Linux におけるファイルシステムの性能及び信頼性の検証

菅谷みどり
doly@dcl.info.waseda.ac.jp

1. はじめに

1.1 背景

Linux が商用化するに従い信頼性やスケーラビリティへの要求が拡大している。特にファイルシステムのデータ保護機能が低い場合には、システム全体の信頼性が損なわれる。カーネル2.4 では、これらの要求に対応するため、ジャーナリング機能を持つファイルシステムである ext3, JFS を取り込んだ。また、ファイルシステムを支えるメモリ管理機構、I/O サブシステムの改善により、信頼性を確実にするための仕組みを備えた。

本稿では特にファイルシステムの信頼性の検証を Linux のカーネルも含めた構造として検証する。

1.2 目的

ファイルシステムは、記録単位あたりのコスト面で優れるハードディスクなどの二次記憶装置を、データの記録媒体として利用するために開発されたソフトウェア機構である。

Linux では複数のファイルシステムが利用可能となっているが、ファイルシステムの選択にあたって、信頼性や性能を具体的に評価するための指標が少ない。

この点を考慮し、本稿では特にファイルシステムの機能として重要な信頼性に着目し、その信頼性を高める機能及び性能に関する検証を行うことを目的とする。

現在の Linux ファイルシステムで実装されているジャーナリング機能の目的は、短い時間でデータを回復させることである[1]。「ジャーナリング」という概念は本来データベースから導入された概念で、ユーザーのログを保存しデータを制御するという役割がある。これに対して、ファイルシステムにおいては主に「メタデータのログを保存する」役割を指す[1] (JFS は除く)。

メタデータには、ファイルシステム内部の制御構造である inode, freeblock allocation maps, inodes-map などが含まれる。速いリカバリを実現するためのジャーナリング機能は、それ以外にもファイルシステムはデータの統合性が確認される点で、fault-recovery といった性質も備えている。つまりデータの更新毎にメタデータを不揮発性ディスクに書き込む処理により、データの安全性も確保される。

こういったジャーナリング機能は、性能面から評価する場合に速度差をうめるための機構について考慮する必要がある。二次記憶として用いられるディスクのアクセス速度は通常 CPU、RAM に比べると 100 万倍の差がある。速度差を埋め、効率的に処理を行うためには、オペレーティングシステムが提供する遅延処理などのシステムを利用する事が必要となる。

Linux では特に仮想記憶を利用した階層的なキャッシュ構造で遅延 (非同期) 処理をサポートしている。ファイルシステムは、こうした機能を利用し、信頼性と性能という目的を達成する。本稿では Linux におけるこれらの兼ね合いを検証し、ファイルシステムの機能を明らかにすることを目的とする。

2. Linux ファイルシステムの特徴と課題

2.1 Linux ファイルシステム

ファイルシステムの目的は物理ディスクを隠蔽し、オペレーティングシステムが提供する遅延処理を利用して効率的にディスクを管理することである。この目的に沿いファイルシステムは様々な管理方法を提示している。

Linux のファイルシステムは、カーネルの階層内では VFS の下位、ハードウェアの上位に位置する。VFS は Virtual File System と呼ばれる OS のソフトウェア機構で、ユーザからファイルシステムの差異を意識することなく利用するためのインターフェイスを提供している。

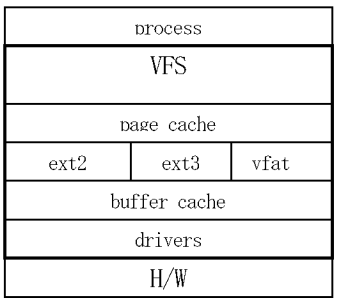

図1: Linux ファイルシステムの階層

2.2 VFS (Virtual File System) 機能

VFS は SunOS4.X, SystemV Release 4 の実装に由来する。仮想記憶を効率的に利用してファイルシステムを実装するために考案されたもので、UNIX で広く利用されている[2]。VFS は設計当初から「オブジェクト指向」を明確にしており、Linux でも i ノード構造体、ファイル構造体、d エントリ構造体などがその設計思想に沿って実装されている。

Linux は、システムコールに対しファイルシステム毎の差異を隠蔽するインターフェイスとして VFS を実装することで、複数のファイルシステムを透過的に扱っている。ユーザプロセスは、ファイルポインタを通じ、VFS 経由で構造体として定義される i ノードなどのオブジェクトを操作する。これらのオブジェクトはキャッシュという形でメモリ上に保持される。VFS はユーザがこれ

らのオブジェクトに効率的にアクセスするための環境を提供する。ファイルシステム側では、VFS の API を利用することで、VFS のオブジェクトやキャッシュ機構を利用することができる。

2.3 Linux へのファイルシステムの移植性

LinuxのVFS はUNIXで広く利用されている機能を実装しているが、特に「ファイルシステムの移植性の高さ」が指摘される。通常ファイルシステムの実装者はI/O部分の操作まで実装を求められるが、Linux ではインターフェイスが汎用的で、利用しやすい事が利点にあげられている[3]。

Linux の VFS はバッファやキャッシュを利用するための汎用的なインターフェイスを提供しているが、ファイルシステム自身がバッファリングやキャッシュングの機能を実装することも可能である。

システムコールをファイルシステム層固有の操作関数に変換する部分の仕組みは Linux のドライバとほぼ同じ仕様でシステムコールを固有関数に置き換えるものとなっている。設計者側では取捨選択の自由度が高い。

ext2, ext3 等のネイティブなファイルシステムはこうしたLinux のAPI の使用率が高く、VFS 層の動作に性能が依存する部分が大きい。このように、Linux のファイルシステムはVFS を中心としたファイルシステムの実装が特徴といえる。

次に各ファイルシステムの特徴と課題について述べる。

2.4 ext2, ext3, ReiserFS, JFS

Linux カーネル2.4 では、40 前後のファイルシステムが利用可能となっている。代表的に利用されているのは現在、ext2, ext3、ReiserFS、JFS である。本稿ではこの4つのファイルシステムの検証を行う。

2.4.1 ext2

ext2 は、Linux で伝統的に使用されてきたディスクベースのファイルシステムである。BSD の FastFile System(BSD FFS)を元にして設計されている。領域はブロックグループに分割されて管理されている。ブロックはグループ化され、inode のエントリの最適化、キャッシュの利用により、データのアクセスの高速化が図られている。データの一貫性に対して保証されていないが、fsck プログラムによるスーパーブロックの回復機能がある。また、全ての書き込みはバッファキャッシュを利用して非同期に行われる[4]。

