2004－OS－96　(6)
2004／6／17

Linux ファイルシステムの信頼性についての一考察

小西 隆介†　天海 良治†　佐藤 孝治†　真鍋 義文†　盛合 敏†

要旨

オープンソースの OS として利用が急速に拡大している Linux は、ジャーナリング機能の導入に伴いファイルシステムの信頼性が大幅に向上したと言われている．しかし，信頼性を担保するには，書き込み処理の同期がきちんと行われていること，ファイルシステムが規定する I/O 順序の制約が守られていることが必要である．本稿では Linux ファイルシステムにおけるこれら信頼性上の観点と検証方法について述べる．今回，我々は同期 I/O 時のディスクへの書き出しについて，システムコールと DMA のイベントをトレースする方法を適用し，最新のバージョンで問題がないことを確認した．

A Study on the Linux File-System Reliability

Ryusuke KONISHI†　Yoshiji AMAGAI†　Koji SATO†　Yoshifumi MANABE†
Satoshi MORIAI†

Abstract

In these years, adoption of Linux, which is an open source OS, is advancing rapidly. It is said that the reliability of Linux file-system was improved significantly by the introduction of journaling technologies. However, in order to assure the reliability, it is required that write synchronization is performed exactly and to keep the I/O order restrictions which the file-system specifies. This paper discusses the viewpoint and the verification method for these reliability issues in the Linux file-system. We applied a method of tracing the event of system calls and DMA on the write operation to disk in synchronous I/O, and confirmed that the write synchronization was performed properly in the newest version.

1. はじめに

近年，オープンソースの OS である Linux[1]の利用が，商用システムや企業内システム，あるいは官公庁系システムへと拡大している．このような利用拡大に伴い，同 OS の信頼性に対する要求は，今まで以上に高まってきている．Linux の信頼性に関しては，ファイルシステムの信頼性の低さがとりわけ問題とされてきた．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)は，(1)障害時にデータを喪失しやすい，(2)データの整合性が損なわれやすい，(3)障害発生後の再起動時の復旧(fsck の実行)に時間がかかる，(4)復旧が途中でアボートすることがある，など問題が多かった．

これに対して，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 ファイルシステム(以下 Ext3)[1]や，XFS や JFS など商用システムで実績のあるファイルシステムが相次いでサポートされることなどにより，Linux ファイルシステムを取り巻く状況はここ最近大きく変わってきている．

† 日本電信電話株式会社 サイバースペース研究所
NTT Corp. Cyber Space Laboratories

[1] Linux は Linus Torvalds の商標または登録商標です．

ここでジャーナリング機能とは，ファイルシステムに対する更新を，ジャーナルという特殊なログファイル(ジャーナルファイル)に一旦記録し，それが済んでからディスク上の本来の領域に更新を行い，最後にジャーナルファイルに書き終わったことを記録するという手順を取る方式である．障害が発生した場合，ジャーナルファイルが参照され，ファイルシステムに対する更新が行われる前の状態に保たれるか，もしくはジャーナル上の更新が完了しているものについては，更新後の状態に復元する作業(Roll-forwarding)が行われ，中間状態に陥らないようにできる．厳密には上記手法をファイルシステムの管理情報であるメタデータだけに行うのか，それともデータの実体に対しても適用するのかという方式上の差異があるが，少なくともメタデータに適用することで，ファイルシステムの管理状態が中間状態，すなわち整合性の損なわれた状態に陥ることを回避でき，障害復旧の時間を短縮できる．

文献[2]では，実際にマシンの電源切断を行って，ディスク上のデータの喪失や整合性の破壊を検査した結果が報告されており，ジャーナリング機能持つファイルシステムの信頼性の優位性が示されている．

このように，信頼性向上に効果があるとされるジャーナリング機能であるが，正しい動作を実現するには，ディスクに対するデータの書き込みの順序や同期を正しく制御する必要があり，ファイルシステム本体だけでなく，下位のデバイスも含めた全体的なサポートが必要となる．

Ext3 に関しては，過去に同期 I/O が非同期 I/O 並みに高い性能を示すことが報告されている[2]．これは，同期処理が正しく行われていなかったためである可能性が高い．我々は，全体を通じて正しい動作が行われているか検証する必要があると考えている．

