2006－OS－103　(8)
2006／7／31

ログ構造化ファイルシステムの同期書き込み性能の最適化

小西 隆介[†]　佐藤 孝治[†]　天海 良治[†]　木原 誠司[†]　盛合 敏[†]

要旨

我々は，ユーザデータ喪失の機会を減らすため，Linux 向けにログ構造化ファイルシステム NILFS を開発している．既存ジャーナリングファイルシステムとの比較を通じ NILFS の同期書き込みにおける性能上の課題として上書きと I/O バリア適用時の性能低下を抽出した．これら課題に対し，本稿では上書きデータのセマンティクス上の順序制約を一部緩める最適化と，ディスク上の書き込み順序の制約を緩める最適化を検討し，それら手法による同期書き込みのスループットの改善効果とトレードオフを検証した．

Performance Tuning of a Log-Structured File-System for Synchronous Writes

Ryusuke KONISHI[†]　Koji SATO[†]　Yoshiji AMAGAI[†]　Seiji KIHARA[†]
Satoshi MORIAI[†]

Abstract

To reduce the possibility of user data loss, we are developing the New Implementation of a Log-structured File-System (NILFS) for Linux. By comparing with an existing journaling file-system, we have clarified two performance problems with synchronous write on the NILFS: a low overwrite performance and degradation due to write barrier. To resolve these issues, we introduce two optimizations, one of which relaxes order semantics for data overwriting and another relaxes a restriction on write order on disk. We verified their improvement effects and trades-offs.

1. はじめに

オープンソースの OS の利用が進むにつれ，これら OS においても信頼性・可用性がより求められるようになっている．これら OS の信頼性の課題の一つにファイルシステムの信頼性がある．例えば，オープンソースの OS である Linux Operating System (以下 Linux[1])は，かつては障害時にデータ喪失が容易に生じる，あるいはファイルシステムの首尾一貫性が損なわれ，その結果復旧に時間を要してしまうなどの問題があったことから，業務への適用をすすめることは困難であった．Ext3 ファイルシステム(以下 Ext3)[1]をはじめとするジャーナリングファイルシステムの採用でこの点は著しい改善があった[2]が，その後もディスクに対する同期書き込みが正しく行われない場合がある[3]など，障害時のデータ喪失の原因となり，またデータベースなどのトランザクションシステムの信頼性を左右する不具合が見られた．しかし，開発者の努力と専心的な取り組み[4]によって，今に至るまで多くの問題が取り除かれてきている．

このような取り組みに加え，我々は Linux 上のデータ喪失を更に減らすべく，ログ構造化方式[5]に基づくファイルシステムである NILFS (the New Implementation of a Log-structured

† 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．

File-System)の開発を進めている[6]. ログ構造化ファイルシステム(以下 LFS)は，ファイルやディレクトリの中身であるユーザデータとそれらの管理用の情報であるメタデータを，ディスク上で上書きするのではなくログのように追記していくことを特徴とするファイルシステムである．LFS では，ファイルシステムの過去の状態がディスク上に残っており，過去のデータの復元に適している．また通常のファイルシステムで過渡的に生じうるデータとメタデータ間の矛盾した状態の発生を回避することが可能で，ジャーナリングファイルシステムと同様，障害時の復旧時間も短く, 高い可用性を実現できる．

NILFS では LFS のこれらの特徴を活かし，ユーザのデータが高い確率で救済可能であり，かつ，通常のファイル操作と同等の容易性で過去のデータが救済できるような，ユーザデータ最優先のファイルシステムの実現を目指している．ユーザデータ喪失の要因には，システムクラッシュ, 不意の電源断, ユーザのミスオペレーションなどがあるが，我々は，NILFS 開発を通じて，障害時の復旧だけでなく，ユーザのミスオペレーションも含めたデータ損失の機会を最小にしたいと考えている．

一方，ファイルシステムにおいて信頼性・可用性を重要視することにより，性能が犠牲になる場合がある．例えば，ジャーナリングファイルシステムは，ディスクへの書き込みをジャーナルファイルと実領域の２箇所に対して行うので，原理的に書き込み性能が低下する．そのため通常は，メタデータのみをジャーナルファイルに書きリカバリ可能とする方法が採られる．