2.4.2 ext3

ext2 にジャーナリング機能を追加したファイルシステムである。Journaling Block Device layer(JBD)と呼ばれるジャーナリング機能のためのシステムコールを利用している。また、トランザクション層とジャーナリング層の機能を分け、効率を向上させている。JBD は独立性と汎用性を持ったジャーナリング機構として設計されており、どのような種類のブロックデバイスに対しても利用できる[8]。

2.4.3 ReiserFS

ReiserFS は、”fast balanced tree“と呼ばれるB-Tree(平衡木) アルゴリズムに基づいたファイルシステムを提供している。特にB-Tree をベースとしたアルゴリズムでは、最悪でもO(log n)に計算量を抑えるなどの効果が得られる。ReiserFS ではB-Tree に拡張を加えたB+Tree を採用している[11]。

また、ReiserFS では動的な inode の割り振りによるディスク領域の使用効率を上げるなど性能向上のための様々な設計が試みられている。

2.4.4 JFS (IBM's Journaled File System)

IBMがエンタープライズ用途でAIX、OS/2 といったUNIXに実装してきたファイルシステムを、Linux に移植したものである。信頼性とスケーラビリティの高さが注目されており、エクステント・アロケーションシステムにより複数ディスクの管理、複数ブロックサイズのサポートや、ダイナミック inode によるメモリ効率の増加、B+Tree によるファイル操作の効率化など、可用性を柱とした実装がなされている。

2.5 ジャーナリング機能

カーネル2.4 からはext3, ReiserFS, JFS が取り込まれた。ファイルシステムのジャーナリングの特性については、「1.2 目的」で述べたとおりであるが、主なメリットは、書き込み時にメタデータのログをディスクに保存する事で、障害時のすばやい復旧、及びデータの一貫性についてある程度の整合性保たれる事が挙げられる。ログの参照による復旧は、ext2 の fsck のようにディスクをスキャンした上で整合性を取るといったプロセスが不要になるため起動時間が大幅に短縮される。

2.6 カーネル2.2, 2.4 までのファイルシステムの課題

主にカーネル2.2 までのファイルシステムの問題点を整理すると次の通りとなる。

1. fsck に時間がかかる
2. ファイルの書き込み処理に時間がかかる
3. ファイルシステムが安定せず、inode の不整合などが生じやすい、といった傾向が指摘されてきた。[1]

[1] メーリングリスト、苦情など

これらの課題については、ジャーナリングファイルシステムの導入によってどの程度の改善がなされているのかを確認する。また、カーネル2.2 時の導入時からカーネル2.4 でのさらなる改善について実際にテストを行って内容を確認する。

3. 検証と評価

3.1 検証の方向性の検討

カーネル2.2 までの問題点を踏まえ、新機能を正しく評価するための方向として次の観点を示す。

[検証の観点]

ファイルシステムの信頼性を多角的に評価するため、次の3つの観点からテストを行う。

1. ジャーナリング機能の検証
2. 性能に関する比較
3. カーネルのバージョンへの依存性の検証

1 では、電源遮断が生じた場合のデータの保全性（復旧率）を検証することでジャーナリング機能の検討をおこなう。

2 では、読み込み、書き込み操作によるパフォーマンステストを行う。

3 では、1,2 を通じて3のカーネルのバージョンへの依存性の関連について考える。

この中で信頼性と性能がどのように両立するかを明らかにすることを目的とする。

検証にあたっては、以下の点を考慮する。

1. CPU キャッシュ、物理ディスク、バス等ハードウェアの差異による影響を最小限にするため、同じテストは同一マシンで実施する。
2. カーネルは2.2系, 2.4系を対象とする。ファイルシステムのバージョンはカーネルのマイナーバージョンで標準的に取り込まれているものを利用する。カーネルの動作に影響が生じると考えられるパラメータ等は、カーネルを通じて同一のものを使用するように調整を行い、条件を極力揃える。

3.2 ジャーナリング機能の検証

ファイルシステムのジャーナリング機能の中でデータの保存性についての機能を検証する。データベース等のアプリケーションテストの場合、データ作成中、作成後等トランザクションのモニタリングも含んだジャーナリング機能の検証を考慮するが、今回の検証ではファイルシステムとしての性能の一部としての評価を行う事にとどめる。

3.2.1 書き込み時のクラッシュにおける復旧率

テストは障害と復旧の時間が明確になるように、書き込み操作の後の経過時間を複数設定し、その経過時間後に計画的にシステムの電源を落とし、起動後のファイルサイズを測った。書き込みはput() で1バイトずつ行いclose()した時点からの時間を 0,5,10 秒の間隔をおいた。通常のデータベースの場合にはトランザクションの途中の処理についてabort 操作などによるデータの復旧を確認するが、今回ファイルシステムの調査にあたっては、そういった実装があるファイルシステムとないファイルシステムがあるために、close()まで完全に終了した段階でのクラッシュという形でテストを行う事にした。ファイルの中に別のデータが混じるなど、整合性が取れなかった場合には、0 とした。テストは1つの項目につき複数回実行しその平均値を結果とした。

3.2.2 バッファのフラッシュタイム

通常書き込み時は速度向上のために遅延書き込みが行われる。書き込みはRAMに対して行われるため、電源遮断時のデータの保全性が悪くなる。テストではこの点を明確にするため、遅延操作を指定するパラメータを変更する事による影響をみる。バッファは「汚れた(dirty)」バッファと呼ばれる書き込み待ちのバッファとなる。値は/proc/sys/vm/bdflush 内のパラメータのうち、age_buffer「更新されたバッファをディスクへ書き込むタイムアウト値」の値を対象とする。

3.2.3 ジャーナリング機能の検証

検証項目と結果を次に示す。

表1 ： 検証項目

1	カーネルバージョン	2.2.18-4. 2.4.18-3
2	ファイルサイズ	1MB, 10MB, 100MB, 1GB
3	システムクラッシュまでの時間	0秒、5秒、10秒
4	バッファのフラッシュまでの時間	30秒、3秒、1秒
5	対象ファイルシステム	ext2、ext3、ReiserFS、JFS

表2 ： 検証機ハードウェア構成

CPU	Memory	HDD	Contoller
Pentium4 1.6GHz	512 MB	IDE	Generic ide

表3 ：ソフトウェア構成

ソフト名	テスト(1)-(3)	テスト(4)-(6)
Kernel	2.2.18-2	2.4.18-3
Glibc	2.1.3-32	2.2.5-34
Gcc	2.95.2.1-16	2.96-110