本稿では，Linux のジャーナリングファイルシステムにおいて信頼性が確保されていることを確認するには，何をどのように検証すれば良いか考察し，システムコールと DMA のイベントのトレースに基づく解析方法を提案する．そして，Ext3 を対象として，最近の Linux カーネルで実際に解析を行った結果を報告する．

2. 要求条件

Linux の適用範囲について，我々は通常のビジネスシステムでの利用に留まらず，ミッションクリティカルな分野への適用も行えるようにしたいと考えている．この前提では，以下の条件が満たされる必要がある．

(1) 同期 I/O 時のディスクへの書き込み
(2) 整合性が保たれるディスクへの書き込み順序
(3) ディスクの障害特性を考慮したディスクの使い方

以下，それぞれについて述べる．

2.1. ディスク書き込みの同期

ディスク書き込みの同期は，アプリケーションやミドルウェアレベルでの状態の整合性を保証するために，ファイルシステムがサポートすべき機能である．サーバシステムの一貫性の保証には，一般にファイルシステムの整合性に加えて，アプリケーションが要求するタイミングでデータに加えられた変更が二次記憶装置に反映される必要がある．ファイルシステムに関し，この機能を与えるシステムコールには以下のようなものがある．

- open()システムコールの O_SYNC もしくは O_DSYNC オプション[2]
- fsync(), fdatasync()システムコール
- sync()システムコール

これらシステムコールの完了までにディスクの書き込みが行われていれば，同期が保証されていると言うことができる．

厳密に言うと，ディスクへの書き込みの保証といった場合，磁気媒体に物理的に記録するまでを保証するのか，ディスクキャッシュに書き込むまで(ディスクへのデータ転送の完了まで)

[2] これらオプションは後続する各 write()システムコールで同期がとられることを保証する意味をもつ．

を保証するのか，区別しなければならない．ディスクキャッシュ上のデータは電源が断たれると喪失する可能性がある．これを回避するには，以下のような方法がある．

- ディスクキャッシュのフラッシュコマンドを発行して同期をとる．
- ディスクキャッシュを無効にする

また，ミッションクリティカルな分野に関しては，バッテリ等でバックアップされたディスクキャッシュメモリを利用することで，同期に伴う性能の低下を避けつつ，この問題を回避する手段がとられる．

2.2. ファイルシステムの整合性とディスク書き込みの順序

ジャーナリングファイルシステムにおいて，整合性を保つために守られなければならないディスクの書き込み順序を図 1 に示す．

(1) **トランザクション書込み**
ファイル操作によりメタデータに変更が生じると，更新されたメタデータのブロックはジャーナルに書き出される．この更新は一貫性を保証する単位毎(トランザクションと呼ばれる)に扱われる．

(2) **トランザクションのコミット**
一連の操作が完了するとジャーナルに完了したことを表す印がつけられる．

(3) **チェックポイント**
メタデータのブロックをディスク上の通常の格納領域に書き出す．

(4) **ジャーナルのクリーンナップ**
ジャーナル上からコミットされたトランザクションを削除する．

障害が発生した場合，コミット完了前のファイル更新は破棄される．一方コミット完了後のファイル更新は，図中(3)'に示すようにジャーナルの記録を元にメタデータのブロックの復旧が行われる．

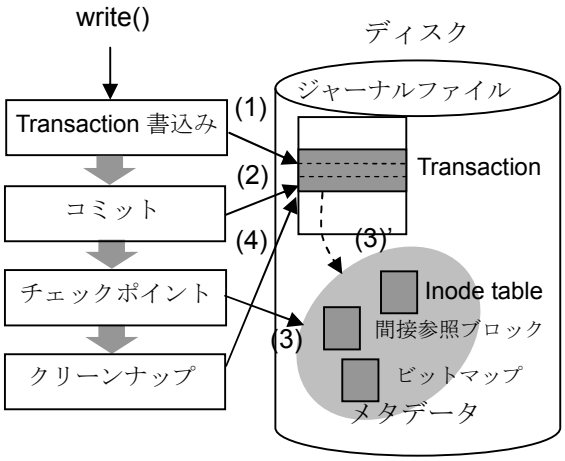


図 1. ジャーナル更新の手順

ファイルシステムの一貫性を保証するには，このようにメタデータのみの順序保証を行えばよい．Ext3 には，さらにファイル実体の更新を救済する ordered モード(デフォルト)と，journal モードがサポートされている．ordered モードは，データの書き込みをメタデータより先に行うことで，コミット後のデータとメタデータの一貫性，特に拡張部分に書かれたデータの正当性を保証する．journal モードは，データに対してもジャーナリングを行うことにより，コミット前と後の両方についてデータとメタデータの一貫性を保証する．

