速度が最も速い．

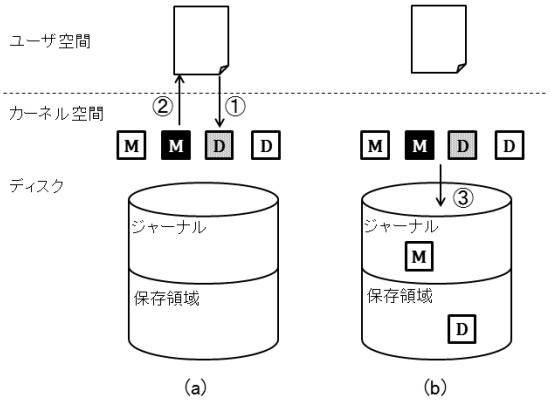


**図 3　非同期書き込みのジャーナリング．(a) write システムコールによる処理 (b) カーネルスレッドによる処理**

非同期書き込みのジャーナリングの処理では，ジャーナリングを行うデータには JBDDirty フラグを立てることで，ジャーナリングを行うデータと行わないデータを区別する． 図 3 に，ordered モードでジャーナリングの処理を行うときの非同期書き込みの様子を示す．ここで，*D* は実データ，*M* はメタデータ，Dirty フラグが立っているデータを灰色，JBDDirty フラグが立っているデータを黒色で表す．この時，ジャーナリング対象のメタデータに JBDDirty，対象ではない実データに Dirty フラグを立て，ユーザに通知する．(図 3 (a)) その後，カーネルスレッド pdflush は

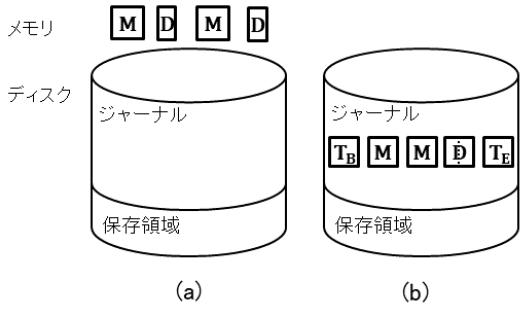


**図 4　Okeanos ファイルシステム**

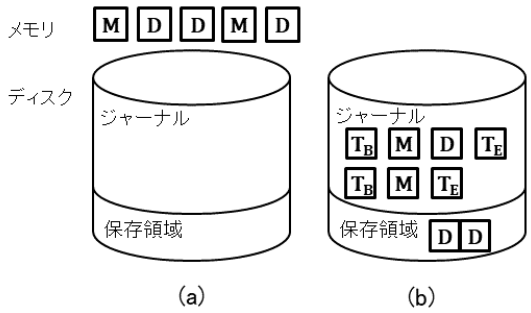


**図 5　Selective Journaling**

Dirty フラグの立っているデータをディスクに書き込み，kjournald は，JBDDirty フラグの立っているデータをコミット，チェックポイントを行う．(図 3 (b))

## 3　関連研究

これまでに，ジャーナリングモードの設定単位を細かくする研究はいくつか行われている．Okeanos [2] は，同期書き込み時に複数のページキャッシュをまとめてコミットする Wasteless Journaling モードと，データサイズが閾値以下の書き込みには Wasteless Journaling を適用する Selective Journaling モードを持つファイルシステムである．Selective Jouranling モードでマウントされたファイルシステムでは，図 4 のようにコミット前にページキャッシュを一つのデータに結合する．このことで，書き込むブロック数を減らし，シーケンシャルアクセスが可能になるため，ディスク I/O 待ち時間を減らすことができる．

Adaptive Journaling [5] は，図 5 のように I/O パターンからジャーナリングモードを自動的に設定する機構である．シーケンシャルアクセスの場合には ordered モードを適用することで，ディスクの無駄なシーク時間を減らすことができる．

Okeanos はデータサイズに応じてジャーナリングモードを自動的に設定する．また，Adaptive Journaling はトランザクション単位でジャーナリングモードを自動的に設定する．このため，どちらに関してもユーザ自身の判断に基づいて，ファイルの重要度に応じた設定をすることができない．