表4 ： ファイルシステムバージョン

	テスト(1)-(3)	テスト(4)-(6)
ext3	0.0.7a	0.9.17-24
ReiserFS	3.5.29	3.6.25
JFS	-	1.0.14

3.2.4 結果

表5: 計画的なクラッシュにおけるファイルの復旧率

	1M			10M			100M			1G		
	0秒	5秒	10秒	0秒	5秒	10秒	0秒	5秒	10秒	0秒	5秒	10秒
1) a	50%	0%	100%	93%	100%	100%	92%	100%	100%	100%	100%	100%
b	50%	0%	100%	99%	100%	100%	94%	100%	100%	100%	100%	100%
c	0%	0%	0%	0%	0%	0%	91%	91%	88%	99%	100%	99%
2) a	50%	100%	100%	103%	100%	100%	97%	100%	100%	100%	100%	100%
b	50%	100%	100%	96%	100%	100%	98%	100%	100%	99%	100%	100%
c	50%	0%	0%	50%	33%	67%	97%	94%	91%	99%	100%	99%
3) a	0%	100%	100%	88%	100%	100%	97%	33%	33%	100%	100%	100%
b	85%	100%	100%	86%	100%	100%	96%	100%	100%	100%	100%	100%
c	0%	0%	100%	98%	63%	96%	96%	95%	96%	99%	100%	99%
4) a	0%	0%	0%	90%	100%	100%	80%	98%	77%	98%	98%	99%
b	0%	100%	100%	89%	100%	100%	91%	100%	100%	99%	98%	100%
c	0%	100%	100%	97%	72%	96%	90%	100%	100%	99%	98%	100%
d	0%	100%	100%	99%	100%	100%	95%	100%	100%	99%	99%	100%
5) a	0%	0%	50%	83%	100%	100%	91%	93%	100%	99%	99%	100%
b	0%	100%	100%	72%	100%	100%	92%	93%	100%	99%	99%	100%
c	0%	100%	100%	72%	50%	100%	92%	93%	100%	99%	99%	100%
d	0%	100%	100%	72%	100%	100%	92%	97%	100%	99%	99%	100%
6) a	0%	100%	100%	98%	50%	100%	87%	92%	81%	100%	99%	100%
b	0%	98%	100%	92%	0%	100%	87%	92%	91%	100%	99%	100%
c	0%	94%	100%	92%	0%	100%	87%	92%	91%	100%	99%	100%
d	0%	98%	100%	92%	100%	100%	87%	100%	100%	100%	100%	100%

注: a: ext2, b: ext3, c: ReiserFS, d: JFS

※ ext3 のジャーナリングモードは orderd

ファイルサイズ 1M, 10M, 100M, 1G

シャットダウンまでの時間 0 秒, 5秒, 10秒

1) カーネル2.2/30秒　　4) カーネル2.4/30秒

2) カーネル2.2/ 3秒　　5) カーネル2.4/ 3秒

3) カーネル2.2/ 1秒　　6) カーネル2.4/ 1秒

表6: テスト結果の内容

	1	2	3
1)	2.2	30	・ReiserFS は 10M までの間は整合性が取れない場合には削除される現象が生じた。10M 以上のファイルの場合復旧後のデータ内容は整合性が取れない現象が頻繁にみられた。 ・ext2 と ReiserFS はデータの整合性に問題が生じるケースが多く見受けられた。
2)	2.2	3	10M の場合の復旧率は1)より改善された。但しファイルの整合性に問題があるケースが存在した。
3)	2.2	1	Rseiserfs の 10M では改善が見られた。また、1M のファイルに関しては、ext2 について 1) よりも改善がみられた。
4)	2.4	30	ジャーナリングの有無によって明確に残存する率が変化した。特にジャーナリング機能を実装していない ext2 に関しては 1M, 10MB の場合には全くファイルが残らなかった。
5)	2.4	3	1M の 0 秒では ext2 の全てのファイルが残らなかった。
6)	2.4	1	6) では、常に 10M の5 秒後の回復時には JFS のみだけが安定して復旧率が保たれた。

※ 1. カーネル, 2. フラッシュまでの時間(秒), 3. 結果

3.2.5 結果についての評価及び考察

(1) 高い JFS の復旧率

今回0 秒以外の全てのケースで JFS は約 100%データが残存し最も信頼性の高い結果となった。データ内容の整合性も問題は見受けられなかった。

(2) ReiserFS の改善

ReiserFS は、B+Tree はファイル操作には適しているが、クラッシュ時の回復に際しては、ディスク配置の Balancing 操作が複雑で大量の余計な I/O が必要となる事が指摘されていた[8]。

実際、カーネル 2.2 の結果では 100%整合性が取れた形で復旧したのは 1 ケースで、それ以外は十分な復旧がなされておらず、データがマージされる事が多かった。さらに起動時の ReiserFS の fsck が障害を見越してファイルを削除するケースが見られた。(表 6: 1),2),3)) この点に関しては、カーネル 2.4 では大きく改善がみられた。2.2 でみられたマージされる現象が無くなり、復旧される比率が高くなった。これは 16KB 以上のファイルに連続したブロックを割り当てる、大きなファイルに割り当てた木のノードのグループを小さいファイルと別のノードに保存するなど、2.2 から 2.4 の対応の中にブロック割り当て、保存における改善がみられた事などが原因の一つと指摘できるだろう。

(3) ext2, ext3

ext2 はカーネル2.2 からカーネル2.4 になることで、数値上では復旧率が低下している。但しカーネル 2.2、2.4 いずれのバージョンであっても、フラッシュのタイミングを早める事で残存率が改善された。 同様に ext3 では、カーネル 2.4 になる事で 0 秒後、5 秒後の値が若干低下しており、フラッシュタイムの操作により一部改善している。 ext2, ext3 は、後に述べる性能に関しては、カーネル 2.4 となる事で向上している。

(4) キャッシュの効果とジャーナリング機能の独立性・フラッシュタイミングの効果

フラッシュのタイミングの操作はカーネル 2.2 及びカーネル 2.4 を通じて ext2 が最も効果があった。特に

カーネル 2.2 の 30 秒では全てが消えたが、3 秒にすることで100%復旧された。
・カーネル 2.4 では、ext2 の 3 秒で復旧率が上がった以外は逆に 1 秒では JFS 以外のファイルシステムは悪化した。これは、カーネル 2.4 では I/O サブシステム全体の効率が上昇しキューの処理が追いつかず性能悪化を引き起こしている事などが考えられるが、具体的内容については次回の検討課題としたい。

・バッファに左右されない独立機構

フラッシュタイミングの効果より、ext2 はバッファキャッシュの影響を受けるのに比べて、ジャーナリング機能を持つ ext3, JFS が、フラッシュのタイミングに影響されない事が指摘できる。(表 6: 4)-6))