こういった順序保証は一見簡単に達成できるように見えるが，ブロックデバイスや更に下位の IDE,SCSI レベルの実装の影響を受けうることに注意しなければならない．例えば，エレベータシークアルゴリズムの適用や DMA の介在により I/O 要求が入れ替わると上記の条件は満たされなくなる恐れがある．下位レイヤのサポートが同時になされていなければ，ファイルシステムの信頼性は確保できない．

同期との関係で言うと，write()などのファイル操作を同期モードで行う場合，システムコールはそのトランザクションのコミットが完了してから返るような順序関係が満たされなければならない．

2.3. ディスクの障害特性の考慮

ハードディスクの故障は，製品と利用形態(書

き方)が影響する. 具体的には以下のような事象が挙げられる.

(1) 書き込み頻度が高いセクタは(代替セクタのカバー分も含め)壊れ易い.
(2) アクセスパターンが同じディスクは，特に製造ロットが同じであると，同時期に壊れることが多い.

(1)は，スーパブロックなどの固定的なメタデータの格納域が物理的には壊れ易いことを意味している．同じことはジャーナルファイルについても言えることである．これら領域が破壊された場合にも，ファイルシステムが致命的な状態に陥らないように，重要かつアクセス頻度の高いメタデータは書込み領域の分散や適切な冗長化が行われていることが求められる．

さらに(2)は，ミラーリングなどにより冗長化を行ったとしても，その効果が小さいこと，可能であればアクセスパターンが同じにならないような制御をするのが望ましいことを示唆している．

3. 検証方法

本稿では，前節で述べた要求条件の主に(1)を対象として調査方法を検討した．具体的な方法としては以下の解析方法が挙げられる．

(A) バスアナライザを用いた観測
(B) ハードディスクの消費電力の観測
(C) システムコールの応答時間の観測
(D) カーネルのイベントログによる解析方法
(E) ソース解析

(A)は，IDE や SCSI のバスの電気信号を観測して，ホストからデバイスに対し，発行されるコマンドの内容とタイミング，またデバイスからホストに通知される完了信号のタイミングを見るものである．IDE や SCSI といったバスに関しては，専用のバスアナライザが市販されている．ただし，こういった観測装置は元々バスコントローラのデバッグを目的としたものであるため，信号レベルの観測機能が中心となっている．コマンドレベルの抽象度でバスの通信を観測する機能を有するものを選ぶ必要がある．また，要求条件(1)の確認のためには，システムコールの発行時刻と戻り時刻に対する前後関係が記録できなければならないが，現時点で Linux を対象としてそのような機能を有する装置は見当たらない.

一方(B)は, ハードディスクの消費電力から内部のアクティビティを推測しようとするものである．消費電力の測定からデバイスの内部動作を解析する手法は，暗号処理デバイスの解析用途で提案されている[4]. ハードディスクに関しては実際の書込み動作のタイミング解析に応用できる可能性がある．詳細は後述する．

(C)は，システムコールの応答時間をテストプログラムあるいはベンチマークツールで測定することで，同期処理の実施の有無を確認するものである．簡単に実施できるが，異常ケースの抽出が主となり，正しい同期処理が行われていることの検証には不十分である.

これに対し(D)は, カーネルのソースコードに修正を加えて，システムコールの発行時刻や DMA の要求時刻，応答時刻の記録をとり，そのログを解析することにより同期処理の完全性や I/O 順序の妥当性を確認するものである．ソースコードが入手可能であることが前提になるが，カーネルの内部動作を捉えるのに効果的である．ただし，観測手段を埋め込むことによるオーバヘッドやスケジューリングへの影響に注意する必要がある.

(E)は，内部動作の理解に欠かせないが，今回のように複数の機能層をまたがる場合，解析する範囲が広くなり時間を要する．また Linux はソースコードの移り変わり早いこともあり，必ずしも効率的でない.