また，ジャーナリング方式の信頼性はディスクプラッタ上での実際の書き込み順序に強く依存しており，ブロックレイヤ，デバイスドライバ，ディスクドライブのどこか一箇所でブロックの書き込み順序が入れ替わると，正しいリカバリが保証できなくなる．その場合，リカバリ処理がスーパブロックのような重要ブロックを破壊してしまうことすらある．この問題にはファイルシステムを実装する上で留意すれば回避できるものと，ディスクコントローラのアウトオブオーダ実行のようにハードウェアに依存したものがある．後者に対しては，ハードウェアサポートを利用した Write barrier (以下バリア)と呼ばれる順序制御を適用することになるが，最近まで Linux では，この機能は限定的にしかサポートされていなかった[2]．また，ファイルシステムレベルでバリアを標準状態で有効にしているのは，XFS[7]だけであり，定着しているとは言い難い．バリア機能の適用は，ディスクシーク時間を最小化する最適化の制限や，場合によってはディスクキャッシュの強制フラッシュを伴うので，性能面にも影響がある．

本稿では，NILFS に関し，前述のデータ優先の考え方と信頼性・可用性を前提に，性能面に目を向ける．すなわちジャーナリングファイルシステムを比較対象としながら，NILFS の現状の性能を評価し，課題を抽出する．その中で前述のバリア導入の影響についても評価を行う．そしてその結果を踏まえ，書き込み性能を改善する方法を検討する．

2. ログ構造化ファイルシステム NILFS

NILFS は，データブロックおよび inode の管理に B-Tree 構造を採用し，64-bit ベースで設計された Linux 向けログ構造化ファイルシステムである[6]．NILFS では更新されたブロックは，セグメントと呼ばれる単位に区切られ，連続的に書き出される．我々はセグメントという用語を多義的に使用しているが，本稿ではファイルシステムの状態の変更に対して atomicity を与える単位の意味で用い，以下これを論理セグメントと呼ぶ．NILFS の論理セグメントは図 1 のようにセグメントサマリというヘッダに，新規に作成された，あるいは変更のあったデータブロックとメタデータブロックが，データブロックから始めて参照される順番に書き足された構造になっている．

NILFS 上のファイル及びディレクトリのデータブロックは，File B-Tree を介して inode により保持される．Inode 自体もファイルと同様に

[2] Write barrier が各種ドライブに対し機能するようになったのは Linux 2.6.16 からである．

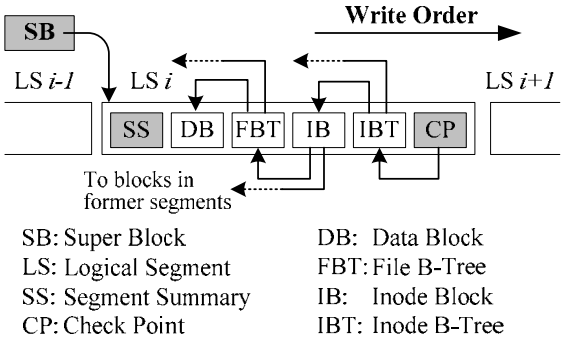

図 1　NILFS-1.0 のディスクレイアウト

B-Tree(Inode B-Tree, IBT)により管理されており，ファイルシステムのそれぞれの一貫したスナップショットが結局チェックポイントというブロックから辿れるようになっている．ファイルシステムの最新状態は，最新のチェックポイントにより与えられる．チェックポイントを含む論理セグメントへのポインタはスーパブロックに格納され，これは定期的に更新される．不意の停電などの際には，このポインタを基点に最新のチェックポイントを探索し，状態を復旧する．リカバリの際には，論理セグメントが完全に書かれているか判別する必要があるが，NILFS はこれを CRC チェックサムにより行っている．CRC はセグメントサマリとチェックポイントに加え，データ重視の観点からセグメント全体に対しても付与している．