これは ext3, JFS のジャーナリング機能であるメタデータの更新記録を独立した領域に書き込む方法に起因すると考えられる。特に ext3 では更新記録のメタデータはカーネルスレッド (kjournald) によってディスクに書き込まれる。

カーネルスレッドは、メタデータをバッファに書き込む機能 (コミット) と、古い情報を持ったログを削除する(チェックポイント)機能を持っている。このスレッドはデフォルトで 5 秒おきに起動されるため、5 秒前後で作成されたデータは、整合性及び一貫性のある形で復旧する。

JFS においては、カーネルスレッド (jfsCommit) として同様の処理が実装されている。jfsCommit は割り込み I/O 要求がある場合に起動され、トランザクションを処理する。スレッドはファイルシステムがマウントされた時点で起動し、独自の設定でジャーナルデータのフラッシュ操作を行う。

ext3, JFS はこのようにジャーナリングデータの保存に対し、システムのバッファリング機能とは独立した保護機能を実装しているため、システムのバッファ動作に左右されず、信頼性のある機能を提供している事が指摘できる。

(5) ジャーナリング機能評価

カーネル 2.2 ではジャーナリングファイルシステムと非ジャーナリングシステムとの差がそれほど明確ではなかったが、前項(1)-(3)で述べたように、カーネル 2.4 では差が明確になった。

ここで述べた以外にも、カーネル 2.2 の ext2 では復旧した状態であっても、整合性が保たれていないケースが多く、結果も不安定であった。しかし、カーネル 2.4 でジャーナリング機能を持ったファイルシステムでは消失、もしくは残存した場合には、整合性が保たれている。これは ext2 は後にみるように、メモリ (バッファキャッシュ) の利用率が高く、さらにジャーナリング機能がないために、データが消失しやすい事が指摘できる。
(表 6: 5),6))
・ハードとの兼ね合いで考慮すると、フラッシュタイミング値を 1 秒の設定＋クラッシュのタイミングが 0 秒で 10MB 前後のファイルが保存されていることから、1 秒 当たり約 10MB のディスクへの書き込みは行われている。検証では 10MB で 10 秒以上経過したデータは、ほぼ 100% 回復した。フラッシュタイミング値を 30 秒、3 秒、1 秒と値を小さくしていくほどキャッシュに残存するデータが減り、ディスクへの残存率が上がる。ハード側の制約によりフラッシュタイミングの値が 1 秒と 3 秒に関しては逆に復旧率の悪化となる事が示された。バッファフラッシュのキューの増加による I/O 側の処理の圧迫が生じるためと思われる。

ジャーナリングを全く考慮しない場合にはデータロスの値は書き込み速度とクラッシュまでの時間を乗じた値にほぼ等しく、当然の事であるが、ファイルが大きくなるにつれロスの比率は下がる。[2]

3.3 結論

以上復旧率の調査により、カーネル 2.4 において個々のファイルシステムによって結果や傾向が異なる事がわかった。復旧率の順序では、JFS、ext3、ReiserFS、ext2 となった。また、カーネル 2.4 では、ext2 の復旧率が 2.2 の場合より低下し、ジャーナリング機能を持つファイルシステムと非実装のファイルシステムで信頼性の優劣が明確になった。
また、信頼性を持つファイルシステムも JFS 以外は今後まだ機能について改善が求められるといえる。

4. 性能調査

4.1 性能測定の指標

前項では信頼性に関するテストを行った。次に性能を測定する。測定はシステムコールごとのスループットを指標とした。特に以下の点を考慮した。
・ベンチマークプログラムは Linux で一般的に使用されている bonnie++を用いる。
・書き込み時に遅延処理のスワップ処理のオーバーヘッドが生じないように、メモリ量は性能で表示される平均スループットで割ったデータ量と物理ディスクのものとの差の最大値が物理メモリを超えないように設定した。
・経過時間が短すぎる場合には、性能測定の平均値にブレが生じるので、測定では 1GB 以上のファイル操作で性能を測定した。

---

[2] (6)において JFS のみが復旧したのは、この機能が 5 秒以下で動作していた為と考えられる。

表8: 検証機ハードウェア構成

CPU	Memory	HDD	Contoller
Duron 1.3GHz	768 MB	SCSI(Ultra2)	Symbios

表9 :ソフトウェア構成

kernel version	2.2.18-2	2.4.18-3
bonnie++	1.02a	

4.2 測定結果

4.2.1 各ファイルシステムの性能における特徴

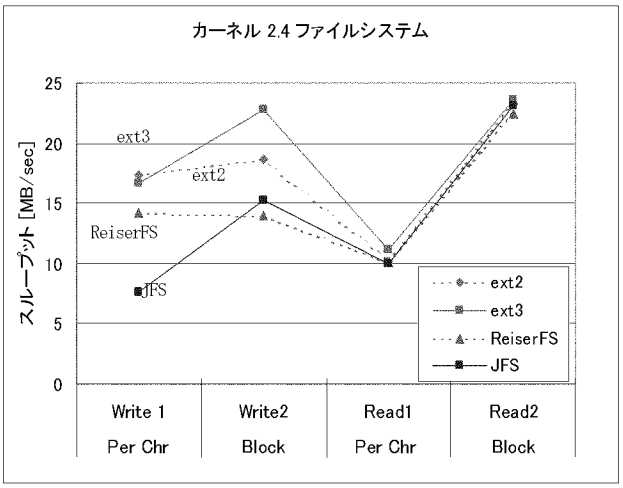

図2: カーネル2.4 におけるファイルシステム

・ジャーナリング機能を持つファイルシステムのスループットが高い事が示された。ext3 はキャラクタ、ブロックベースの Write/Read 共に 10MB 以上のスループットを示した。

ブロックサイズの変化による影響を見るためサイズを 1KB に変更してテストを行った。その際の性能比を次に示す。

表10 : 異なるブロックサイズでのファイルの性能比

[4KB/1KB, 単位:100%]

	Write 1	Write2	Read1	Read2
	Per Chr	Block	Per Chr	Block
ext2	1.6	2.1	1.4	1.9
ext3	1.0	1.7	1.2	1.7
Riserfs	1.0	1.4	1.2	1.7
JFS	1.2	1.2	1.2	1.6

結果よりブロックサイズは、Linux のページフレームの単位でもある 4KB の方が 1KB よりも平均して1.4倍効率がよい。 また、特に ext2 では 1KB と 4KB の差が2倍あった。

・CPU 使用率、ファイル操作

・CPU の使用率は性能のグラフと逆で、パフォーマンスがあまり出ない処理に対して使用率が高くなっている。ブロック単位のRead, Write に対しては性能/CPUの効率が良いことが分かる。また、JFS ではキャラクタベースの Write については性能も低いが、CPU の使用効率も低くなっている。