以上の考察から，今回の調査では，信頼性に関する要求条件が満足されているかどうかを容易に確認できると思われる(B)と(D)の方法について検討する.

3.1. 消費電力測定の詳細

市販のデジタルマルチメータには，時間分解能は高くないものの，測定データを PC にリアルタイムで取り込み, 履歴をとれるものがある．例として，表 1 のデジタルマルチメータを用い

て，Linux カーネルのソースを解凍後，コンパイルを実行する際の消費電流の時間変化を採取した参考データを図 2 に示す．全般的にディスクの書込みが行われている期間に消費電流値が高くなっている．これはディスクの読み書き動作やシーク動作によるものと推測される．

この手法は，やはりシステムコール発行や DMA との時間的な対応関係がとれないため，現時点では間接的な情報である．ただし，ディスク媒体への実際の書き込みを観測しうるという点で，ハードディスクのブラックボックス化に対する有効な手段として期待が持てる．

表 1 デジタルマルチメータの諸元

型式	Sanwa Digital Multimeter PC5000
サンプルレート	1.5 回/秒
インタフェース	RS-232C or USB

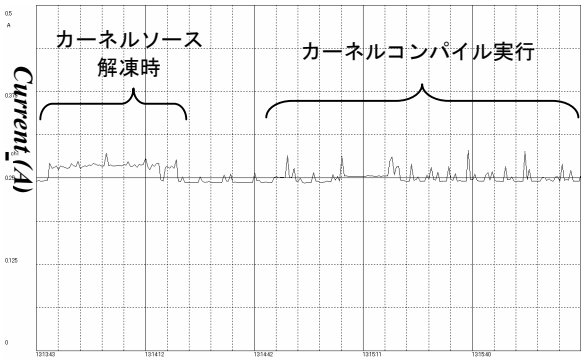


図 2 ハードディスクの消費電流測定の例

3.2. イベントログによる解析の詳細

この方法の有効性を判断するために作成した，一次的な解析環境の概略を図 3 に示す．

イベントの記録には，今回専用の機構を実装せず，カーネルデバッグ関数の **`printk()`** を用いる．ただし試験対象への影響を極力抑えるため，(1)試験対象ディスクと作業用ディスクの IDE バスを分離する，(2)メッセージレベルによりログに出力するメッセージを絞る，(3)出力先のログファイルを絞る，などの調整を行う．また，カーネルに挿入するコードでは，デバイス名やファイル名との照合を行い，試験に無関係のシステムコール呼び出しや，DMA 発行がログに出力されないようにする．この制約がないと，ログ自体の書き出しがイベントを生成し，再帰的な呼び出しになってしまうので注意が必要である．

カーネルに挿入するコードの規模は，全体で 300 行程度である．試験を行うカーネルのバージョンごとにパッチを作成する必要があるが，観測用コードを挿入するのは，それぞれカーネルの最上位と最下位のインタフェース部分であり，修正は容易である．

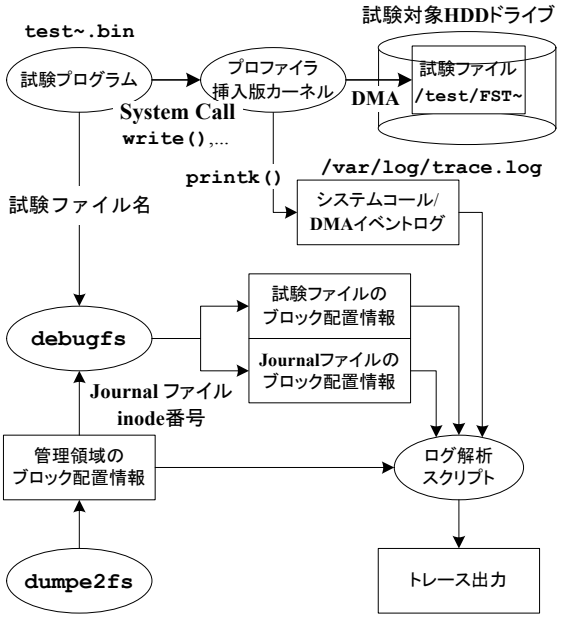


図 3 解析環境の構成

イベント発生時刻の計測には，システム開始からの実行クロック数を返す read time stamp counter (RDTSC)命令[5]を用いる．この命令は CPU 内のレジスタを読み出すものであり，負荷が少なく正確な時刻を求められる．