3. NILFS の性能上の課題

3.1. 現行バージョンの評価

現行の NILFS(NILFS-1.0)の性能面の課題を抽出するために，iozone benchmark プログラムを利用して性能測定を行った. 測定に使用した PC サーバのハードウェア構成と使用ハードディスクの諸元データを表 1 に示す．本稿では主に測定機 A(PC-A)を使う．測定機 B(PC-B)はディスクによる違いを見るために使用する．Linux カーネルは version 2.6.16 を使用した．NILFS は version 1.0.9 を，また iozone は version 3.257 を使用した．比較対象としては Ext3 を用いたが，ジャーナリングモードはデフォルトの ordered モードである．なお NILFS-1.0 には，連続空き領域を作成する Garbage Collection (GC, Cleaner) と呼ばれる LFS の必須機能が備わっていない．従って, 本稿での性能データは Ext3 に対する優位・不利を論ずるものではなく，改善点を抽出するための参考データであることを予めお断りしておく．GC については次のメジャーリリース(NILFS-2.0)で対応する予定である．

図 2 左に同期書き込みのスループットを示す．Ext3 と NILFS のそれぞれについて新規書き込み(write), 上書き(rewrite), ランダムライト(rand. rewrite)のスループットを示している．テスト用ファイルのサイズは 128 Mbytes とした．書き込みの同期方法は，テスト用ファイルを O_SYNC オプション付きでオープンする方法を用いている．このモードでは，write システムコールの呼び出しごとにディスクへの書き出しが強制され，システムコール完了時には，データは二次記憶装置に記録されていることが保証されなければならない．本測定では従って，横軸のバッファサイズは同期書き込みの単位と一致する．同期書き込みの方法には，他にも同期のタイミングをユーザがシステムコールで制御できる fsync() などがある．同期書き込みの単位を測定の基準に考える意味では O_SYNC で十分であったので，本稿ではその他の方法は取り扱わない．この同期書き込みの測定条件に関しては，以後の測定データに関しても同様である．

図 2 左の結果では，NILFS のスループットは新規書き込みとランダムライトで Ext3 を上回っている. これは, NILFS では, 全てのブロックの書き込みが追記的に行われ，ディスク上の

表 1 測定マシンの諸元

	PC-A	PC-B
CPU	Pentium-4 3.2GHz	Dual Core Xeon 2.8GHz×2
Memory	2048MB	4096MB
Bus Controller	Intel 6300ESB SATA Storage Controller	LSI-Logic53c1030 Fusion-MPT Ultra320 SCSI HBA
ドライブ	Maxtor 6Y080M0	Maxtor ATLAS15K2_73SCA
容量	80GB	73.4GB
Buffer Size	8BM	8MB
回転数	7,200rpm	15,000rpm
シーク時間	9ms	3ms

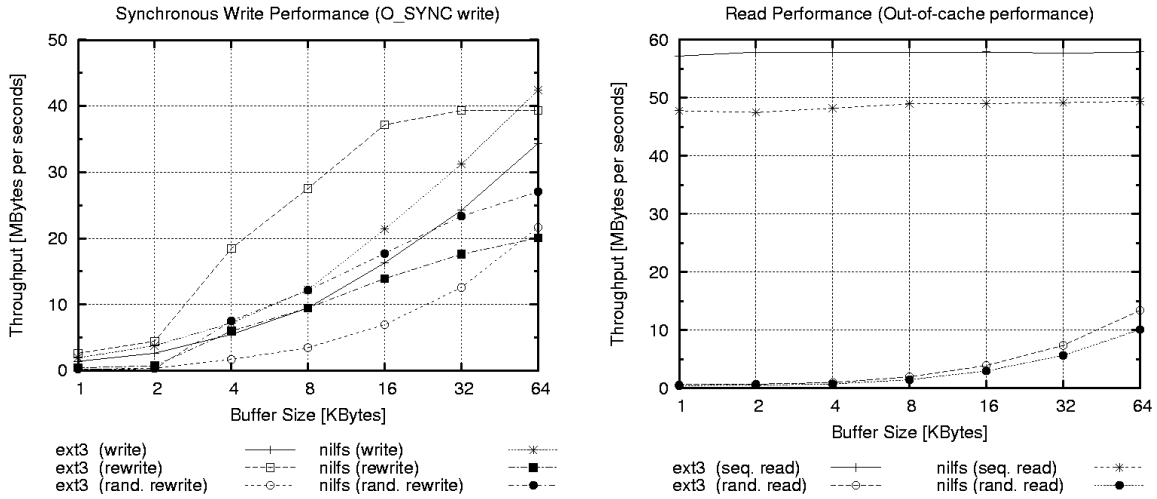

図 2 NILFS-1.0 と Ext3 の同期書き込み性能及び読み出し性能の比較

表 2 バリア適用の有無と同期書き込みのスループット [Mbytes/sec]