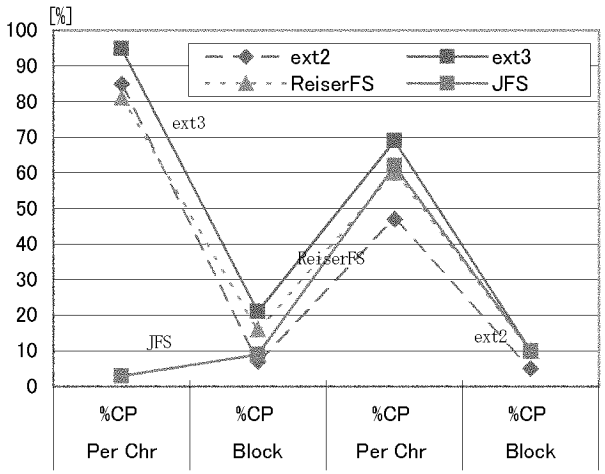

図3: CPU の利用率

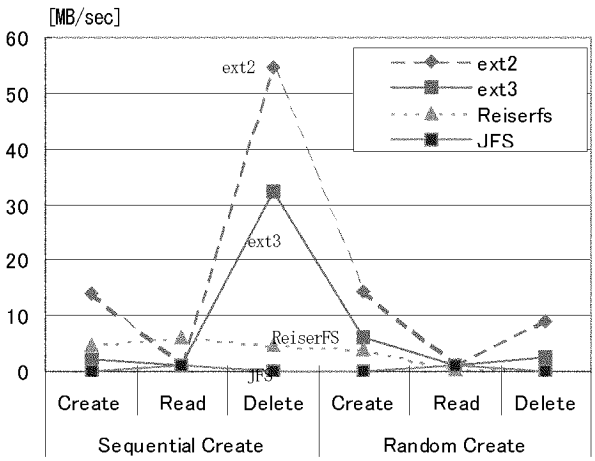

図4 : ファイル操作

図4では、1K から 8K バイトのファイル 100,000 個の操作を行った場合の性能を測定した。このケースでは ext2 の delete のスループット値が最も高い結果となった。

Delete に関して ext2 はジャーナリング機能を実装しておらず、処理が軽量である事が原因といえる。ジャーナリング処理を行う ext3 などでは、削除を行う時点でディスク上のメタデータに同時に記録する追加処理のためのオーバーヘッドがかかる。

これに対し ext2 の unlink() 処理では、アドレス空間のマッピング、inode のマッピングを親のエントリに戻す処理といったアドレスの操作のみである。

4.3 性能に関する検討

ここでは性能テストの結果から検討できる点について述べる。

4.3.1 ジャーナリングファイルシステムの性能の高さ

今回の性能テストでは、ジャーナリングファイルシステムの書き込み時の性能が非ジャーナリングシステムに劣らない事が示された。通常ジャーナリング機能を実装するファイルシステムは、1度以上ディスクに書き込む操作が必要であるため、性能面でのパフォーマンスが通常のファイルシステムに劣ると考えられる。

この点に関し、特にLinuxネイティブのext3 のスループットの高さについて考えられる理由を述べる。

(1) ファイルシステムと、ジャーナリング機能の分離

特に ext3 ではブロックデバイスのトランザクションをモニタするファイルシステム層と、ジャーナリングを管理する層(JBD)の機能を分けた設計がなされている[7]。互いに機能が独立している事で、ファイルシステム層が性能に関して影響を受けない設計となっている。

(2) 更新記録のライトバック

3.2.5「フラッシュの効果とジャーナリング機能の独立性」で示したように、ext3 においても独自のカーネルスレッドの実装によりライトバックによる遅延書き込みが行われている[8]。つまり、ジャーナリングのコミット部分に関しても遅延操作により性能向上が図られている事が指摘できる。

また信頼性保護の観点からこの値は 5 秒と低く設定されているがハード機能がそれに見合うスループットを提供できる場合性能が向上することが指摘できる。

4.3.2 カーネル2.2 から 2.4 への改善

性能測定調査を元に、カーネル 2.2 と 2.4 の性能の比較を Read/Write 操作時についてまとめた。

表11 : カーネル2.2 から カーネル2.4 への改善比

[単位:100%: 2.2 を1とした場合]

	Write 1	Write2	Read1	Read2
	Per Chr	Block	Per Chr	Block
ext2	1.8	1.8	1.4	2.9
ext3	7.2	13.0	0.8	2.0
ReiserFS	2.8	4.2	0.8	2.0

表より ext3, ReiserFS のキャラクタベースの Read 以外は全て 1.5 倍から 13 倍という高い伸びを示している事が分かる。この改善の原因としては幾つかの要因が考えられる。それらを次に示す。

(1) ファイルシステム毎の改善による性能向上

はじめにバージョンの変化による性能の向上が指摘できる。ext3、ReiserFS、JFS のファイルシステムはそれぞれ速いスピードで開発が進み、カーネル2.4 時にバージョンアップが行われた。特に ext3 では、カーネル2.2 の際に非常に書き込みの性能が低かった事から大幅な改善がなされた。また、これらの改善時にほとんどのファイルシステムで gcc の拡張命令であるインライン関数による最適化が追加された事などが指摘できる。

また、特にカーネル2.4 に関してはカーネル本体の改善の影響が大きい。カーネル2.2 からは本体のI/Oサブシステムが大幅に書き換えられた。ここでは変更された部分と改善内容について述べる。

(2) カーネルサブシステムの改善

・インライン展開する関数の導入による速度向上

カーネル 2.4 では、ファイル関連オブジェクトのカウントを行う部分がatomic_add(),atomic_sub(),atomic_set(),atomic_del()といった関数を使用した形へ大量に書き換えが行われた。これらの関数は内部的にはアセンブラで記述されており、インライン命令で処理されているため高速である。特に、inode オブジェクト構造体の i_count, ファイルオブジェクト構造体の f_count, dentry 構造体の d_count といった参照頻度の高いオブジェクトの扱いに関して大量に用いられている効果は大きいといえる。

・書き込み、読み込み操作の統一

カーネル2.4 では、ページキャッシュとバッファキャッシュの操作が統一された。バッファキャッシュと、ページキャッシュは、前者がブロックデバイスのための I/O の非同期を実現するためのキャッシュ、後者は仮想記憶の単位であるページフレームのキャッシュを行うものであるが、カーネル2.2 までは、読み込みと書き込みで別のキャッシュが使用されていた。