DMA 部分では，I/O 対象のセクタ番号とセクタ数が抽出でき，そこからブロック番号とブロック数が特定できる．解析では I/O 対象のブロックがメタデータなのかデータなのか，あるいはどういったメタデータなのか識別する必要がある．今回は，対象とするファイルシステムを Ext3 に絞り，Ext3 のデバッグツールを利用する．具体的には，スーパブロック，ブロックビットマップ， i ノードテーブルなどの管理領域のメ

タデータの識別には **dumpe2fs** の出力情報を用い，ユーザデータ領域中のデータブロック，間接参照ブロックなど識別には **debugfs** より得られた配置情報を利用する．

Ext3 ではジャーナルログもユーザデータ領域中のファイルであるため，同じく **debugfs** により配置情報が取得できる．これらの出力ファイルを Perl で記述したスクリプトで解析し，最終的なトレース出力を自動生成する．

各テストはクリーンな状態で行うため，試験プログラムの開始時にテスト対象の HDD ドライブをマウントし，終了時にアンマウントする．Ext3 では，メモリ中のトランザクションのコミットとクリーンナップは kjournald カーネルスレッドにより 5 秒周期で行われている．また，チェックポイント処理は通常のページキャッシュの書き出し処理として 30 秒周期で行われている．そこで，これらデーモンによるディスク書き込みを確かめられるように，ファイルのクローズからアンマウントするまでの期間を調整できるようにする．後述する試験では 40 秒のマージンを挿入している．

4. 調査結果

要求条件の(1)が Ext3 で満たされているかを，Debian[3] GNU/Linux 3.0 (Kernel 2.6.5 及び 2.4.25)，それに RedHat[3] Enterprise Linux 2.1 (Kernel 2.4.9-e.12)について確認した．検証マシンのハードウェア構成を表 2 に，また使用ハードディスクの諸元データを表 3 に示す．2.1 節で述べた各同期手法について検証を行った．Kernel 2.6.5 のトレース結果の一部を図 4 に示す．

表 2 検証機のハードウェア構成

CPU	Athlon XP 1600+
動作 Clock 周波数	1405.96MHz
Mother Board	Gigabyte 7VTXE 1.0
Memory	1GB
IDE Controller	VIA VT8233

[3] Debian は Software in the Public Interest, Inc.の登録商標です．また RedHat は RedHat Software,Inc.の商標または登録商標です．

表 3 検証に用いたハードディスクの諸元

型式	Western Digital WD205BA
ディスク容量	20GB
バッファサイズ	2MB
回転数	7200rpm
シーク時間	9ms
転送モード	IDE Ultra DMA 4
データ転送速度	23.8MB/sec (実測)

図 4 は既存ファイルを上書きする例で，ordered モードと journal モードのそれぞれにつき，同期せずに書き込みを行った場合と，O_SYNC オプションを指定して同期モードで書き込みを行った場合の組み合わせの 4 パターンを掲載している．実際には新規ファイル作成の場合，書き込みを繰り返し行う場合，更に高負荷状態で書き込みを行う場合などについても試験を行っている．

図 4 の例では，書き込みはファイル上をブロック(4KB 単位)で数えて，0 ブロック目，12 ブロック目，1036 ブロック目の位置に計 3 回行っている．同期処理を行わない図 4(a)および(c)では，write システムコール期間中に，ディスクへの書き込みは一切行われず，12 ブロック目と 1036 ブロック目のデータブロックの所在を知るための間接参照ブロックの読み込みが行われているだけである(0 ブロック目の所在は既読の inode に書かれているため，間接参照ブロックの読み込みは伴わない)．実際の書き込みは，(a)では kjournald によって行われている．そしてデータ書き込みの後，更新日付の変更を反映するため inode テーブルがジャーナルに書かれている．

この場合，ジャーナルへの書き出しの J[1]はトランザクションの開始を表すディスクリプタブロック，J[2]は inode テーブルの写し，そして J[3]はトランザクション終了を表すコミットブロックに対応する．J[0]は，ジャーナルのスーパブロックにあたり，この書き出しは初期化のためである．