File-Adaptive Journaling [8] は，ファイル単位でジャーナリングモードを設定できる機構である．ユーザは，予めファイルにモードを設定することで，当該ファイルに対する操作は設定したジャーナリングモードに基づいて処理されるようになる．このことで，ファイルシステム内に複数のモードを持つことが可能としている．しかし，File-Adaptive Journaling では，ファイル一つ一つにジャーナリングモードを設定する必要があり，設定が煩雑である．例えば，新規ファイルにジャーナリングモードを設定する場合，ファイルの書き込み前にユーザがジャーナリングモードを設定しなければならない．それに対して本研究では，ディレクトリ単位でジャーナリングモードを設定するため，設定の煩雑さが軽減されている．

## 4　設計

### 4.1　システムの概要

本システムは，ディレクトリ単位でジャーナリングモードを設定可能なファイルシステムである．同一ディレクトリ直下の全てのファイルに，そのディレクトリに設定されたジャーナリングモードが適用される．子ディレクトリには，親ディレクトリのジャーナリングモードとは独立に，ジャーナリングモードを設定できる．設定できるジャーナリングモードは，ext3 と同じく，data，ordered，writeback のいずれかとする．ここで，ジャーナリングモードを設定していないディレクトリは，無モードとして扱い，ジャーナリングの処理を行わずにファイル操作をする．

同じディレクトリ内のファイルは共通する性質を持ち重要度が共通することが多いことから，モード設定

の細かさの単位をディレクトリとする本システムの設計は適切であると考えられる．

本ファイルシステムは，ext3 と同様にブロックデバイスに mkfs でファイルシステムを生成，mount でファイルシステムをマウントして利用する．mkfs によるファイルシステムの生成時にルートディレクトリのジャーナリングモードを設定することができる．また，ユーザは，ファイルシステムのマウント後に，本システムが提供するインタフェースを用いてディレクトリのジャーナリングモードを設定・変更することができる．

本ファイルシステムは，ext3 と互換性があるため ext3 と ext2 として使用していたパーティションを初期化せずにマウントすることが可能である．これらのファイルシステムは，ディレクトリにジャーナリングモードを持たないため，全てのディレクトリのジャーナリングモードは無モードで開始する．

### 4.2 システムの動作例

本ファイルシステムでは，ファイルに対する操作は親ディレクトリのジャーナリングモードに従い，ディレクトリに対する操作は自身のジャーナリングモードに従い，ファイル操作処理を行う．しかし，Linux ではファイルやディレクトリにリンクを貼ることでファイルに別名をつけ，異なる名前で同一ファイルにアクセスできため，ファイルと親ディレクトリの関係が一対一になるとは限らない．そこで，複数の親ディレクトリが参照される可能性がある以下の 3 つの状況については，特別な処理をする．

- ファイルの移動による操作は，移動先のディレクトリのジャーナリングモードに従うことにする．ただし，移動元と移動先のディレクトリからファイルエントリを削除，追加する操作はそのディレクトリのジャーナリングモードに従う．
- ハードリンクに対する操作は，最後に貼られたリンク先ファイルの親ディレクトリのジャーナリングモードに従う．
- シンボリックリンクに対する操作は，リンク元ファイルの親ディレクトリのジャーナリングモードに従う．

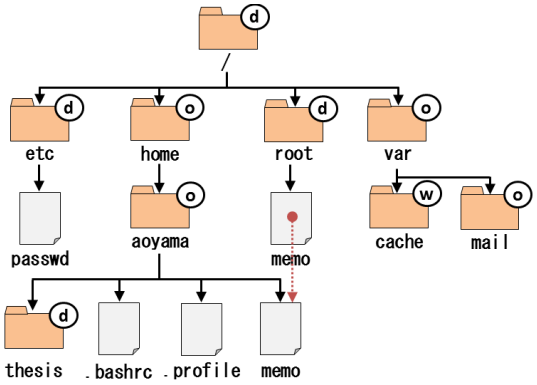


**図 6　ディレクトリ構造例**

図 6 を用いて，本システムの動作の流れを説明する．ディレクトリ右上の丸は，d は data，o は ordered，w は writeback モードであることを表し，赤い点線は矢印先のファイルのシンボリックリンクであることを表す．また，カレントユーザは `aoyama` とする．ここで，`/var/mail/`のハードリンクとして`~/mail/`を作成する．このとき，前述で取り上げた 5 つのパターンについて考える．