書込方法	Buffer Size	PC-A (SATA)				PC-B (SCSI)			
		Ext3 (ordered)		NILFS-1.0		Ext3 (ordered)		NILFS-1.0	
Write Barrier		OFF	ON	OFF	ON	OFF	ON	OFF	ON
Write	4KB	5.490	0.115	7.182	0.232	0.310	0.269	0.917	0.478
	64KB	33.405	2.053	42.606	3.511	4.553	4.559	12.376	6.990
Rewrite	4KB	19.550	19.352	6.302	0.231	0.986	0.986	0.915	0.478
	64KB	43.492	42.794	19.972	3.379	15.321	15.290	11.890	6.841
Random rewrite	4KB	1.944	1.833	7.528	0.231	1.230	1.240	0.913	0.477
	64KB	22.506	21.676	27.162	3.407	15.355	15.409	11.826	6.849

書き込みも連続的に行えるためと考えられる．一方上書きに関しては, 測定範囲で Ext3 が 4KB 以上で 2〜3 倍, 4KB 未満で 6 倍程度高いスループットを示しており，大きな開きが見られた．

同様に Ext3 と NILFS について，既存ファイルの逐次読み出し(seq. read)とランダム読み出し(rand. read)を行った時のスループットの測定結果を図 2 右に示す．

結果，NILFS の逐次読み出しとランダム読み出しのスループットは，前者が 15〜18%, 後者が 25%程度 Ext3 に劣っている．ただし，この測定では実際のディスク I/O 性能を見るため，データ部分がキャッシュにヒットしないよう，測定毎にディスクパーティションを一旦アンマウントしてページキャッシュを強制的にフラッシュしている. フラッシュを行わない場合には，キャッシュの効果により両ファイルシステムの測定結果はほとんど変わらなくなる．

3.2. I/O バリアの影響

次にバリアの有無による同期書き込み性能の違いを評価した結果を表 2 に示す．Ext3 では，ジャーナルファイル上にメタデータを記録した後，その完了を示すコミットブロックの書き込みにバリアが使用されている．バリアを指定したブロックの書き込みは，図 3 に示すように，ブロック I/O 層やディスクドライブで前後のブロックと順序の入れ替えが起きないように実行される．コミットブロックにバリアを適用することで，全てのメタデータブロックがジャーナルに書き終わってから，コミットブロックが書かれることが保証され，ブロック書き込みがどの時点で中断しても，論理的に正しいリカバリが可能になる．ただし最初に述べたように Ext3 はデフォルト状態でこの機能をオフにしている．

表 2 の結果によると，PC-A では Ext3 の新規の書き込み(write)で著しい性能の劣化が生じて

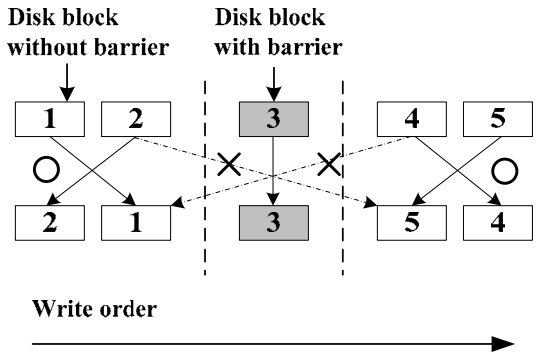

図 3　I/O バリアによる順序制約

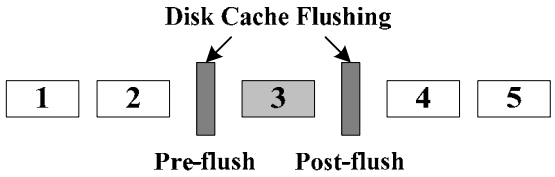

図 4 ネイティブサポートのないドライブでの I/O バリアの実現方法

いるが，上書き(rewrite, random rewrite)に関して差異は見られない．一方 NILFS については全ての書き込みで著しい劣化が見られた．これは，Ext3 の上書きは，メタデータの更新を生じず，バリア書き込みが生じていないためである[3]．現行 NILFS-1.0 ではチェックポイントブロック(コミットブロックに相当)の書き込みにバリアを適用しているため，新規作成・上書きと関係なく，バリア書き込みが生じてしまい，結果としてバリア適用の影響を受けてしまっている．