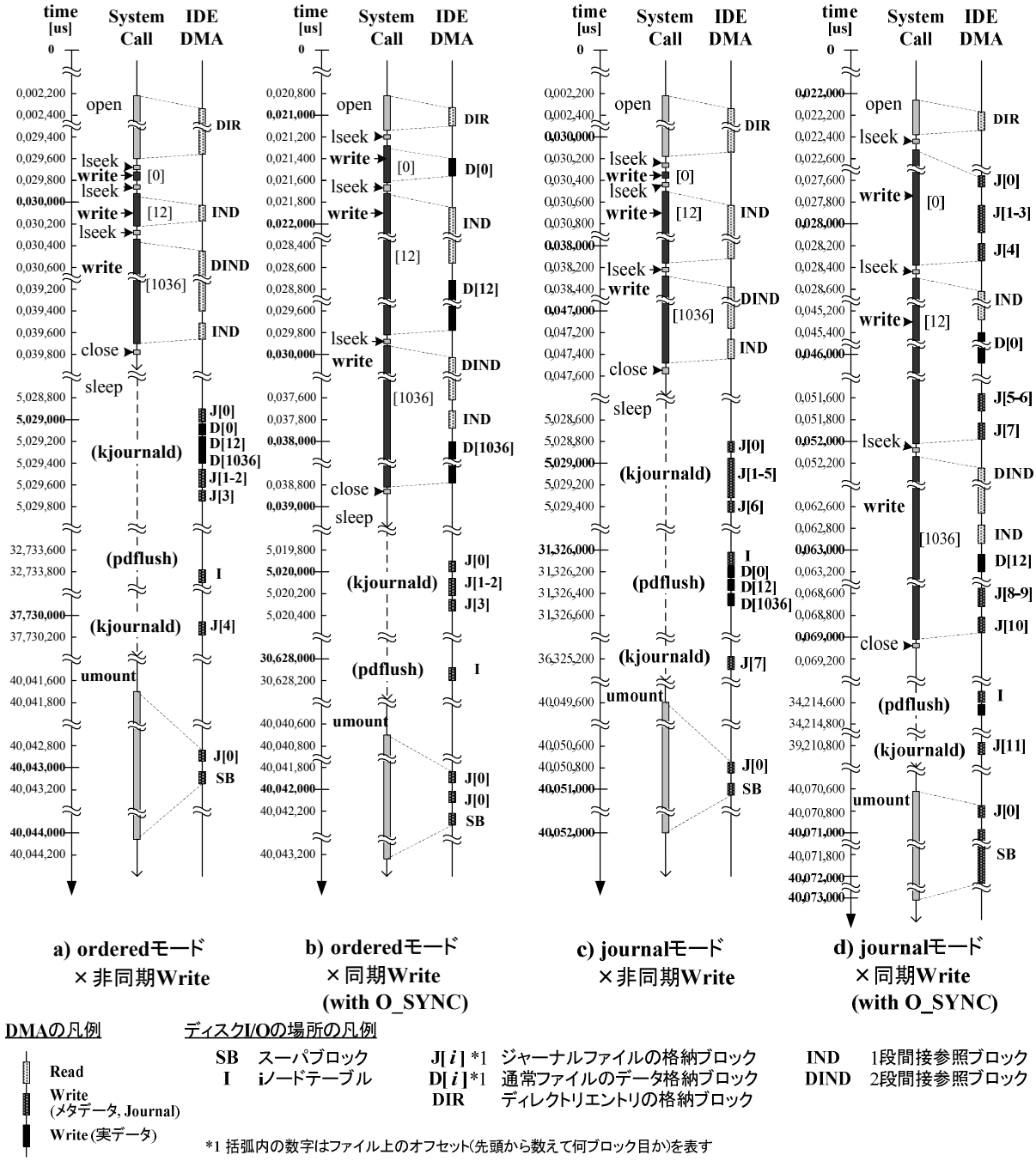


図 4 Ext3 のシステムコールと DMA の順序のトレース結果(既存ファイルの上書きの場合)

そして，その後 30 秒おきに起動している pdflush によって inode が本来の場所に書かれ，更にその 5 秒後の kjournald によって，ジャーナルからトランザクションを除去することを表すコミットブロック(J[4])が書かれている．