1. **ファイルの変更** ユーザが `~/.bashrc` を編集する場合，親ディレクトリの `~/`を参照する．従って，ordered モードが適用される．
2. **ディレクトリの変更** ユーザが `~/thesis/`にファイルを新規作成する場合，自身のジャーナリングモードの data モードでディレクトリのエントリを追加する．
3. **ファイルの移動** ユーザが `~/.profile` を `/root/`に移動する場合，移動先の`/root/`を参照する．従って，data モードで`.profile` を更新する．また，`~/`から`.profile` エントリの削除には ordered モード，`/root/`から`.profile` エントリの追加には data モードで実行される．
4. **シンボリックリンクに対する操作** ユーザが `/root/memo` を編集する場合，リンク元の`~/memo` を参照する．従って，`~/`の ordered モードでデータを編集する．
5. **ハードリンクに対する操作** ユーザが `~/mail/`以下のファイルを削除する場合，最後にリンクが

貼られた~/mail/を参照する．従って，ordered モードでディレクトリのエントリを削除する．

## 5 実装

本システムは，Linux Kernel 4.2 の ext3 とファイルシステムをメンテナンスするユーティリティ e2fsprogs[†1] に拡張を施して実現する．

### 5.1 ジャーナリングモードのデータ構造

ディレクトリにジャーナリングモードを格納するため，ディスク上に i ノードが格納されるブロック (i ノードブロック) とメモリ上にキャッシュされる i ノードキャッシュにジャーナリングモードの情報を追加する．しかし，ディスク上のブロックは固定長であるため，i ノードブロックに新しく変数を追加すると，既存のデータ構造を破壊してしまう．そこで，ジャーナリングモードをファイルのフラグとして，i ノードブロックに保持する．ext3 では，i_flags の 32 ビットでファイルのフラグを管理しているが，12 ビットの使用されていない部分がある．本システムでは，data モードを表す DATA_MODE_FL，ordered モードを表す ORDERED_MODE_FL，writeback モードを表す WRITEBACK_MODE_FL，ジャーナリングを行わない JOURNAL_NONE_MODE_FL の 4 つのフラグを追加し，i_flags のあいている 2 ビットに格納する．

### 5.2 ジャーナリングモードの設定・取得

ディレクトリの新規作成時には，親ディレクトリのジャーナリングモードを引き継ぐ必要があるので，mkdir の処理を変更する．ext3 のディレクトリを生成する ext3_mkdir() では，新規ディレクトリの i ノードを生成する処理の時に，引数のオプションから i_flags を適切に設定する．そこで，i ノードを生成する時に，親ディレクトリの i_flags からジャーナリングモードの情報のみを取得して，作成するディレクトリの i_flags に追加する．

また，ユーザがディレクトリに付与されたモードを更新・参照するインタフェースとして，ioctl システムコールを利用する．本システムでは，ioctl システムコールに，ジャーナリングモードの設定・取得するためのリクエスト (IOC_SETJOURNAL・IOC_GETJOURNAL) を追加した．

IOC_SETJOURNAL では，以下の処理を行う．

1. ユーザ空間からモード情報をコピーする．
2. 安全のために，ジャーナルを全てフラッシュする．
3. 現状のジャーナリングモードに従って，i_flags にモード情報を設定する．

IOC_GETJOURNAL では，以下の処理を行う．

1. i_flags からモードを取得する．
2. ユーザ空間にモードの情報をコピーする．

ここで，ファイル操作時のジャーナリングモードを参照する時間を減らすために，ファイルの i ノードにもジャーナリングモードを自動的に格納する．ファイルの i ノードにジャーナリングモードをキャッシュとして格納するタイミングは以下の通りである．

- ファイルを新規作成した (i ノードにリンクが追加された) 時
- ファイルを移動した (親ディレクトリが変更された) 時
- 親ディレクトリのジャーナリングモードが変更された時