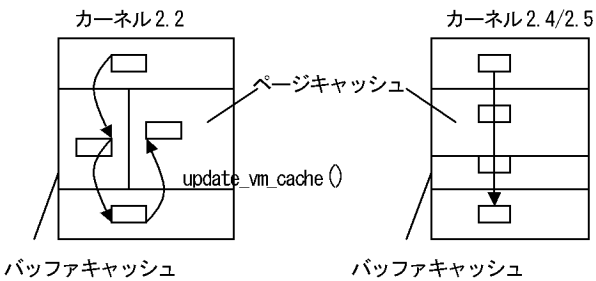

図5 : バッファキャッシュとページキャッシュ

このため、いずれかの処理を行う場合にはキャッシュ同士のデータの整合性をとるためにコピー処理が必要となり、(図 5: update_vm_cache() 操作) オーバーヘッドが生じていたものが、今回統一された。

・サブシステムインターフェイスの統合

inode, dentory, filep, page 等の主記憶のカーネル内にキャッシュされるオブジェクト部分のインターフェイスがハッシュテーブルで管理されることになった。

従来もキャッシュオブジェクトの管理にはハッシュによるリストが使用されていたが、カーネル 2.4 では部分的ではなく、統一的にインターフェイスが書き換えられた。ハッシュによる検索によりキャッシュに対しての要求が生じた場合には高速に目的のオブジェクトにアクセスする事ができる。また、ハッシュインターフェイスに統合されることで、引数の inode 番号、ブロック番号を渡せば目的のオブジェクトが得られ、カーネルの API としてもより汎用的なものとなった。

5. メモリ、キャッシュの影響

ここまでのところで、ファイルシステムの基本的な性能と、それがカーネル 2.4 で大きく向上したことについて述べた。次に性能が、メモリなどの要件によってどの程度変化するのか、ここではメモリを大量に搭載するマシンで、検証を行った。

5.1 検証内容

メモリの変化量に応じてファイルシステムの性能の変化をみるために、メモリを 256MB から 4GB まで 256MB ずつメモリ量を変化させ、性能テストを行った。

表 12: 検証機ハードウェア構成

CPU	Memory	HDD	Contoller
Pentium III 700MHz (x8)	4GB	SCSI(Ultra2)	QLogic

5.2 検証結果

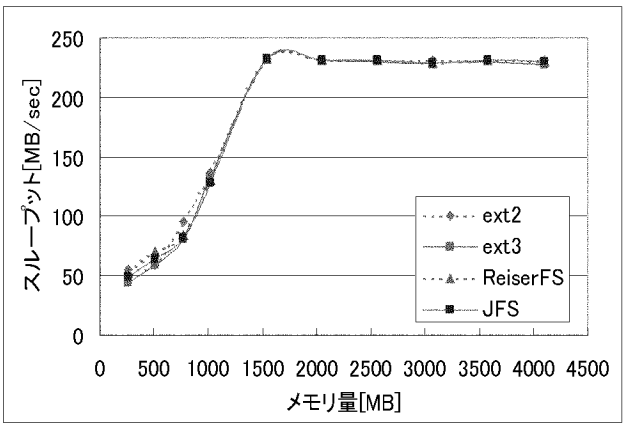

図6 : メモリを増加させた場合の性能の変化 (Read)

ブロック単位での Read では、いずれのファイルシステムでも最大 240MB/sec の性能が示された。また、図6をみても明らかなように、1.5 GB まではメモリの増加は性能に寄与するが、後は向上していない事が指摘できる。

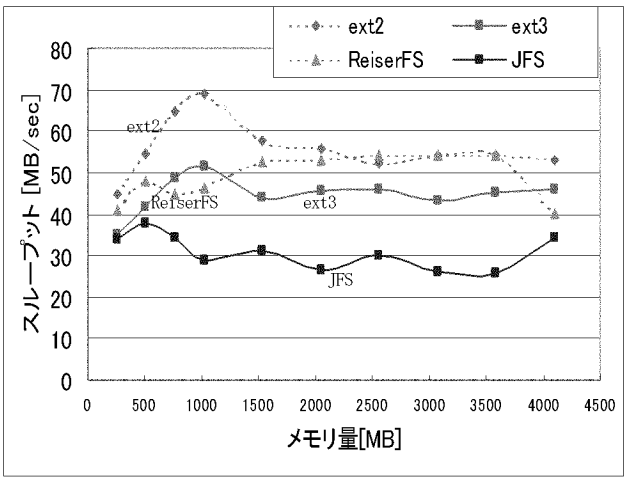

図7: バッファのフラッシュ値変更による性能の変化 (Write)

Write に関してはファイルシステムごとにばらつきがある。先ほどの図5の結果と比較すると、ext2 の値が最も高くなっている。ext2, ext3 では 1G バイトまでは性能に寄与するが、後は低下する傾向にある。JFS はメモリ量の変化による影響がみられない。

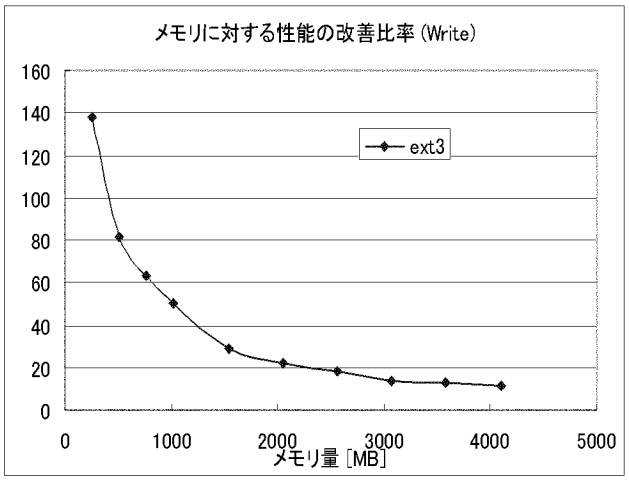

図8: メモリに対する性能の改善比率 (ext3)

次に改善比の増加分をみると、図8の通りとなる。1.5 GB 以降はy(スループット)軸の増加分は、x(メモリの増加分)を下回る値となる。表 13 のように 2GB までの平均増加率をまとめた。

表 13 : スループットの改善率の比較

	Write (Block)	Read (Block)	Write (Char)	Read (Char)
ext2	94	148	17	16
ext3	72	135	16	16
ReiserFS	78	145	15	16
JFS	60	140	13	16