(c)の場合は，データも inode と同じように一旦ジャーナルに書かれた後，pdflush により本来の場所に書き出されるため，データブロックの分，ジャーナル上の書き込みブロック数が増えている．

一方，同期処理を行う図 4(b)の場合は，write システムコールの期間中にデータブロックの書き込みが行われていることが分かる．また図 4(d)の場合は，データを本来の場所に書く代わりにジャーナルに書いてコミットしたことをもって同期がとれたとみなされている．ファイルを

新規に作成する場合や拡張する場合には，書き込み時に inode や間接参照ブロック，ブロックビットマップその他のメタデータも変わるが，同期 I/O 時にはこれらも write システムコール期間中にジャーナルに書かれコミットされていた．

以上のような解析の結果，Kernel 2.6.5 と 2.4.25 については，調査範囲内で同期処理が正しく行われていることが分かった．ジャーナルの書き順については，中身が特定できないためその点厳密ではないが，I/O 順序の制約を満たさない結果は今のところ得られていない．なおメタデータをジャーナルに書き出すポリシーは，バージョンにより微妙に異なっていることが分かっており，図 4 が Ext3 のセマンティクスを代表するものではないことを断っておく．

一方，Ext3 を先駆けて取り込んでいる Kernel 2.4.9-e.12 については，journal モードで O_SYNC(および O_DSYNC)オプションが効いていないことが確認された(図 5)．2.4 系の初期のカーネルを利用している場合，アプリケーションによっては保証されているはずのデータの喪失が起こりうるので注意が必要である．

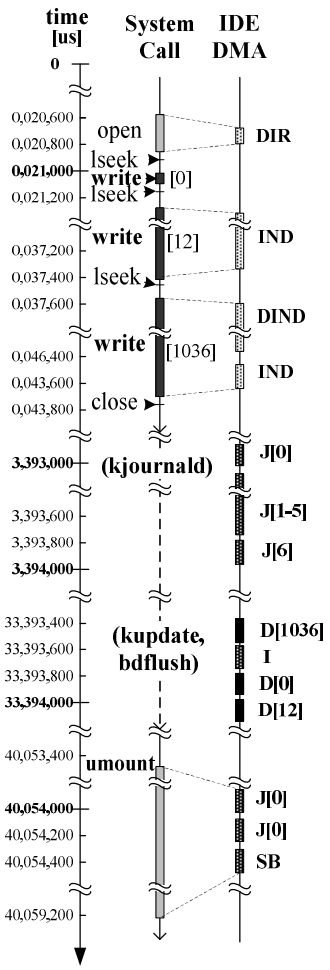
5. まとめと今後の課題

ミッションクリティカルな分野への適用を前提として，Linux ファイルシステムの信頼性の要求条件を述べ，その検証手段について考察した．そして要求条件の一つである同期 I/O 時のディスクへの書き出しについて，システムコールと DMA のイベントをトレースする方法を適用して検証を行った．結果，最新のバージョンでは問題がないが，2.4 系の初期のバージョンには問題が見られることが分かった．

今後，他のファイルシステムに適用を進めるとともに，他の要求条件の検証方法についても，検討を行う予定である．

謝辞

Linux ファイルシステムの信頼性に関しては，NTT データ先端技術の奥山健一氏，三浦広志両氏に議論頂き，貴重なご意見を賜りました．深く感謝いたします．



e) journalモード×同期Write (with O_SYNC)

RedHat Enterprise Linux 2.1 (kernel 2.4.9-e.12)

図 5 同期に問題があるバージョンの例

文　献

[1] S.Tweeide: “Journaling the Linux ext2fs Filesystem,” LinuxExpo ’98. 1998.

[2] 菅谷みどり:「Linux におけるファイルシステムの性能及び信頼性の検証」, Linux Conference 2002.

[3] William von Hagen: “Comparing Journaling Filesystem Performance,” Linux Filesystems Chapter 10, pp.235-247, Sams Publishing, ISBN0-672-32272-2, 2001.

[4] Paul Kocher, Joshua Jae, and Benjamin Jun: “Differential Power Analysis,” CRYPTO’99, LNCS vol. 1666, pp.388-397, Springer-Verlag, 1999.

[5] Intel Corp.: “Using the RDTSC Instruction for Performance Monitoring,” tech. rep., Intel Corporation, 1997.