また SCSI ドライブを搭載する PC-B では，Ext3 は新規書き込みの場合も影響が小さい．これは I/O の順序付け機能を SCSI ドライブ自体がサポートしており，バリア機能が効率的に実現されているためと考えられる．これに対し，IDE や SATA ドライブ[4]に対しては，通常図 4 のように，ディスクキャッシュを 2 回フラッシュすることで，バリア機能を実現している．性能への影響が大きいのはそのためと考えられる．ただし NILFS の場合, PC-A 程顕著ではないが，PC-B においても 40〜50%と大きな性能低下を招いている．

3.3. 課題の整理

以上の結果から，NILFS の性能上の課題として以下を抽出した．

(1) 同期書き込みの上書きの性能
(2) 同期書き込みのバリア適用時の性能
(3) 読み出し性能

(1)は，Ext3 がデータブロックの上書きだけで済む場合が多いのに対し，NILFS ではメタデータの更新がない場合でもチェックポイントも含めたセグメントの書き出しが行われてしまう(a), ディスク上で中間ブロック(B-Tree の構成ブロック)がフラグメンテーションを起こしている(b)，といった理由が考えられる．(3)についてもメタデータのフラグメンテーションが要因として考えられる．(2)はチェックポイント単位でバリアが適用されており, 同期書き込みの場合，頻繁にバリアを伴う書き出しが発生していることが原因と考えられる．

本稿ではこの内，(1)の(a)と(2)を取り上げる．これらは後述するようにディスクフォーマットに大きな変更を加えることなく実施可能であり，問題の切り分け点としても全体のマイルストーンとしても好適と考えたからである．

4. 同期書き込み性能の改善方法

前節の問題の改善方法として，本稿では次の３つのアプローチを考える．

(A) 書き出し順序のセマンティクスの緩和:
　更新データブロックのみを独立して書き出せるようにする．
(B) チェックポイント書き出し頻度の削減.
(C) バリア適用タイミングの見直し．

(A)は前節の課題(1)の(a)の軽減するためもので，(B)と(C)は課題(2)に対応するものである．ただし，(A)は(B)の効果も含むので，以下では(A)と(C)に絞って述べる．

[3] 厳密には inode の更新時刻が変わる場合にはメタデータの書き込みは生じる．タイムスタンプの精度は秒単位であるため，1 秒に一回はバリアを伴う書き込みが生じる．

[4] SATA-II では Native Command Queuing と呼ばれるハードウェアサポートが導入されつつある．

4.1. 書き込み順序セマンティクスの緩和

伝統的な UNIX ファイルシステムでは，ファイル操作やディレクトリ操作の発生順序はディスク上でも保たれる．例えば，同一のマウントポイント内で 2 つのファイル A, B をファイル A →ファイル B の順序で作成している最中にシステムが停止したとする. この場合, 復旧時にファイル A,B が存在するかどうかは不定だが，ファイル B が存在する場合にはファイル A も存在することが要求される．このような順序に関するセマンティクスは解釈次第で実装に大きな違いが出るが, 明確な定義は POSIX にも見当たらない．Ext3 など Linux のジャーナリングファイルシステムでは比較的厳しいセマンティクスが採用されており，特定のファイルやディレクトリを同期書き込みする場合には，その他の全ての操作をディスクに反映させる，つまり「特定操作の同期的実行」が実質「ファイルシステム上の全更新のディスクへのフラッシュ」となる．

Ext3 の ordered data モードでは，対象ファイルのメタデータに変更がなければ，データブロックの書き込みは，指定ファイル内では順序を厳密に保証するが，ファイルシステム上のその他のイベントとは独立した順序で(メタデータの更新よりは前倒しで)行われる．3 節で述べたケースで Ext3 が性能低下を免れているのはこの最適化が奏功しているためである．