この時，ジャーナルにモードを設定してしまうとシステムが正常に動作しなくなるため，対象のファイルにジャーナルであることを表す EXT3_JOURNAL_DATA_FL フラグが立っていないことをか確認してから行う．

また，ファイルシステムの生成時にルートディレクトリのジャーナリングモードを設定可能にするため，e2fsprogs の e2mkfs プログラムを拡張する．e2mkfs は，ext2，ext3，ext4 専用の mkfs であり，ファイルシステムの初期化やルートディレクトリの生成を行う．そこで，ルートディレクトリの生成が確認できたら，ioctl システムコールを用いてルートディレクトリのモードを設定する．

### 5.3 設定したジャーナリングモードでの処理

ext3 では，ファイルシステムのマウント時にファイル操作処理を決定してしまう．本ファイルシステムでは，設定したモードに従ってファイル操作処理をす

†1 http://e2fsprogs.sourceforge.net/

るために，以下のファイルシステムに書き込むための関数を拡張する．

1. **書き込み前に準備を行う `write_begin` 関数**
   ext3 では，`write_begin` 関数でページキャッシュの生成と i ノードに JBDDirty フラグを立てる．ext3 は，ジャーナリングモード無モードが存在しないため問題はないが，本ファイルシステムは，ジャーナリングモード無モードが存在するので，JBDDirty を立ててはいけない場合もある．そこで，本ファイルシステムでは，書き込み対象のジャーナリングモードから，ジャーナリングモード無モードであれば i ノードに Dirty フラグを，それ以外のモードであれば JBDDirty フラグを立てるように処理を拡張する．
2. **書き込み処理を行う `write_end` 関数**
   ext3 では，data モード用，ordered モード用，writeback モード用の `write_end` 関数が定義されている．data モードと ordered モードの `write_end` 関数では，ページキャッシュに JBDDirty フラグを立て，writeback モードでは，ページキャッシュに Dirty フラグを立てる．本ファイルシステムでは，ジャーナリングの処理はファイル書き込み時まで決定することができない．そこで，`write_end` 関数のラッパ関数を用意し，書き込み対象のジャーナリングモードから，各モード用の `write_end` 関数を呼び出すように処理を拡張する．
3. **ディスクに書き込みを行う `writepages` 関数**
   `write_end` 関数と同様に `writepages` 関数では，data モード用，ordered モード用，writeback モード用の関数が定義されている．各モード用の関数では，Dirty フラグの立ったディスクキャッシュをディスクに書き戻す，JBDDirty フラグの立ったディスクキャッシュを適切な順番でディスクに書き戻す．そこで，`write_end` 関数と同様に，`writepages` 関数のラッパ関数を用意し，書き込み対処のジャーナリングモードから，各モード用の関数を呼び出すように処理を拡張する．

また，e2fsprogs の e2fsck を拡張することで，ジャーナルに記録されたデータから，それぞれのジャーナリングモードに応じたファイルシステムの復旧を可能にする．

## 6 実験

拡張元である ext3 と比較することで提案手法のオーバーヘッドを計測した．実験に使用したシステムの構成を表 1 に示す．

実験には，ファイルシステムの性能を計測するマイクロベンチマーク IOzone[†2] を使用した．IOzone を用いて，書き込み，再書き込み，ランダム書き込みの 3 つのパターンについて計測する．ベンチマークの設定として，レコードサイズを 64KB，ファイルサイズを 1GB，`O_RSYNC,O_SYNC` を有効，測定には fsync によるフラッシュが含まれる．

図 7 に，IOzone の実行結果を示す．左から writeback モード，ordered モード，data モードにおける実行結果を示している．各グラフは，左から書き込み，再書き込み，ランダム書き込みのパターンを示している．ここで，ext3 のジャーナリングモードが，writeback モードを ext3-w，ordered モードを ext3-o，data モードを ext3-d とする．実験で実装したファイルシステム myext3 も同様に，myext3-w，myext3-o，myext3-d とする．

実行結果より，オーバーヘッドの増加量は 5%以内に収まっていることが分かった．IOzone によるファイルの生成パターンによっては，実行結果が 0.01 ほど変化することがあったので，writeback モードにおけるオーバーヘッドは誤差によるものだと考えられる．