表13より

(1) ブロックベースのRead/Write 共にext2 の性能に対する改善率が最も高い。

(2) キャラクタベースに関しては、大きな差はないがWrite についてはext2 が最も性能改善率が高い。

メモリに対する性能改善率は、大きい順から ext2, ReiserFS, ext3 、JFS となった。ここでメモリに関して改善比率の高いファイルシステムの順位は、信頼性の結果と逆の順である事が指摘できる。つまり、メモリを増やした際に性能が向上する傾向が大きいファイルシステムほど、キャッシュなどのメモリを効率的に使用する実装となっており、この事によって逆にクラッシュが生じる際のRAM メモリへの依存度のために信頼性にあたる復旧率が低下してしまうことが指摘できる。つまりメモリに対する改善率と信頼性はトレードオフ要件であることが指摘できる。

通常物理RAM メモリはファイルシステムではバッファキャッシュとして利用されるため、ここではキャッシュの利用効率とする。

5.3 トレードオフの解消 フラッシュタイム操作

このトレードオフを解消には、復旧率の値を改善し、かつ性能を保つ（向上させる）という形で少なくとも双方の値の改善できる事が望ましい。「2、復旧率」みたように、復旧率を向上させる方法としては、フラッシュタイムの操作が有効であった。これは、遅い二次記憶媒体への書き込みを、速いRAM メモリへの書き込みによって一時的に代用し遅延を隠蔽する「非同期書き込み」の際に、バッファキャッシュをディスクへフラッシュするフラッシュタイム(age_buffer の値)を短くすることにより、復旧率を改善させる方法である。

「2. 信頼性の検証」において、フラッシュタイムは復旧率の改善に寄与した。ここではフラッシュタイムの変化が性能に影響を及ぼすのかを調査した。

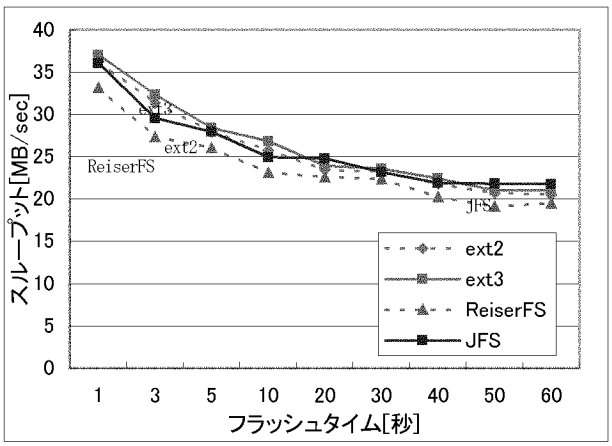

図9: フラッシュタイムとスループット (Read(block))

Readに関しては、図9よりバッファフラッシュは短い方が性能がよいことが示された。デフォルトでは30秒となっているので、それよりも値を小さくする事で最大60%の改善が見込まれることが示された。

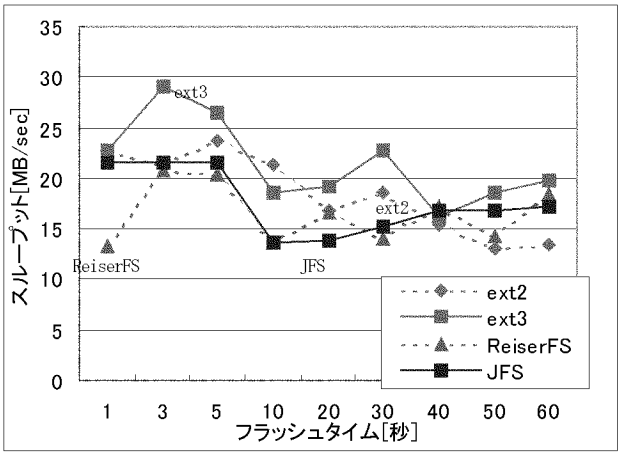

図 10: フラッシュタイムとスループット (Write(block))

また、Write についてはデフォルトの 30 秒の設定よりも値を小さくする事が、全体的に有効である事が示された。特に 10 秒以下の方が性能の改善率が高い。また、10秒以下では JFS 、ext2, ReiserFS でも改善が見られている。ただし、3 秒より小さい1秒とした場合には逆に性能が悪化する傾向となった。これらの値はハードウェアの速度やバス、ディスクのスピード、さらにシステム全体の性能に依存する内容であると考えられるので、実測しながら最適な値を選択する事が望ましいだろう。

このように、ここではバッファのフラッシュのタイミングの操作を行う事でファイルシステムの信頼性を上げるのみならず、性能に関しても向上が見込まれた事を示した。

5.4 性能向上のためのその他のアプローチ

ハイアベラビリティマシンでのテストを通じ、ファイルシステムによってキャッシュの利用率が高いファイルシステムであるほど高い比率で性能が向上する事を確認した。ただし、キャッシュの利用率が高いファイルシステムは逆に信頼性の評価が低く、トレードオフの関係がみられた。さらに、バッファフラッシュの値を変化させる事により、このトレードオフを解消することができる方向を示すことができた。

ここでは、さらに信頼性を保ちつつ性能を向上させる方法について考察する。具体的には変更可能な値を操作することで現在の性能を引き出す方向性を探る。

5.4.1 ユーザーモードからのチューニング

カーネルには、ユーザモードからのチューニングパラメータが複数用意されており、ハイエンドマシンを利用す

る際には変更を行う事ができる。ここではカーネル内で使用されるパラメータを利用し、改善を達成する事を目的とする。

5.4.2 パラメータの検討

カーネルには、キャッシュの扱いにおいて複数の操作パラメータが存在する。ここでは特にカーネル2.4についてパラメータについて影響度を調べることで、性能向上に寄与度が高い値について考慮する。特に次の値の影響について調査を行った。

[考慮点]

1. 先読みキャッシュの増加による、先読み効果
2. ページのクラスタリング数の増加による効果
3. 上限値をページサイズの2の累乗倍とする効果

表14 : ファイルシステムの性能評価

パラメータ	内容
nfract	bdflushが起動されるための更新されたバッファの閾値（単位%）
age_buffer	更新されたバッファをディスクへ書き込むタイムアウト値（tick）
nfract_sync	kupdateが起動するまでのバッファの閾値
kswapd	swap-clusterは、vm.kswapdの3番目の値。が一度に書き込むページ数。この値を大きくする事でディスクのシーク数を減らす。
page_table_cache	vm.pagetable_cache ページテーブルキャッシュの値
over-commit	vm.overcommit_memory メモリのオーバーコミット許諾
max-readahead	vm.max-readahead ブロックの先読みバッファの最大値
min-readahed	vm.min-readahead ブロックの先読みバッファの最小値
page_cluster	vm.page-cluster ページのクラスタリング数