NILFS-1.0 ではイベント順序を忠実に再現する厳しい順序セマンティクスを採用しているが，これは更新ブロックを区別せず，ある時点で区切って全て書き出すという LFS の書き出しをシンプルに実現する方法が，結果として順序セマンティクスの厳密解になっているだけである．そこで，ordered data モードと同様に順序セマンティクスを緩めることを検討する．具体的には

(1) 何らかのメタデータの更新がある場合には，通常通りチェックポイント込みの論理セグメントを作成させる．
(2) メタデータの更新がない場合は，上書きされたデータブロックとセグメントサマリのみを含む論理セグメントを作成する．

という変更を加える．

ここで，(2)のセグメントにはチェックポイントがないので，リカバリ処理のロールフォワードにより復旧可能でなければならない．NILFS はセグメントを構成する全ブロックの CRC (Data CRC)を取っているので，これによりセグメントの正当性を判断し，ロールフォワードを実現する．具体的には，CRC チェックにより全てが書き込まれたことが確認された論理セグメントのみをチェックポイント以後の有効な変更と見なし，復旧対象とする．そしてその論理セグメントに含まれるデータブロックを，File B-Tree が指し示すようにメモリ上でポインタを置き換える．後は通常のチェックポイントを含むセグメントの書き出しを行えば，(2)のセグメント中のデータブロックがチェックポイントから全て可達であるような通常状態に戻せる．

この変更は，データブロックの上書きの順序のセマンティクスを Ext3 の ordered data モード相当に緩める．しかし，データの書き込み操作の atomicity は依然保たれており，ordered data モードとは異なり信頼性は犠牲にしていない．

4.2. バリア機能の適用方法の見直し

現行 NILFS-1.0 では I/O バリアをチェックポイントに適用しており，図 5(a)のようにディスク上の論理セグメントの書き順は厳密にセグメントの順番どおりとなる．本節ではこれをスーパブロック毎に減らすことを検討する．スーパブロックはチェックポイントのある論理セグメントへのポインタを持っているが，更新頻度は高くない(Linux では 5 秒に 1 回). 従ってバリアによる性能低下の大幅な改善が見込める．

バリアの適用をスーパブロック毎とする場合以下の 2 点に注意する必要がある．

- リカバリ処理で論理セグメントの完全性の検証に，トレーラブロック(チェックポイント)ではなく Data CRC を使う必要がある．
- 図 5(b)のようにディスク上でセグメントの書き順の入れ替わりが考えられ，結果としてリカバリ不能な論理セグメントが増える可能性がある．

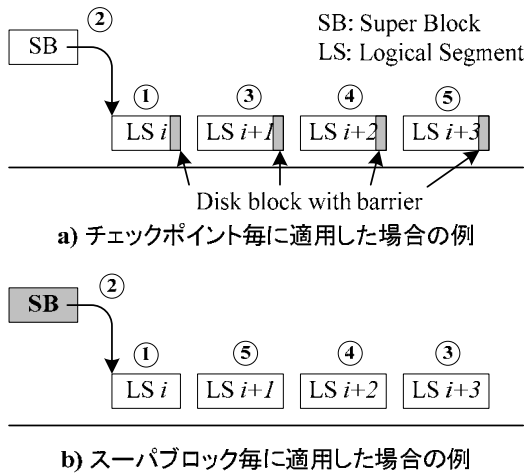

図 5　バリアの適用方法とブロックの書き順

例えば，図 5(b)で⑤の書き込みの途中で失敗すると,④と③のセグメントが救済できなくなる．これは信頼性を損なうように思われるが，前提条件を振り返って考えると実はそうではない．ディスクキャッシュ上のデータは通常揮発であり，そもそもバッテリバックアップなどの補助手段がなければ，不意の電源断に対しプラッタ上への書き込みは保証されない．ドライブがバッテリバックアップされるなどディスクキャッシュが不揮発であることを前提とすると,ディスクドライブがデータを受け取った時点で書き込みが完了したとみなせるので，ディスクドライブにセグメントの書き出しを要求する順序が保たれていれば，図 5(a)と(b)の場合で違いは生じない．そしてこの順序の維持は，NILFS 側でセグメントの書き出しに際して，前のセグメントの I/O 完了を待つことで充足できる．