以上のことから，同期書き込みにおいて本システムは，既存システムとほぼ同じ性能で，問題なく使用できることが示された．

**表 1　実験評価に使用したシステムの構成**

| | |
|---|---|
| CPU | Intel Core i5 M520 2.40GHz |
| メモリ | 2GB |
| HDD | 100GB Serial ATA/2.5 5400rpm |
| Linux | Ubuntu16.04 x86_64 |
| カーネル | Kernel 4.2 |

†2 `http://www.iozone.org/`

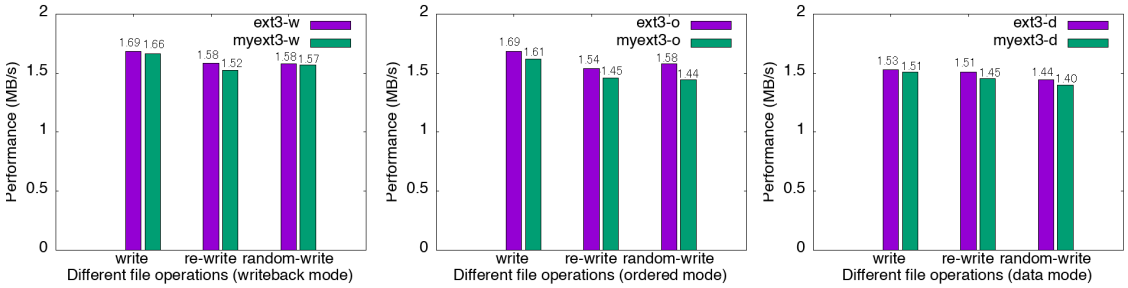


**図 7　IOzone による性能評価 (MB/s)**

## 7　おわりに

本稿は，既存ファイルシステムにおけるジャーナリングモードの設定粒度が荒く，個々のファイルの重要度に対応できないという問題を解決するために，ディレクトリ毎に設定できるファイルシステムを提案した．ジャーナリングモードをディレクトリ単位で設定することで，既存研究のような設定の煩雑さがなく，ファイルの重要度に対応した設定が可能である．

現状は，提案したシステムの実装を行っている途中であり，ディレクトリに writeback モード，ordered モード，data モードを設定してジャーナルへコミットとチェックポイントできるという段階である．今後は，ジャーナリングモード無モードの実装や，シンボリックリンクの対応，異常終了後のリカバリ処理の実装を完成させる．実装が終わり次第，システムの評価を行い，提案手法の有用性を確認し，ext3 の後続のファイルシステムでもある ext4 への実装を考えている．

## 参 考 文 献

[1] Hagmann, R.: Reimplementing the Cedar File System Using Logging and Group Commit, *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, SOSP '87, New York, NY, USA, 1987, pp. 155–162.

[2] Hatzieleftheriou, A. and Anastasiadis, S. V.: Okeanos: Wasteless Journaling for Fast and Reliable Multistream Storage, *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, Berkeley, CA, USA, 2011, pp. 19–19.

[3] Kowalski, T. J.: UNIX Vol. II, 1990, chapter Fsck - the UNIX File System Check Program, pp. 581–592.

[4] McKusick, M. K. and Ganger, G. R.: Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem, *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, Berkeley, CA, USA, 1999, pp. 24–24.

[5] Prabhakaran, V., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H.: Analysis and Evolution of Journaling File Systems, *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, Berkeley, CA, USA, 2005, pp. 8–8.

[6] Rocha, P. E. and Bona, L. C. E.: Analyzing the Performance of an Externally Journaled Filesystem, *Proceedings of the 2012 Brazilian Symposium on Computing System Engineering*, SBESC '12, Washington, DC, USA, 2012, pp. 93–98.

[7] Rosenblum, M. and Ousterhout, J. K.: The Design and Implementation of a Log-structured File System, *ACM Trans. Comput. Syst.*, Vol. 10, No. 1(1992), pp. 26–52.

[8] Shen, K., Park, S., and Zhu, M.: Journaling of Journal is (Almost) Free, *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, FAST'14, Berkeley, CA, USA, 2014, pp. 287–293.