※上位3つは、vm.bdflush 1,6,7 番目の値

5.4.3 測定結果

上記の値を元に、複数性能テストを行った。テストの結果、swap-cluster で指定した、一度に書き込むページ数の操作を行うことで改善がみられた。特にJFSに関してはディスクに書き込むページ数はデフォルトで 8 ページのところを 16 ページと 2 倍にすることで図のようにWrite時の性能が向上した。また、

・kupdate の起動までのバッファの閾値を下げる事により、RAM の利用率が高いファイルシステムの性能が悪化

・ 書き込み対象のファイルの容量を100M（今までのテストの1/10 ）にしたところ、ext2 の最大で 120MB、JFS で105MB、ReiserFS で60MBまで性能が改善

などが観測された。これらのパラメータの影響はさまざま見られるので、今後これらの因果関係も含めて継続して調査して行きたい。

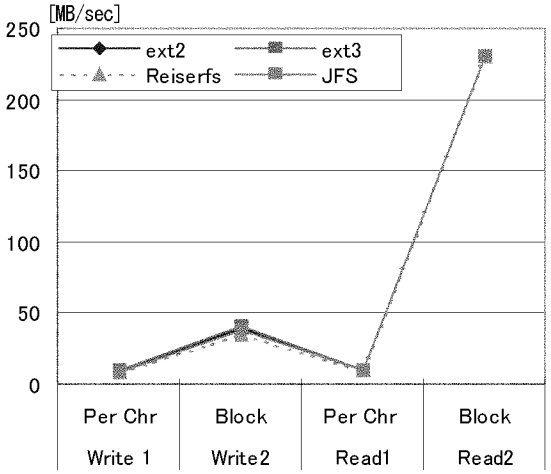

図12: swap-cluster 数の増加による JFS の改善

6. ファイルシステムまとめ

現在までの結果を踏まえて、ファイルシステムの評価と検討を行った。全体としてカーネル2.4 では信頼性と性能の向上が両立してきている事が指摘できる。結果は次の表のようにまとめることができる。

6.1 比較

表15: ファイルシステムの比較

	ext2	ext3	ReiserFS	JFS
設計	Block	Block +JBD	B+Tree	エクステント B+Tree 動的 inode
メリット、デメリット	Linux ネイティブ 長時間使用によるブロックの内部断片化	JBD レイヤーの分離 長時間使用によるブロックの内部断片化	検索、追加削除などの高速操作	スケーラビリティ 信頼性
ジャーナリング機能	なし	JBD	Fast Journaling	Journald File System
キャッシング	高	中	中	中[3]

[3] ジャーナリング部分でのキャッシュは利用していないが非ジャーナリング部分で利用しているため「低い」と断定できない。

機構の利用				
システム機能からの独立性	低	中	中	高
信頼性	低	中	中	高

3項、4項、5項のテストを通じて「信頼性」と「性能」という視点でファイルシステムの検証を行った。

この中で信頼性の低いシステムはメモリ利用効率がよく、メモリの増加に対する性能改善率が高くなる傾向がある事、またジャーナリングシステムはバッファやキャッシュから独立して機能するために信頼性が高いが、性能の向上に関してはやや低く、個別の対処などが有効である事を示した。

バッファキャッシュは起動時に通常 10%確保される[6]事から、物理メモリ量が増大すればバッファキャッシュの使用効率が高いファイルシステムは自然に性能に反映する。こうした改善に対して、5 項では信頼性を追求するファイルシステムでも、バッファのフラッシュタイムの処置を適切に行う事で、トレードオフを改善し性能と信頼性の双方を改善できる事を示した。

6.2 信頼性と性能の今後

信頼性と性能はディスクの遅さという決定的な要因をもとに十分に両立しないとされてきた。但し、今回のテストに関していえば、ジャーナリング機能を持つファイルシステムが設計に優れる事で両者がLinuxにおいても、両立してきている事が指摘できた。

特にJFSでは開発サイドでもLinuxへの移植に際してバッファキャッシュからページキャッシュへの移行に合わせた設計の変更を打ち出している事や [9]、次に示すReiserFS なども性能に関してより積極的な方向性が示されているなど期待が持てる。

6.3 カーネル2.5におけるファイルシステムの性能

カーネル2.5 における改善に関しては、kio-buf の実装でさらに効率性が高まるという報告もあり[7]、取り込みが待たれる。カーネル2.5でも性能調査は行ったものの、I/O を含むサブシステムでは大きく変更はされておらず、結果の数値に関しても大きな変更はなかった。唯一ReiserFS のRead/Writeで性能が向上している結果となった。

ReiserFSでは新たにバージョン4が開発され[11]、この 2.5 への取り込みが起因していると思われる。ReiserFS4ではさらに”fast journaling”機能が強化されており、今後は信頼性に関しても期待される。

7. 今後の課題

本稿では、信頼性という指標の検討、ジャーナリングシステムの検証、及びファイルシステムの性能比較を行った。検証によってファイルシステムごとの信頼性と性能について、複数の観点から調査を行う事により、ファイルシステム毎の差異を明らかにし、6 項で示した課題としてまとめた。

今後はこれらを継続的に調査し、ボトルネックの改善、開発要件へと発展させてゆきたい。さらにテスト方式のパッケージングなどを通じて、ファイルシステムの調査についての汎用的な手段や指標を提示できるようにしたい。より幅広い条件設定や設計も含め今後の課題としてゆきたい。

[参考文献]

[1] Ricardo Galli, Journal File Systems in Linux, UPGRADE, Vol.II, No.6, December 2001.
[2] Rosenthal, David S. H. ″Evolving the Vnode Interface,″ Proceedings of the Summer 1990 USENIX Conference. June, 1990, pp. 107-118.
[3] Erez Zadok, Ion Badulescu, Stackable File Systems as a Security Tool, January 1998
[4] Christian Czezatke, M. Anton Ertl, LinLogFS - A Log-Structured Filesystem For Linux, Usenix 2000 Freenix Track, pages77-88
[5]フィリップ・A・バーンスタイン、エリック・ニューカマー「トランザクション処理」1998, 日経BP
[6] ユーレッシュ・ヴァハリア著,『最前線UNIXカーネル』,ピアソン・エディケーション, 2001
[7] Stephen Tweedie, EXT3 Journaling Filesystem, OLS2000, Released on 31 October, 2000.
[8]Daniel Robbins, Advanced File System Implementor’s guide, Part7. Introducing ext3. November 2001.
[9]SteveBest,” How the Journaled File System handles the on-disk layout.May 2000
[10]LinuxJapan,Linuxファイルシステム最新事情, 2002.2
[11] Namesys, http://www.namesys.com
[12] 岡島順次郎, Journaling Filesystem, LinuxWrold 2002, VA Linux Kernel Forum.