一方，バッテリバックアップなどの補助手段がない多くのディスクドライブ上のデータに関しては,バリアは書かれているデータ･メタデータの一貫性の保証を与え，破壊的な振る舞いを回避するのに不可欠である．しかしこの観点では, NILFS では Data CRC による正当性のチェックが可能であり，バリア適用の粒度はスーパブロックの書き込み毎に行えば十分である．

5. 改善方法の評価

前節の改善策を適用した場合の同期書き込みの性能(どちらも PC-A で計測)を 3 節と同様に iozone で測定した結果を図 6,図 7 に示す．

まず図 6 の結果から，従来セマンティクス適用の場合(strict)に比べて，書き込み順序セマンティクスを緩和した場合(relaxed)，同期書き込みの上書きについて 1.2 倍(64KB 単位の場合)から 2 倍以上(4KB 単位の場合) の改善があった．Ext3 との差はまだ大きいが，同期単位が細かい部分では性能差の半分程度は改善できている．ランダムライトの場合にも 1.5 倍(64KB 単位)から 2.3 倍(4KB 単位)の改善が見られ，こちらは NILFS の性能面での強みを補強する結果を得た．

次に図 7 の結果から，バリアの適用をスーパブロックの書き込みに絞ることで，新規作成，

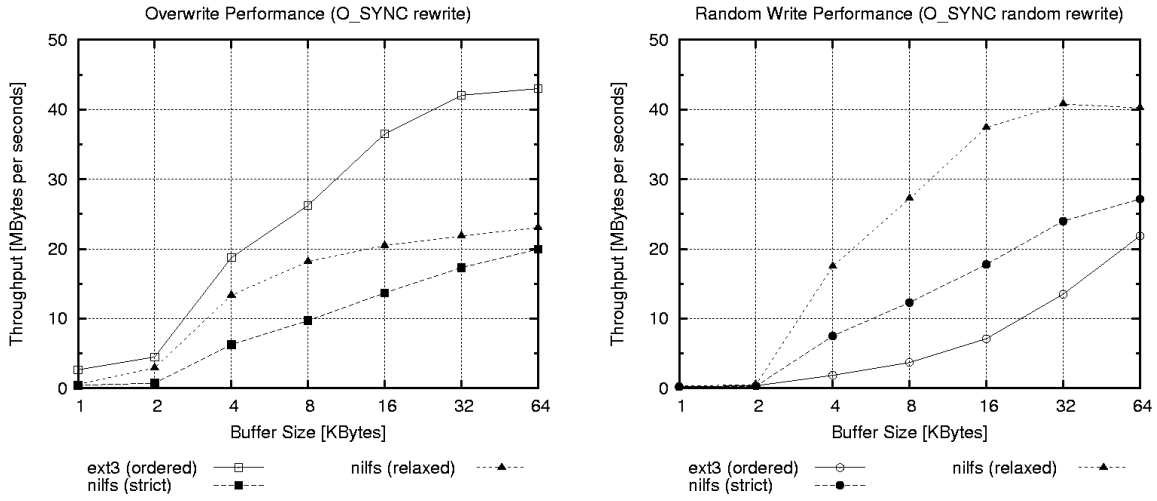

図 6 Relaxed order semantics 適用時の同期書き込み性能の改善

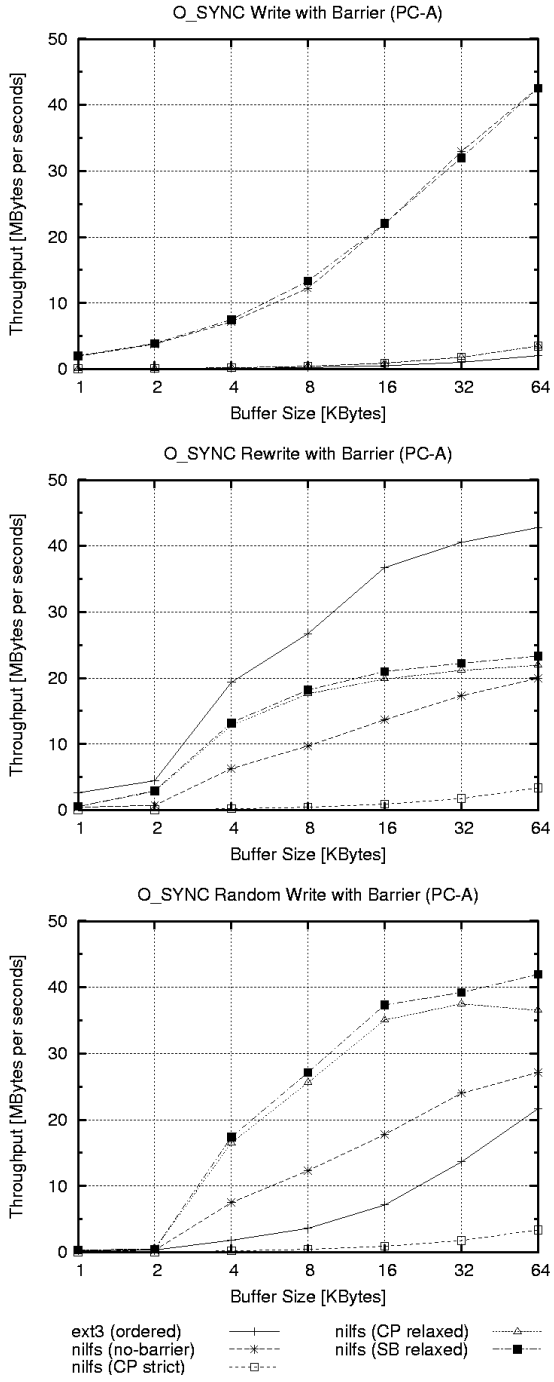

図 7 Barrier 適用時のオーバヘッドの改善

上書きの全ての場合 (write, rewrite, random rewrite)において期待通りバリアのオーバヘッドを回避できることを確認した(CP strict 対 SB relaxed). 上書きの場合には，順序セマンティクスを緩和する場合(CP relaxed)と同等にできる．

一方，バリア適用を省くと，リカバリ時に Data CRC をチェックする必要が生じ，リカバリの所要時間が長くなる．書き込み中にランダムなタイミングでリセットをかけた後，リカバリ時間(マウント開始から完了までの所要時間)を PC-A で実測したところ，100 回の試行で，平均 11.7 秒，+1σが 17.8 秒，+2σが 24.0 秒となった． NILFS-1.0.9 では 1 秒以内であったので無視できない低下を生じている．ただこれについては，セグメントの連続性を生かしてリカバリ時の読み込みをバルク的に行うなど，まだチューニングの余地はある．折衷案の検討と合わせて，引き続き改善に取り組んでいる．

6. まとめと今後の課題

本稿では，既存ジャーナリングファイルシステムを基準としながら，LFS の性能面の課題を抽出した．そして信頼性，リカバリ処理の実現性，ファイルシステムのセマンティクスを考慮しながら，その改善方法を検討し，有効性とトレードオフを検証した．

今後は，GC の実装や，今回取り上げなかった性能の最適化を進め，NILFS 開発を通じオープンソースに基づく高信頼なプラットフォーム技術の確立に貢献していきたいと考えている．

文　献

[1] S.Tweedie: “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] 小西隆介, 天海良治, 佐藤孝治, 真鍋義文, 盛合敏：「Linux ファイルシステムの信頼性についての一考察」, 情処研報, 2004-OS-96, pp.37-44, 2004.

[4] Project DOUBT. http://developer.osdl.jp/projects/doubt/

[5] M. Rosenblum and J.K. Ousterhout: “The design and implementation of a log-structured file system,” ACM Transactions on Computer Systems, 10(1): pp.26-52, 1992.

[6] 天海良治, 一二三尚, 小西隆介, 佐藤孝治, 木原誠司, 盛合敏：「Linux 用ログ構造化ファイルシステム nilfs の設計と実装」, 情処研報, 2005-OS-99, pp. 61-68, 2005.

[7] Project XFS Linux. http://oss.sgi.com/projects/xfs/.